UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

ON MULTIMEDIA CONTENT DELIVERY AND MULTICASTING

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

YUH-RONG CHEN
Norman, Oklahoma
2012

ON MULTIMEDIA CONTENT DELIVERY AND MULTICASTING

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

Dr. Sudarshan Dhall, Chair

Dr. Sridhar Radhakrishnan, Co-Chair

Dr. S. Lakshmivarahan

Dr. John K. Antonio

Dr. Suleyman Karabuk

To my parents, Chin-Hua Tseng and Chieh-Fu Chen

Acknowledgements

This dissertation could not have been finished without the help and support from a great number of individuals. I would like to express my deepest appreciation to all of you.

First of all, I am deeply grateful to my advisor and committee chair Dr. Sudarshan Dhall. He guided me through this wonderful journey and provided valuable feedback for the dissertation.

I also would like to express the deepest appreciation to my co-advisor Dr. Sridhar Radhakrishnan, for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research.

Furthermore, I would like to thank Dr. Suleyman Karabuk, who spent countless hours helping me develope Integer Programming models. My thanks also go out to other members of my dissertation committee Dr. S. Lakshmivarahan and Dr. John K. Antonio for their understanding, patience and valuable feedback.

I further like to thank the professors with whom I had taken classes, which had laid a solid foundation for my research work. I also would like to thank Dr. Qi Cheng, Dr. Sudarshan Dhall, Dr. S. Lakshmivarahan, Dr. Sridhar Radhakrishnan, Dr. Deborah Trytten with whom I had worked. I learned a lot from being a teaching assistant in their courses.

My special gratitude goes to my friends and colleagues: Tachun Lin, Muhammad Javed, Aravind Mohanoor, Amlan Chatterjee, Daniel Bailey, Asif M Adnan, Professor Tian-You Yu, Yih-Ru Huang, Chih-Yu Lai and Longman Weng for stimulating discussions and their readiness to help.

My special thanks are extended to the wonderful staff members in the School of

Computer Science at the University of Oklahoma, Barbara Bledsoe, Jonathan Mullen, Emily Pierce, Jim Summers, Virginie Perez Woods, Chyrl Yerdon, for their timely help, encouragement, and advice since my first day in this program.

I also sincerely appreciate my former advisor Dr. Jy-chyi Yuan at Fu-Jen Catholic University and Dr. Hsueh-Chih Chen at National Taiwan Normal University. I learned a lot from them when I was working on my Master's degree in psychology.

Last but not least, I would like to thank all my family members for their love, encouragement and support. Especially my parents, Chieh-Fu Chen and Chin-Hua Tseng, who raised me and supported me in all my pursuits; and my grandfather Song-Ching Tseng, brother Jhih-Gang Chen, sister Yu-Chin Chen who were always gave me their best wishes.

Table of Contents

List of Tables

List of Figures

Abstract

Multimedia content now contribute to a huge amount of the Internet traffic due to the popularity and availability of anytime anywhere Internet connection. Unlike the circuit-switched telephone network – in which necessary resources are reserved for communication between two parties at the time the connection is established, a packet-switched network, like the Internet, only guarantees the reachability when the connection between two parties is established. In other words, the end-to-end delay and available bandwidth between two hosts depend on the amount of traffic on the network. The communication paths between the participating hosts are also determined by the routing policies and hence are not under control of the participating hosts. Hence how to improve the performance of delivering multimedia content on the Internet has become an interesting research topic.

In this dissertation, we consider the problem of delivering multimedia contents using multicast wherein a group of participants are participating in the same communication session. We assume the networks are flexible such that the end hosts can specify the communication paths. A few examples of this type of networks are overlay networks and IPv6 network with source routing support. This problem is addressed from both routing and network traffic perspectives.

First, we assume a two-layer approach which includes a well-provisioned service overlay network and the regular Internet. The participants in the multimedia group communication can take the advantage of the service overlay network by connecting to the nodes in the service overlay network through the Internet. We consider two major assignment problems – Server and Client Assignment Problem (SCAP, Client-Server model) and Client Assignment Problem (CAP, Peer-to-Peer model) as well as

several variants of these problems. These problems are $\mathcal{NP}$-hard and we have developed polynomial-time heuristic algorithms to assign the participants to appropriate service nodes such that some real-time constraint(s) are satisfied and the number of service nodes involved are minimal. Integer programming (IP) models for solving these problems are also developed for performance evaluation purpose. Empirical results show that the solution quality of the proposed algorithms compares favorably with the optimal ones obtained from the execution of IP models, while keeping the execution times significantly low.

We have also considered the Multi-stream Multi-source Multicast Routing Problem (MMMRP). Given a network and a set of multicast sessions, each with one or more sources and multiple destinations. The goal of MMMRP is to determine multiple multicast tree for these multicast sessions on the given network in such a way that the overall residual bandwidth on the links that are shared among the trees is maximized. We prove that MMMRP is $\mathcal{NP}$-hard and apart from providing an IP formulation, we have also provided a heuristic algorithm *MMForests* which runs in polynomial-time. We compared and contrasted the performance of MMMRP with known algorithms for the multicast tree packing problem. Our exhaustive empirical evaluations show that our heuristic has a very low execution-time while achieving the optimal residual bandwidth. In addition, our heuristic is very scalable as it is able to produce results for networks with thousands of nodes, unlike the other ones which are based on Steiner tree heuristics.

Chapter 1

Introduction

Network Multimedia applications such as music streaming, video streaming or online gaming, have become popular nowadays because of affordable broadband and mobile networks. These applications now contribute to a huge amount of the Internet traffic. For example, Netflix streaming accounted for about one third Internet traffic during prime time in North America as of Fall 2011 [52]. Since the amount of multimedia data is usually large, there are issues to be addressed when transmitting multimedia data using the Internet. In this chapter, we provide an overview of the Internet, communication models for network multimedia applications, and the issues that arise when delivering multimedia content.

1.1 The Internet

The Internet is a packet-switched (or store-and-forward) network consisting of links connecting nodes such as switches, routers and end hosts [47, 53]. The data transmitted from an end host s to another end host t is divided into smaller units (packets). Each packet is sent from a node to another node and stored in the memory (buffer) waiting to be forwarded to the next node (hop) until it reaches the destination t. During the transmission, the next hop for a packet is determined by the routing policies of the network and hence it is possible multiple packets of a message from the same source s to the same destination t could travel through different paths and arrive at t out of order.

On the other hand, a circuit-switched network requires a setup stage before the

Table 1.1: A Comparison of Circuit-switched and Packet-switched Networks.

Item	Circuit Switched	Packet Switched
Call Setup	Required	Not Needed
Dedicated Physical Path	Yes	No
Each Packet Follows the Same Route	Yes	No
Packets Arrive in Order	Yes	No
Is a Switch Crash Fatal	Yes	No
Bandwidth Available	Fixed	Dynamic
Time of Possible Congestion	At Setup Time	On Every Packet
Potentially Wasted Bandwidth	Yes	No
Store-and-Forward Transmission	No	Yes
Transparency	Yes	No
Charging	Duration	Per Packlet
		From [53]

transmission begins. A path between s and t is established and all the packets of a message follow the same path. The major advantages and disadvantages of packet-switched and circuit-switched networks are shown in Table1.1. In this dissertation, we assume the network (Internet) is packet-switched.

1.2 Challenges

There are a number of challenges when transmitting data from one host to another on the network. The fundamentals of transmitting data using the Internet are addressed by communication protocols in the Internet protocol suite. IEEE 802 standards [34] define standards of services and protocols for lower layers (from physical layer to data link layer) and the higher layer protocols are developed by Internet Engineering Task Force (IETF) [35]. However, there are issues that are not practically addressed in these standards and protocols. For example, end-to-end Quality of Service (QoS) protocol and multicasting are defined but not widely deployed to the end users. There exist alternative approaches to address these issues such as using overlay networks (which will be discussed later). In this dissertation, we focus on two of the major metrics for measuring the Quality of Service: (a) network latency, which is the time taken

for a packet to travel from one node to another node and (b) bandwidth provisioning, which guarantees sufficient bandwidth for communication between two nodes.

Network multimedia applications, such as online games or online video chat, are sensitive to network latencies. Claypool et al. [19] have shown that user performance in the games is directly related to network latency. Massively online game traffic analysis by Chen et al. [10, 11] also shows that online game players may stop playing the game when the network latencies become too high. Hence, in order to have a better Quality of Service, it is essential to control the end-to-end latency within a desired bound. Note that this real-time requirement (desired bound) depends on the type of the application. For example, Cisco Systems suggests that one-way latency for Voice over IP should be no more than 150 milliseconds [18]. The requirement for online games varies from 100 milliseconds to 1 second [19].

Network latency (or end-to-end delay) is the time to transmit 1-bit of data from one node to another node. But usually, we measure it as the time to transmit a certain amount of data, one packet for example. First, the packet needs to be pushed into the medium and then travel to the destination . Generally, it consists of processing delay, queueing delay, transmission delay and propagation delay. The time to push the data into the medium is transmission delay and the time for the first bit to arrive at the destination is the propagation delay. These two types of delays are caused by the nature of using the medium to transmit data. On the other hand, processing delay and queueing delay are caused by the computational task and the number of packets queued in the buffer at the intermediate nodes (routers). There are also higher level factors that contribute to the end-to-end latency between two hosts, transmitting paths and network congestion for example. In this dissertation, we attempt to address the latency issue from *routing* and *network traffic* perspectives. Although we assume the use of *overlay networks* (defined in Section 1.4), which is running at the application layer, the results can be applied to anywhere from the

network layer to application layer of the Internet Protocol suite.

Routing

Various factors affect routing on the Internet, such as political issues, commercial issues or peering agreements between the networks. Ly et al. [43] conducted experiments on the PlanetLab [48] and showed that the routing on the Internet is not always optimal in terms of end-to-end latencies. This is the so called Triangle Inequality Violation (TIV). Based on this fact, sometimes it is possible to reduce the latency between two nodes on the Internet by one-hop detouring through a third node [42]. In order to satisfy the real-time requirement between two end hosts, there must exist path(s) satisfying this requirement in the underlying network. One of the main contribution of this dissertation is the development of algorithms to find alternative routing path(s) satisfying some given delay constraint(s) while keeping the number of hops minimal. By reducing the number of hops, we also can (a) reduce the total processing delay at the intermediate nodes on the path and (b) lower the rate of packet dropping which results in retransmission. As a result, more simultaneous session can be served by the same amount of resources.

Network Traffic

Since resources on a packet-switched network are shared, the amount of traffic flows through the path between two hosts will impact the end-to-end delay between them. The end-to-end delay also comes from the various delay sources we mentioned earlier. When there is more traffic, although the sum of transmission delay and propagation delay remains the same, other two types of delay could increase. Technically, transmission delay does not increase. But an Internet path consists of several hops. If there are other hosts transmitting data at each hop, an intermediate node has to wait until the channel is clear before it can start pushing the data into the medium. Fur-

ther, more traffic will also cause the processing delay and queueing delay to increase since there are more data needs to be processed at the intermediate nodes or routers. Therefore, it is desirable to reduce the traffic on the congested links of the network. Another main contribution of this dissertation is that we have developed algorithms to route traffic of different sessions through the network to maximize the residual bandwidth, which is the remaining bandwidth between two nodes. Hence the impact of the network traffic on the end-to-end delays can be reduced.

1.3 Group Communication Models

Given a packed-switched network $G = (V, E)$ where $V = \{v_i | 1 \leq i \leq n\}$ is the set of n vertices and $E = \{e_j | 1 \leq j \leq m\}$ is the set of m edges. Each of the vertices could be a router, a switch or an end host. Let $C = \{c_k | 1 \leq k \leq p\} \subset V$ be the set of p participants involved in the same network multimedia application session.

During the lifetime of the session, a global state needs to be maintained especially for interactive applications. For example, when player A performs an action that changes the state of an object in an online game session, player B from the same session should also see the state change of this object. The communication model for the network multimedia application session dictates the mechanisms how the states of the session are maintained or synchronized. Generally, the communication model can be classified into two main types: *client-server* (centralized) and *peer-to-peer*, which are illustrated in Fig 1.1.

- Client-Server (Figure 1.1a): In the client-server (centralized) architecture, all updates generated by the participants are sent to the central server first. Then the central server computes the new state of the session and sends necessary updates to all the participants. The nodes (vertices) and links (edges) involved in the same session form a tree (or a star graph if we only consider the data

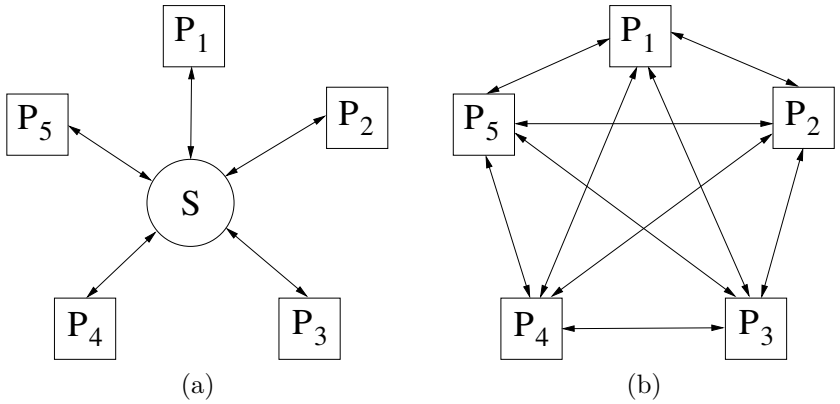


Figure 1.1: (a) Client-Server Model and (b) Peer-to-Peer Model where S is the central server for the client-server model, $\{P_1, P_2 \cdots P_5\}$ are the participants in the session. The arrows show the direction of data flows.

flows) with the central server as the root. Most video streaming services and massively multiplayer online games (MMOG) use this model.

- Peer-to-Peer (Figure 1.1a): In the peer-to-peer approach, there is no central server. All the updates are exchanged between the participants directly and the new state of the session is computed at each client. The nodes (vertices) and links (edges) involved in the same session form a complete graph. Some video/audio conferencing systems and Real-time Strategy (RTS) Games use this architecture.

Both architectures require some synchronization mechanism to restore the order of the events (updates), especially for a competitive environment like online games. However, the peer-to-peer approach is less scalable due to the direct message exchange between participants and requires more sophisticated synchronization methods to maintain a persistent state due to lack of a central control [27]. There are also other architectures proposed, such as multi-server or mirrored server models [20, 30], which are variations or hybrid of the two basic model described above.

Note that it is not necessary for the participants to keep sending updates during the lifetime of a session. How often the participants send out their updates and how

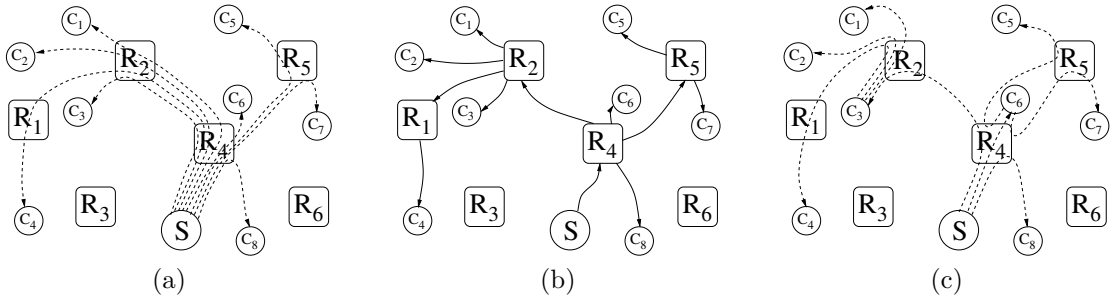


Figure 1.2: Multicast session with the source S and the destinations $\{C_1, C_2 \ldots C_8\}$. (a) Simulating multicast with multiple unicasts. (b) Multicast with hardware (router) and protocol support. (c) Underlying data flow of overlay multicasting.

large the updates are (in terms of packet sizes) depends on the application. For applications such as video/audio streaming, majority of the network traffic comes from the central server (where the multimedia content is stored or generated) to the participants. There are virtually no updates from the participants. On the other hand, in applications such as video/audio conferencing, the amount of incoming/outgoing multimedia data for each participant is huge and frequent.

1.4 Multicast and Overlay Networks

Multicast is an approach to reduce the network traffic to deliver the same data from a single source to multiple destinations. With multicast, multiple unicast sessions that share subpaths on the network can be combined with *packet duplication* and *packet aggregation*. We use Figure 1.2 to illustrate the idea of multicast. Consider a source S that needs to transmit data to the set of destinations $\{C_1, C_2 \ldots C_8\}$. $\{R_1, R_2 \ldots R_8\}$ are Internet routers.

Figure 1.2a shows how multicast can be simulated by using multiple unicast communications. In this case, the same copy of data flows through the link (S, R_4) for 8 times, once for each destination nodes. Hence the bandwidth of the link (S, R_4) becomes critical and could impact the performance of the system if this bandwidth is not sufficient. In Figure 1.2b, multicast is supported at hardware and protocol

level. Here the routers are capable of duplicating the packets and forwarding them to the destinations. Bandwidth can be conserved since each packet only travels through each communication link at most once. However, routers that support IP multicast protocols such as IGMP (IPv4) [54] and MLD (IPv6) [55] are not widely deployed and available to end users. An alternative approach called *overlay multicast* is proposed to address this issue. Examples include Application Layer Multicasting (ALM) or End System Multicast (ESM) [9, 16, 58].

In an overlay network, the nodes are end-hosts and links are the Internet paths connecting them. By using an overlay network, the data can be sent to some of the end hosts then these end hosts can duplicate the data and send the data to the destinations. This is the so called *overlay multicast* and Figure 1.2c illustrates how it works. Consider the subset of the destinations $\{C_5, C_6, C_7\}$. If multiple unicasts are used, S will send the three copies of the data through (S, R_4) to deliver them to $\{C_5, C_6, C_7\}$. In the case that overlay multicast is used, S sends one copy of the data to C_6 first, then C_6 uses two unicasts to deliver the data to C_5 and C_7. This reduces the bandwidth usage on the link (S, R_4) from 3 units to 1 unit. Similarly, the same strategy can be applied to other subsets ($\{C_8\}$ and $\{C_1, C_2, C_3, C_4\}$) of the destinations to conserve more bandwidth as shown in the same figure. The results in a huge improvement over using multiple unicasts as in Figure 1.2a.

One drawback of using an overlay network in a multicast session can be illustrated with the examples in Figure 1.3. Consider an end-host computer S used by a user with residential Internet connection to participate in an interactive application session. This computer connects to a series of routers (in some cases wireless) to access the Internet router (R_1) located at the Internet Service Provider (ISP), which is the entry point to access the Internet and the content is distributed to other peers through this router. The path from this participant node to the nearest Internet router suffers from *upstream* bandwidth limitations when compared to available bandwidths on the

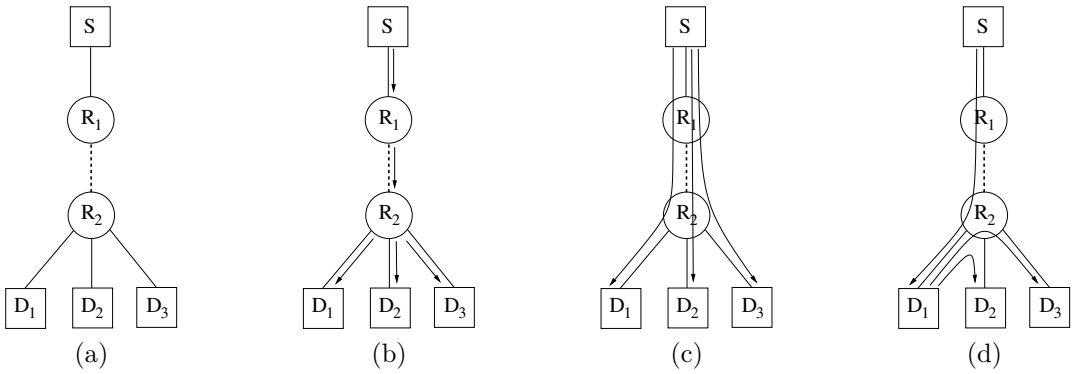


Figure 1.3: Different approaches for multicast. (a) The underlying structure of the network. S is the source, D_1, D_2, D_3 are the destinations. S, D_1, D_2, D_3 are end hosts. R_1 and R_2 are their nearest Internet routers. (R_1, R_2) is an Internet path that may consists of several Internet routers. (b) IP multicast. (c) Multiple unicasts. (d) Overlay multicast.

Internet backbone. The problem is more acute when S is a multicasting node – here the participant node has to duplicate every message it receives from its parent (equal to the number of children in the multicasting tree) and each message will follow the same bandwidth constrained path to the nearest Internet router. For instance, the latest ADSL standard ITU G.992.5 Annex M (ADSL2+M) connection from S to R_1 is 3.3 Mbit/s upstream (upload) vs 12 or 24 Mbit/s downstream (download). During a group communication session, S needs to send multimedia content to D_1, D_2, D_3. When the network supports multicast protocol, only one copy of the contents flows through the link $(S \rightarrow R_1)$ (Figure 1.3b). In the case without multicast support (Figure 1.3c), S has to use unicast to achieve the same objective, i.e., the same content has to flow through $(S \rightarrow R_1)$ for three times. Hence the upstream bandwidth available to an end host is the bottleneck and the situation gets worse when the number of children of this client or the data generation rate becomes larger. Figure 1.3d shows how overlay multicast can be used to lift this issue. However, the same content flows through $(D_1 \rightarrow R_2)$ twice in this case and once through $(R_2 \rightarrow D_1)$.

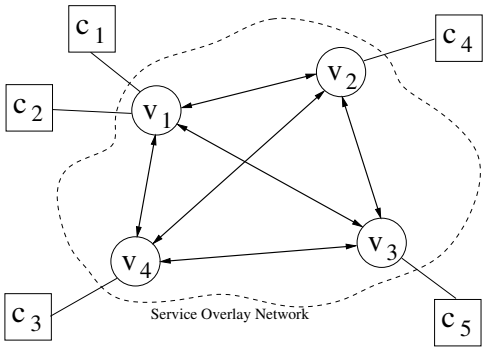


Figure 1.4: Two-Layer Approach

Service Overlay Networks

A service overlay network (SON) is an overlay network built on top of the Internet in which the nodes are capable of performing application layer data forwarding with end-to-end QoS support (on the overlay paths) [1, 8, 22]. Overlay multicast and packet aggregation can also be implemented to conserve the bandwidth usage and results in smaller congestion and latency. We assume the service overlay network is *well-provisioned*, i.e., lower latency with higher bandwidth. The service nodes are hosted at the ISPs in different regions for more consistent latencies and bandwidth. Because the routing on the Internet is not always optimal in terms of latencies [43], sometimes the latency is smaller for two nodes communicate via a third node than directly. Hence it is also possible to deploy the service nodes at some strategic locations to lower round trip time (RTT). A two-layer approach as shown in Figure 1.4 can also be used to relieve the burden of overlay multicast (unicasts) from the end hosts as follows. Each participant in an application session can connect to a service node (gateway) in the SON and the service overlay network copies and forwards the data to the destinations.

1.5 Organization of the Dissertation

The rest of this dissertation is organized as follows. In the first half of this dissertation (Chapter 2 and 3), we assume the two-layer approach and address the problem of finding the paths with latency-related constraints for both Client-Server and Peer-to-Peer communication models – Server and Clients Assignment Problem (SCAP) using the Client-Server communication model in Chapter 2, followed by Client Assignment Problem (CAP) using the Peer-to-Peer communication model in Chapter 3. We have developed algorithms and integer programming models for both these problems. In the second half, we address the multi-stream multi-source multicasting problem (MMMRP) covered in Chapter 4. In MMMRP, we assume multiple concurrent multicast sessions in a given network and present a heuristic algorithm MMForests as well as an integer programming model to coordinate the multicast sessions in such a way that the overall residual bandwidth is maximal. Conclusions are given in Chapter 5 along with the direction of future work.

Chapter 2

Server and Clients Assignment Problem

2.1 Introduction

Network applications involving groups of users participating in a communication session have become more popular today with the availability of anytime-anywhere networks. A few examples include Internet-based concerts, network video conferences, virtual environments, online games, live video broadcasting. These applications can be distinguished by the *degree of interaction* among the users (or clients) involved in the same application session. For instance, Internet-based concerts or jamming sessions have a high degree of interaction, wherein the musicians located at different locations to join the network, play their instrument of choice while listening to others, and create music as if they are all at the same location. Video or audio broadcasting on the other hand has no interaction among its clients.

We have briefly discussed two major communication models – Client-Server (or centralized) model and Peer-to-Peer model for group communication in Chapter 1. In this chapter, we will address the problems using the Client-Server model.

Consider the p clients (or users) along with 1 central server involved in a client-server (or centralized) type application session as shown in Figure 2.1a. The clients in the same application session send their updates to the central server periodically. Then the central server computes and sends the new state of the session to the clients in order for the clients to maintain a consistent view of the session. One way the server can send data to each of the clients is via p unicast connections. Depending on

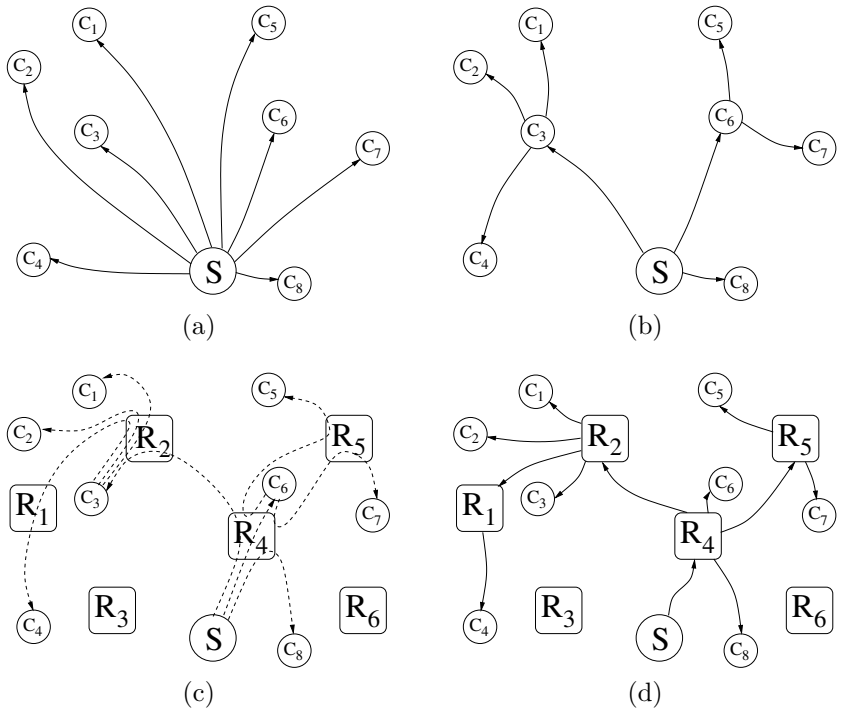


Figure 2.1: (a) The logical data flow of multicasting in an application session. S is the source and C_j's are clients. (b) Same session using overlay multicast, where the clients need to help multicasting. (c) The underlying data flow of previous example, $\{R_1, R_2 \cdots R_6\}$ are IP routers. (d) Multicast using an SON, $\{R_1, R_2 \cdots R_6\}$ are service nodes.

the application and the number of clients participating in this session, the server to clients and clients to the server unicast connections could result in plenty of network traffic, consuming bandwidth and increasing latency.

