DISTRIBUTED CONSTRUCTION OF RESOURCE-EFFICIENT OVERLAY TREE BY APPROXIMATING MST

LI YUAN

(B.Sc., Nanjing University, China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgement

This thesis is not something I could have completed alone. As the culmination of my short journey through graduate life in National University of Singapore, it is time to look back and thank all of the people who gave me countless support, patience and encouragement while I am growing.

I am deeply indebted to my advisor Dr. Ooi Wei Tsang for his much guidance and constant support during my research. I joined his research group when I was a fresh and naive young research student. He introduced me to the world of research and taught me many valuable skills required for research, from paper writing and reviewing to social skills in academe. He guided me to the area of multimedia networking and showed me a research direction. Working with Wei Tsang has helped me grow academically in ways I hadn't even realized when I first started. He generously lets me draw ideas, insight, and experience from him whenever I reach my hand for help. His guidance and ability to foster independent research skills have helped make me the researcher I am today. His vision, intelligence and passion are an inspiration and an example towards which I strive. I've learned so much from him, not only about how to be a good researcher, but also about how to be a good person. I am looking forward to continuing the interactions

with Wei Tsang after my departure from NUS.

I also would like to thank Dr. Pung Hung Keng and Dr. Sandeep Kumar for serving on my committee and providing me valuable comments and suggestions for my thesis revision.

I would like to thank my fellow members of Networked Media Systems Research Group: Wang Na, Ma Lin, Yanghong and Qingrui. They have provided friendship, conversation, and feedback about a lot of topics, along with a fun and stimulating environment in which to work. They generously shared with me their knowledge of software tools and provided many useful references and friendly encouragement. My former labmates: Ding Chen, Yuan Junli, offer much help and references in getting me familiar with tools and programming under linux. My classmates: Zhou Yongluan and Zhang Xi also teach me a lot in general research. I am really grateful to all of them.

My time at Singapore has not been only about work. I had the pleasure of knowing so many wonderful friends here. They made my life colorful and memorable. Liping is always the quiet listener and good comforter when I am down in spirits. Liang Hui treated me as a younger sister, gave me valuable advices and influenced much in my important decisions. Playing with Jinye, Zhang Xi, Karen couple, Cao Yue, Jimmy and Anne is really fun. I'll never forget those wonderful experiences in hiking, searching for delicious cuisine, barbecues at seashore and the exciting trips to Malaysia. Thanks to Orson for the painful introduction to the art of roller-skating. Thanks to Xingsen and Haiping for playing squash and tennis with me. My weekends could never been so relaxing without them. Thanks to Frederic Chopin and Sergei Rachmaninov for the

night companion with their piano masterpieces when I was working.

The Research Scholarship in NUS, which was awarded to me for the period of my staying, was crucial to the successful completion of this project.

My seniors, Fu Wei and Wu Wen, helped me much both in my study and life on my early days at NUS, and I am also grateful to them for their encouragement and help in my application to NUS. I am lucky to have a friend like Wang Na to grow together for so many years. She gave me much of friendship and caring in daily life besides support in my work. My life at Singapore would be different without her.

Finally, I own the most gratitude to my parents for making me the person I am today. Mom and Dad, thank you for teaching me to appreciate for others. Without your unwavering love and supporting, I could not have reached this point in my life.

Singapore Li Yuan

January 1, 2004

Contents

List of Figures

List of Tables

Summary

This thesis describes the design, simulation, and evaluation of a distributed routing protocol called RESMO (Resource-Efficient Scalable Multicast Overlay) for constructing overlay tree to support video streaming applications. RESMO reduces network resource usage by approximating MST and achieves low end-to-end latency between the sender and each receiver at the same time. The resulting overlay is a compromise between minimum spanning tree and shortest path tree.

RESMO is a mesh-first protocol – nodes in RESMO maintain a mesh and the overlay tree is build on top of the mesh. The tree is constructed in a stepwise manner initiated from the sender. The end-to-end latency is dynamically measured as overlay edge weight during tree construction process. Each end host in the multicast group only maintains states for a small number of neighbors and uses soft-state to keep them up-to-date. In order to adapt to network conditions and group membership changes, the tree is reconstructed periodically without suspending data transmission.