Although *multicast* can reduce network traffic by using *packet duplication* and *packet aggregation*, IP multicast protocols such as IGMP (IPv4) [54] and MLD (IPv6) [55] are not widely deployed and available to end users. A popular approach to address this issue is to construct an overlay network and use *overlay multicast* [9,16,58]. In an overlay network, the nodes are end-hosts and links are the Internet paths connecting them. The example in Figure 2.1b shows the server and clients form a multicast tree where the server S is the root of the tree and each link of the tree is an Internet path connecting two nodes. The internal nodes in the tree behave like routers and

simulate multicast by using multiple unicasts. Overlay multicast is flexible in the sense that new protocols can be easily incorporated, but are less efficient because the multicasting paths may involve *overlapping Internet paths* [26]. Example in Figure 2.1c shows that for the same data flows through the link between R_2 and c_3 for 4 times.

We have introduced the notion of service overlay networks (SONs) in Chapter 1, which is an overlay network built on top of the Internet with bandwidth-provisioned and end-to-end Quality-of-Service (QoS) support [1, 8, 22]. The nodes in a SON are also capable of performing application layer data forwarding. Hence it can be an approach to relieve the burden of end hosts (especially the server node) that perform overlay multicasting as well as conserve the bandwidth on the network as following.

In an application session, each participant connects to a service node (we call it a contact node) in a SON. When a node needs to multicast, it first sends the data to its contact node. Then the contact node sends the data to other participants' contact nodes using the SON wherein application layer multicast can be used. As we can observe, the node that performs multicasting only needs to send out one copy of data using this two-layer approach, as shown in the example utilizing SON (Figure 2.1d). Overlay multicast and packet aggregation can also be implemented to conserve the bandwidth usage, which results in smaller congestion and latency.

However, building such a network is costly since the Overlay Network Operator needs to deploy the service nodes at different locations and purchase network bandwidth to connect them under Service Level Agreements (SLAs) for bandwidth-provisioning. Hence, one of the goals in this research is to reduce the number of nodes utilized by an application session for the following two reasons. First, each overlay path consists of several IP network links and reducing the number of nodes in the multicasting tree has the benefit of reducing packet loss. As more number of nodes participate in multicasting the probability of packet loss in the queues of the nodes

increases and hence keeping the number of multicasting nodes smaller is beneficial. Second, it is cheaper to increase the computational power for each node when compared to the link cost. Less number of nodes used by an application results in less bandwidth consumption and more sessions can be served.

In this research, we consider several variants of *Server and Client Assignment Problems (SCAP)*. The objective of SCAP is, to find a contact service node for each of the participants (this is called an assignment) in a client-server application session and then construct a multicasting tree satisfying some latency-related criteria. The rest of this chapter is organized as follows. In Section 2.2 we provide the information on the related work. Then system model and notations are described in Section 2.3. Integer Programming (IP) formulations and our proposed algorithms are provided along with problem formulations in Section 2.4. Performance evaluation is given in Section 2.5 and conclusions are drawn in Section 2.6.

2.2 Related Work

Vik et al. [58] has provided a thorough and complete survey of diameter and degree bounded Steiner tree heuristics. They have also evaluated these heuristics in terms of their execution times as it relates to various network sizes. One way to use the heuristics mentioned in [58] is to create a larger network consisting of the service nodes and clients as nodes, and links from both the SON and each client to each service nodes. But when we execute the heuristics on this larger graph, we not only increase the execution time many folds, we might also end up with the situation wherein the client node may be an internal node of the tree thereby taking the additional responsibility of performing multicasting.

Lee et al. [40] proposed a well-provisioned network of *mirrored servers* for massively multiplayer online games (MMOG). A distributed algorithm called Zoom-In

Zoom-Out (ZIZO) is proposed to reduce the number of *contact servers* used by a game session in this research which has a similar goal of our research with three major differences. First, ZIZO works for the mirrored server architecture, which is Peer-to-Peer and we focus on the client-server architecture. Second, they try to minimize only the number of *contact nodes* in the resulting network, not the total number of nodes as in our work. Third, several variants of the problem are considered in our research, which are not part of the work in [40] .

Service Overlay Networks (SONs) [8, 41, 56, 57] have recently emerged as an alternative to the IP multicast since IP multicast is not widely available to end users. Vieira et al. [57] assume that the end systems and service nodes are interconnected by different Internet Service Providers (ISPs) where each link is associated with an access cost. They present several heuristic algorithms to find a topology such that the total cost of links is minimized and the resulting subgraph includes all end systems and are connected.

We differentiate this research with the works on SONs referenced above in the following aspects. First, we assume the topology of SON pre-determined with a fixed routing mechanism between nodes. Second, instead of demand matrices, we consider application sessions where *groups of users* are involved. Hence our goal is to assign end hosts to service nodes (accessible through the Internet) such that certain QoS constraints are satisfied. Third, we also require that the number of service nodes involved is minimal. Fourth, we assume that multicast and packet aggregation are supported by the SONs for bandwidth conservation.

2.3 System Model and Notations

We define the terminology, notations, system model and describe our problems in this section.

2.3.1 Service Overlay Networks (SONs)

We use $G = (V, E)$ and delay function $d()$ to denote the well-provisioned SON, where $V = \{v_i | 1 \leq i \leq n\}$ is the set of n *service nodes* (or service gateways) which are capable of overlay multicast and E is the set of m links connecting them. We use $d(v_i, v_{i'})$ to denote the latency between two nodes for $v_i, v_{i'} \in V$ and $v_i \neq v_{i'}$. When there is a direct link between $(v_i, v_{i'}) \in E$, $d(v_i, v_{i'})$ is the latency of this link. Otherwise, $d(v_i, v_{i'})$ is the sum of the latencies of the links on the shortest path between v_i and $v_{i'}$.

The well-provisioned SON has lower latency and higher bandwidth than the Internet. The service nodes are hosted at the Internet Service Providers in different regions of the world for more consistent latencies and bandwidth. It is also possible to strategically deploy the service nodes at some specific locations to lower round-trip time (RTT) [43].

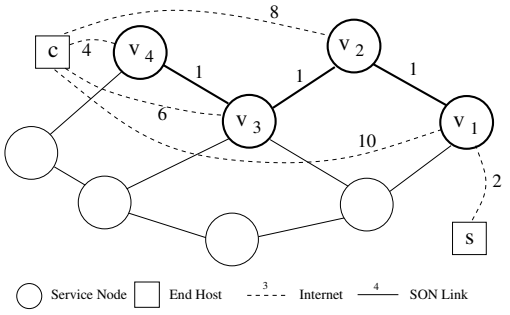


Figure 2.2: Figure illustrating the trade-off between end-to-end latency and number of services nodes involved.

Since the latencies on the SON are low, it is possible to reduce the number of service nodes used by an application session and we illustrate this using Figure 2.2. Let V_1 be the contact node of s and the shortest path from V_1 to V_4 be $V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4$. The numbers on the link denote their latencies. Client c can communicate with s by connecting to V_4 with the latency 9 and 4 service nodes are used in this case. Note that although V_2 and V_3 are not contact nodes, yet the traffic has to go through

them for communication between s and c. When c is assigned to V_1, there is only one service node involved but with a higher latency of 12. The trade-off between the latency and the number of service nodes involved makes the problem more interesting when we consider the real-time requirements of any application session wherein data flows from the central server can share the same subpaths.

2.3.2 Client-Server Application Session

A client-server application session includes a source of a data stream (or a central server in the case of an interactive application) and a set of clients that will be receiving the data (and/or sending updates to this source or server). We use s to denote the data source or central server, $C = \{c_j | 1 \leq j \leq p\}$ to denote the set of p participating clients and $d(h, v_i)$ to denote the Internet latency between $h \in \{s\} \cup C$ and $v_i \in V$. An application session is denoted as $D = (s, C, G)$. We also assume the tree structure is used for client-server communication in a client-server application session.

2.3.3 Assignment

Our goal is to construct a network for the application session (or group) $\{s\} \cup C$ using the service overlay network G according to the following two requirements. First, each of the group members needs to connect to a service node in G in order to utilize the service overlay network. We call the service nodes that the server and clients connect to as *contact service nodes* or *contact nodes* for short, we call the contact node of the server s *root service node*. We will use a_j, $1 \leq j \leq p$, to denote the service node for $c_j \in C$ and a_s for the root service node.

Second, we need to construct a multicasting tree T (within G) that spans the contact nodes $\{a_1, a_2 \cdots a_p\}$ with the root service node a_s as the root. Multicasting tree T can be easily derived once an assignment is determined since the path between

any pair of nodes in G is predefined (shortest path in our assumption). T is actually the shortest path tree rooted at a_s that spans all a_j's, $1 \leq j \leq p$. Hence finding a good mapping from the participants to the service nodes is the key to finding a good multicasting tree. We call this mapping an *assignment* and use $A = \{a_s, a_1, a_2, \cdots a_p\}$ to denote it, where $a_s, a_1, a_2, \cdots a_p \in V$ are the contact nodes for $s, c_1, c_2, \cdots c_n$ respectively. Note that $a_j = a_{j'}$ if client c_j and client $c_{j'}$ are assigned to the same service node.

Since multicasting tree T can be derived from the assignment A, the solution to our problem can also be represented by the assignment A. We use $T_D(A)$ to denote the shortest path tree (of G) derived from the assignment A for the application session D, and $|T_D(A)|$ for the number of nodes in $T_D(A)$, which is also called the cardinality of the assignment A.

2.3.4 Communication Model

The client-server communication of the application session D using the service overlay network can be performed as follows. The server s can *multicast* data to all participating clients by sending one copy of the data to the root service node a_s and the rest of work is done by overlay multicast using the multicast tree $T_D(A)$. For interactive application sessions, the clients send updates to their contact nodes (through the Internet) and the service nodes aggregate and forward the data to a_s through the shortest path in G to a_s (overlay) and to s (Internet). The server uses this data, performs computations if necessary, and sends the state update to the clients using the method described earlier. The major advantage of this is that the burden of multicasting is relieved from end hosts which usually have lower upstream bandwidth and computational power.

2.3.5 Delays (Latencies)

Usually, we use the end-to-end latency or synchronization delay to measure Quality-of-Service (QoS) of the network connection between two end hosts. Similarly, we measure the latency perceived by each client in an application session to evaluate the QoS of the session. We define the latency for a client c_j in an application session $D = (s, C, G)$ under an assignment A as the propagation delay from client c_j to the server s. The latency is denoted by $\Lambda_D(A, c_j) = d(c_j, a_j) + d(a_j, a_s) + d(s, a_s)$. This is the sum of the propagation delays of the links from c_j to a_j, then to a_s and finally to the server s under the assignment A. We further define the delay for the application session D under the assignment A as the largest delay of the clients' delays:

$$\begin{aligned}\Lambda_D(A) &= \max \Lambda_D(A, c_j) \\ &= \max[d(c_j, a_j) + d(a_j, a_s) + d(s, a_s)] \quad \forall \quad c_j \in C\end{aligned}$$

2.3.6 Delay variation

The synchronization of application state among participating clients is important for certain types of interactive applications such as video/audio conferencing or online games. In these applications, ideally the clients should receive the update sent by the server at approximately the same time for the sake of fairness. Rouskas et al. [51] first defined the term *delay variation* as the difference between the maximum delay and the minimum delay from any client to the server node. Approaches for solving delay variation problems including finding alternative paths or packet buffering are presented in [2, 50, 51]. We adopt the definition for an application session as follows:

$$\Delta_D(A) = \max |\Lambda_D(A, c_j) - \Lambda_D(A, c_{j'})| \quad \forall \quad c_j, c_{j'} \in C$$

2.4 Problem Formulations and Algorithms

With multicasting on the well-provisioned service overlay network, the loads on participants for an application layer multicast session are relieved and we also can achieve goals such as lower latencies or serving more number of users. However, how the servers and clients are assigned to the service nodes is critical to QoS. The assignment depends on the requirements of the application.

In this chapter, we consider the problem that deals with network construction and server and client assignment to support multimedia group communication sessions. Given a well-provisioned service overlay network $G = (V, E)$, server s, the set of clients C and the latency function $d()$ (for both G and from hosts to V), we formulate several variants of *Server and Client Assignment Problems* (SCAP) with different real-time related constraints with the requirement number of service nodes involvement in the resulting multicast tree $T_D(A)$ is minimum.

Consider the set of latency requirements Γ, there is a limited number of service nodes that each client can connect to. How to coordinate these choices among the clients to minimize the total number of service nodes involved in an application session is a $\mathcal{NP}$-hard problem, which can be proved by reduction from SetCover problem [40]. We will show that SCAP-DMC problem is $\mathcal{NP}$-hard later, the proof for other problems are similar. We attempt to address this problem in this chapter. We provide heuristic algorithms and integer programming (IP) models for solving some variants of this problem, which are summarized in Table 2.1.

2.4.1 SCAP with Minimum Delay (SCAP-MD)

Most online interactive applications have their real-time requirements. In the first variant of SCAP – SCAP-MD, our goal is to find an assignment A with the lowest possible delay for a given application session D, i.e., that $\Lambda_D(A)$ is minimum. The

Table 2.1: Summary of SCAP Problems and Corresponding Algorithms

Problem	Problem Complexity	Algorithm	Algorithm Complexity
SCAP-MD	Polynomial-time Solvable	SCAP-MD-A†	$O(n^2p)$
SCAP-DMC	$\mathcal{NP}$-Hard	SCAP-DMC-IP†	Exponential
		SCAP-DMC-H	$O(n^3p)$
SCAP-DV	Polynomial-time Solvable	SCAP-DV-A†	$O(n^3p^2)$
SCAP-DVMC	$\mathcal{NP}$-Hard	SCAP-DVMC-IP†	Exponential
		SCAP-DVMC-H	$O(n^4p^2)$
		SCAP-DVMC-IP-CHAIN†	Exponential

† denotes the algorithm gives an optimal solution. * n: number of application routers in the SON, p: number of clients.

number of service nodes involved is not a consideration in SCAP-MD and it can be solved exactly within the polynomial-time using Algorithm SCAP-MD-A.

Algorithm SCAP-MD-A

Algorithm SCAP-MD-A is presented in Algorithm 1, in which we only show the key ideas of the algorithm. The algorithm works as follows. For each service node r as the *root service node*, we create an assignment A' such that $d(c_j, v_i) + d(v_i, r) + d(s, r)$ is smallest for each client $c_j \in C$ (and $v_i \in V$). Then we compare A' and d' with the best known assignment A and its delay d. A and d are replaced if A' has a smaller delay. The assignment with smallest possible delay is returned by the algorithm. Line 4 can be done in $O(pn)$ time and it is executed for n times. Hence the overall time complexity of SCAP-MD-A is $O(n^2p)$.

2.4.2 SCAP with Delay Bound and Minimum Cardinality (SCAP-DMC)

In SCAP-DMC, we attempt to coordinate the paths from the clients to the server and find the trade-off between the real-time requirement and number of service nodes used in an application session. Let the real-time requirement for the given application session be μ (the given delay bound), the goal is to find an assignment such that $\Lambda_D(A) \leq \mu$ and $|T_D(A)|$ is minimum. In this section, we first show that SCAP-DMC

Input: s, C, $G = (V, E)$, $d()$
Output: assignment A
Create an empty assignment A;
$d = \infty$;
foreach $r \in V$ **do**
    Create an assignment A' by assigning each client c_j to a service node v_i such that $d(c_j, v_i) + d(v_i, r) + d(s, r)$ is minimum;
    $d' = \Lambda_D(A')$;
    if $d' < d$ **then**
        $d = d'$, $A = A'$;
    end
end
return A

Algorithm 1: SCAP-MD-A

is $\mathcal{NP}$-hard, then provide an Integer Programming formulation along with a heuristic algorithm.

Theorem 1. *SCAP-DMC is $\mathcal{NP}$-hard.*

Proof. Lee et al. [40] have shown minimum game server allocation problem is $\mathcal{NP}$-hard. A similar proof can be used to show the $\mathcal{NP}$-hardness of SCAP-DMC, the proof for other variants of SCAP are similar.

Set Covering Problem (SetCover)

Given a family of n finite sets $\{V_j\} = \{V_1, V_2, \cdots V_n\}$. The goal of *SetCover* is to find a subfamily $\{T_h\} \subseteq \{V_j\}$ such that $\bigcup\ T_h = \bigcup\ V_j$ and the cardinality of $\{T_h\}$ is minimum. The optimization version of set covering problem is known to be $\mathcal{NP}$-hard.

SetCover $\leq_p$ SPD-CS

Here we show SetCover is polynomial-time reducible to a special case of SCAP-DMC. For convenience, let the universe of SetCover be $\bigcup V_j = \{C_1, C_2, \cdots C_p\}$. First, we construct a network with a dummy node r as the root and let the rest of the nodes be $v_1, v_2, \cdots v_n$. This network is a star graph and the set of service nodes is $\{r, v_1, v_2, \cdots v_n\}$. Now, we set the latencies of the edges from $v_1, v_2, \cdots v_n$ to r with

delays as 1. Let v_j corresponds to V_j, $1 \leq j \leq n$.

Next, we add p client nodes to the network: $c_1, c_2, \cdots c_p$ and let c_k corresponds to C_k, $1 \leq k \leq p$. There are edges between each c_k and service node v_j pair (recall that the service nodes and clients form a complete bipartite graph). The delays will be set as follows. First, we set the delay of (c_k, r) to 3 for all c_k. Then, if $C_k \in V_j$, the delay between c_k and v_j is 1. Otherwise, it is set to 3. Finally, we add the server s to the network with latency to r being 0 and 5 to other v_j's.

Now an instance of SCAP-DMC is constructed with the set of service nodes $\{r, v_1, v_2, \cdots v_n\}$, the set of clients $\{c_1, c_2, \cdots c_p\}$ and edges described earlier. The delay bound is set to 2 which is the maximum allowance from a client to r in the solution. The construction of this problem instance can be done in $O(n + np)$.

In this instance, r will to be chosen as the root service node in the solution. The set of service (v_j's) nodes chosen in an optimal solution of this SCAP-DMC problem also gives the solution to the *SetCover* problem instance by simply mapping v_j to V_j. Hence SCAP-DMC is $\mathcal{NP}$-hard. □

Integer Programming Approach (Alg. SCAP-DMC-IP)

We use the strategy similar to Algorithm *SCAP-MD-A* that iterates through all service nodes as the root service node r. In each iteration, the following IP model (*SCAP-DMC(r)*) is used to find an assignment A that satisfies the delay bound μ and $|T_D(A)|$ is minimal. The one that gives the fewest number of service nodes is minimum. Model *SCAP-DMC(r)* is built as follows.

We define a_i, which is a column vector of dimension n that represents the nodes on the shortest path between a service node v_i and the chosen root service node r. An entry $a_{ii'}$ is equal to 1 if $v_{i'}$ is on the path and 0 otherwise.

Define binary decision variables Y_{ji} that take the value 1 if client c_j is assigned to service node v_i and 0 otherwise. Y_{ji} also represents the path selection from a client

Model SCAP-DMC(r)

$$\text{minimize} \qquad \sum_{v_i \in V} X_i \tag{2.1}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \;\; c_j \in C, \Lambda_D(c_j, v_i, r) \leq \mu \tag{2.2}$$

$$\sum_{c_j \in C} \sum_{v_{i'} \in V} a_{i'i} Y_{ji'} \leq p X_i \qquad \forall \;\; v_i \in V \tag{2.3}$$

Figure 2.3: Model SCAP-DMC(r)

c_j to its contact node v_i then follows the shortest path to the root node r and to s. Binary decision variables X_i are defined to take the value 1 if service node v_i is used in the solution, 0 otherwise. The delay constraints in the IP models are imposed in the preprocessing stage and hence do not appear in the IP model. Model SCAP-DMC(r) is shown in Figure 2.3.

The objective function (2.1) measures the total number of service nodes used (selected). Constraints (2.2) ensure that each client is assigned to exactly one service node and delay bound μ is satisfied. Constraints (2.3) are used to make sure a service node is set to *selected* when the path Y_{ji} is selected and it is on this path.

Algorithm SCAP-DMC-H

SCAP-DMC-H is a heuristic algorithm that also uses the same strategy as SCAP-MD-A. It iterates through all possible root service nodes as the root service node. Each iteration with the root service node r works as follows.

First we assign the clients to the root service node r if their delays ($\Lambda_D(c_j, r, r)$) are within the delay bound μ and mark r as used. Next step is to evaluate unused service nodes that are *neighbors of used service nodes* and choose the one (v^*) that can take the largest number of clients without violating μ. Then we assign possible clients to v^* and mark v^* as used. By repeating this procedure, we will find a solution with

r as the root service node if such solution exists. By comparing all the assignment returned, we choose the one with fewest number of service nodes marked as used as our heuristic solution. The overall complexity of algorithm *SCAP-DMC-H* is $O(n^3p)$, which is presented in Algorithm 2.

```
   Input: s, C, G = (V, E), d(), μ
   Output: assignment A
 1 Create an empty assignment A, n = ∞;
 2 foreach r ∈ V do
 3 |   Create an empty assignment A';
 4 |   mark all service nodes as unused, not_done = false, found = true, d = ∞;
 5 |   foreach c_j ∈ C do
 6 |   |   if d(c_j, r) + d(s, r) ≤ μ then
 7 |   |   |   Assign c_j to r in A';
 8 |   |   else
 9 |   |   |   not_done = true, found = false;
10 |   |   end
11 |   end
12 |   mark r as used;
13 |   while not_done do
14 |   |   V' = unused neighbors of used service nodes, C' = unassigned clients;
15 |   |   Find the v* ∈ V that can take the most number of clients (n*) without violating μ;
16 |   |   if n* ≠ 0 then
17 |   |   |   Assign unassigned c_j to v* in A' if μ is not violated;
18 |   |   |   mark v* as used;
19 |   |   |   if all clients are assigned then
20 |   |   |   |   not_done = false, found = true;
21 |   |   |   end
22 |   |   else
23 |   |   |   not_done = false, found = false;
24 |   |   |   clear A', d = ∞;
25 |   |   end
26 |   end
27 |   n = cardinality of A';
28 |   if n' < n then
29 |   |   n = n', A = A';
30 |   end
31 end
32 return A
```

Algorithm 2: SCAP-DMC-H(s, C, G, μ)

2.4.3 SCAP with Delay Bound and Minimum Delay Variation (SCAP-DV)

In SCAP-DV, we consider delay variation minimization problem for an application session such that $\Lambda_D(A) \leq \mu$ (delay bound) and $\Delta_D(A)$ is minimum (delay variation). SCAP-DV problems can be solved in polynomial time by using Algorithm SCAP-DV-A. Note that if we relax the delay bound requirement, we may get a solution with a tighter delay variation. We present Algorithm SCAP-DV-A (Algorithm 3) to solve this problem. There is no requirement on the number of service nodes used.

Algorithm SCAP-DV-A

Algorithm *SCAP-DV-A* iterates through all possible service nodes as the root service node r as our previous algorithms do (line 2 – 20). In each iteration, a modified version of the Algorithm *Chain* [2] is used to find the assignment with minimal delay variation. Each iteration works as follows.

First a list L which contains all the tuples $(c_j, v_i, \lambda(j,i))$ whose $\lambda(j,i) = \Lambda_D(c_j, v_i, r)$ $\leq \mu$ (line 3 – 9) is created then sorted by $\lambda(j,i)$ in non-descending order. We ignore r in $\lambda(j,i)$ since the context is clear. Each tuple represents a possible assignment for a client and we apply Algorithm Chain [2] to find an assignment A such that the delay variation is minimal. For convenience, we use L_k to denote the k-th element in L and use $(\hat{c_k}, \hat{v_k}, \hat{d_k})$ to denote its content.

Before describing the following steps, here we define the term *chain* that will be used in the explanation. A *chain* is a set of consecutive elements in L that starts with L_h (head) and ends with L_t (tail) where $h < t$ with the two conditions satisfied. (a) each client must appear *at least once*, and (b) the client at the tail ($\hat{c_t}$) only appears *exactly once*. A chain can be converted to an assignment with delay variation $\hat{d_t} - \hat{d_h}$ by assigning $\hat{c_h}$ to $\hat{v_h}$ and $\hat{c_t}$ to $\hat{v_t}$ and arbitrarily assigning other clients using the

```
   Input: s, C, G = (V, E), d(), μ
   Output: Assignment A, delay variation d_v
1  Create an empty assignment A, d_v = ∞;
2  foreach r ∈ V do
3  |  Create an empty list L;
4  |  foreach (c_j, v_i) pair do
5  |  |  λ(j, i) = d(c_j, v_i) + d(v_i, r) + d(r, s);
6  |  |  if d(j, i) < μ then
7  |  |  |  Add the tuple (c_j, v_i, λ(j, i)) to L;
8  |  |  end
9  |  end
10 |  Sort L by λ(j, i) in non-descending order;
11 |  foreach L_h ∈ L do
12 |  |  Find a chain starts with L_h which ends at L_t;
13 |  |  if a chain is found then
14 |  |  |  if (d̂_t − d̂_h) < d_v then
15 |  |  |  |  d_v = d̂_t − d̂_h;
16 |  |  |  |  Create A from items L_h through L_t (and replace the old A);
17 |  |  |  end
18 |  |  end
19 |  end
20 end
21 return A, d_v
```

Algorithm 3: SCAP-DV-A(r, s, C, G, μ)

tuples in the chain. Finding a chain starts with L_h as the head then by scanning from L_h toward the end of L and stop when all the clients are visited. The last element visited is the tail of the chain.

Next step (line 11 – 19) is to find all possible *chains* and compare their delay variations. The one with smallest delay variation is minimum. The overall time complexity of Algorithm *SCAP-DV-A* is $O(n^3p^2)$.

2.4.4 SCAP with Delay & Delay Variation Bounds and Minimum Cardinality (SCAP-DVMC)

We further extend *SCAP-DV* problem by relaxing the delay variation minimization in *SCAP-DVMC*. Given a delay bound μ and a delay variation bound ν, the goal is to find an assignment A satisfying the following requirements:

1. $\Lambda_D(A) \leq \mu$ (delay bound)

Model SCAP-DVMC(r)

$$\text{minimize} \quad \sum_{v_i \in V} X_i \tag{2.4}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \ c_j \in C \tag{2.5}$$

$$\Lambda_D(c_j, v_i, r) \leq \mu$$

$$\sum_{c_j \in C} \sum_{v_{i'} \in V} a_{i'i} Y_{ji'} \leq p X_i \qquad \forall \ v_i \in V \tag{2.6}$$

$$[\lambda(j,i) - \lambda(j',i\prime)](Y_{ji} + Y_{j'i'} - 1) \leq \nu \qquad \forall\ c_j, c_{j'} \in C \tag{2.7}$$

$$c_j < c_{j'}$$

$$\forall\ v_i, v_{i'} \in V$$

$$\lambda(j',i\prime) \leq \ \lambda(j,i) \leq \mu$$

Figure 2.4: Model SCAP-DVMC(r)

2. $\Delta_D(A) \leq \nu$ (delay variation bound)

3. $|T_D(A)|$ is minimum (minimum cardinality)

We present two integer programming approaches (Algorithm SCAP-DVMC-IP, Algorithm SCAP-DVMC-IP-CHAIN) and a heuristic algorithm (SCAP-DVMC-H) for this problem.

Algorithm SCAP-DVMC-IP

Algorithm *SCAP-DVMC-IP* iterates through all possible service nodes as root service node r and uses *Model SCAP-DVMC(r)* to find the optimal solution.

Similar to previous sections, we use $\lambda(j,i)$ to denote the communication cost (delay) and use column vector a_i to represent the path. Binary decision variables Y_{ji} and X_i are also the same as in Model SCAP-DMC(r). The model is shown in Figure 2.4:

The model is actually identical to *Model SCAP-MC(r)* except for the addition of constraint set (2.7), which is used to enforce the delay variation (ν).

Algorithm SCAP-DVMC-H

We developed a heuristic algorithm *SCAP-DVMC-H* (Algorithm 4) for SCAP-DVMC which iterates through all possible service nodes as the root service node r (line 3 – 34) and to find the minimal of local minimals. Each iteration works as follows. The list L is constructed and sorted as in Algorithm *SCAP-DV-A* (line 4). Next step is to find all what we call *non-left-extendable chains* satisfying the delay variation bound ν (line 5). We define a non-left-extendable chain as a chain starting with L_h and ending with L_t such that there is no chain in L which starts with some $L_{h'}$ for $h' < h$ and ends with L_t. Note that $\hat{c}_t$ only appears in this chain once. Each non-left-extendable chain is a candidate for our solution.

Now we are going to construct an assignment A' from all the non-left-extendable chains found individually and choose the assignment with minimum cardinality as the local optimal solution (line 7 – 32). If the cardinality of a chain γ equals the number of clients, then we can simply create an assignment based on γ (line 7 – 9). Otherwise, we will create an assignment as follows.

We first assign $\hat{c}_t$ to $\hat{v}_t$ for the client at tail of the chain (line 12). Then we find the set of all the service nodes V' on the shortest path from $\hat{v}_t$ to the root service node r (line 13), and assign client c_j to $v_i \in V'$ if $(c_j, v_i, \lambda(j,i))$ appears in γ (line 14 – 18). By doing this, the cardinality does not increase since all the service nodes on the shortest path from $\hat{v}_t$ to r will be used during the communication. Then we choose the service nodes in γ that can take the largest number of clients and assign clients to it using the same method and repeat this until all clients are assigned (line 19 – 27). The tree induced by the assignment with minimal cardinality is chosen as our final solution (line 39 – 41).

In Algorithm SCAP-DVMC-H, creating L is $O(np)$ and sorting it takes $O(np\log(np))$ time. Finding a chain starting with a particular element as its head takes at most

Input: s, C, $G = (V, E)$, $d()$, μ, ν
Output: Assignment A

Create an empty assignment A;
$N = \infty$;
foreach $r \in V$ **do**
    Create and sort the list L by $\lambda(j, i)$ in non-descending order;
    Γ = set of all *non-left-extendable chains* satisfying ν;
    foreach *chain* $\gamma \in \Gamma$ **do**
        if $|\gamma| = |C|$ **then**
            Create an assignment A' from γ;
            $N' = |T_D(A')|$;
        else
            h = head of γ, t = tail of γ;
            Assign $\hat{c_t}$ to $\hat{v_t}$ in A';
            V' = set of service nodes on the path from $\hat{v_t}$ to r;
            foreach $(\hat{c_j}, \hat{v_j}, \hat{d_j}) \in \gamma$ **do**
                if $\hat{v_j} \in V$ **then**
                    Assign $\hat{c_j}$ to $\hat{v_t}$ in A';
                end
            end
            while *not all clients are assigned* **do**
                Find the service node v_i appears in γ that can take the largest number of clients;
                V' = set of service nodes on the path from v_i to r;
                foreach $(\hat{c_j}, \hat{v_j}, \hat{d_j}) \in \gamma$ **do**
                    if $\hat{v_j} \in V$ **then**
                        Assign $\hat{c_j}$ to $\hat{v_t}$ in A';
                    end
                end
            end
            $N' = |T_D(A)|$;
        end
        if $N' < N$ **then**
            $A = A', N = N'$;
        end
    end
end
return A

Algorithm 4: SCAP–DVMC-H(r, s, C, G, μ, ν)

$O(np)$. The resulting complexity is $O(n^2p^2)$ since the size of L is at most np. Identifying non-left-extendable chains from the chains only needs a single pass through all possible chains which takes $O(np)$. Creating an assignment from a chain takes $O(n^2p)$ (n service nodes and a chain is at most np long). Hence the complexity of each iteration is $O(n^3p^2) = O(n^2p) \times O(np)$ and the overall time complexity of *SCAP-DVMC-A* is $O(n^4p^2)$.

Algorithm SCAP-DVMC-IP-CHAIN

Two of the major disadvantages of Algorithm SCAP-DVMC-IP and Model *SCAP-DVMC(r)* are (a) extreme long execution time and (b) huge memory consumption. These are due to the huge search space created by the size of constraint set (2.7) in Model *SCAP-MC(r)*. The model ran out of 12 GB of memory easily when solving larger instances ($p \geq 200, n = 30$). Based on the fact the delay variation must be less than ν in a feasible solution, we can effectively reduce the search space at the cost of searching multiple smaller overlapped search spaces. The idea is to find all search spaces such that the delay bound μ and delay variation bound ν are satisfied, then the optimal in each smaller search space is found and the one with smallest cardinality is optimum. We developed Algorithm SCAP-DVMC-IP-Chain (Algorithm 5) using this idea wherein Model *Chain-MC(r, L)* is used to find the local optimal within each small search space.

Similar to our previous algorithms, Algorithm *DVMC-IP-Chain* iterates through all possible r as the root service node. Each iteration works as follows. First a list L is constructed as done in Algorithm SCAP-DVMC-H (line 4). We use $l_i = (\hat{c}_i, \hat{v}_i, \hat{d}_i)$ to denote i-th element in L and let the size of L be q. Then for each (distinct) delay value $\hat{d}_h$ in L, we find all the elements in the list such that their delays are between $\hat{d}_h$ and $\hat{d}_h + \nu$ (line 6). We use $L_h = [l_h \ l_t]$ to denote this sublist. If all clients are covered in L_h, Model *Chain-MC(r, l)* is used to solve this subproblem with smaller

search space (line 8 – 13). Otherwise, there's no feasible from $[l_h\ l_t]$. Best known assignment A is updated if the returned assignment A' uses less number of service nodes. This procedure is repeated until we reach the tail of L.

Input: s, C, $G = (V, E)$, $d()$, μ, ν
Output: Assignment A

```
1  Create an empty assignment A;
2  n = ∞;
3  foreach r ∈ V do
4  |   Create and sort the list L = {l_1, l_2, ··· l_q} (|L| = q) by λ(j, i) in non-descending order;
5  |   for h = 1 to q do
6  |   |   Scan from l_h until the largest t such that l_t such that d̂_t ≤ d̂_h + ν;
7  |   |   L_h = [l_h l_t], the sublist of L from l_h to l_t;
8  |   |   if all the clients are covered in L_h then
9  |   |   |   A', n' = Model Chain-MC(r, L_h) ( assignment A' and its cardinality n');
10 |   |   |   if n' < n then
11 |   |   |   |   A = A', n = n';
12 |   |   |   end
13 |   |   end
14 |   |   if t == q then
15 |   |   |   break;
16 |   |   end
17 |   |   while d̂_{h+1} == d̂_h do
18 |   |   |   h = h + 1;
19 |   |   end
20 |   end
21 end
22 return A
```

Algorithm 5: DVMC-IP-Chain(s, C, G, μ, ν)

Model *Chain-MC(r, l)* (which is shown in Figure 2.5) and Model *SCAP-DMC(r)* only differ in the input. Model *SCAP-DMC(r)* constructs the constraints from the application session D, but Model *Chain-MC(r, l)* constructs the constraints only from L_h as follows. Recall that each element in $[l_h\ l_t]$ consists of the tuples $(c_j, v_k, \lambda(j, k))$. For any two tuples $(c_j, v_k, \lambda(j, k))$ and $(c_{j'}, v_{k'}, \lambda(j', k'))$ in $[l_h\ l_t]$, $|\lambda(j, k)) - \lambda(j', k')| \leq \nu$. Hence we only add the binary decision variables Y_{ji} to the model if $(c_j, v_i, \lambda(j, i))$ in the model and they take the value 1 if client j is assigned to service node v_i, 0 otherwise. Binary decision variables X_i are the same as defined in Model *SCAP-DMC(r)*.

Model Chain-MC(r, l)

$$\text{minimize} \quad \sum_{v_i \in V} X_i \tag{2.8}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \ c_j \in C, \Lambda_D(c_j, v_i, r) \leq \mu \ (c_j, v_i, d_{ji}) \in l \tag{2.9}$$

$$\sum_{c_j \in C} \sum_{v_{i'} \in V} a_{i'i} Y_{ji'} \leq p X_i \qquad \forall \ v_i \in V, \ (c_j, v_i, d_{ji}) \tag{2.10}$$

Figure 2.5: Model Chain-MC(r, l)

2.5 Performance Evaluation

In our experimental evaluation, 30 different networks are randomly generated for each configuration listed in Table2.2 by using Tiers [7], a random Internet topology generator. The latencies of the links in the SON are reduced to 70%, 80% or 90% to represent the well-provisioned links. Since the links on SON have smaller latencies, it is reasonable to multiply the minimum delay found by SCAP-MD-A by a factor (100%, 110% or 120%) as the delay bounds. The configurations are summarized in Table 2.2.

Table 2.2: Network Generation Parameters in SCAP Experiments

Parameter	Values
Size of SON	10, 20, 30, 40, 50
Num. of Clients	50, 80, 100, 200, 300, 400, 500
SON Delay Reduction	70%, 80%, 90%
Delay Bound Adjustment	100%, 110%, 120%

The algorithms are implemented using C/C++ with *Gurobi Optimizer* C++ library (Version 4.6.1) [31] using GNU C++ compiler. The experiments are done on a 16-core Intel Xeon (*E5520* at 2.27 GHz) machine with 12 GB of RAM installed and running *Ubuntu* (*3.0.0-12-generic* kernel). Multithreading (up to 16 threads) is

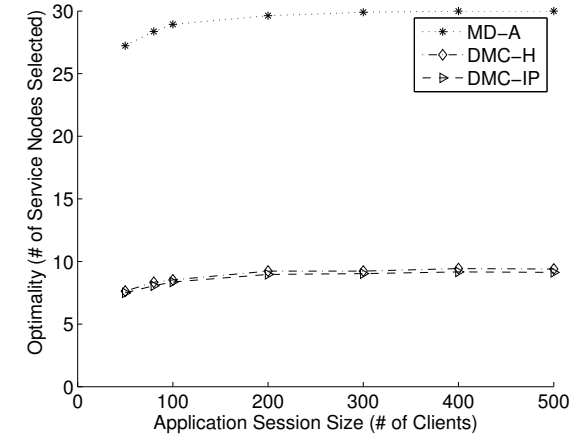
used when possible for parallel barrier in Gurobi solver. The algorithms compared are shown in Table 2.1. We measure computational complexity (execution time) and optimality (number of service nodes selected) in our experiments, and take the average from 30 instances for different network configurations. We only present part of the results from the instances with 30 servers, 70% SON delay reduction and 110% delay bound adjustment. The results for other configurations are similar.

2.5.1 SCAP-MD and SCAP-DMC

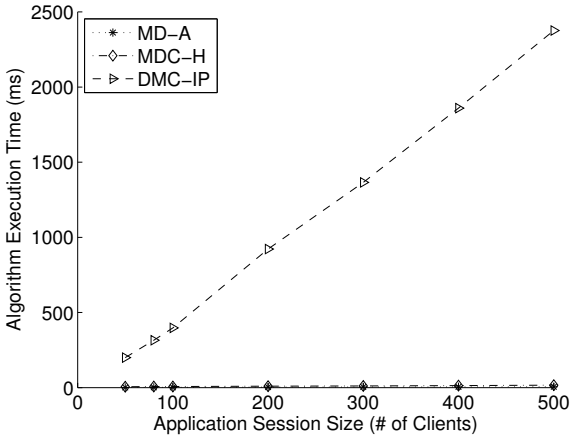
We first compare Algorithms SCAP-MD-A, SCAP-DMC-IP and SCAP-DMC-H in Figure 2.6. We can observe that almost all service nodes are used in order to achieve the lowest latency (SCAP-MD-A). On the other hand, the number of service nodes selected in a solution can be significantly reduced when we are looking to satisfy a reasonable delay bound instead of minimum delay. Algorithm SCAP-DMC-H performs pretty well in terms of the number of service nodes selected in comparison to the optimum from Algorithm SCAP-DMC-IP. Our heuristic is significantly faster than Algorithm SCAP-DMC-IP. However, the average time to find optimal using Algorithm SCAP-DMC-IP is only about 2.5 seconds for $n = |V| = 30$ and $p = |C| = 500$, which is reasonable. It appears that our heuristic is more practical for solving large scale instances.

2.5.2 SCAP-MD and SCAP-DV

We first use Algorithm SCAM-MD-A to find the lowest possible delay for an application session and use this value as the delay bound μ for Algorithm SCAP-DV-A. From Figure 2.7, we can observe that Algorithm SCAP-DV-A achieves the same delay bound while reducing delay variation by about $\frac{4}{5}$ (Figure 2.7a) with increased execution time (Figure 2.7b). With $n = 30, p = 500$, the average execution time is about 80 seconds and this may be reasonable for application sessions that will last



(a)



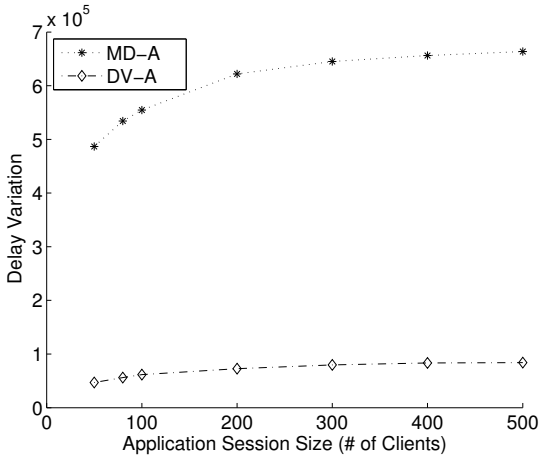
(b)

Figure 2.6: Comparisons between SCAP-MD and SCAP-DMC algorithms, which include SCAP-MD-A(MD-A), SCAP-DMC-IP (DMC-IP) and SCAP-DMC-H (DMC-H). (a) Number of service nodes used. (b) Execution time.

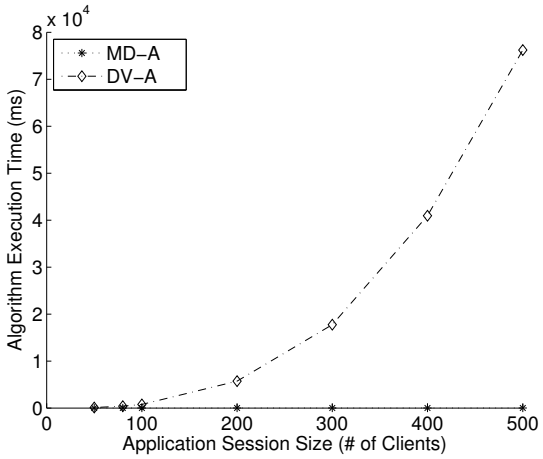
for hours. But for larger instances, a faster algorithm is needed. We omit the figure that shows the numbers of service nodes selected for these two algorithms since both algorithms select about the same number of service nodes.

2.5.3 SCAP-DVMC

Figure 2.8 presents the results of solving SCAP-DVMC using different approaches, the algorithms included are Algorithm SCAP-DVMC-IP, Algorithm SCAP-DVMC-



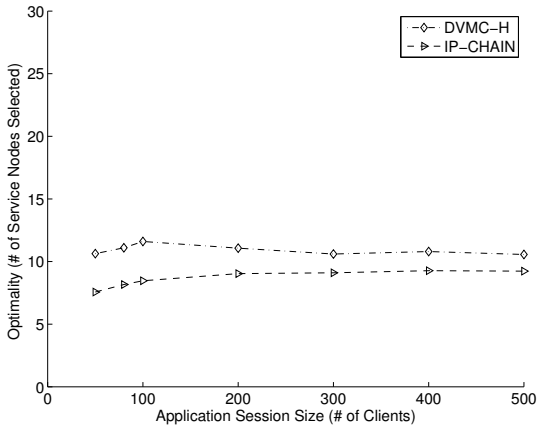
(a)



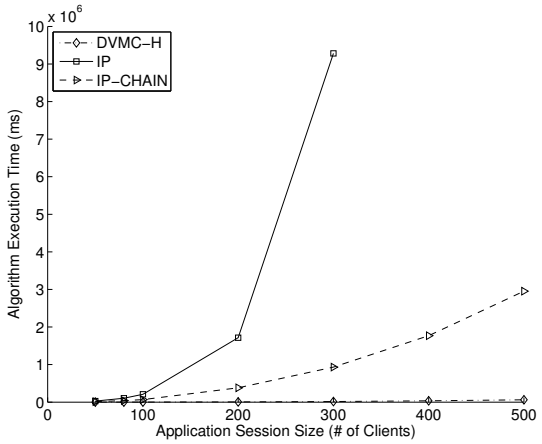
(b)

Figure 2.7: Comparison of SCAP-MD-A (MD-A) and SCAP-DV-A (DV-A) algorithms. (a) Delay variations. (b) Execution time.

H and Algorithm SCAP-DVMC-IP-Chain. Here we use Algorithm SCAP-MD-A to find best possible delay for an application session and multiply it by 1.1 as the delay bound μ. Delay variation bound ν is set to $\frac{\mu}{3}$. Both Algorithms SCAP-DVMC-IP and SCAP-DVMC-IP-Chain give optimal solutions in terms of the number of service nodes used. Algorithm SCAP-DVMC-H gives the results that are close to optimal (Figure 2.8a). We can also observe from Figure 2.8b that the execution time is improved by a large amount from SCAP-DVMC-IP to SCAP-DVMC-IP-Chain. On the other hand, our heuristic SCAP-DVMC-H has significantly shorter execution time than both of



(a)



(b)

Figure 2.8: Comparisons of SCAP-DVMC algorithms: SCAP-DVMC-IP (DVMC-IP), SCAP-DVMC-H (DVMC-H) and SCAP-DVMC-IP-Chain (IP-CHAIN). (a) Number of service nodes used, optimal is obtained by SCAP-DVMC-IP-CHAIN. (b) Execution time.

them. Although it does not show clearly in the figure, SCAP-DVMC-H takes about 60 seconds (average) for the problems with $n = 30, p = 500$. Note that Algorithm SCAP-DVMC-IP creates a huge model and it can only solve a subset of the problems with $p = 200, 300$ ($n = 30$) before it runs out of 12 GB RAM. It is not able to solve larger problems (for $p = 400, 500$).

2.6 Summary

In this chapter, we have considered Server and Client Assignment problem (SCAP) using a service overlay network (SON) to improve the performance of application session where a group of users are involved. We presented several algorithms to balance the user experience and resource usage for several variants which include SCAP-MD, SCAP-DMC, SCAP-DV and SCAP-DVMC. The contributions and results of this research are summarized as follows.

- We provided exact algorithms for SCAP-MD and SCAP-DV which are polynomial-time solvable
- SCAP-DMC and SCAP-DVMC are $\mathcal{NP}$-hard and we presented algorithms using Integer Programming (for finding optimal solution) and heuristic algorithms (for finding approximation).
- We exploited the problem structure of SCAP-DVMC and provided an alternative algorithm using Integer Programming that significantly reduces the execution time.
- The quality of our heuristic algorithms are good (compared to optimal). They run in a short amount of time and are suitable for large scale instances.
- Integer Programming approach takes significantly longer time to find the optimal solutions.

The material presented in this chapter also appears in [13, 14] wherein a similar problem – Server Selection Problem in which the server is chosen from one of the service nodes. The results of Server Selection Problem are presented in Appendix A.

Chapter 3

Clients Assignment Problem

3.1 Introduction

We have addressed the group communication using the client-server architecture in Chapter 2. In this chapter, we propose to construct and use a peer-to-peer (P2P) overlay network for group communication among clients involving multimedia data. Similar to the two-layer architecture in Chapter 2, the communicating clients connect to appropriate nodes (also called *service nodes*) that forms a P2P network. The service nodes are responsible for distributing the multimedia content to other service nodes using this P2P network, and eventually to the clients that connect to it.

There has been a number of research papers on the design of P2P networks for group communication [4, 29, 32, 46, 58]. The use of overlay networks can be witnessed by the successful file-sharing services such as Bit-Torrent [5], Napster [45], Gnutella [61], eDonkey [60], and others. However, the inability of the Internet routers to fully-support multicasting and related communication resulted in the development of algorithms for the design of *overlay networks* and protocols to carry content on them [23]. Examples of overlay multicast include Application Layer Multicasting (ALM) and End System Multicast (ESM) [9, 16, 58]. In an overlay network, the nodes are end hosts and links are the Internet paths connecting them. Each node has the ability to *multicast* to its neighbors by using multiple *unicasts* to conserve the network bandwidth.

Duan et al. [22] first proposed the notion of *Service Overlay Networks* (SONs) as

a design mechanism to bring about value-added Internet services by placing *service nodes* (gateways) at strategic locations on the Internet. These service nodes are programmable with the ability to offer a wide-variety of end-to-end Quality-of-Service (QoS) guarantees [1, 8, 22]. The service nodes are connected by the path of Internet routers forming an overlay network and they can incorporate the overlay multicast functionality. Bandwidth can be provisioned on the links along the path between any two service nodes. This overlay network of service nodes with well-provided bandwidths becomes an important resource for many delay-sensitive multimedia applications such as game playing, video conferencing, and others.

Most group communication proposals on P2P networks consider each participant (or client) as a peer node. The participant node usually connects to a series of routers (in some cases through wireless routers) to access the Internet router. The content travels through a series of routers to reach other peers. The path from the participant node(s) to the nearest Internet backbone router(s) suffers from bandwidth limitations when compared to available bandwidths on the Internet backbone. The problem is more acute when a participant node is a multicasting node – here the participant node has to duplicate every message it receives from its parent (equal to the number of children in the multicasting tree) and each message will follow the same bandwidth constrained path to the nearest Internet router. The above has been illustrated and discussed in Chapter 1.

To overcome some of issues above, the multicasting responsibilities can be handed over to the service nodes which send the messages to the other service nodes and to the clients that are connected to it. Given a SON, a set of clients (that are significantly larger than the number of nodes in the SON), and delays from each client to each of the nodes in the SON, our goal is select for each client a node in SON (contact node) that the client should connect to while satisfying a variety of desirable properties. The contact nodes and other service nodes on the SON that act

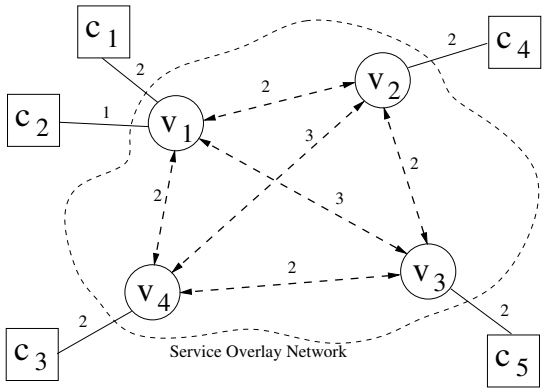
as intermediate routers communicate with each other in a P2P fashion. We illustrate this two-layer approach in Fig. 3.1a. Fig. **??** illustrates how the multimedia content can be delivered from c_5 to other clients. Note that there may be other service nodes along the path between the two contact nodes that are not by themselves contact nodes. For example, although there is a logical data flow path between v_2 and v_3, the actual data flow goes through v_5 as shown in Fig. 3.1c. However, there is only one copy of data is sent from v_3 to v_2 when multicasting is used.