We evaluated the tree constructed by RESMO through simulations and compared it with NICE and Narada application-layer multicast protocols, minimum spanning tree, shortest path tree on the same network scenarios. Simulation results support that RESMO gives significant improvement over existing protocols in terms of link stress, relative delay penalty and resource usage.

Chapter 1

Introduction

The explosive growth of the Internet and increasing demand for multimedia information make media streaming applications a significant fraction of the Internet traffic [1]. Real-time transport of live or stored multimedia content always has real-time constraints and consumes much bandwidth of network link due to the large amount of data content. Therefore, the network support for low latency and high bandwidth data delivery is necessary.

The original one-to-one communication model – unicast fails to efficiently support group media communication due to its high consumption of network bandwidth. An alternative approach – IP-multicast [2] was introduced in the late 1980s by Deering. IP-multicast allows an efficient one-to-many data delivery by eliminating data duplicates on network links and therefore reduces network resource usage to the minimum. However, due to its lack of scalability and support for higher level functionality, IP-multicast is not widely deployed by Internet Service Providers (ISPs) [3].

In recent years, many application-layer multicast protocols [4, 5, 6, 7, 8, 9] have been proposed to address the problems with IP-multicast. In application-layer multi-

cast, routing and data forwarding is carried out in the application layer instead of the network layer. The multicast tree in application-layer multicast (also known as *overlay tree*) is a virtual delivery tree built on top of underlying network where each edge consists of a unicast route between two overlay nodes. Unlike IP-multicast, application layer multicast introduces duplicate packets on physical links and may incur longer end-to-end latency than IP-multicast. In order to reduce the efficiency penalty introduced by application-layer multicast, many current researchers have proposed a variety of protocols for building an efficient overlay tree.

In this thesis, we revisit the existing application-layer multicast protocols and propose the design of a new distributed protocol – RESMO (**R**esource **E**fficient **S**calable **M**ulticast **O**verlay) for constructing an overlay tree in a distributed environment with limited topological information. The resulting tree reduces resource usage by approximating minimum spanning tree and achieves low end-to-end latency between the sender and each receiver at the same time.

1.1 Media Streaming Applications

In recent years, real-time multimedia applications for communication and entertainment have gained tremendous popularity. Advances in computer hardware, compression technology, high-bandwidth storage devices, and high-speed networks have fostered the growth of media streaming applications such as video conferencing and videophone, internet entertainment broadcast, distance learning, network computer games and surveillance.

Video has been an important media in multimedia streaming applications. Video content is typically reworded in the following steps in a typical streaming application: captured, encoded, transmitted, decoded and displayed. This thesis will focus only on the video transmission.

Although the current compression technologies used for video streaming – H.263 V2 and MPEG-4 have increased the compression efficiency drastically [1], video transmission still consumes a large amount of network bandwidth as compared to other media. Video content can be pre-encoded (stored) or real-time encoded (live). The application can be either interactive or non-interactive. Video conferencing and videophones are examples of real-time encoding and interactive applications whereas video-on-demand (VOD) and video streaming over the internet are examples of remote stored video applications. Video streaming is different from transmission of stored video in that the video content is not being downloaded in full before playback, but is being decoded and played out while parts of the content are received. Receivers only buffer part of the content, and "late" data that arrives after playback deadline (defined in terms of buffer size and link transmission delay) may be useless. Therefore, there is a real-time constraint in video streaming applications. In case of interactive applications, the time constraint will be tighter.

To sum up, media streaming applications have the following major properties:

- *they are bandwidth-intensive, and*
- *they are delay-sensitive.*

Video transmission in these applications should consider optimizing bandwidth con-

sumption on network links and reducing end-to-end delay for each receiver.