The contact nodes determined as part of this research are not only based on satisfying one or more the properties below, but are also chosen in such a way that the total number of nodes required to establish the P2P communication is minimal. Some of the desirable properties include:

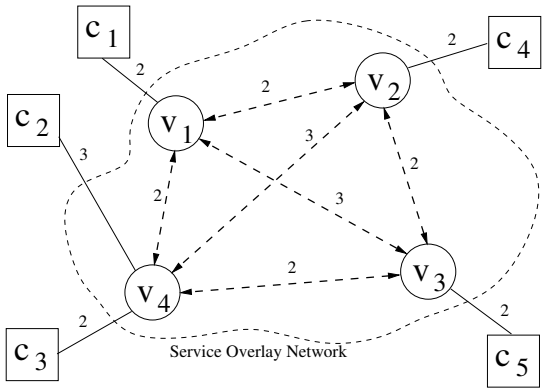
a. *Delay*: In a group communication set up, we would like to keep end-to-end delay between pairs of participants to a minimum or bounded by a desired delay.

b. *Delay Variation*: It is undesirable or even in some applications unfair for certain participants to receive messages early while others receive it late. Hence a desirable property would be to keep the variations in delay among pairs of participants to a minimum or bounded.

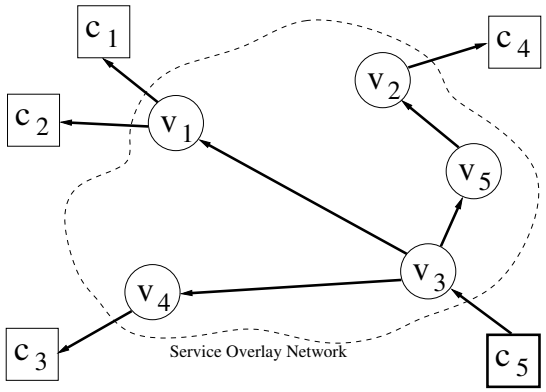
We have developed IP models (interesting in its own right) and polynomial-time heuristics for several variants of the overall goal stated above. The IP models are used to examine the quality of the solutions that our heuristics produce. The rest of this chapter is organized as follows. In section 3.2, we provide additional information on some related works. System model and notations are described in more detail in section 3.3. Section 3.4 gives the problem formulation along with corresponding Integer Programming (IP) formulations and proposed algorithms. Performance evaluation is presented in section 3.5 and conclusions are summarized in section 3.6.



(a)



(b)



(c)

Figure 3.1: Two-layer approach for peer-to-peer communication using a SON (a) An assignment for participants $\{c_1, c_2 \ldots c_5\}$ and the P2P network formed by service nodes $\{v_1, v_2, v_3, v_4\}$. Intermediate nodes are not shown in the figure. The links shown by dashed lines between the service nodes represent logical data flows and the numbers denote the delays of thm. The delay for this assignment is 7 and delay variation is 4. (b) A different assignment with a higher delay of 8 but a lower delay variation of 3. (c) Distributing multimedia content from c_5 to other clients.

3.2 Related Work

Lee et al. [40] proposed a well-provisioned network of *mirrored servers* for massively multiplayer online games (MMOG). A distributed algorithm called Zoom-In Zoom-Out (ZIZO) is proposed to reduce the number of *contact nodes* used by a game session. ZIZO works for the mirrored server architecture, which is similar to the P2P model in our assumption. There are two major differences between work of Lee et al. [40] and our research. First, they try to minimize only the number of *contact nodes* in the resulting network, not the the total number of nodes involved in the P2P communication as in our work. Second, several variants of the problem (bounded delay and delay variation) are considered in our research, which are not part of the work in [40] .

3.3 System Model and Notations

We define the terminologies, notations, system model that are used to describe our problems in this section.

3.3.1 Service Overlay Networks (SONs)

We use the same notations from Chapter 2 to denote the well-provisioned service overlay network – $G = (V, E)$ and delay function $d()$, where $V = \{v_i | 1 \leq i \leq n\}$ is the set of n *service nodes* which are capable of overlay multicast and E is the set of m links connecting them. We also use $d(v_i, v_{i'})$ to denote the latency between $v_i, v_{i'} \in V$ and $v_i \neq v_{i'}$. Other details and properties are described in Chapter 2.

3.3.2 Application Session

An interactive application session is denoted as $D = (C, G)$, which consists of a group of p participants (or clients) $C = \{c_j | 1 \leq j \leq p\}$. Each participating client

has access to any service node in V through the Internet. We use $d(c_j, v_i)$ to denote the Internet latency between $c_j \in C$ and $v_i \in V$. During an application session, each client periodically generates updates, which are delivered to all other clients in the same application session.

3.3.3 Assignments

As we briefly mentioned earlier in Section 3.1 and Figure 3.1, the participants need to connect to the service nodes in order to utilize the service overlay network. We call the service nodes that the participants connect to as *contact service nodes* (or simply *contact nodes*) and the mapping from the clients to the contact nodes an assignment. We use $A = \{a_1, a_2, \cdots a_p\}$ to denote an assignment where $a_1, a_2, \cdots a_p \in V$ are the contact nodes for $c_1, c_2, \cdots c_n$, respectively. Note that $a_j = a_{j'}$ if client c_j and client $c_{j'}$ are assigned to the same contact node.

A communication subnetwork of G for the application session D can be derived from an assignment A and we use $H_D(A)$ to denote it.

3.3.4 Delays (Latencies)

End-to-end network delay (latency) is usually used to measure the Quality-of-Service (QoS) of the network. We use $\Lambda_{D,A}(c_j, c_{j'})$ to denote the delay between participants c_j and $c_{j'}$ under the assignment A for application session $D = (C, G)$. $\Lambda_{D,A}(c_j, c_{j'})$ is the sum of the delays of the links from c_j to a_j (through the Internet), then from a_j to $a_{j'}$ (within SON) and from $a_{j'}$ to $c_{j'}$ (again, through the Internet). We further define the delay for an application session D under the assignment A as the largest end-to-end delay between any two clients:

$$\Lambda_D(A) = \max \Lambda_{D,A}(c_j, c_{j'})$$
$$= \max[d(c_j, a_j) + d(a_j, a_{j'}) + d(c_{j'}, a_{j'})] \quad \forall \quad c_j, c_{j'} \in C, c_j \neq c_{j'}$$

3.3.5 Delay variation

The synchronization of application state among participating clients is important for certain types of interactive applications such as video/audio conferencing or online games. For these types of applications, ideally the packet sent by one participant should arrive at other participants at approximately the same time to address fairness. Rouskas et al. [51] first defined the term *delay variation* in the context of client-server communication. The delay variation was defined as the difference between maximum and minimum delays from any client to the server node. Approaches for solving delay variation problems including finding alternative paths or packet buffering and some of these results are presented in [2, 50, 51]. We extend this definition of delay variation for an application session using the peer-to-peer model as the difference between maximum and minimum end-to-end delays betweeen pairs of clients as follows:

$$\Delta_D(A) = \max |\Lambda_{D,A}(c_j, c_{j'}) - \Lambda_{D,A}(d_j, d_{j'})| \quad \forall \quad c_j, c_{j'}, d_j, d_{j'} \in C, c_j \neq c_{j'}, d_j \neq d_{j'}$$

3.4 Problem Formulations and Algorithms

Given a well-provisioned service overlay network $G = (V, E)$, the set of participants C, latency function $d()$ (for G and from $C \times V$), we consider two variants of the *Client Assignment Problem* (CAP).

- *CAP with Delay Bound and Minimum Cardinality (CAP-DMC)*: In this problem, our goal is to find an assignment A along with the induced subgraph such that the following requirements are satisfied:

  a. The given delay bound μ is satisfied ($\Lambda_D(A) \leq \mu$).

  b. The size of the induced subgraph ($|H_D(A)|$) is minimum.

- *CAP with Delay Bound and Delay Variation Reduction (CAP-DVR)*: In this problem, we aim to refine the assignment A by CAP-DMC-H to find a new assignment A' while the following requirements are satisfied:

  a. The delay bound is satisfied ($\Lambda_D(A) \leq \mu$).

  b. A' has less delay variation ($\Delta_D(A') < \Delta_D(A)$), or ideally $\Delta_D(A')$ is minimum.

  c. The size of the induced subgraph does not increase ($|H_D(A')| \leq |H_D(A)|$).

In this section, we provide nontrivial integer programming (IP) models for CAP-DMC and heuristic algorithms for both CAP-DMC and CAP-DVR.

3.4.1 CAP with Delay Bound and Minimum Cardinality (CAP-DMC)

CAP-DMC problem is $\mathcal{NP}$-hard and can be proved by reduction from set covering problem as the Server Allocation Problem [40]. We provide two different IP models (Model CAP-DMC-A and Model CAP-DMC-B) and a heuristic algorithm for CAP-DMC in this section.

Model CAP-DMC-A

We introduce the following notations to better describe the Model CAP-DMC-A. Let $K = \{(c_j, v_i, v_{i'}, c_{j'})|\ c_j, c_{j'} \in C,\ c_j > c_{j'}, v_i, v_{i'} \in V\}$ be the set of all possible tuples representing the connecting paths between participants c_j and $c_{j'}$, for which we define

Model CAP-DMC-A

$$\text{minimize} \quad \sum_{v_i \in V} X_i \tag{3.1}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \ \ c_j \in C \tag{3.2}$$

$$\sum_{k \in K_{ji}} Z_k = (p-1) Y_{ji} \quad \forall \ \ c_j \in C, v_i \in V \tag{3.3}$$

$$\sum_{k \in K} Z_k a_{ki} \leq p(p-1) X_i \quad \forall \ \ v_i \in V \tag{3.4}$$

Figure 3.2: Model CAP-DMC-A

d_k as the delay value for $k \in K$. A tuple $k \in K$ specifies a participant c_j, connected to a service node v_i, which in turn is connected to another participant $c_{j'}$ via contact node $v_{i'}$. Let a_k be a column vector of dimension n that represents the service nodes on a path connecting two service nodes in k. An entry $a_{ki\prime}$ is 1 if server $v_{i\prime}$ is on path k, 0 otherwise. Further, we define $K_{ji} \subset K$ such that paths $k \in K_{ji}$ start with c_j with contact node v_i.

We use decision variables Y_{ji} and X_i to represent the assignment of c_j to service node v_i, and selection of service node v_i (if it is used), respectively. In addition, we define binary variables Z_k that take the value 1 if path k is selected, meaning that the path between two participants is decided, and 0 otherwise. We impose the maximum delay constraint implicitly in a preprocessing stage and hence it does not appear in the model. If for any $k \in K$ $d_k > \mu$, then we eliminate the corresponding Z_k from the formulation. The complete IP formulation for problem CAP-DMC is presented in Figure 3.2.

The objective (3.1) is to minimize the total number of service nodes that are needed to accommodate a given P2P network. Each participant has to be assigned to exactly one service node, as described in (3.2). If c_j is assigned to v_i, then we have to select as many paths as there are remaining clients $(p-1)$ that start with c_j, v_i, thus

ensuring full client-to-client connectivity (3.3). Constraints (3.4) ensure that path k cannot be selected, unless all servers that are on k are also selected.

Model CAP-DMC-B

In model *CAP-DMC-A*, we explicitly capture all possible paths between any pair of clients using all possible contact nodes. This approach, in effect, creates a large number of decision variables – one for each path, $O(p^2n^2)$ total. It also removes the need to explicitly impose delay constraints, because only those paths that are within the delay bound are retained as input to the model.

An alternative approach is to identify the participant with the largest delay assigned to each of the contact nodes, then measure and control the delay of the paths between all such participants. This would ensure that all other participants connecting through the same pair of contact nodes will have equal or less amount of delay (thus satisfying delay constraints implicitly), by virtue of the fact that communication between any pair of service nodes always takes place on the same shortest path.

Based on this observation, we present an alternative formulation that has a less number of decision variables, but more constraints. Here we define $d_{ii'}$ as the delay between two service nodes $v_i, v_{i'}$ (through the shortest path). Further we define $a_{ii'}$ as a column vector of dimension n that represents a path of connected servers. An entry $a_{ii'i''}$ is 1 if server $v_{i''}$ is on the path from server v_i to $v_{i'}$, 0 otherwise.

We retain decision variables Y_{ji} and X_i as in Model *CAP-DMC-A*. We redefine path selection variables $Z_{ii'}$ to take value 1 if both v_i and $v_{i'}$ are selected as contact nodes, thereby selecting the path in between, and 0 otherwise. The paths between $v_i, v_{i'}$ such that $d_{ii'} > \mu$ are identified in preprocessing and related $Z_{ii'}$ variables are removed from the formulation. In addition, we define F_i a continuous valued non-negative variable that measures the largest delay among all clients that are assigned to v_i, and Q_i a binary variable that takes the value 1 if v_i serves as a contact node.

Model CAP-DMC-B

$$\text{minimize} \quad \sum_{v_i \in V} X_i \tag{3.5}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \ j \in C \tag{3.6}$$

$$\sum_{c_j \in C} Y_{ji} \leq pQ_i \qquad \forall \ v_i \in V \tag{3.7}$$

$$Z_{ii'} \geq Q_i + Q_{i'} - 1 \qquad \forall \ v_i, v_{i'} \in V \tag{3.8}$$

$$\sum_{v_i, v_{i'} \in S, v_{i'} > v_i} a_{ii'i''} Z_{ii'} \leq p(p-1) X_{i''} \qquad \forall \ v_{i''} \in V \tag{3.9}$$

$$F_i \geq d_{ji} Y_{ji} \qquad \forall \ v_i \in V, c_j \in C \tag{3.10}$$

$$Y_{ji} + Y_{j'i} \leq 1 \qquad \forall \ v_i \in V, \ c_j, c_{j'} \in C, \ d_{ji} + d_{j'i} > \mu \tag{3.11}$$

$$Q_i + Q_{i'} - 1 \leq 0 \qquad \forall \ v_i, v_{i'} \in V, \ v_{i'} > v_i, \ d_{ii'} > \mu \tag{3.12}$$

$$F_i + d_{ii'} Y_{ii'} + F_{i'} \leq \mu \qquad \forall \ v_i, v_{i'} \in V, \ v_{i'} > v_i, \ d_{ii'} < \mu \tag{3.13}$$

Figure 3.3: Model CAP-DMC-B

Next, we present the alternative formulation in Figure 3.3.

The objective (3.5) is to minimize the total number of service nodes that are needed to accommodate a given P2P network. Constraints (3.6) and (3.7) are the same as in model CAP-DMC-A. Constraints (3.8) ensure that when a pair of service nodes v_i, and $v_{i'}$ are selected as contact nodes, the path in between them is also selected. Constraints (3.9) are similar to (3.4) in the previous model and serve the same purpose. The rest of the constraints, (3.10) − (3.13), are the ones that are extra in this formulation (compared to the previous one), and collectively they impose the delay constraints between all pairs of participants. In (3.10), at each service node, the largest delay from all assigned participants is measured. In (3.11), same service node assignments that violate the delay bound are eliminated. Constraints (3.12) ensure that any two contact nodes whose connecting path in between violates the

delay bound are not both selected. Finally, constraints (3.13) measure the largest delay path in between every pair of contact nodes and enforces the delay bound.

Algorithm CAP-DMC-H

We have developed a heuristic algorithm called SPD-CS-H(r) for the server selection problem using client-server model in [13], which is presented in Procedure SPD-CS-H. Procedure SPD-CS-H is used to find an assignment such that the delay from any client c_j to r is bound by $\bar{\mu}$ while keeping the number of service nodes involved minimal. It works as follows. It starts with a service node r and the tree T only consists of r. It assigns the clients to this tree T without violating the delay constraint $\bar{\mu}$. If all the clients are assigned, we have found an assignment. Otherwise, we choose a service node s that is a neighbor of T (a neighbor of some node in T) such that s can serve as a contact node for a maximum number of unassigned clients without violating $\bar{\mu}$. Now the node s is added to T and more clients are assigned. The above process is repeated until all clients are assigned or all service nodes are in T.

The desired peer-to-peer communication subnetwork of CAP-DMC problem can be found by using SPD-CS-H(r) by setting $\bar{\mu}$ to $\frac{\mu}{2}$ based on the following fact. Suppose we have an assignment A with the central server $s \in V$ given by SPD-CS-H(r) such that the maximum delay between a client c_j and s through c_j's contact node a_j is bounded by $\frac{\mu}{2}$. Let $c_j \neq c_{j'}$ be any two clients with contact nodes $a_j, a_{j'}$ in A, respectively. Then $d(c_j, a_j) + d(a_j, s) + d(c_{j'}, s_{j'}) + d(s_{j'}, s) \leq \mu$. If the communication between $c_j, c_{j'}$ is done in a peer-to-peer manner, then $d(c_j, c_{j'}) = d(c_j, a_j) + d(a_j, a_{j'}) + d(c_{j'}, s_{j'})$. There are two cases, s lies on one of the shortest paths between a_j and $a_{j'}$ or not. In either case, $d(c_j, a_j) + d(a_j, a_{j'}) + d(c_{j'}, s_{j'}) \leq d(c_j, a_j) + d(a_j, s) + d(a_{j'}, s) + d(c_{j'}, a_{j'}) \leq \mu$. Hence A can be converted to a solution for *CAP-DMC* instance with the same delay bound μ.

Input: $G = (V, E)$, C, latency function $d()$, delay bound μ, root r
Output: assignment A

```
1  T = {r},not_done = false, found = true;
2  foreach c_j ∈ C do
3  |  if d(c_j, r) < μ then
4  |  |  Assign c_j to r;
5  |  else
6  |  |  not_done = true, found = false;
7  |  end
8  end
9  Mark r as used;
10 while not_done do
11 |  s = NULL, n_max = 0;
12 |  V' = neighbors of service nodes in T;
13 |  foreach v_i ∈ V' do
14 |  |  n = number of clients can be assigned to v_i;
15 |  |  if  n > n_max then
16 |  |  |  s = v_i, n_max = n;
17 |  |  end
18 |  end
19 |  Mark s as used;
20 |  foreach unassigned client c_j ∈ C do
21 |  |  if d(c_j, s) + d_s(s, r) < μ then
22 |  |  |  Assign c_j to s;
23 |  |  end
24 |  end
25 |  if all clients are assigned then
26 |  |  found = true, not_done = false;
27 |  end
28 |  if all service nodes are used & found = false then
29 |  |  found = false, not_done = false;
30 |  |  A = φ, n = 0;
31 |  end
32 end
33 return A
```

Procedure SPD-CS-H(r)

Input: $G = (V, E)$, C, latency function $d()$, delay bound μ
Output: subnetwork H, assignment A

```
A = SPD-CS-H(G, C, d(), μ/2, v_1);
Find the induced subgraph H_D(A);
foreach v_i ∈ V − {v_1} do
    A' = SPD-CS-H(G = (V, E), C, d(), μ/2, v_i);
    Find the induced subgraph H_D(A');
    if |H_D(A)| > |H_D(A')| then
        A = A';
    end
end
V' = contact nodes used in A;
H = subgraph of G derived from A;
return H, A
```

Algorithm 6: CAP-DMC-H

Algorithm 6 is designed for CAP-DMC based on Procedure SPD-CS-H(r). First we use SPD-CS-H(r) to find an assignment A. Then we derive the subgraph H from A, which is the communication subgraph used by the application session. The complexity of the SPDCS-H(r) algorithm is $O(n^2p)$, deriving the subgraph H can be done in $O(n^3)$. The above steps are performed n times, hence the overall complexity of this algorithm is $O(n^3p)$ with $n < p$.

Zoom-In Zoom-Out Algorithm (ZIZO)

ZIZO algorithm by Lee et al. [40] is a heuristic for the *mirrored server* architecture, in which the underlying communication is the peer-to-peer model. Hence it can be used to solve CAP-DMC problem with a slight modification. The *ZIZO* algorithm first allocates the clients to the nearest servers (service nodes, in our term) and migrates them toward the *core server* s^* (that minimizes the longest shortest distance to all the clients) to reduce the number of servers used. Example shown in Fig 3.4 illustrates the existence of an assignment with the delay bound $\mu = 16$. However, ZIZO fails to find a solution in this example for $\mu = 18$. The initial assignment gives the minimal delay of 19 and the ZIZO algorithm stops. Our algorithm finds the solution with delay bound 16. Note that we consider all nodes used by an application session but

Lee et al. [40] consider only the number of contact nodes.

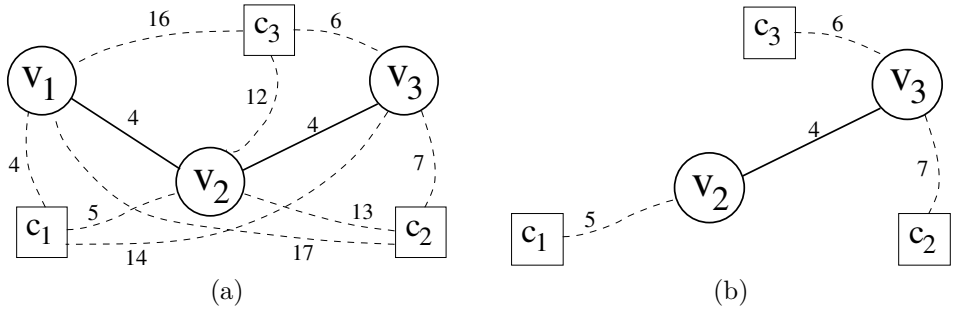


Figure 3.4: (a) An example shows that ZIZO fails to find a solution with $\mu = 18$ where v_1, v_2, v_3 are servers (service nodes) and c_1, c_2, c_3 are clients. (b) A shortest path tree rooted at v_3 has the depth of 9 which gives the delay bound $\mu = 18$. Our heuristic will find a solution with the delay 16 if we set the delay bound parameter to 18.

3.4.2 CAP with Delay Bound and Delay Variation Reduction (CAP-DVR)

We developed Algorithm CAP-DVR-H to refine an assignment A for a CAP-DMC problem. For the new assignment A', (a) the delay variation is reduced ($\Delta_D(A') < \Delta_D(A)$) and (b) the number of service nodes used is no more than $A(\ |H_D(A')| \leq |H_D(A)|$. Note that although we do not provide an IP model for CAP-DVR, The two IP models for CAP-DMC presented earlier can be used for CAP-DVR by only considering the nodes in the induced subgraph.

Algorithm CAP-DVR-H

To reduce the delay variation, we use a *reassignment* approach that modifies current assignment A by decreasing the maximum delay or increasing the minimum delay in A. However, the delay variation is not necessary reduced since all the latencies corresponding to this participant have changed after the reassignment because of the P2P communication model.

Input: $G = (V, E)$, C, latency function $d()$, assignment A, delay bound μ
Output: h, d, dv, Γ

Create the set of service nodes involved in A as S';
Construct and sort the list L by the delay values in non-decreasing order;
$done = false$;
while !$done$ **do**
    Let $c_j, c_{j'}, d$ be the two participants and the delay value of the **first** element in L;
    $(A', L') = Reassign(c_j, G, C, d(), \mu, A, S', L)$;
    if $A'! = \phi$ **then**
        $A = A', L = L'$;
    else
        $(A', L') = Reassign(c_{j'}, G, C, d(), \mu, A, S', L)$;
        if $A'! = \phi$ **then**
            $A = A', L = L'$;
        else
            $done = true$;
        end
    end
end
$done = false$;
while !$done$ **do**
    Let $c_j, c_{j'}, d$ be the two participants and the delay value of the **last** element in L;
    $(A', L') = Reassign(c_j, G, C, d(), \mu, A, S', L)$;
    if $A'! = \phi$ **then**
        $A = A', L = L'$;
    else
        $(A', L') = Reassign(c_{j'}, G, C, d(), \mu, A, S', L)$;
        if $A'! = \phi$ **then**
            $A = A', L = L'$;
        else
            $done = true$;
        end
    end
end
return $d_v(L)$, A

Algorithm 7: CAP-DVR-H

Algorithm CAP-DVR-H (Algorithm 7) is based on this reassignment approach and works as follows. For a given assignment A, we find the set of service nodes involved in the assignment S'. A list L sorted by $d_{i,i'}$ (in non-decreasing order) which contains the set of tuples $(c_j, c_{j'}, d^A_{j,j'})$ is created, where $d^A_{j,j'}$ is the delay between $c_j, c_{j'}$ under A. The delay variation of L is the difference between the delays of the first and the last elements.

The algorithm goes through in two phases. In the first phase (line 4 – 17), the goal is to reduce the delay variation by increasing the smallest delay in the list L. This is done by reassigning one of the two clients $c_j, c_{j'}$ at the beginning of L to service nodes in S'. We use *Procedure Reassign()* to reassign c_j, get a new assignment A' and a new list L' with a smaller delay variation if it exists. In the case such an assignment is found, A and L are replaced by A' and L', respectively. This procedure is repeated until no improvements can be made. We next use *Procedure Reassign()* to find a new assignment by reassigning the second client $c_{j'}$. If reassignment can be made to improve the delay variation, we go back to the beginning of the first phase with the new assignment and list. Otherwise, we move forward to the second phase. The second phase (line 19 – 32) is similar to the first phase, but instead of reassigning the clients at the beginning of L, the clients at the end of L are reassigned to reduce the largest delay.

The set of used service nodes S' and L constructed in lines 1–2 can be done in $O(n^2)$ given the all-pair shortest path between servers. A single iteration in the first loop (lines 4 – 17) is dominated by *Procedure Reassign()* which can be done in $O(pn^2)$. The second loop from lines 19 – 32 also takes $O(pn^2)$ time for a single iteration. Let us denote delay variation of the given assignment as D_v, assume only integer values are involved. These two loops are executed at most D_v times which could not be determined before L was constructed. However, D_v is also bounded by μ, which is considered to be a constant. Hence the algorithm has pseudo-polynomial time

Input: c_j, G, C, delay function $d()$, μ, A, S', L
Output: new assignment A', new list L'

```
1  A' = A, L' = L;
2  foreach v_k ∈ S' do
3  |   Create assignment A'' by reassigning c_j to v_k;
4  |   Construct the new list L'' from A'';
5  |   if d_v(L'') < d_v(L') and d(L'') ≤ μ then
6  |   |   A' = A'', L' = L'';
7  |   |   reassigned = true;
8  |   end
9  end
10 if ! reassigned then
11 |   A' = φ, L' = φ;
12 end
13 return A', L'
```

Procedure Reassign$(c, G, C, B, \mu, A, S', L)$

complexity of $O(pn^2)$. In our experiments, the first loop (phase) is only executed a few times (< 10) and the second phase even fewer times.