Internet is a best effort shared network based on packet-switched mechanism where individual packets of different applications may encounter variable delays, arrive out of order, or may be lost if congestion happens. Recently, there is a trend in research to provide application-level QoS (e.g., congestion control, error control, etc.) for continuous media distribution applications such as media streaming [10]. The deployment of these techniques needs support from the application. On the other hand, although IP-mulitcast for delivery of multicasting data is efficient by its original design, it has its own limitations and deployment issues in supporting application level functionality. This has drawn much attention from research community and the industry [3]. Due to these two reasons, application-layer multicast which migrates data replication and forwarding from the IP layer to the application layer was introduced around year 2000.

1.2 Multicast in Group Communication

In this section, we revisit some data delivery techniques for group communication, such as unicast, IP-mulicast, and application-layer mulicast. We also point out the limitations of IP-mulicast and explain the reason for introducing application-layer multicast. An example of the above three techniques is provided.

Traditional one-to-one transmission mode – unicast is not feasible in supporting media streaming applications despite of its widely deployment in today's Internet. Unicast from a source to all receivers introduces duplicate data on a single link which makes the link at the source congested. IP-multicast is an efficient data delivery mechanism that

eliminates duplicates carried on network links. However, the current service model in IP-multicast is designed without a commercial service in mind. This is possibly why it is still under slow commercial deployment 20 years after its invention.

1.2.1 Problems with IP-multicast

The service model and architecture for IP-multicast has the following limitations in its original design:

- Scalability problem in number of per-group states maintained at intermediate routers. One widely used IP multicast protocol, DVMRP requires routers remember routing information for every group G and every source S. The number of states maintained at these routers is $O(|S||G|)$, where $|S|$ is number of sources per group and $|G|$ is number of multicast groups. This results in serious scaling constraints.

- Lack of sender and receiver authentication. The current IP multicast model allows for an arbitrary source to send data to an arbitrary group. This makes the network vulnerable to flooding attacks by malicious sources.

- Scalability and difficulty in global multicast address allocation. IP multicast requires every group to dynamically obtain a globally unique address from the limited multicast address space, and it is difficult to ensure this in a scalable, distributed and consistent fashion. The address collision causes receivers to receive unwanted data, and introduces a serious inefficiency risk for network utilization.

- Difficulty in supporting higher level functionality such as reliability, congestion

control, flow control, and security.

Besides the above practical difficulties of IP-mulicast in supporting wide-area group communication, it also presents a number of challenges to streaming media systems. Firstly, the problems of heterogeneity in today's internet make multicast complicated. Not only the link capacity is various throughout the network. End hosts are also heterogeneous with respect to CPU and storage capacity [11]. In IP-multicast, heterogeneity is typically solved by using multiple layered multicast to provide choices for the receivers [12]. This mechanism needs support from compression technology for layered-encoding. The receiver can therefore elect to join several layers of multicast according to its capacity and requirement. But this is at the price of loss of compression efficiency and additional complexity at routers. Secondly, retransmission, generally used in error control, may cause problems when using with IP-multicast. For instance, both the retransmission request and actual retransmission are transmitted to all the receivers in the multicast group, which obviously leads to a waste of link bandwidth.

To address the problems with IP-multicast, recent research has proposed to implement multicast service at the application layer instead of the IP layer.

1.2.2 Application-Layer Multicast

Application-layer multicast migrates the multicast function from the network layer to the application layer. Therefore, routing and data forwarding is carried on end hosts, which frees intermediate routers from maintaining per group state. The multicast tree in this scheme is a virtual data delivery tree consisting of end-to-end unicast connec-