3.5 Performance Evaluation

We used Tiers [7] to generate Internet-like graphs to evaluate our algorithms. For each different graph size configuration (summarized in TABLE 3.1), we generated 30 instances of input and calculated the average as the result. The latencies of the links are reduced to 70% to represent the well-provisioned service overlay network. For each instance of the input, the minimum possible delay Γ for a given session is found and then multiplied by a factor $f \in \{1.0, 1.1, 1.2\}$ to represent the different real-time requirement for different application types.

The algorithms are implemented in C/C++ with *Gurobi Optimizer 4.5* C++ library [31] for solving the integer programming models. The experiments are done on a 16-core Intel Xeon (*E5520* at 2.27 GHz) machine with 12 GB of RAM running *Ubuntu* (*2.6.28-11-generic* kernel). Multithreading (up to 16 threads) is used when possible for parallel barrier in Gurobi solver.

For the problem CAP-DMC, we compared the performance of different algorithms

Table 3.1: Network Generation Parameters in CAP Experiments

Parameter	Values
Size of SON (n)	$10, 20, 30, 40, 50$
Num. of Clients (p)	$50, 80, 100, 200, 300, 400$
SON Delay Reduction	$70\%, 80\%, 90\%$
Delay Bound Adjustment (μ)	$100\%, 110\%, 120\%$

in terms of number of service nodes selected and execution time. For the problem CAP-DVR, we compare the amount of reduction on the delay variation and the execution time of the algorithm. We only show the results of the networks with 50 servers with delay bound 1.1, results of other configurations are similar. Note that integer programming approach takes longer to find the solutions for larger delay bound since there are many more possible choices.

3.5.1 CAP-DMC

We compared 4 different approaches which include (a) Nearest: assign the clients to their nearest contact node (b) CAP-DMC-H: our heuristic algorithm (c) IP: Model CAP-DMC-B and (d) ZIZO: from Lee et al. [40]. Due to the $\mathcal{NP}$-hardness of the problem, we were not able to use IP approach to find the optimal solutions within an acceptable amount of time. Hence we limit the execution time of the solver for each instance to 1 hour and compare the results found (if any) with our heuristic. Although we provided two IP models, we observed that Model CAP-DMC-B gives better results within the time limit and we used it in all comparisons.

- Optimality: Figure 3.5a shows the number of service nodes selected by different algorithms. Our heuristic CAP-DMC-H is able to find the solutions that are close to the solutions given by *Model CAP-DMC-B*. When the number of clients is greater than 300, our heuristic found better solutions than *Model CAP-DMC-B*. The figure also shows *ZIZO* does not perform as good as CAP-DMC-H. Note

that in some instances, integer programming approach was not able to find the solutions within the time limit.

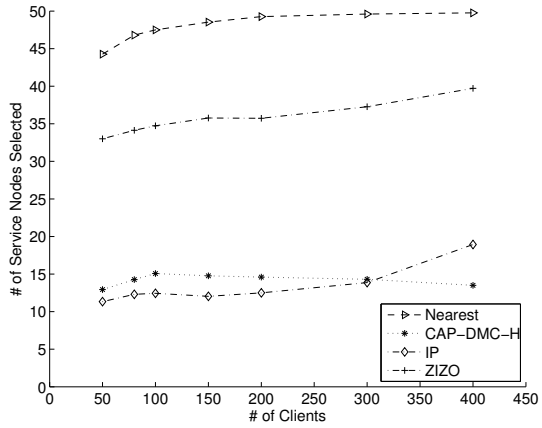
- Time Complexity: The execution time of different algorithms are shown in Figure 3.5b. *Model CAP-DMC-B* was not able to find the optimal solutions for the input size greater than 200 clients although Gurobi solver utilizes all 16 cores when solving the instances. We also observe that in most cases, *Model CAP-DMC-B* spent a certain amount of time to find a solution and then used an enormous amount of time (hours to days) to verify its optimality during our pilot study. We further compared the execution time of CAP-DMC-H and ZIZO in Figure 3.5c and the results show that CAP-DMC-H has shorter execution time.

3.5.2 CAP-DVR

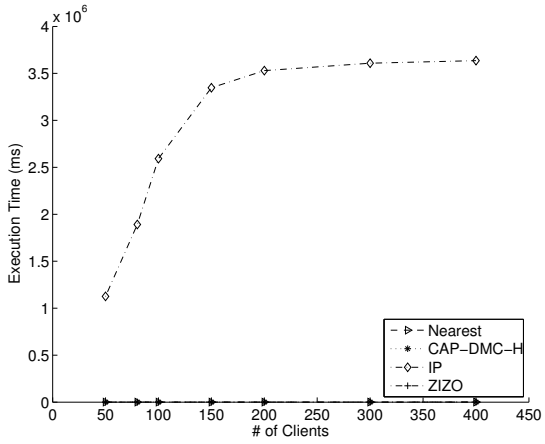
We evaluated the performance of CAP-DVR-H algorithm (Algorithm 7) as follows. For each of the instances, CAP-DMC-H algorithm was first used to solve the instance. Then CAP-DVR-H is used to reduce the delay variation of the solution. The results are shown in Figure 3.6.

- Delay Reduction: From Figure 3.6a, we can observe that the delay variation values range from 80% – 90% of the delay bound μ for the initial solution. After we applied CAP-DVR-H algorithm, the values drop to about 50%, which is about a 30% – 40% improvement.

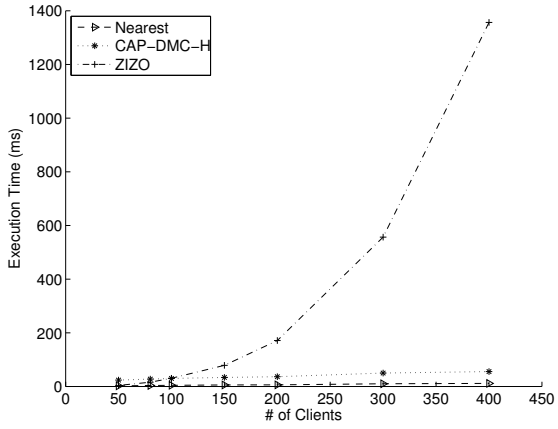
- Execution Time: Figure 3.6b shows the execution time of CAP-DVR-H algorithm (and CAP-DMC-H algorithm). Although the algorithm has psuedo-polynomial time complexity of $O(pn^2)$, it still takes about 90 seconds for instances with 400 clients. The figure also shows that the execution time is polynomial in the number of participants.



(a)



(b)



(c)

Figure 3.5: Results of CAP-DMC (50 servers, Bound = 1.1): (a) Optimality (b) Execution time of algorithms with different number of clients (c) Execution time of Nearest, CAP-DMC-H and ZIZO.

- Delay: We are also interested in the maximum delay values for each solution before and after applying CAP-DVR-H algorithm. The delay values (in terms of % of delay bound μ) are shown in Figure 3.6c and we can observe that the delay value increases and approaches μ after CAP-DVR-H algorithm is applied (about 10% increase). However the delay bound is still satisfied.

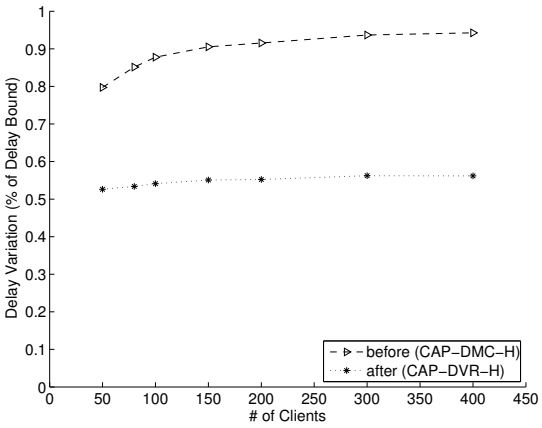
3.6 Summary

In this chapter, we have considered Client Assignment problem (CAP) using a service overlay network (SON) to improve the network performance of application session where a group of participants are involved. Two different versions of CAP are addressed in this chapter which include CAP with delay bound and minimal cardinality (CAP-DMC) and CAP with delay bound and delay reduction (CAP-DVR). We presented two nontrivial integer programming models and a heuristic algorithm (CAP-DMC-H) for CAP-DMC. For CAP-DVR, we presented a heuristic algorithm (CAP-DVR-H) to reduce the delay variation. Experiments were conducted to evaluate different algorithms, the results are summarized as follows.

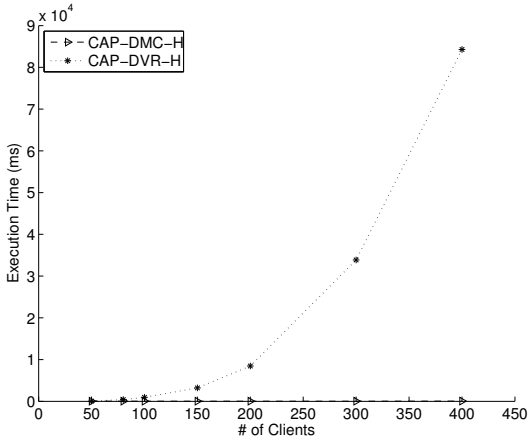
- The number of service nodes selected by CAP-DMC-H algorithm is close to the solutions found by using integer programming approach which we ran for a maximum time of one hour (Model CAP-DMC-B). On the other hand, ZIZO [40] found solutions that use more than twice of the service nodes in comparison with the solution found by the CAP-DMC-H algorithm.

- CAP-DMC-H also has faster execution time than both the integer programming approach and ZIZO [40].

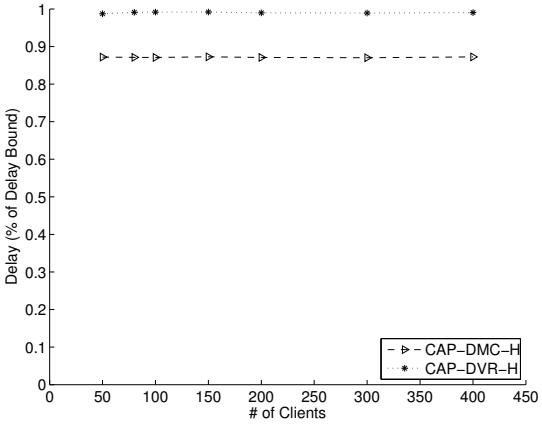
- CAP-DVR can efficiently reduce the delay variation of any solution by about 30% to 40% of the given delay bound μ. This is done at the cost of increasing



(a)



(b)



(c)

Figure 3.6: Results of CAP-DVR (50 servers, Bound = 1.1): (a) Delay variation reduction by CAP-DVR (b) Execution time (c) New delay values after CAP-DVR-H

of maximum delay of a solution while still keeping it within the delay bound μ.

The material presented in this chapter also appears in [13, 14] wherein the Server Selection Problem is considered.

Chapter 4

Multi-stream Multi-source Multicast Routing

4.1 Introduction

Multicasting is an efficient way to deliver the multimedia contents or large files from a single source to multiple destinations. It can be performed at the network layer taking into account the Internet routers that support Internet Group Management Protocol (IGMP) in IPv4 [54] or Multicast Listener Discovery (MLD) in IPv6 [55]. However, IGMP and MLD are not widely available to end users. An alternative approach called Application layer multicasting [3,9,16,58] is done using the concept of *overlay network* where the nodes are the end-hosts and the links are paths formed by Internet routers. Application layer multicasting is very flexible in the sense that newer protocols can be easily incorporated at the end-hosts, but are less efficient because the multicasting paths may sometimes involve overlapping Internet paths [26]. Multicast backbone (Mbone) [24] uses IP tunneling to connect "multicast islands" and allow end users to access it.

There has been a plethora of research activity dealing with the construction of multicasting trees that satisfy various constraints. For example, the problem of constructing a single source serving a single multimedia stream wherein minimum delay is desired can be solved efficiently by constructing a single source shortest path tree and pruning subtrees that do not have a destination node. In cases where the delay bound, delay variation bound, node degree bound, and others are desired the multicasting tree construction problems have been shown to be $\mathcal{NP}$-hard [2, 58].

There has also been a growing interest in building multiple multicast trees. Castro et al. [9] developed SplitStream where they split the source stream into k stripes and multicast them using disjoint multicast trees, i.e, the trees do not share common interior nodes. The destinations (or subscribers) then obtain each stripe from different trees. S. Birrer et al. [4] address the issue of bandwidth, especially it being the bottleneck as we move closer to the root (or source). They do this by building *fat-trees* for multicasting, wherein the outgoing links near the root have higher bandwidth compared to links that are further away from the root.

One approach to solving the multi-stream multi-source problem is to build multicasting trees for each stream and combine the multicast trees. This approach may not always produce a result (or one that is desirable). For example, say we have a source s and a destination t and there are two video streams needed to be sent from s to t and each consume 1 unit of bandwidth. If we solve the problem for each of the streams individually, we may get two edge-joint paths which consume 2 units of the bandwidth on the common edges. A better solution with less congestion could be two edge-disjoint paths from s to t which results in 1 unit of the bandwidth usage. Several papers have addressed these issues for different scenarios such as minimum interference routing between source-destination pairs in multi-protocol label switched (MPLS) networks [28, 36] or multicasting group packing [12, 39, 59].

In this chapter, we consider the problem of delivering multiple multimedia streams to their destinations taking into consideration that each stream can originate from one or more sources. Our goal is to develop algorithms to reduce the congestion on the communication links and increase their residual bandwidths. The rest of this chapter is organized as follows. Several relevant research works are reviewed in Section 4.2. In Section 4.3, we introduce the notations, define Multi-stream Multi-source Multicast Routing Problem MMMRP, and prove the $\mathcal{NP}$-hardness of this problem. Integer programming (IP) formulations are then provided in Section 4.4 and the heuristic

algorithms based on widest path algorithm is presented in Section 4.5. Performance evaluation and results are presented in Section 4.6 with conclusions drawn in Section 4.7.

4.2 Related Works

There have been a number of techniques for creating multiple multicasting trees that optimize various resources. For example, there have been several works that try to reduce the number of nodes that participate in the multicasting trees [58]. A number of researchers have developed techniques to minimize the total resources consumed by all multicast trees [59], and others that try to reduce the number of shared links among the multiple trees [12, 39]. There are also approaches that combine many constraints such as the number of nodes, total bandwidth, and bandwidth constraints on links [12, 39, 59].

The minimum interference routing problem is discussed in [28, 36]. Kar et al. [36] considered the problem of routing data between source-destination pairs in MPLS networks. Data from the source is routed to destinations using one more more edge-disjoint paths. Figueiredo et al. [28] later developed an algorithm which improved its computation time.

Chen et al. [12] considered the multicast tree packing problem wherein groups of participants communicate with other participants within the same group. Each group uses a multicast tree for many-to-many multicasting as illustrated in Figure 4.1. The goal of multicast packing problem is to minimize the maximum congestion (the number of times a link is shared) among the communication links while keeping the size of each multicast tree within a bound. Chen et al. [12] developed IP models together with a heuristics algorithm called *TreePacking*. Their solution methodology involves solving multiple Steiner tree problems individually and then improving the

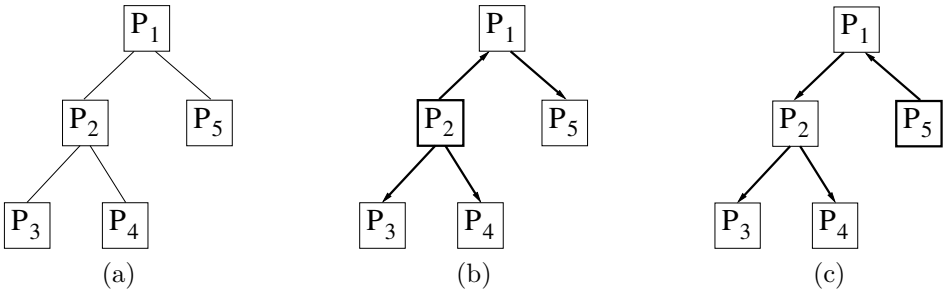


Figure 4.1: (a) Using a multicasting tree for many-to-many communication among $\{P_1, P_2 \cdots P_5\}$. (b) P_2 as the source. (c) P_5 as the source.

solution by reconstructing the trees that use the most congested link(s).

The problems considered in [12] assume that each multicast tree requires the same amount of bandwidth, in other words all multimedia streams served require the same bandwidth. Lee and Cho [39] considered the same problem in which the bandwidth consumption are all different and provided an algorithm called *MMTA*. Wang et al. [59] address the similar problem but with a different objective wherein they aim to reduce the total cost of the multicast trees (cost on the communication links) while satisfying the bandwidth constraints of the communication links.

The research works mentioned above assume that the multicast sessions consume constant bandwidth during their lifetime. Ravindran et al. [49] considers the problem of changes to the bandwidth that can occur at various points in a multimedia streaming environment and provide technique to find routing paths.

The main difference between the problem addressed in this chapter and the work in [12, 39, 59] is that our work considers the case in which each multicast session has *one or more sources* that can provide the data stream. The multicast trees that are constructed in [12, 39, 59] are used for group communication and there is no requirement to take into consideration source nodes. That is, each member in the group performs peer-to-peer communication with others in the group. Existing solutions [12,39,59] are not suitable for the problem under consideration based on the following reasons. First, the existing solutions use Steiner tree heuristics to reduce

the number of participating nodes. If we relax the number of nodes constraint and focus on just the bandwidth related constraints, it may be possible to find better solutions that maximize minimal residual bandwidth. Second, we cannot remove the Steiner tree construction parts from the existing heuristics as they are their core. Third, in order to apply the existing solutions mentioned above, we have to for each stream, treat the sources and destinations as a group. Since no distinction between sources and destinations are made, the algorithm may unnecessarily try to optimize bandwidth related constraints between source nodes.

The heuristics proposed in [12, 39, 59] involve construction of Steiner trees which is an expensive computation (since they all try to minimize the number of nodes in the solution). The proposed solution in this chapter does not involve Steiner trees and hence shown to be very scalable. We have shown that heuristic produces very good results in few seconds on networks whose sizes are in thousands.

4.3 Problem Definition

In this section, we introduce the notations for *Multi-stream Multi-source Multicast Routing Problem* (MMMRP), define and model the problem, then summarize the variants of problems similar to MMMRP (see Table 4.1).

4.3.1 Notations

Let $G = (V, E)$ be an undirected graph representing the communication network, where $V = \{v_i | 1 \leq i \leq n\}$ is the set of n nodes and $E = \{e_j | 1 \leq j \leq m\}$ is the set of m communication links connecting them. We use c_j to denote the capacity (bandwidth) of link e_j. For convenience, we use A_i to denote the set of neighbor(s) of node v_i in G and assume each v_i has the capability to multicast to its neighbors.

Assume there are r data streams supplied and requested by some of the nodes in

V. Let $W = \{w_k |\ 1 \leq k \leq r\}$ be the set of r distinct data streams and b_k be the bandwidth requirement of data stream w_k. We use S_k to denote the set of source nodes that can supply data stream w_k and D_k for the set of nodes that demands data stream w_k. Let $|S_k|$ and $|D_k|$ be the cardinality of S_k and D_k, respectively. Note that $|S_k| \geq 1, |D_k| \geq 1,\ 1 \leq k \leq r$. We denote $\delta_k = (w_k, S_k, D_k)$ as a multicast session or multicast group and $\delta_k = (w_k, s_k, D_k)$ for a multicast session when there is only one source s_k in S_k.

4.3.2 Multi-stream Multi-source Multicast Routing Problem

The Multi-stream Multi-source Multicast Routing Problem (MMMRP) can be formulated using the notations introduced earlier as follows. Given a network $G = (V, E)$, a set of data streams $W = \{w_k |\ 1 \leq k \leq r\}$, and for each stream w_k the set of nodes $S_k \subset V$ that can supply $w_k \in W$ and the set of nodes $D_k \subset V$ that demands $w_k \in W$.

The goal of MMMRP is to find a *multicast forest* F_k for each data stream w_k to deliver w_k from any of the sources in S_k to their destinations in D_k such that the minimum residual bandwidth $(c_j - \sum_{k=1\cdots r}^{e_j \in F_k} b_k,\ 1 \leq j \leq m)$ is maximum.

A multicast forest F_k is defined as the set of $|S_k|$ trees satisfying the following conditions.

- For any tree $t_i^k \in F_k$, it is rooted at a node in S_k.
- For any two trees $t_i^k \neq t_j^k \in F_k$, they do not share any link or nodes. That is, when a single stream is supplied by two or more sources, the trees from each of these sources do not share links.
- Every node in D_k is in exactly one tree in F_k.

Table 4.1: Problems Similar to MMMRP

Item	(For Each Session/Group) Num. of Sessions	Num. of Sources	Num. of Destinations	Problem Description
1.	1	1	1	Shortest Path Problem [21] Widest Path Problem [33]
2.	1	N/A	> 1 (Group size)	Steiner Tree Problem [6, 58] $*$ The tree has minimum cost
3.	r	1	1	Minimum Interference Routing Problem [28, 36] $*$ A flow is split into multiple integral flows $r \geq 1$
4.	1	1	> 1	SplitStream [9] $*$ A stream is split into multiple chunks
5.	r	N/A	> 1 (Group size)	Multicast Tree Packing [4, 12, 39, 49, 59] $*$ With tree size constraint $r \geq 1$
6.	r	> 1	> 1	MMMRP (this chapter and [15]) $r \geq 1$

Lower bound for a special case

When the links are homogenous (in terms of available bandwidth), the problem is equivalent to minimizing the maximum bandwidth consumption among the links. If we also assume that all the data streams consume the same unit bandwidth, a loose upper bound for the maximum bandwidth consumption is $|W| = r$ and a loose lower bound is $\max\lceil \frac{|\bar{W}_i|}{deg(v_i)} \rceil,\ \forall\ v_i \in V.\ |\bar{W}_i|$ is the total number of data streams v_i supplies or demands, $deg(v_i)$ is the degree of v_i in G.

4.3.3 Similar Problems

We have reviewed several related works in Section 4.2, and we summarize these problems in this section and Table 4.1 along with MMMRP.

1. A single stream with a single source and a single destination. The problem is well-known as the shortest path problem, which can be solved using Dijkstra's

algorithm [21]. As far as residual bandwidth is concerned, a modified version of Dijkstra's algorithm can be used to solve the problem.

2. A single multicast group with multiple participants. This is similar to previous works where in the problem is modeled as Steiner tree problems [6, 58] when the total cost of the tree and/or the diameter of the tree are considered.

3. Multiple streams with a single source and a single destination for each session. This is the minimum interference routing problem addressed in [28, 36].

4. A single stream with a single source and multiple destinations. This is the problem addressed by SplitStream [9] where the stream can be divided into multiple chunks and delivered to the destinations using different multicast trees. This is a special case of 6 below where each stream has a single source and each session has the same source and destinations.

5. Multiple multicast tree packing. This is addressed in [4, 12, 39, 49, 59] where the members of each group use the same multicast tree for communication. The total cost of the tree is considered in the problem.

6. Multi-stream Multi-source Multicast Routing Problem. This is the problem we are addressing in this research, where multiple streams coexist and each stream can have multiple sources and multiple destinations.

As we mentioned previously, 5 and 6 have the same optimal value when (a) the tree cost constraint is relaxed in 5 and (b) there is only one source for each session in 6. For convenience, we call this special version of 5 and 6 as Simple MTP and Simple MMMRP, respectively. Here we briefly define these two problems and show the proof of this claim.

Given a network $G = (V, E)$ as described earlier and a set of sessions $\Delta = \{\delta_k = (w_k, s_k, D_k) |\ 1 \leq k \leq r\}$. The goal of Simple MMMRP is to find a multicast tree $\hat{t}_k$

rooted at s_k for each session δ_k such that the minimal residual bandwidth is maximum. Also, let each multicast group be $\lambda_k = \{s_k\} \cup D_k,\ 1 \leq k \leq r$. The objective of the corresponding Simple MTP is to find a multicast tree $\bar{t}_k$ for each group λ_k such that the minimal residual bandwidth is maximum.

Theorem 2. *Simple MMMRP and Simple MTP have the same optimal value.*

Proof. It is possible that a Simple MMMRP has more than one optimal solutions with the same objective value X. Similarly, Simple MTP also has more than one optimal solutions that give the same objective value Y. Let one of the the optimal solutions of a Simple MMMRP instance that gives X be $\hat{T} = \{\hat{t}_k^* | 1 \leq k \leq l\}$. Clearly each $\hat{t}_k^*$ spans $\{s_k\} \cup D_k$ which is also the multicast group λ_k. Hence $\hat{T}$ is also a feasible solution of the corresponding Simple MTP instance and $Y \leq X$.

Similarly, let one of the optimal solutions of a Simple MTP instance that gives the optimal objective value Y be $\bar{T} = \{\bar{t}_k^* | 1 \leq k \leq l\}$. Since each $\bar{t}_k^*$ spans $\lambda_k = \{s_k\} \cup D_k$, we can simply re-orient each $\bar{t}_k^*$ such that s_k is the root. Hence $\bar{T}$ is also a feasible solution of the corresponding Simple MMMRP instance and $X \leq Y$.

From above, we can conclude $X = Y$. □

4.3.4 $\mathcal{NP}$-Hardness of MMMRP

The multicast tree packing problems discussed in [12, 39, 59] are $\mathcal{NP}$-hard because solutions for the multicast tree packing problems are answer to Steiner tree problems. Here we show that MMMRP is $\mathcal{NP}$-hard even without the tree cost constraints.

Theorem 3. *MMMRP is $\mathcal{NP}$-hard.*

Proof. Consider the multi-commodity integral flow problem which is shown to be NP-complete by Karp [37]. Note that Even et al. [25] also show that it is true even if the number of commodities is 2. We will show that multi-commodity integral flow problem is polynomial-time reducible to MMMRP.

Given a graph $G = (V, E)$ with its capacity function $c : E \rightarrow N$ where N is the set of nonnegative integers. We are also given commodity source-terminal pairs $\{(s_1, t_1), (s_2, t_2) \cdots (s_k, t_k)\}$ and nonnegative-integer requirements of these pairs $\{R_1, R_2 \cdots R_k\}$. We use e_v to denote the edges that have one end point at $v \in V$. The multi-commodity integral flow problem is to determine whether there exist the flow functions $\{f_1, f_2 \cdots f_k\}$ from $E \rightarrow N$ such that

- The capacity constraint $c(e)$ for each edge e is satisfied.
- The total outgoing flow equals the total incoming flow for each commodity i at each $v \in V - s_i, t_i$.
- The net incoming flow for each commodity i at t_i is greater than R_i.