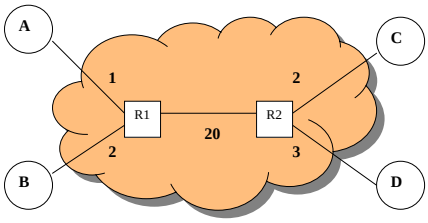

Figure 1.1: Network Topology

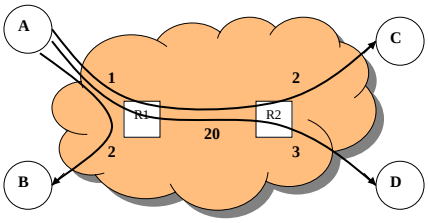

Figure 1.2: Unicast Tree

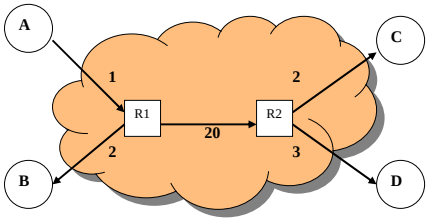

Figure 1.3: IP-Multicast Tree

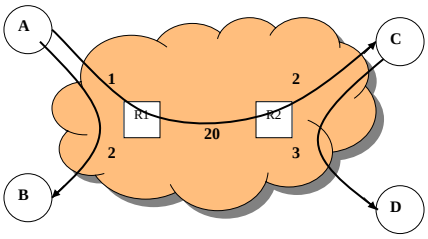

Figure 1.4: Application-Layer Multicast Tree

tions. Therefore, multicast routing information is only maintained at end hosts without additional supporting for maintaining routing tables at underlying routers. Periodically exchanging routing tables at intermediate routers is also eliminated. Compared to IP-multicast, application-layer multicast can be easily deployed on Internet. In addition, solutions for supporting higher layer functionalities such as error, flow, and congestion control, transcoding can be significantly simplified by leveraging well understood unicast solutions for these problems.

Consider Figure 1.1 which depicts an example physical network topology: R1 and R2 are underlying routers, while A, B, C, and D are end hosts. Link delay is also specified in the figure. We assume A is the sender.

Figure 1.2 depicts how unicast tree maps onto the physical topology. It is clear that the link near to the sender: A – R1 carries three copies of a transmission. The most

costly link R1 – R2 carried two copies.

Figure 1.3 and Figure 1.4 highlights the contrast between IP-multicast and application-layer multicast. The IP-multicast tree in this example is constructed by DVMRP [13]. Each path from a sender to a receiver in the IP-multicast tree is the reverse shortest path from the receiver to the sender. R1 and R2 are responsible for copying and forwarding data to the multiple interfaces: B, C, D. At most one copy of a packet is delivered over any physical link. Each receiver encounters a same delay as in unicast.

Unlike IP-multicast, however application-layer multicast introduces duplicate data on physical links. This also can be seen in Figure 1.4 where link A – R1 and R2 – C carry two identical data packets. We know that end hosts are usually located at the edge of networks. Data that arrives at some receivers is forwarded by other end hosts, for example, C forwards data to D in Figure 1.4. The transmission introduces data duplicates on the physical links near the forwarding end host (link C – D). This is not as efficient as IP-multicast. On the other hand, the delay from sender to certain end host may also be increased due to data forwarding by other end hosts. For example, in Figure 1.4, delay in D is increased by 4 (double link delay of R2 – C).

To evaluate the efficiency of overlay trees, Chu et. al. [5] define several metrics which are widely used by researchers. We introduce these metrics as follows:

- *link stress*: number of duplicate packets carried by each link.
- *relative delay penalty* (RDP): the ratio of the delay between the source to a receiver along the overlay tree to the unicast delay between the source and the receiver.

- *resource usage*: $\sum_{i=1}^{L} d_i * s_i$ where L is the number of active physical links covered by the overlay tree, d_i is the delay of link i and s_i is the link stress of link i.

In Figure 1.4: the maximum link stress is 2 of link A – R1 and link R2 – C; RDP for receivers B and C is 1 since the routes for these two nodes in overlay tree are the same with unicast routes. Delay for D is increased from 24 to 28, hence the RDP is 28/24 which is larger than 1. The network resource usage in IP-multicast is 28 whereas in application-layer multicast is 31. Therefore, IP-mulicast has the minimum network resource usage.

1.3 Contributions

In the rest of thesis, we will present a new distributed application-layer multicast protocol called RESMO. By approximating a minimum spanning tree, RESMO builds an efficient tree with less efficiency penalties (described in the previous section) compared with other protocols. Our contribution can be summarized as follows:

1.3.1 Comparable Resource Usage with MST

By definition, minimum spanning tree (MST) has minimum resource usage among all the overlay trees. The first contribution of this thesis is proposing a new distributed algorithms to build a multicast tree with lower resource usage comparable to MST whereas keeping the RDP much lower than other influential published schemes [5, 7].