Now we will construct an instance of MMMRP from the multi-commodity integral flow problem as follows. Consider each commodity i with the requirement R_i. Since R_i is an integer, we can divide i into R_i unit flows and create R_i source-terminal pairs $\{(s_{i,1}, t_{i,1}), (s_{i,2}, t_{i,2}) \cdots (s_{i,R_i}, t_{i,R_i})\}$. This is actually an instance of MMMRP where each session has 1 source and 1 destination with 1 unit bandwidth requirement. The conversion can be done in $\sum_{i=1}^{k} |R_i|$ time. Hence multi-commodity integral flow problem $\leq_P$ MMMRP and the later is $\mathcal{NP}$-hard.

□

4.4 Integer Programming Formulations

One of the important properties of the classic transshipment problems or network flow problems is the total supply equals the total demand for the nodes [17]. However, a data packet can be duplicated at any intermediate nodes that support multicasting. Hence the linear programming models for solving classic network problems cannot be used directly to solve MMMRP. Here we will treat the problem as a network flow

Model MMMRP

Maximize Z

Subject to:

$$\sum_{v_i \in S_k} \sum_{v_{i'} \in A_i} X_{ii'k} = |D_k| \quad \forall \ w_k \in W \tag{4.1}$$

$$\sum_{v_{i'} \in A_i} X_{i'ik} - 1 = \sum_{v_{i'} \in A_i} X_{ii'k} \quad \forall \ w_k \in W, v_i \in D_k \tag{4.2}$$

$$\sum_{v_{i'} \in A_i} X_{ii'k} = \sum_{v_{i'} \in A_i} X_{i'ik} \quad \forall \ w_k \in W, v_i \in V, \ v_i \notin S_k, v_i \notin D_k \tag{4.3}$$

$$X_{ii'k} \leq cF_{ii'k} \quad \forall \ w_k \in W, \ v_i \in V, v_{i'} \in A_i \tag{4.4}$$

$$\sum_{v_{i'} \in A_i} F_{i'ik} = 0 \quad \forall \ w_k \in W, v_i \in S_k \tag{4.5}$$

$$\sum_{v_{i'} \in A_i} F_{i'ik} \leq 1 \quad \forall \ w_k \in W \ v_i \in V, v_i \notin S_k \tag{4.6}$$

$$c_j - \sum_{w_k \in W} [b_k \cdot (F_{ii'k} + F_{i'ik})] \geq Z \quad \forall \ (v_i, v_i') = e_j \in E \tag{4.7}$$

Figure 4.2: Model MMMRP

problem, but add some additional decision variables and constraints to incorporate multicasting in this problem. The following decision variables are defined to be used in Model MMMRP.

- $X_{ii'k}$: non-negative integer variables that represent the total number of w_k's flow from the edge (v_i, v_i') when treated as a network flow problem.
- $F_{ii'k}$: binary variables that take the value 1 if $X_{ii'k}$ is *positive*, 0 when $X_{ii'k}$ is 0. This also represents if w_k flows through the edge v_i to v_i' in MMMRP.
- Z: A non-negative integer variable for measuring the minimum residual bandwidth among the links, which is also the objective function.

We define a constant c, an integer, greater than or equal to $\max |D_k| \ \forall \ w_k \in W$. The model is presented in Figure 4.2.

The objective function Z measures the maximum residual bandwidth among the links. Constraints (4.1) to (4.3) are as used in the classic network flow problems. Constraints (4.1) ensure the copies a data stream w_k sent out by its source nodes equal the number of requests. On the other hand, constraints (4.2) enforce the property that the number of copies of incoming data stream w_k is exactly 1 more than that of outgoing copies at a node that demands w_k. Constraints (4.3) assures for each intermediate node v_i, the number of outgoing and incoming copies of w_k are the same. Constraints (4.4) are used to determine if a data stream w_k flows from v_i to v_i'. If there is at least one copy of w_k that flows from v_i to v_i', $F_{ii'k}$ is set to 1 by this constraint, 0 otherwise. We use constraints (4.5) and (4.6) to remove the cycles based on the following two observations: (i) there should not be any incoming data stream w_k from any neighbor of v_i if $v_i \in S_k$ (Constraints (4.5)) and (ii) there should be at most 1 neighbor of v_i supplying data stream w_k to v_i if $v_i \notin S$ (Constraints (4.6)). Constraints (4.7) measure the residual bandwidth on each link using Z. When Z is less than 0 the instance is infeasible.

4.5 Algorithm MMForests

We present a heuristic algorithm *MMforests* based on *widest-path algorithm* for MMMRP in Algorithm 8.

The idea of MMforests is as follows. First, we set the capacity c_j of each communication link e_j to $|W|$ (the number of data streams), which is the loose upper bound (line 2). In the case of maximizing the minimum residual bandwidth, capacities are set to the given values. Then for each of the data streams w_i, we construct a *multicast forest* f_i that spans the destination set D_i (line 5, 10) and each tree is rooted at one of the source nodes in S_i. The trees in f_i do not have nodes or edges in common. Then we update the residual bandwidth by subtracting 1 from c_j if e_j is in f_i, and

Input: $G = (V, E)$, data stream set W, sources and destinations of each stream w_k: $\{S_k\}$, $\{D_k\}$.
Output: Set of multicast forests F

```
1  F = φ;
2  Set the capacity {c_j} of each e_j ∈ E to r (|W| = r );
3  C = {c_j};
4  G' = (V, E, C);
5  Forest f_1 = DijkstraForest(G', S_1, D_1);
6  foreach e_j ∈ f_1 do
7  |   c_j = c_j − 1;
8  end
9  foreach w_i ∈ W − {w_1} do
10 |   Forest f_i = WPForest(G', S_i, D_i);
11 |   foreach e_j ∈ f_i do
12 |   |   c_j = c_j − 1;
13 |   end
14 end
15 F = {f_i};
16 return F
```

Algorithm 8: MMForests Algorithm

repeat this until all data streams are processed (line 6 – 8, 11 – 13).

Each *multicast forest* is constructed using Widest-Path Forest Algorithm (Algorithm 10) which is based on the widest path tree algorithm (Algorithm 9, a modified version of Dijkstra's algorithm [21]) except the original Dijkstra's algorithm is used for the first data stream. Widest-Path Forest Algorithm works as follows.

We first construct the single source widest paths for each of the sources (line 2 – 4). Then we find the path from each of the destinations to one of the sources (line 5 – 22) as follows. For each of the destinations d_j, we set d_j as the current node. There will be a "widest-path" from the current node to each of the sources. We then find the widest among them and the next node on this path (line 8 – 15). Then we add the edge from current node to next node to the f and set the next node as the current node (line 16, 17). We repeat this procedure until one of the sources is reached then continue for next d_j. The resulting graph will be a forest where each tree is rooted at one of the source nodes and there is no overlapping of nodes or edges among the trees.

```
Input: G = (V, E, C), source s
Output: Widest Path Tree t
U = V;
foreach v ∈ V do
    cap[v] = 0;
    prev[v] = 0;
end
cap[s] = ∞;
while U is not empty do
    u = vertex in U with largest capacity;
    remove u from U;
    if cap[u] == 0 then
        return NULL;
    end
    foreach neighbor v of u do
        alt = min(cap[u], capacity_between(u, v));
        if alt > cap[v] then
            cap[v] = alt;
            prev[v] = u;
        end
    end
end
Construct the tree t from previous;
return t
```

Algorithm 9: Widest Path Tree Algorithm

```
Input: G = (V, E, C), S = {s_1, s_2 ···}, D = {d_1, d_2 ···}.
Output: Multicast forest f
f = φ;
foreach source s_i ∈ S do
    Tree t_i = WidestPathTree(G, s_i);
end
foreach d_j ∈ D do
    current_node = d_j;
    while true do
        next_node = next node on the widest path from current_node to s_1 in t_1;
        next_width = the bottleneck bandwidth from current_node to s_1 in t_1;
        foreach s_i ∈ S − {s_1} do
            if the bottleneck bandwidth from d_j to s_i in t_1 > next_width then
                next_node = next node on the widest path from d_j to s_i in t_i;
                next_width = the bottleneck bandwidth from d_j to s_i in t_i;
            end
        end
        Add (current_node, next_node) to f;
        current_node = next_node;
        if next_node ∈ S then
            break;
        end
    end
end
return f
```

Algorithm 10: Widest-Path Forest (WPForest) Algorithm

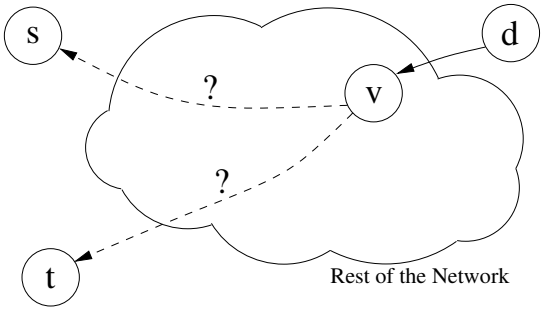


Figure 4.3: Widest Path Selection.

The reason that we do not directly construct a path from each d_j to one of the sources but instead we build the path step by step is explained using Figure 4.3. Suppose we are constructing the widest path from d to one of the source nodes (s, t). We will choose the one that is wider (say to s) and determine the next node on the path which is v in the example. If we keep going from v to s all the way, we may miss some "wider" paths if (d, v) is the bottleneck. In this case, two paths $d \leftarrow v \leftarrow \cdots \leftarrow s$ and $d \leftarrow v \leftarrow \cdots \leftarrow t$ have the same residual capacities. But our goal is to try to use the link with higher capacities, and hence we need to make a decision again at each node and so on.

Complexity of MMForests

First we consider the complexity of *WPForest*. The complexity of Dijkstra's algorithm is $O(m + n \log n)$, where m is the number of edges and n is the number of nodes. The loop from line 5 to 22 runs in $O(n \cdot |D_k||S_k|)$ time for each data stream w_k; hence, the overall complexity is $\max(O(m + n \log n), O(n \cdot |D_k||S_k|))$. *WPForest* is called r (total number of data streams/sessions) times in *MMForests* and hence the overall complexity of *MMForests* is $O(rn^2)$ when $m > n$, $|D_k|$ is bound by n and $|S_k|$ is bound by a small constant that is much less than n.

Table 4.2: Network Generation Parameters in MMMRP Experiments

Exp.	n	m	r	p	Note
1	200	$2n, 3n$	$20, 40, 60, 80, 100$	30	†
2	200	$2n, 3n$	40	$10, 20, 30, 40, 50$	†
3	$40, 80, 120, 160, 200$	$2n, 3n$	40	30	†
4	200	$2n, 3n$	40	$10, 20, 30, 40, 50$	‡
5	200	$2n, 3n$	40	30	† ◇
6	$200, 500, 1000, 2000, 4000$	$2n, 3n$	40	30	† ◇

n: num. of nodes, m: num. of edges, r: num. of sessions, p: num. of participants
†: unit bandwidth ‡: various bandwidth, equally distributed
◇: multiple sources

4.6 Performance Evaluation

We conducted experiments to evaluate the performance of different algorithms in terms of execution time and optimality (minimal residual bandwidth). For these experiments, first we used BRITE [44] to generate the underlying network $G = (V, E)$ based on Waxman's probability model. Our experiments used a variety of values, for number of nodes $n \in \{40, 80, 120, 160, 200\}$ and number of edges $m = \{2n, 3n\}$. Here we only show the results with $m = 3n$ due to space limitations but the results for $m = 2n$ are similar. We also generate input instances with $n \in \{200, 500, 1000, 2000, 4000\}$ for the purpose of evaluating the speed of our algorithm MMForests. For each of the instance, we randomly generate r sessions with p participants. The parameters used in the experiments are summarized in Table 4.2. We assume a homogenous network, in which the available bandwidths for the links are identical. Hence instead of showing the minimum residual bandwidths, the maximum bandwidth usages are show in the results. The results of 30 network instances of each input configuration are averaged and compared.

The algorithms from [12] and [39] are compared with MMForests Algorithm and Model MMMRP. These algorithms are summarized in Table 4.3. The core ideas behind Chen's algorithm [12], modified Chen's algorithm [39], MMTA [39] are similar.

Table 4.3: Algorithm Compared in Different MMMRP Experiments

Experiment	Algorithm	Notation	Description
1, 2, 3	IP	IP600	Model MMMRP, best feasible
		IPLB	Model MMMRP, lower bound
	MMForests	MMForests	Algorithm 10
	Chen's Algorithm	Chen	[12]
	Modified Chen's	mChen	[39]
	MMTA	MMTA	[39]
4	IP	IP600	Model MMMRP, best feasible
		IPLB	Model MMMRP, lower bound
	MMForests	MMForests-VBS	Sorted by bandwidth
	MMForests	MMForests	Unsorted
	MMForests	MMForests-VBRF	Sorted with refining
	MMTA	MMTA-VBAG	Alternative gain [39]
	MMTA	MMTA-VBHBF	Highest bandwidth first [39]
5	IP	IP600	Model MMMRP, best feasible
		IPLB	Model MMMRP, lower bound
	MMForests	MMForests	Algorithm 10
6	MMForests	MMForests	Algorithm 10

†Algorithms not cited are from this chapter.

First, each multicast session is solved individually as a Steiner Tree Problem. Link congestions are computed and the most congested link is identified. Second, a group using this link is selected and the corresponding multicast tree is reconstructed without using this link. The newly added links should have more residual bandwidth than the minimal residual bandwidth before reconstruction. The algorithms stop when no such group exists.

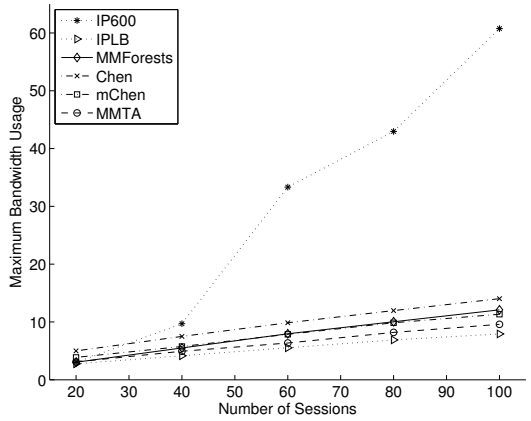
There are two approaches used for tree reconstruction. The first approach is used in Chen's algorithm [12] and MMTA [39]. Here the most congested link is removed from the tree and a path connecting these two components whose residual bandwidth is more than the minimal is inserted. The second approach is used in modified Chen's algorithm [39] wherein the links with residual bandwidth less than minimal residual bandwidth plus the bandwidth of the group are removed. A new multicast tree is constructed from the resulting graph. Both approaches guarantee an improvement *if* the tree is reconstructed, i.e., the newly added links have more residual bandwidth

than the minimal residual bandwidth. Solving Steiner tree problems is $\mathcal{NP}$-hard and we use the well-known heuristic KMB [38] in Chen's algorithm instead of solving it exactly using Integer Programming. KMB is also used in the original MMTA. Note that we do not include the algorithms from [59] because they are designed to minimize *the total costs of all trees*.

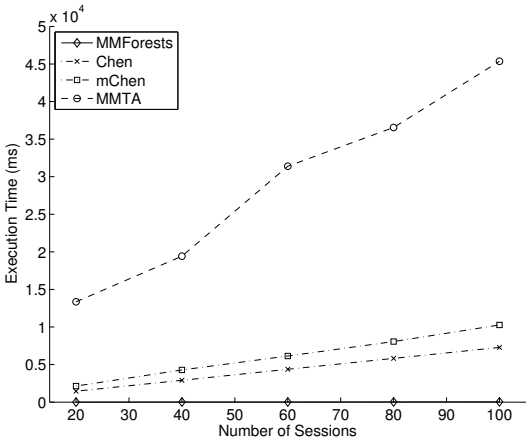
These algorithms are implemented in C/C++ and compiled using g++ 4.4.6 with Gurobi Optimizer [31] 5.0.1 C++ Library for solving the IP model. The workstation used in the experiments is an Intel Xeon (E5520 at 2.27 GHz) machine with 12 GB of RAM running Linux kernel 3.0.0-16. Multithreading (up to 16 threads) is used (when applicable) for parallel barrier in Gurobi Optimizer while the rest of the algorithms only utilize a single thread. Finding an optimal solution to a problem instance using Integer Programming may take a long period of time (hours to days) for large instances due to the $\mathcal{NP}$-hardness of the problem, hence we limit the execution time of the solver to 600 seconds and obtain *best known solution* (*IP600*) and *best available lower bound on solution* (*IPLB*). IPLB is obtained from the solver when solving the IP model. Although we do not know how close IPLB is to the optimal, but if the gap between IP600 and IPLB is small, then IP600 is close to optimal. Comparing IP600 and IPLB gives us an idea of the optimal solution and hence allows us to compare it with our heuristic. In experiments $1, 2, 3, 5$ and 6 we assume that each session consumes the same unit bandwidth. The bandwidth consumption varies in experiment 4 and each session can have multiple sources in experiments 5 and 6. Note that we assume each link has the same available bandwidth, we simply show the maximum bandwidth usage instead of minimum residual bandwidth in the figures.

4.6.1 Experiment 1: Number of Sessions

In experiment 1, our goal is to evaluate the impact of the number of simultaneous multicast sessions and the results are shown in Figure 4.4. The numbers of sessions



(a)



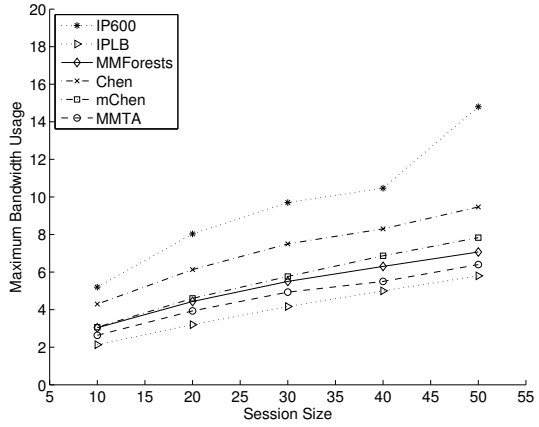
(b)

Figure 4.4: Experiment 1 Results: (a) Bandwidth Usage. (b) Execution Time.

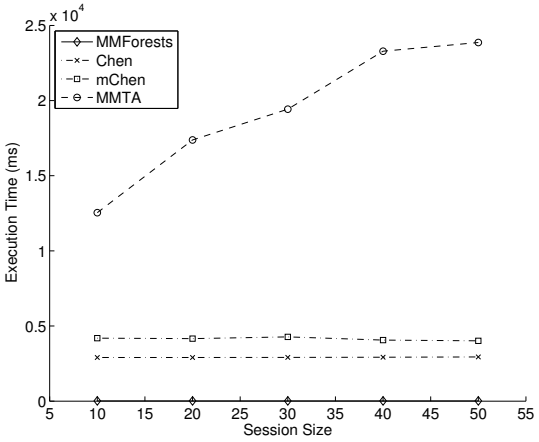
are $20, 40, 60, 80, 100$ while other parameters are fixed as shown in Table 4.2. As we can observe from Figure 4.4a, Integer Programming approach (IP600) can only find good solutions (in terms of bandwidth usage) for instances with fewer simultaneous sessions $(20, 40)$ due to the 600-second time limit. All other algorithms can efficiently find good solutions, and MMTA [39] gives best result overall and the solutions from Chen's algorithm (Chen) [12] use largest amount of bandwidth. The performance of MMForests is about half-way between MMTA [39] and Chen's algorithm [12] and approximately the same as modified Chen's algorithm (mChen) [39]. Note that the bandwidth usage grows linearly with the number of sessions.

Due to the $\mathcal{NP}$-hardness of the problem, the average execution times of the IP approach are close to 600 seconds (except for 20 sessions) and we omit it from Figure 4.4b. MMForests algorithm is the fastest among algorithms while MMTA is the slowest.

4.6.2 Experiment 2: Size of Sessions



(a)



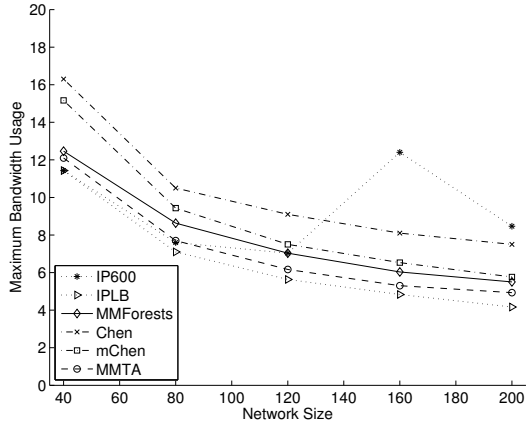
(b)

Figure 4.5: Experiment 2 Results: (a) Bandwidth Usage. (b) Execution Time.

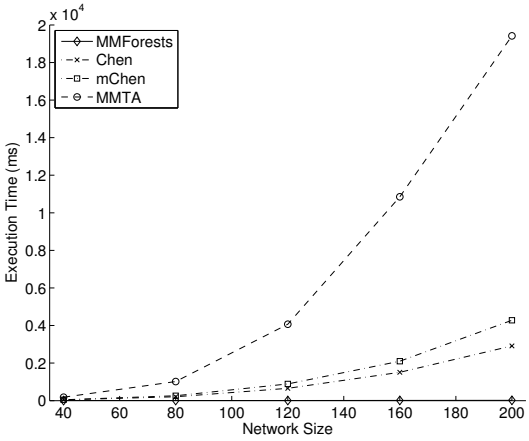
In this experiment, we varied the size (number of participating nodes) of each session while keeping other parameters the same. The results (Figure 4.5) are similar to experiment 1. The bandwidth usage grows as the size increases. MMTA [39]

takes significantly longer time but provides the best results among all algorithms. Interestingly, while the execution time of MMTA increases linearly with the size of the session, the execution times of MMForests algorithm, Chen's algorithm [12] and modified Chen's algorithm [39] are not affected by the size of the session. MMForests algorithm only takes a few milliseconds while providing reasonably good solutions in terms of bandwidth usage.

4.6.3 Experiment 3: Size of the Network



(a)



(b)

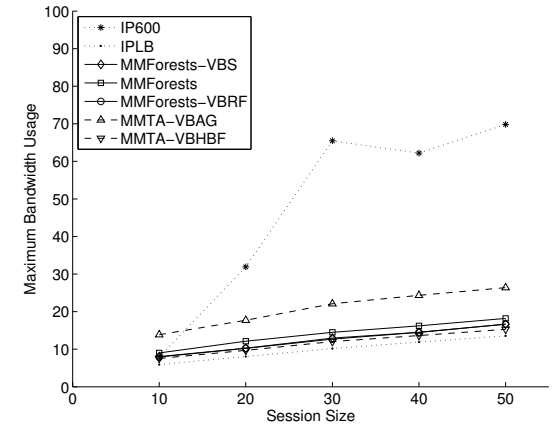
Figure 4.6: Experiment 3 Results: (a) Bandwidth Usage. (b) Execution Time.

We evaluated the effect of the network size in experiment 3. From Figure 4.6, we

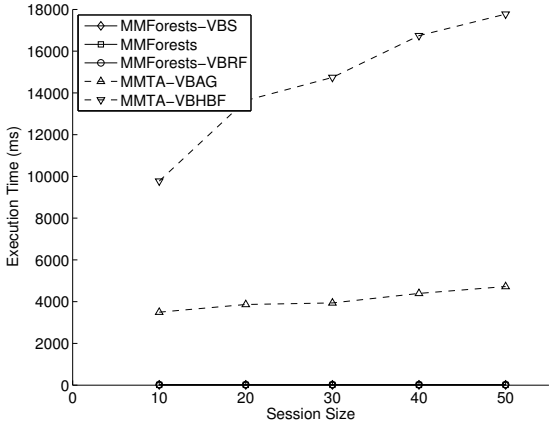
can observe the following facts. First, the bandwidth usage decreases as the network size becomes larger. This is due to the fact that there are more alternative paths to choose in a larger network. Second, the execution times of Chen's algorithm [12], modified Chen's algorithm [39], and MMTA [39] are polynomial in the network size. Among the algorithms, MMTA [39] grows much faster than the other two. The execution time of MMForests Algorithm is also polynomial in the network size but it grows at a more gentle rate in comparison to other algorithms.

4.6.4 Experiment 4: Sessions with Various Bandwidths

In experiment 4, we assume that each the sessions could have different bandwidth consumption. Two types of bandwidth models are tested. In Case 1, each session may consume 1, 2 or 3 unit(s) of bandwidth, which are randomly, equally distributed in the 30 sessions of the same input instance. In Case 2, the possible bandwidths are 1, 4 or 9 unit(s). Note that 1, 4 or 9 represent the ratios for NTSC, HD 720, and HD 1080 resolutions. The algorithms we compared include variants of MMForests algorithm and MMTA [39]. In the first variant MMForests-VBS, we first sort the various sessions in the descending order of their bandwidth requirement and then run the MMForests algorithm. The second variant MMForests-VBRF uses the same strategy as MMForests-VBS but performs *refinement* by using the tree reconstruction technique used in Chen's algorithm. Two variants of MMTA [39] are also considered. In MMTA-VBAG, a session with largest *alternative gain* (the one that can increase the most residual bandwidth) is chosen for reconstruction. On the other hand, the variant MMTA-VBHBF chooses the session that consumes the most bandwidth. The results are shown in Figure 4.7 and Figure 4.8 and they are similar for both Case 1 and Case 2. From Figure 4.8a and 4.7a, we can observe that the bandwidth usages in MMForests-VBS, MMForests-VBRF and MMTA-VBHBF are similar, while bandwidth usage of MMForests-VBUS is a little bit higher and that of MMTA-VBAG is the highest.



(a)



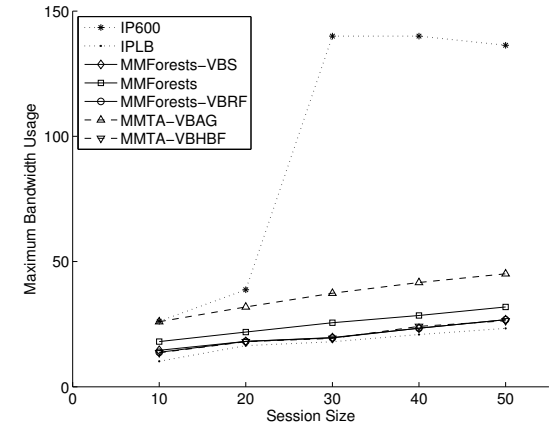
(b)

Figure 4.7: Experiment 4 Results – Case 1: (a) Bandwidth Usage. (b) Execution Time.

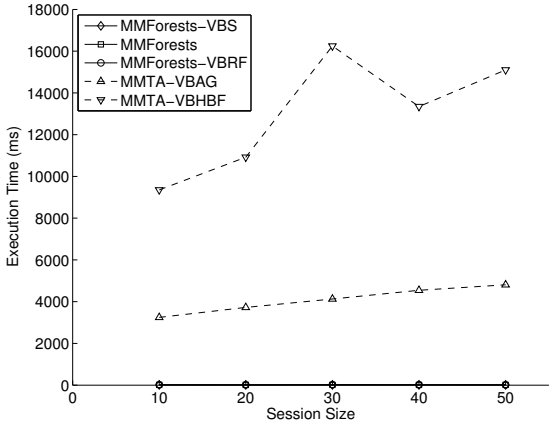
On the other hand, three variants of MMForests algorithm have similar execution time and two variants of MMTA take significantly longer time to solve the problem. MMTA-VBHBF has the longest execution time among all algorithms.

4.6.5 Experiment 5: Sessions with Multiple Sources

In experiment 5, we evaluated and compare the performance of MMForests algorithm and the IP solution for MMMRP with multiple sources. While controlling other factors, the number of sources for each session is varied from 1 to 5 and the results



(a)



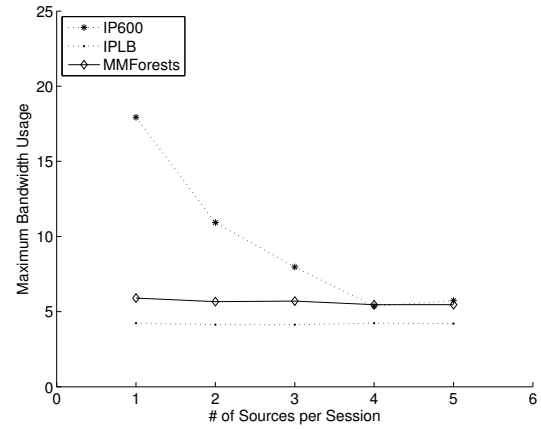
(b)

Figure 4.8: Experiment 4 Results – Case 2: (a) Bandwidth Usage. (b) Execution Time.

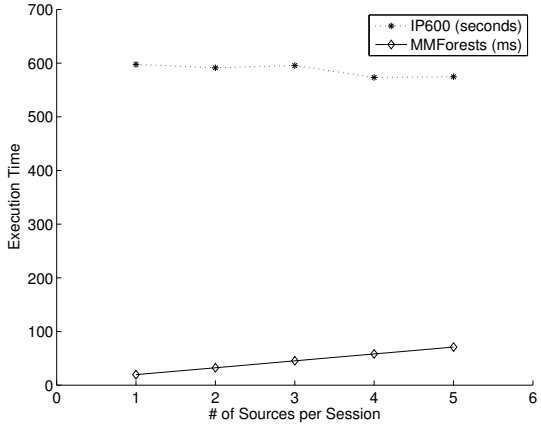
are shown in Figure 4.9. We can observe from Figure 4.9a, that the average bandwidth usages are not affected by the number of sources for each session. This could imply that there are some links that are critical in the network. The congestion can be improved by increasing the available bandwidth on those links.

4.6.6 Experiment 6: Scalability of MMForests

We tested the scalability of our MMForests algorithm in experiment 6 with large input instances in terms of number of nodes. The number and size of sessions are fixed at 40



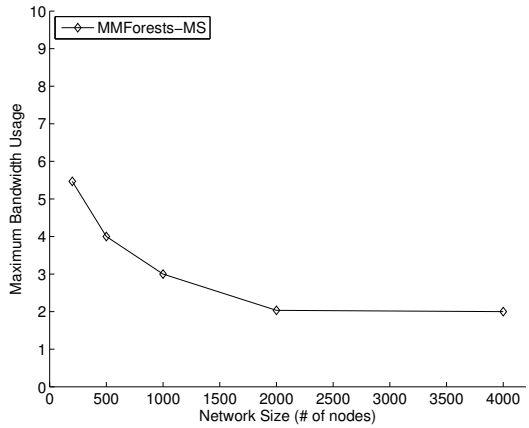
(a)



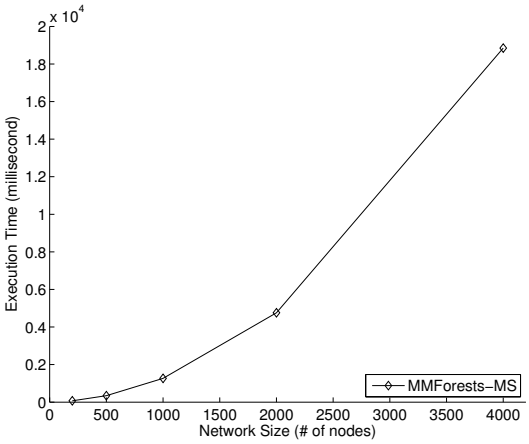
(b)

Figure 4.9: Experiment 5 Results: (a) Bandwidth Usage. (b) Execution Time.

and 30, respectively. There are 5 sources for each session. We manipulated the size of the network from 200 to 4000 and the results are shown in Figure 4.10. The execution time of MMForests algorithm is polynomial in the network size which is consistent with our complexity analysis. From Figure 4.10, we can observe MMForests algorithm can solve the problem with 2000 nodes (and 6000 edges) within 5 seconds. For 4000 nodes and 12000 edges, it takes about 18 seconds.



(a)



(b)

Figure 4.10: Experiment 6 Results: (a) Bandwidth Usage. (b) Execution Time.

4.7 Summary

In this chapter, we defined Multi-stream Multi-source Multicast Routing Problem (MMMRP), which is a generalized version of many multicast streaming problems and prove that MMMRP is $\mathcal{NP}$-hard. We compare our solution methods with other similar ones in the literatures and show that MMMRP and multicast tree packing problem have the same optimal value if the following conditions holds: (a) there is only one source for each data stream and (b) the tree size constraint is relaxed in multicast tree packing problem.

We proposed MMForests algorithm for solving MMMRP and evaluated it empirically. The performance and the results are summarized as follows.

- In experiments $1, 2, 3$, we first compare MMForests algorithm with Integer Programming approach and algorithms for multicast tree packing problems in the case of single-source MMMRP. The results show that MMForests algorithm performs well in terms of optimality and uses very short execution time for various input instances.

- In experiments 4 and 5, we consider the case of MMMRP with multiple sources which is not studied in the literatures. MMMForests algorithm performs pretty well in terms of optimality and execution time. The results also show MMMForests algorithm is capable of solving large input instances due to its low computational complexity.

- The initial solutions from the algorithms we compared in this chapter can be further improved in the tree reconstruction step. However, the improvement is not much when a similar tree reconstruction algorithm is included in MMForests algorithm.

- The results from experiment 6 suggest that the locations of the sources play an important role on bandwidth usages (or residual bandwidths) of the links.

MMForests is also an online algorithm, which makes it applicable for solving real world problems such as multimedia content distribution. A challenging research topic continuing this work is to use a similar model in multi-commodity integral flow problem in which the flows can be separated into smaller integral flows. This will help us utilize the network bandwidth more efficiently. Another topic of future study is the choices of the source locations. Although this is similar to facility location problem, the ability of duplicating data at the nodes will make it more interesting.

We also performed evaluation with a different input generation model in our preliminary research, which can be applied to different applications. Details can be found in Appendix B. The material presented in this chapter and Appendix B also appears in [15].

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this dissertation, we addressed several Quality-of-Service (QoS) related multimedia content delivery issues using multicast in packet-switching networks.

First, we considered two $\mathcal{NP}$-hard problems – Server and Clients Assignment Problem (SCAP, Chapter 2) and Client Assignment Problem (CAP, Chapter 3), wherein a two-layer architecture is assumed. The two-layer architecture consists of the Internet and a well-provisioned Service Overlay Network (SON) that is capable of multicasting. The participants (clients and the server for the client-server model, clients for the peer-to-peer model) of an application session are the Internet nodes with access to the nodes in the SON (through the Internet paths). The application then can take the advantage of the well-provisioned SON to reduce the end-to-end latencies and conserve the bandwidth (using multicast). Given a service overlay network with n nodes and p participants, we have developed algorithms to assign the participants to the service nodes in such way that the number of the service nodes involved is minimized while satisfying various criteria. These results are summarized below.

- Server and Clients Assignment Problem (Client-Server Model):

    a. Minimum Delay (SCAP-MD): In this problem we give a polynomial time algorithm to find an assignment of the server and clients to service nodes such that the maximum delay among clients is minimized.

b. Delay Bound and Minimum Cardinality (SCAP-DMC): In this problem we find an assignment of the server and clients to service nodes such that the number of service nodes used is minimized and the maximum delay among clients is bounded by a given value. This is an $\mathcal{NP}$-hard problem. Here we give an Integer Programming (IP) model solution that uses exponential time, and a heuristic algorithm with complexity $O(n^3p)$ to find the assignment. A somewhat related problem was addressed in [40] that attempted to minimize the number of contact servers. The performance of our algorithm compares favorably with the optimal solution found using the IP formulation.

c. Delay bound and Minimum Delay Variation (SCAP-DV): In this problem we find an assignment of the server and clients to service nodes such that the maximum delay is bounded by a given value with the condition that the variation in the delays experienced by different clients is minimized. For this problem we give an algorithm that finds the assignment in $O(n^3p^2)$ time, where n is the number of service nodes and p is the number of clients.

d. Delay bound, Delay Variation bound, and Minimum Cardinality (SCAP-DVMC): In this problem we want to find an assignment requiring minimum number of service nodes with a given bound on delay and a given bound on delay variation. This problem is shown to be $\mathcal{NP}$-hard. We give an IP for this problem that takes exponential time to find the solution. To further reduce time for finding the exact solution we give another IP formulation in which the search space is sub-divided into overlapping search sub-spaces. Though there are multiple search spaces in this case, this formulation takes much less time than the first one because the size of the individual search spaces is small. A heuristic algorithm with a time complexity of $O(n^4p^2)$ is also given. The performance of this algorithm compares favorably with

the optimal solution found using the IP formulations.

- Client Assignment Problem (Peer-to-Peer Model):

  a. Delay Bound and Minimum Cardinality (CAP-DMC): In this problem we find an assignment of clients to service nodes such that the number of service nodes used is minimized while keeping the maximum delay between any pair of clients bounded by a given value. This is an $\mathcal{NP}$-hard problem. Here we give an Integer Programming (IP) model solution that uses exponential time, and a heuristic algorithm with complexity $O(n^2p)$ to find the assignment. A similar problem was addressed in [40] that attempted to minimize the number of contact servers using the *ZIZO* algorithm. We have modified ZIZO to also take into account the intermediate nodes. The performance of our algorithm compares favorably with the optimal solution found using both the IP formulation and ZIZO.

  b. Delay Bound and Delay Variation Reduction (CAP-DVR): In this problem we find an assignment of the clients to service nodes such that the maximum delay is bounded by a given value with the condition that the variation in the delays experienced by different pairs of clients is minimized. For this problem, we first solve it by using the algorithm we developed for CAP-DMC, then reduce the delay variation by reassigning the clients. We give an algorithm with pseudo polynomial time complexity of $O(n^2p)$ that can effectively reduce the delay variation by 30% – 40% of the delay bound.

In the second part of this dissertation (Chapter 4), we attempt to address the Multi-stream Multi-source Multicast Routing Problem (MMMRP). Given a network with bandwidth-constrained links and multiple multicast sessions each with one or more sources and multiple destinations. The goal of MMMRP is to find a way to route the data from the source(s) to the destinations using multicast such that the

overall residual bandwidth is maximized. We have proved that MMMRP is a generalization of many multicast streaming problems. We have also proved it is $\mathcal{NP}$-hard. We developed a heuristic algorithm called *MMForests* which is based on Dijkstra's algorithm that runs in $O(rmn)$ time and can solve MMMRP both online and offline. A nontrivial Integer Programming formulation (Model MMMRP) for this problem is also developed for the purpose of performance evaluation. Experimental results show that our algorithm has the performance similar to best known algorithm in terms of the overall residual bandwidth. Our algorithm also takes less execution time and hence is more scalable, less than 20 seconds for a network with 4000 nodes for example.

5.2 Future Work

In this dissertation, we have considered (a) Server and Clients Assignment Problem (SCAP)/Client Assignment Problem(CAP) with various real-time requirements and (b) Multi-stream Multi-source Multicast Routing Problem (MMMRP) with bandwidth constraints independently. A direction to extend the work in this dissertation is to consider SCAP/CAP with multiple concurrent sessions in which the overall bandwidth usage in the service overlay network needs to be minimized. In practice, it would be a good idea to use *tunneling* in such two-layer architectures so that the application sessions are transparent to the end users. Protocols need to be developed to handle this and the routing since source routing is not efficient due to the fact that the routing information is embedded within the packets.

Bibliography

[1] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe, "Optimal design of service overlay networks with economic and performance constraints," *Int. J. Commun. Syst.*, vol. 23, no. 3, pp. 369 – 389, Mar. 2010.

[2] S. M. Banik, S. Radhakrishnan, and C. N. Sekharan, "Multicast routing with delay and delay variation constraints for collaborative applications on overlay networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 3, pp. 421 – 431, 2007.

[3] B. Bhattacharjee, "Project NICE at the University of Maryland," [Online; accessed 27-August-2012]. [Online]. Available: http://www.cs.umd.edu/projects/nice/

[4] S. Birrer, D. Lu, F. E. Bustamante, Y. Qiao, and P. Dinda, "Fatnemo: Building a resilient multi-source multicast fat-tree," in *In Proc. of IWCW*, 2004, pp. 182–196.

[5] BitTorrent, Inc, "BitTorrent - Delivering the World's Content," [Online; accessed 20-September-2012]. [Online]. Available: http://www.bittorrent.com

[6] E. Brosh, A. Levin, and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application-layer multicast trees," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 473–484, Apr. 2007.

[7] K. Calvert, M. Doar, and E. Zegura, "Modeling internet topology," *Communications Magazine, IEEE*, vol. 35, no. 6, pp. 160 –163, Jun. 1997.

[8] A. Capone, J. Elias, and F. Martignon, "Routing and resource optimization in service overlay networks," *Comput. Netw.*, vol. 53, no. 2, pp. 180 – 190, Feb. 2009.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: high-bandwidth multicast in cooperative environments," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 298 – 313, Oct. 2003.

[10] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "An empirical evaluation of tcp performance in online games," in *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2006, p. 5.

[11] K.-T. Chen, P. Huang, and C.-L. Lei, "Effect of network quality on player departure behavior in online games," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 593–606, 2009.

[12] S. Chen, O. Günlük, and B. Yener, "The multicast packing problem," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 311 – 318, Jun. 2000.

[13] Y.-R. Chen, S. Radhakrishnan, S. Dhall, and S. Karabuk, "Server selection with delay constraints for online games," in *IEEE Globecom 2010 Workshop on Multimedia Communications and Services (MCS 2010)*, Miami, Florida, USA, Dec. 2010.

[14] ——, "On the game server network selection with delay and delay variation constraints," in *The Third International Conference on COMmunication Systems and NETworkS (COMSNETS 2011)*, Bangalore, India, Jan. 2011.

[15] ——, "On multi-stream multi-source multicast routing," in *IEEE Globecom 2012 - Communications Software, Services and Multimedia Symposium (GC12 CSSM)*, Anaheim, CA, USA, Dec. 2012.

[16] Y.-H. Chu, S. Rao, S. Seshan, and H. Zhang, "A case for end system multicast," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 8, pp. 1456 – 1471, Oct. 2002.

[17] V. Chvatal, *Linear Programming*, 1st ed. New York: W. H. Freeman, 1983.

[18] Cisco Systems, Inc., "Quality of Service Design Overview - QoS requirements of VoIP," [Online; accessed 2-September-2012]. [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=357102

[19] M. Claypool and K. Claypool, "On latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[20] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," in *University of Michigan*, 2001, p. 01.

[21] E. W. Dijkstra, "A note on two problems in connexion with graphs." *Numerische Mathematik*, vol. 1, pp. 269 – 271, 1959.

[22] Z. Duan, Z.-L. Zhang, and Y. Hou, "Service overlay networks: Slas, qos, and bandwidth provisioning," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 6, pp. 870 – 883, Dec. 2003.

[23] A. El-Sayed, V. Roca, and L. Mathy, "A survey of proposals for an alternative group communication service," *Network, IEEE*, vol. 17, no. 1, pp. 46 – 51, Jan./Feb. 2003.

[24] H. Eriksson, "Mbone: the multicast backbone," *Commun. ACM*, vol. 37, pp. 54 – 60, Aug. 1994.

[25] S. Even, A. Itai, and A. Shamir, "On the Complexity of Timetable and Multicommodity Flow Problems," *SIAM Journal on Computing*, vol. 5, no. 4, pp. 691 – 703, 1976.

[26] S. Fahmy and M. Kwon, "Characterizing overlay multicast networks and their costs," *IEEE/ACM Trans. Netw.*, vol. 15, pp. 373 – 386, Apr. 2007.

[27] S. Ferretti, M. Roccetti, and C. E. Palazzi, "An optimistic obsolescence-based approach to event synchronization for massively multiplayer online games," in *INTERNATIONAL JOURNAL OF COMPUTERS AND APPLICATIONS*, vol. 29, no. 1, 2007, pp. 33–43.

[28] G. Figueiredo, N. da Fonseca, and J. Monteiro, "A minimum interference routing algorithm," in *Communications, 2004 IEEE International Conference on*, vol. 4, Jun. 2004.

[29] A. Ganesh, A.-M. Kermarrec, and L. Massouli, "Peer-to-peer lightweight membership service for large-scale group communication," in *Networked Group Communication*, ser. Lecture Notes in Computer Science, J. Crowcroft and M. Hofmann, Eds. Springer Berlin / Heidelberg, 2001, vol. 2233, pp. 44–55.

[30] F. Glinka, A. Ploss, S. Gorlatch, and J. Müller-Iden, "High-level development of multiserver online games," *Int. J. Comput. Games Technol.*, vol. 2008, pp. 1–16, 2008.

[31] Gurobi Optimization, Inc, "Gurobi Optimizer," [Online; accessed 27-August-2012]. [Online]. Available: http://www.gurobi.com

[32] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: a scalable peer-to-peer network for virtual environments," *Network, IEEE*, vol. 20, no. 4, pp. 22–31, Aug. 2006.

[33] T. C. Hu, "The maximum capacity route problem," *Operations Research*, vol. 9, no. 6, pp. pp. 898–900, 1961.

[34] IEEE Standards Association, "IEEE 802 standards," [Online; accessed 27-August-2012]. [Online]. Available: http://standards.ieee.org/about/get/

[35] Internet Engineering Task Force, "Internet Engineering Task Force," [Online; accessed 27-August-2012]. [Online]. Available: http://www.ietf.org

[36] K. Kar, M. Kodialam, and T. V. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with mpls traffic engineering applications," *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 12, pp. 2566 – 2579, Dec. 2000.

[37] R. M. Karp, "On the computational complexity of combinatorial problems," *Networks*, vol. 5, pp. 45 – 68, 1975.

[38] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for steiner trees," *Acta Informatica*, vol. 15, pp. 141–145, 1981.

[39] C. Y. Lee and H. K. Cho, "Multiple multicast tree allocation in ip network," *Computer Operations Research*, vol. 31, no. 7, pp. 1115 – 1133, Jun. 2004.

[40] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2004, pp. 152 – 157.

[41] Z. Li and P. Mohapatra, "On investigating overlay service topologies," *Comput. Netw.*, vol. 51, no. 1, pp. 54–68, Jan. 2007.

[42] C. Lumezanu, R. Baden, N. Spring, and B. Bhattacharjee, "Triangle inequality and routing policy violations in the internet," in *Proceedings of the 10th International Conference on Passive and Active Network Measurement*, ser. PAM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 45 – 54.

[43] C. Ly, C.-H. Hsu, and M. Hefeeda, "Irs: A detour routing system to improve quality of online games," *Multimedia, IEEE Transactions on*, vol. 13, no. 4, pp. 733 – 747, Aug. 2011.

[44] A. Medina, A. Lakhina, I. Matta, and J. Byers, "Boston university Representative Internet Topology gEnerator (BRITE)," [Online; accessed 27-August-2012]. [Online]. Available: http://www.cs.bu.edu/brite/

[45] Napster, Inc, "Napster," [Online; accessed 20-September-2012]. [Online]. Available: http://www.napster.com

[46] G. Pandurangan, P. Raghavan, and E. Upfal, "Building low-diameter peer-to-peer networks," *Selected Areas in Communications, IEEE Journal on*, vol. 21, no. 6, pp. 995 – 1002, Aug. 2003.

[47] L. L. Peterson and B. S. Davie, *Computer Networks: A Systems Approach*, 4th ed. San Francisco: Morgan Kaufmann, 2007.

[48] PlanetLab. https://www.planet-lab.org:443/.

[49] K. N. Ravindran, A. Sabbir, D. Loguinov, and G. S. Bloom, "Cost optimal multicast trees for multi-source data flows," in *INFOCOM*, 2001, pp. 966–975.

[50] G. N. Rouskas and I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 346 – 356, 1995.

[51] G. Rouskas and I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints," *Selected Areas in Communications, IEEE Journal on*, vol. 15, no. 3, pp. 346 – 356, Apr. 1997.

[52] Sandvine Incorporated, "Global Internet Phenomena Spotlight, North America, Fixed Access, Fall 2011," [Online; accessed 28-August-2012]. [Online]. Available: http://www.sandvine.com/downloads/documents/10-26-2011_phenomena/Sandvine%20Global%20Internet%20Phenomena%20Spotlight%20-%20North%20America%20-%20Fixed%20Access%20-%20Fall%202011.pdf

[53] A. S. Tanenbaum, *Computer Networks*, 4th ed. Upper Saddle River, New Jersey: Pearson Education, Inc., 2003.

[54] The Internet Engineering Task Force, "RFC 3376: Internet Group Management Protocol, Version 3," [Online; accessed 27-August-2012]. [Online]. Available: http://tools.ietf.org/html/rfc3376

[55] ——, "RFC 3810: Multicast Listener Discovery Version 2 (MLDv2) for IPv6," [Online; accessed 27-August-2012]. [Online]. Available: http://tools.ietf.org/html/rfc3810

[56] H. T. Tran and F. Bessler, "A framework toward optimal operation of service overlay network assuring inter-domain qos," in *in: Proceedings of the International Network Optimization Conference, INOC*, 2005, pp. 1–67.

[57] S. Vieira and J. Liebeherr, "Topology design for service overlay networks with bandwidth guarantees," in *Quality of Service, 2004. IWQOS 2004. Twelfth IEEE International Workshop on*, Jun. 2004, pp. 211 – 220.

[58] K.-H. Vik, P. Halvorsen, and C. Griwodz, "Evaluating steiner tree heuristics and diameter variations for application layer multicast," *Elsevier Computer Networks*, vol. 52, no. 15, pp. 2872 – 2893, 2008, special issue on Complex Computer and Communication Networks, Elsevier.

[59] C.-F. Wang, C.-T. Liang, and R.-H. Jan, "Heuristic algorithms for packing of multiple-group multicasting," *Computer Operations Research*, vol. 29, no. 7, pp. 905 – 924, 2002.

[60] Wikipedia, the free encyclopedia, "eDonkey network," [Online; accessed 20-September-2012]. [Online]. Available: http://en.wikipedia.org/wiki/EDonkey_network

[61] ——, "Gnutella," [Online; accessed 20-September-2012]. [Online]. Available: http://en.wikipedia.org/wiki/Gnutella

Appendix A

Server Selection Problem

A.1 Introduction

We addressed Server Selection Problem for both Client-Server and Peer-to-Peer architectures (SSP-CS, SSP-P2P) in [13, 14]. SSP-CS is similar to Server and Client Assignment Problem (SCAP) in Chapter 2. Two problems only differ from each other in the way the server node is modeled. In SCAP, the server node is an end host outside the service overlay network and our goal is to find the contact service node for it. On the other hand, we assume the service nodes are also capable of acting as a game server in SSP-CS and the server node is selected from one of the service nodes. Although the models of two problems are slightly different, the approaches for solving them are similar. Hence we present the algorithms and integer programming models for the variants of SSP-CS in this appendix. Note that SSP-P2P is identical to Client Assignment Problem (CAP) discussed in Chapter 3 and hence is not included in this appendix.

A.2 System Model and Notations

Although we use different terminology for describing SSP-CS in [13, 14], here we will use the notations from Chapter 2 for consistency. We use $G = (V, E)$ for the well-provisioned service overlay network, C for the set of clients involved in the same game session and the delay function $d()$ as defined in Chapter 2. We use $D = (C, G)$ for the game session. The goal of SSP-CS is to find an assignment A that maps

each client c_j in C to a service node a_j in V and a server node r in V such that some requirements are satisfied. Similar to SCAP, the client-server architecture used in SSP-CS is represented as a tree $T_D(A, r)$ with root r being the server. We use $|T_D(A, r)|$ to denote the cardinality of $T_D(A, r)$, which is the number of service nodes in the tree. We define the delay (latency) of a client under the assignment A as $\Lambda_D(A, c_j) = d(c_j, a_j) + d(a_j, r)$. The delay of an application session D under the assignment A as

$$\begin{aligned}\Lambda_D(A) &= \max \Lambda_D(A, c_j) \\ &= \max[d(c_j, a_j) + d(a_j, r)] \quad \forall \quad c_j \in C\end{aligned}$$

Similarly, we define the delay variation of an application session D under the assignment A is

$$\Delta_D(A) = \max |\Lambda_D(A, c_j) - \Lambda_D(A, c_{j'})| \quad \forall \quad c_j, c_{j'} \in C$$

A.3 Server Selection Problem for Client-Server Architecture (SSP-CS)

We consider two variants of SSP-CS – SSP-CS with Delay Constraint (SPD-CS) and SSP-CS with Delay Constraints and Delay Variation Reduction (SPDVR-CS).