In RESMO, we dynamically measure end-to-end latencies between relevant mem-

bers and use them as the edge weights. Unlike existing distributed MST algorithms (base algorithm [14] and it's further improvements [15, 16]) which need additional fragment, edge label and layer naming schemes or more complicate message content, our algorithm is quite easy to deploy. It is a short-length message based protocol. By introducing some timers, RESMO reduces the number of messages for tree building.

1.3.2 Scalability without Depending on Hierarchical Mechanism

Scalability is the key concern of network protocol design due to the growth of Internet and its applications. Existing protocols such as NICE [7] achieve scalability by using cluster-based hierarchy to build the overlay. But the price of this scalability is heavier link stress near cluster leaders and increased end-to-end latency caused by each packet passing through cluster leaders first to reach its destinations.

Another contribution of this thesis is proposing an scalable and fully distributed application-layer multicast protocol which has stable performance when group size increases. It does not depend on hierarchical clusters to form multicast tree, hence avoids the hot spot problems which potentially exist in the cluster leaders and rendezvous point.

1.4 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 presents some background information, including the Internet and multicast infrastructure, graph and spanning trees. We also give introduction to related network techniques and software tools we use in our work. Some related work on application-layer multicast is also discussed. In Chapter

3, we described the detailed design of our protocol. For an easy understanding, we also provide a example tree built by RESMO given a small topology and the step-by-step building procedures. We explain and analyze the simulation results in Chapter 4. Finally we end with conclusions and future work in Chapter 5.

Chapter 2

Background and Related Work

In this chapter we first provide background information about the Internet infrastructure and a various classes of traditional multicast technologies used in Internet backbone, as a preface for understanding how application-layer multicast is different from IP-multicast and hence is able to get fast deployment without additional support from underlying routers and OS. Next, we discuss the spanning trees in graph theory in an effort to understand efficient routing in finding paths connecting a sender and many receivers. Third, we look at several related techniques commonly used in network protocol design such as *soft state* and *expanding ring search*. Finally, we survey other related work in application layer mulicast.

2.1 The Internet Infrastructure and Multicast

The Internet is a collection of individual networks known as *autonomous systems* (ASes). ASes are groups of nodes that are under a common administration and share routing information. They are typically owned and operated by different Internet Service Providers (ISPs). The Internet Protocol (IP) is the common underlying communication protocol

shared by these networks. Individuals and smaller companies usually attach to the Internet via an local ISP. Home users typically connect to Internet via modems (33.6 or 56kbps), Digital Subscriber Lines (DSL, 128kbps - 1Mbps), or cable modems (128Kbps - 2Mbps). Corporations, universities, and service providers attach in a similar manner, but with higher speed T1 (1.5Mbps) or T3 (45Mbps) links. In this architecture, the Internet is a heterogeneous network consisting of various link bandwidth and a diversity of user capacities. Furthermore, as an exponential increasing network, the Internet lacks of a centralized administration which makes routing in such a huge network complicated. In next section, we will give a model which is approved to best reflect the Internet.

2.1.1 Transit-Stub Network Model

In Transit-Stub model [17], the domains in Internet can be classified as either *transit* domains or *stub* domains. A transit domain comprises a set of *backbone* nodes, which are core routers in Internet. Stub domains comprise of *leaf* networks, which have links to one or more transit domains. The responsibility of transit domains is to interconnect stub domains efficiently. A stub domain can be linked to more than one transit domains. In this case, it is called multi-homed. Nodes from different stub domains can also be connected by Stub-Stub edge.

Figure 2.1 gives an example of Transit-Stub domain structure. Edges within each domain is considered as intra-domain links whereas edges connecting different domains are considered as inter-domain links.

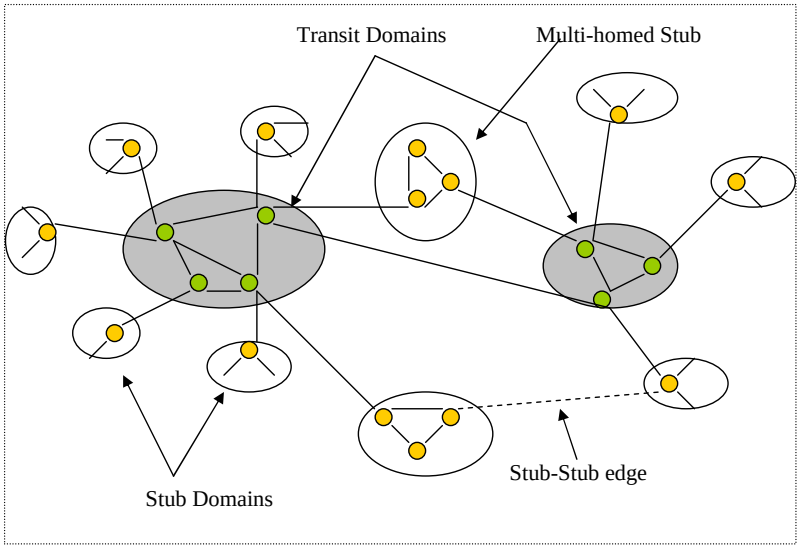

Figure 2.1: Example of Transit-Domain structure

2.1.2 Multicast Techniques

We have discussed in Chapter 1 that many fast increasing applications (media streaming applications) fall in the category of group communications which have several sources and a number of receivers. These applications drove the development of the multicast service. Multicast communication is capable of distribute information to one or many receivers in such a way that each link carries only one data packet.

In IP-mulicast, mulitcast routers are responsible for building and managing the multicast distribution tree. These routers can be classified as either leaf routers (i.e. with end-hosts connected) or core routers (i. e. on a transit network). Edge routers use Internet Group Management Protocol (IGMP) to discover the presence of local receivers which are hosts willing to receive traffic destined to a multicast group. Core routers participate in the distribution tree management and multicast packet forwarding. Distance Vector Multicast Routing Protocol (DVMRP) is a classical protocol running on

core routers. It is used in the MBONE (Multicast Backone), where the data delivery tree consists of reverse shortest paths from sender to each receiver.

Multicast protocols fall into either **dense mode protocols** or **sparse mode protocols**. Dense mode protocols always use *broadcast-and-prune* mechanism. The multicast tree is a reverse shortest path tree rooted at the source; Sparse mode protocols are based on *explicit join* mechanism. In this mode, either a reverse shortest path tree or a shared tree can be used. A shared tree uses a core or a rendezvous point to connect senders and receivers together.

Detailed description of all the IP-multicast protocols is out of scope of this thesis. We give the brief introduction above of IP-multicast in order to note that there is no a common protocol used in the networks. It is difficult for ISPs to make agreement in using a standard multicast protocol. This is also the current deployment issue with IP-mulicast besides the ones we talked in Chapter 1. All of the limitations with IP-multicast drove the emerging of application-layer multicast.

2.2 Theory for Multicast Routing Problems

Graphs are commonly used to model the structure of networks, for the study of problems from routing to resource reservation. Routing is, in essence, an art of graph theory [18]. Consider a graph $G = (V, E)$, consisting of a set of nodes (vertices) V and a set of links (edges) E. M is a subset of set V, including the nodes of a multicast group. The multicast routing problem can be defined as finding one or more interconnection topologies that span all nodes included in M. Typically, such topology is a source-

specific tree or a shared tree for multiple sources.

There are two well known spanning trees in an edge-weighted graph, namely, *minimum spanning tree* (MST) and *shortest path tree* (SPT). We will give definitions of the two trees as following. In real network, the edge weight is always defined as link latency between a pair of nodes.

Minimum Spanning Tree The minimum spanning tree of a weighted graph is a set of edges of minimum total weight which form a spanning tree of the graph. In a centralized manner, the minimum spanning tree can be found in polynomial time. Common algorithms include those from Prim (1957) and Kruskal (1956).