A.3.1 Server Selection Problem for Client-Server Architecture with Delay Constraint (SPD-CS)

The goal of SPD-CS is to find a central server r along with an assignment A that maps each client to its contact node such that the following two requirements are satisfied:

Model SPD-CS(r)

$$\text{minimize} \quad \sum_{v_i \in V} X_i \tag{A.1}$$

subject to:

$$\sum_{v_i \in V} Y_{ji} = 1 \qquad \forall \ c_j \in C \tag{A.2}$$

$$\sum_{c_j \in C} Y_{ji} \leq m X_i \qquad \forall \ v_i \in S \tag{A.3}$$

$$\sum_{v_{i'} \in S, v_{i'} \neq v_i} X_{i'} a_{i'i} \leq n X_i \qquad \forall \ v_i \in S \tag{A.4}$$

Figure A.1: Model SPD-CS(r)

a. The maximum delay is less than μ ($\Lambda_D(A) \leq \mu$).

b. The number of service nodes in the tree $T_D(A, r)$ is minimum ($|T_D(A, r)|$).

Here we provide an integer programming formulation and a heuristic algorithm for solving SPD-CS.

Integer Programming Formulation

We use the same strategy as in Chapter 2 to solve the SPD-CS problem. We iterate through all service nodes as the server r, and use Model SPD-CS(r) to find the optimal solution when using r. The one with minimum number of service nodes is optimum.

Model SPD-CS(r) is similar to SCAP-DMC but simpler. We define a_i as a column vector of dimension n that represents the path from v_i to r. An entry $a_{ii'}$ is equal to 1 if service node $v_{i'}$ is on the path and 0 otherwise. We define binary decision variables Y_{ji} that take the value 1 if client c_j is assigned to service node v_i and 0 otherwise. Similarly, binary decision variables X_i are defined and take the value 1 if service node v_i is selected in the solution, 0 otherwise. The delay constraints are imposed in the preprocessing stage. The formulation is presented in Figure A.1.

The objective function (A.1) measures the total number of service nodes that we

have to turn on. Constraints (A.2) ensure that each client is assigned to exactly one contact node. The remaining two constraints make sure that a service node is selected if it is a contact node (A.3), or it is on the shortest path from another contact node to the server r (A.4).

The total delay associated with an assignment can easily be calculated as described previously. Therefore in a preprocessing stage we identify and set to 0 the Y_{ji} variables that violate the delay constraint. Hence we do not have any explicit delay constraints.

Algorithm SPD-CS-H

Algorithm SPDCS-H is a heuristic algorithm that uses a greedy strategy while keeping the number of servers selected to a minimal. It iterates through all possible service nodes as the server r and uses Procedure SPD-CS-H(in Chapter 3) to find an assignment. The assignment with the smallest cardinality is returned as the solution. We omit Algorithm SPDCS-H here due to its similarity to other algorithms. The complexity of SPDCS-H is $O(n^2p)$ for $n < p$.

A.3.2 Server Selection Problem for Client-Server Architecture with Delay Variation Reduction (SPDVR-CS)

Given an assignment A satisfying the delay constraint μ, the goal of SPDVR-CS is to find a new assignment A' using the same set of service nodes in A such that the delay is less than μ and the delay variation is minimal. Note that since $A' \subseteq A$, the number of service nodes used in the new network is no more than the original solution. In fact, the approaches for solving SCAP-DVMC including the IP models and Algorithm *SCAP-DVMC-H* can be used for solving this problem by setting a dummy *server* whose latencies to all the service nodes are 0. Hence we omit these approaches in this section.

A.4 Performance Evaluation

Tiers [7] is used to generate Internet-like graphs for evaluations of our algorithms. Graphs of different sizes are generated with 30 instances for each configuration and the results are averaged for each configuraton. The latencies of the links on the service overlay network are reduced to 70% to represent the well-provisioned network. For each instance, the minimum possible delay Γ for a given session is found. Then Γ is multiplied by a factor $f \in \{1.0, 1.1, 1.2\}$ to represent the different real-time requirement for different game types.

The algorithms are implemented using C/C++. *Gurobi Optimizer 4.5* C++ library [31] is used to implement the IP models. The evaluation is done on a 16-core Intel Xeon (*E5520* at 2.27 GHz) machine with 12 GB of RAM installed running *Ubuntu* (*2.6.28-11-generic* kernel) . Multithreading (up to 16 threads) is used when possible for parallel barrier in Gurobi solver.

A.4.1 SPD-CS

- *Optimality*: We first compare the size of the service overlay network (number of servers selected) in the solutions, Figure A.2a shows our heuristic is able to find the solutions where the number of servers selected is close to optimal given by integer programming approach (*Model SPD-CS(r)*). We also have an interesting finding that when the number of participating clients is greater than a particular value (150 in this case), the number of servers selected converges. This probably implies that we only need a certain number of server nodes to satisfy the realtime requirement by choosing the locations of the server nodes properly. Note that in *Nearest*, clients are assigned to the servers (service nodes) closest to them.

- *Time Complexity*: Results show the execution time of the algorithms increases

with the size of input (Figure A.2b through A.2d). The number of clients is fixed at 400 in Figure A.2b while the number of servers is fixed at 50 for Figure A.2c and A.2d. We can observe that the execution time increases exponentially with the number of servers and it increases linearly with the number of clients. We also attempted to increase the number of the clients (number of available servers is fixed at 50) to check the limitation of our heuristic and integer programming formulation in terms of execution time. The result shows that IP approach takes more than 5 minutes to solve the network of 10000 clients while our heuristic only takes less than 1 second (Figure A.2c). It is easy to see that our heuristic runs in linear time (Figure A.2b). The quality of the solutions (in terms of number of server selected) is similar to previous results. We also found that because of the properties of the integer programming model (*Model SPD-CS(r)*), the solver only utilizes 1 core when solving the problems.

A.4.2 SPDVR-CS

We compare the improvement on delay variation after the modified Algorithm SCAP-DVMC-H is applied. Modified Algorithm SCAP-DVMC-H significantly reduces the delay variation of the assignments. The result of the networks with 25 service nodes, $f = 1.3$ is shown in Figure A.3a while other configurations give similar results. We also compared the change on latencies after applying the algorithm. Interestingly, we found that the algorithm does not only reduce the delay variation, but it also reduces the latencies by a small amount (Fig A.3b). Similar results are observed for other input data sets. The material presented in this appendix appears in [13, 14].

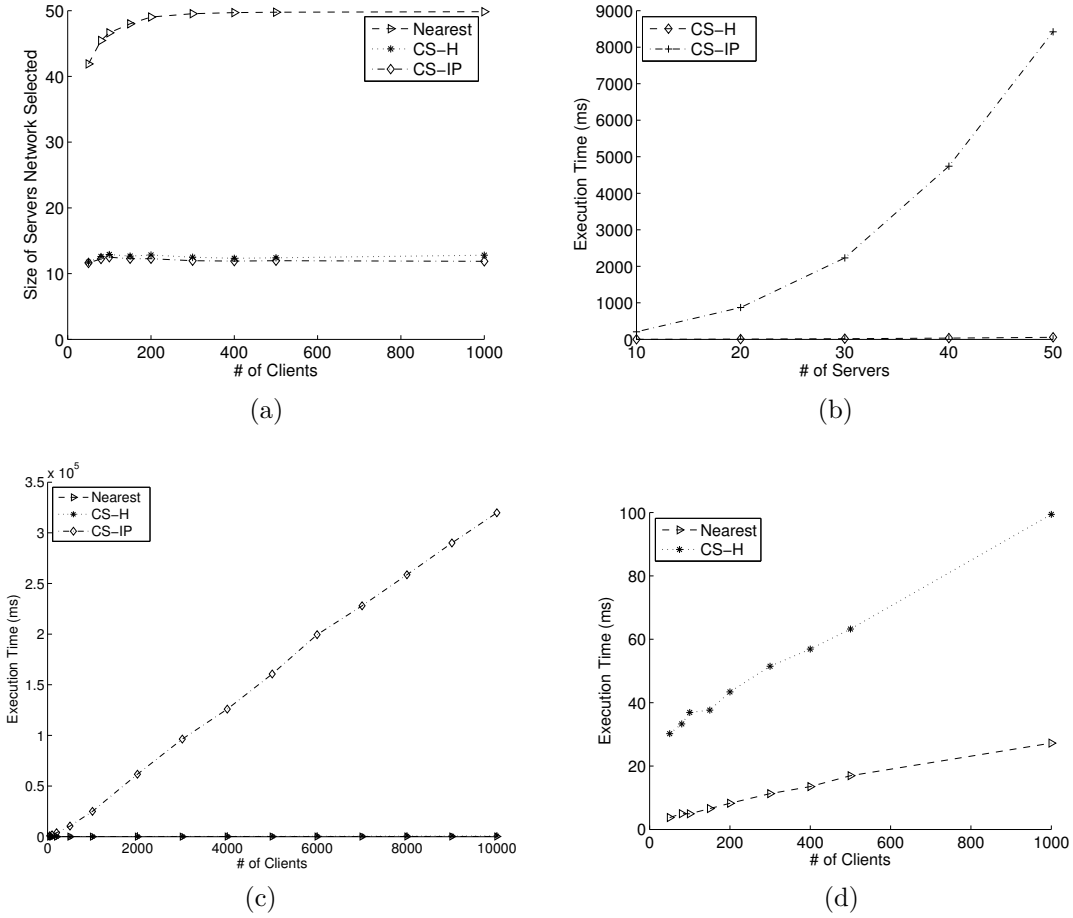


Figure A.2: Results of SPD-CS: (a) Optimality (b) Execution time for 400 clients with different sizes of server network (c) Execution time for different number of clients (50 servers) (d) The growth of execution time for the heuristic.

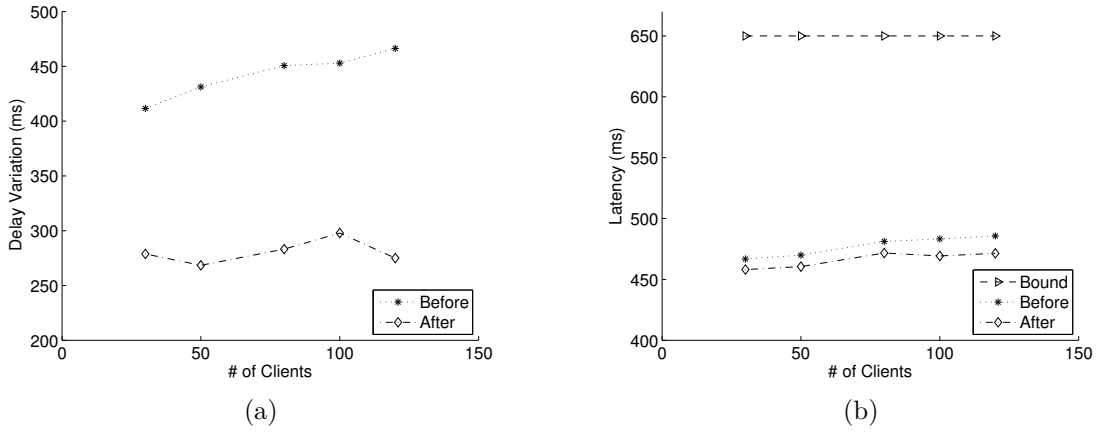


Figure A.3: Results for SPD-DVR-CS: (a) Delay variations before and after applying modified Algorithm SCAP-DVMC-H (25 servers, $f = 1.3$). (b) Delay before and after applying modified Algorithm SCAP-DVMC-H (25 servers, $f = 1.3$).

Appendix B

Preliminary Results of MMMRP

B.1 Network Generation and Experimental Setup

In the preliminary research, we assume each node collects data from several radars. Under this assumption each node can provide several data streams and there are groups of nodes that demand these data streams. We generate 30 random instances for each different configuration using the parameters in TABLE B.1 with the average node degree of 3. We assume a homogenous network, in which each source node supplies the same number of data streams, each destination node requests the same number of data streams and each data stream is supplied by the same number of source nodes. For simplicity, we assume that a node can be either a source node or a destination node but not both.

Table B.1: Network Generation Parameters in Preliminary MMMRP Exps

Name	Description
N	Number of nodes
N_S	Number of source nodes
N_W^S	Number of data streams supplied by a source node
N_D	Number of destination nodes
N_W^D	Number of data streams requested by a destination node
N_W	Number of data streams
N_S^W	Number of source nodes that supply a data stream ($N_S^W = N_S \cdot N_W^S / N$)

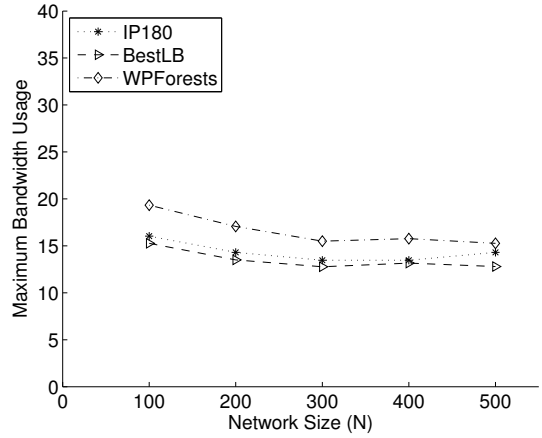
We evaluate the impact of different parameters and compare the performance of *Model MMMRP* with our heuristic *MMForests*. MMForests and Model MMMRP are

implemented in C++ with Gurobi Optimizer [31] (version 4.6) C++ Library for the IP model. The environment for running the experiments are the same as in Chapter 4 – an Intel Xeon (E5520 at 2.27 GHz) machine with 12 GB of RAM running Linux kernel 3.0.0-16. Multithreading (up to 16 threads) is used when possible for parallel barrier in Gurobi optimizer, *MMForests* algorithm only uses a single thread. Note that finding an optimal solution using the IP model may take a long period of time (hours to days) for large instances. Hence we limit the running time of the solver to 180 seconds and obtain *best known solution* (*IP180*) and *best available lower bound on solution* (*BestLB*). BestLB is obtained from the solver during solving the IP model. If the gap between IP180 and BestLB is small, IP180 is close to optimal. This gives us an idea about the optimal solution and allows us to compare it with our heuristic. We compare (a) *maximum bandwidth usage* (in terms of the number of data streams) among the links and (b) *execution time* for both approaches in the experiments. The result figures show the average of 30 instances of each network configuration. Note that in the results of execution time, the unit for IP180 is in *seconds* and MMForests is in *milliseconds*.

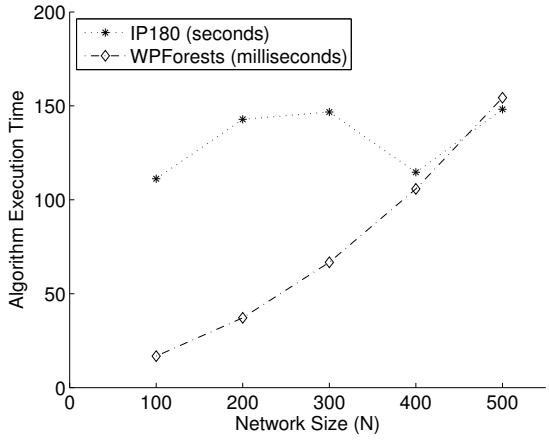
B.2 Experimental Results

B.2.1 Size of the Network (N)

For network size (N) from 100 to 500 and keep other parameters the same ($N_S = 10, N_D = 40, N_W^S = 8, N_W^D = 20, N_S^W = 2$), The results (Figure B.1) show that *IP180* and *BestLB* are very close, which implies IP180 is close to optimal for the network size up to 500 nodes. The maximum bandwidth used given by from *MMForest* is a little more than that given by IP180 but with very short running time ($\leq 150ms$) comparing to more than 100 seconds for IP180. One thing that surprised us is that BestLB does not decrease much as the network size increases, this may imply the



(a)



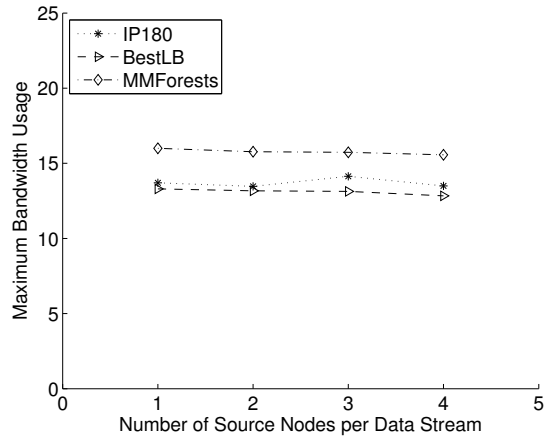
(b)

Figure B.1: Impact of the number of nodes in the network ($N_S = 10, N_D = 40, N_W = 40, N_W^S = 8, N_W^D = 20, N_S^W = 2$). (a) Bandwidth usage. (b) Execution time (Notice that IP180 is in seconds and WPForests is in milliseconds).

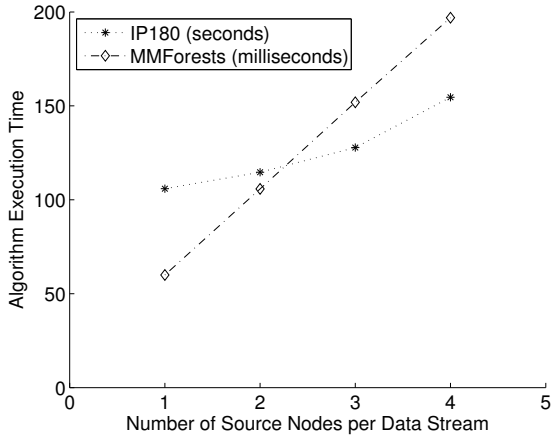
bottleneck in the network is implicitly affected by the average degree of the nodes.

B.2.2 Number of Source Nodes for a Data Stream (N_S^W)

We varied the number of sources per data stream (N_S^W) while keeping other parameters the same ($N = 400, N_S = 10, N_D = 40, N_W = 40, N_W^D = 20$). Note that N_W^S is also affected by N_S^W. N_W^S are set to $4, 8, 12, 16$ with resulting N_S^W as $1, 2, 3, 4$ respectively. The results (Figure B.2) shows the maximum bandwidth usage does not



(a)



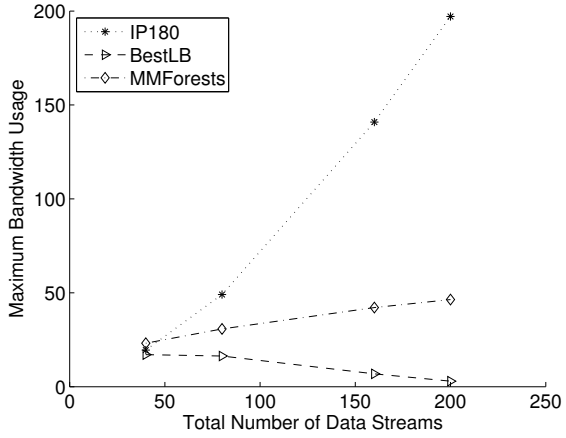
(b)

Figure B.2: Impact of the number of sources per data stream. (a) Maximum bandwidth usage among links. (b) Execution time.

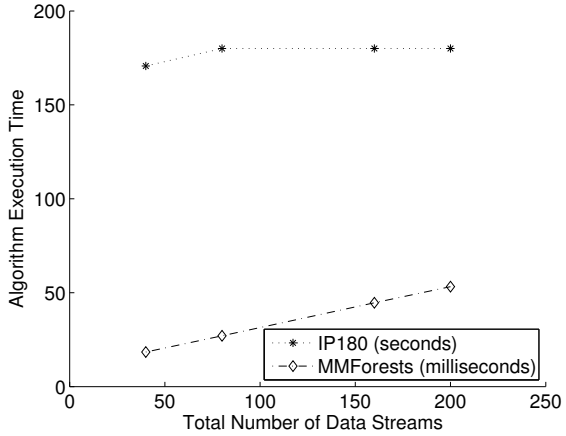
decrease much as N_S^W increases. Our heuristic can quickly find solutions that are close to optimal. We also observe that the execution time for both approaches increases as N_S^W increases.

B.2.3 Number of Data Streams (N_W)

We experiment with the effect of total number of data streams ($N_W = 40, 80, 160, 200$) keeping other parameters the same ($N = 100, N_S = 10, N_D = 80, N_W = 40, N_W^S = 8, N_W^D = 20$). We find that the solver starts failing to find a good feasible solution



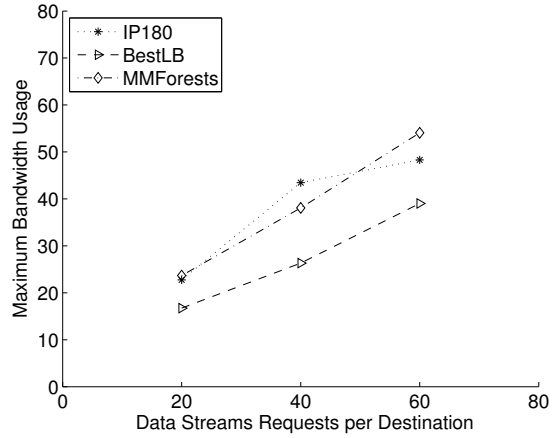
(a)



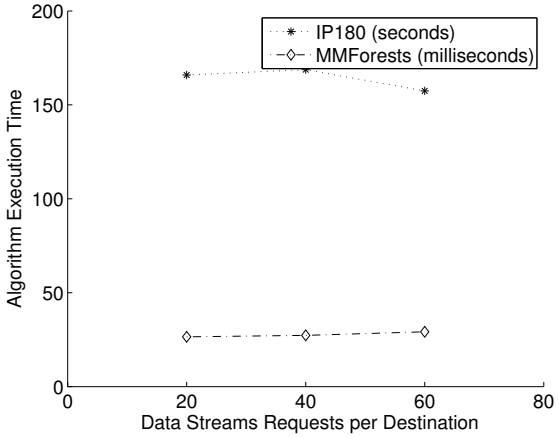
(b)

Figure B.3: Impact of the total number of data streams. (a) Maximum bandwidth usage among links. (b) Execution time.

(and a good BestLB) as the number of data streams increases. As the gaps between the best solution found by IP180 and the BestLB value become large, especially for $N_W = 160, 200$. This information is less meaningful to us. From Figure B.3, we can see that MMForests performs well when $N_W = 40$. We also believe that the optimal value also increases when N_W increases. If our assumption holds, we have confidence that MMForests finds good solutions because the slope of MMForests in Figure B.3a is small. Note that in Figure B.3a, BestLB decreases because the solver could not to improve it much (from 0) when problem sizes get larger.



(a)

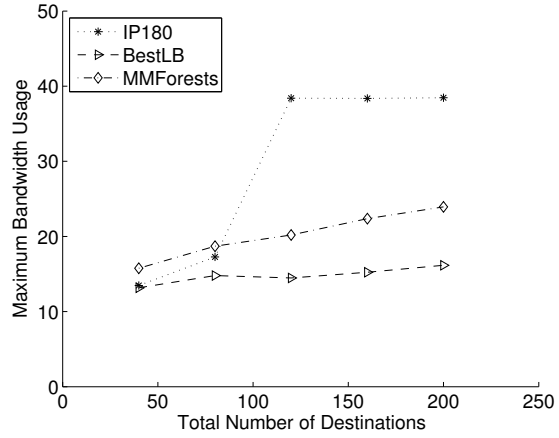


(b)

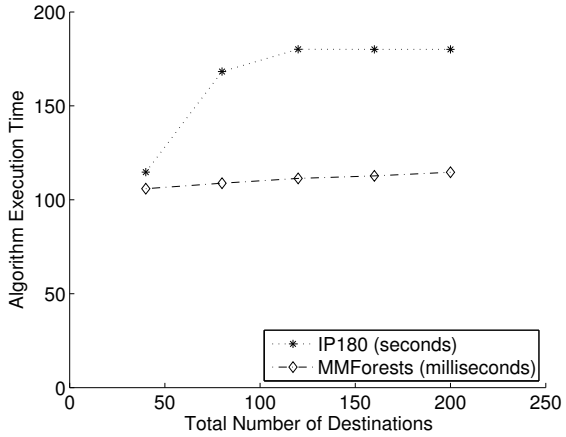
Figure B.4: Impact of the number of data streams requested per destination. (a) Maximum bandwidth usage among links. (b) Execution time.

B.2.4 Number of Data Streams Requested per Destination

Figure B.4 shows the results of varying the number of data streams requested by each of the destinations ($N_W^D = 20, 40, 60$). Other parameters are $N = 100, N_S = 10, N_D = 40, N_W = 80, N_W^S = 16, N_S^W = 2$. Gurobi uses more than 150 seconds in average to find good solutions in all cases while the solutions found by MMForest is comparable with the solver (Figure B.4a). MMForest has fast execution time as in other results.



(a)



(b)

Figure B.5: Impact of total number destination nodes. (a) Maximum bandwidth usage among links. (b) Execution time.

B.2.5 Total Number of Destination Nodes

In this experiment, we change the number of destinations ($N_D = 40, 80, 120, 160, 200$). Other parameters are kept the same $N = 400, N_S = 10, N_W = 40, N_W^S = 8, N_W^D = 20$. Model MMMRP hits its limitation for $N_D = 120, 160, 200$ but still can provide some meaningful BestLB values. Our heuristic algorithm MMForests finds good solutions (compared to BestLB's) by using a little more than 100 milliseconds. The result (Figure B.5) also shows N_D only slightly affects the execution time of MMForests.

B.3 Summary

In this preliminary research, our experimental results show that MMMForests finds good multicast forests in terms of maximizing residual bandwidths while being efficient in execution time. The results also show that the execution time increases significantly in proportion to the size of the network, number of source nodes per data stream, and total number of data streams. It does not increase proportional to the number of data streams requested per destination and total number of destinations. Experimental results also indicate that the structure of the network (degree of the nodes) has impact on the optimal objective value (maximum bandwidth usage among the links). The purpose of this research is to improve multimedia streaming service. The material presented in this appendix appears in [15].