By definition of *resource usage* described in Chapter 1, we can easily deduce that in an application-level overlay, it is equivalent to the sum of virtual edge delays in the overlay multicast tree. MST rooted at the sender therefore is the optimal tree for minimizing resource usage, but it may not be suitable for streaming applications due to the long end-to-end latency it introduced.

Shortest Path Tree Another well-known tree is Shortest Path Tree, which consists of shortest paths between source and each receiver. The shortest path is defined as a path with minimum end-to-end delay.

In application-layer multicast, SPT is optimal with respect to end-to-end delay from the source. But is has its own limitation: SPT will lead to more resource usage by introducing heavier link stress, and the sender's bandwidth may also become a bottleneck.

As we have argued in Chapter 1 that media streaming applications are delay sensitive

and bandwidth intensive. A good routing for this kind of application therefore should optimize end-to-end latency and network resource usage as well (balance between MST and SPT).

2.3 Related Network Techniques

In this section, we look at various techniques crucial in network protocol design. Our protocol also uses these techniques to achieve better performance.

2.3.1 Soft States

In network protocols design, *state* refers to information stored by network nodes. The content of information can be various. For example, Internet Group Management Protocol (IGMP) in a host stores the information of the multicast groups which the host joins; Some multicast protocols such as DVMRP [13], Protocol Independent Multicast (PIM) [19] and Core Based Tree (CBT) store multicast routing state in the routers. The network nodes exchange with each other the states in order to adapt to the network condition. Therefore, the states must reflect the changes in network conditions quickly and accurately.

Soft state uses refresh messages to keep it alive and is discarded after some time interval if the state is not refreshed [20]. The term is first introduced in [21]. Unlike *hard state* which is installed in nodes upon receiving a set-up message and is discarded on receiving an explicit tear-down message, soft state is controlled only by periodically arriving refresh messages. The refresh message sender sends message periodically after

a *refresh period*. In general, the receiver which maintains corresponding state waits for a period of small multiple of the refresh period before discarding that state. Soft state protocols can achieve great robustness and faster adaption to the changes in the network condition. Therefore, it is commonly used in writing light-weight protocols such as Resource Reservation Protocol (RSVP) and PIM.

2.3.2 Expanding Ring Search

"Expanding Ring Searching" is first introduced by Boggs in his dissertation on inter-network broadcasting [22]). The main mechanism in searching related nodes is broadcasting query to increasingly larger concentric circles with a scope constrain in order to limit the distance a searching packet may travel. An example of its use is in multicast protocol design. Some protocols include a *time-to-live* (TTL) field in the packet header for the purpose of bounding the amount of time a packet may travel in a large scaled and multi-hop internetworks [23]. By using a very small TTL value, a sender may limit the packet to reach only nearby neighbors and also reduce the number of responses when multicasting to a large group.

2.4 Software Tools

2.4.1 Tcl and OTcl

Tcl, or Tool Command Language was originally designed as a reusable command language and evolved to a widely used scripting language. As an interpreted, scripting

language, Tcl has very simple syntax which treats all data types as a string. The distinguished feature of Tcl is that it has the best interaction with C. Tcl is the first scripting language which has simple and clean interface with C. Another feature is the extensibility using C language.

OTcl [24] is the object-oriented extension of Tcl. It is built on Tcl syntax and concepts but added object-oriented features. OTcl inherits the extensibility and interactions with C. It is used as base language for NS-2 [25] which we will introduce later.

2.4.2 GT-ITM

Georgia Tech Internetwork Topology Models [26] is a software toolkit to generate graphs that models a variety of internetworks topologies. The graphs are generated in Donald Knuth's SGB (Stanford GraphBase [27]) format. GT-ITM assigns edge weights representing delay based on Euclidean distance between nodes placed on a plane with a uniform random distribution.

GT-ITM can generate three kinds of topologies: Flat random graphs, N-level hierarchical graphs, and Transit-Stub graphs. The authors in their paper [17] compared properties of graphs generated using various method with those of real Internet. They concluded that Transit-Stub model is an efficient method for generating topologies with properties correlated well with Internet structure.

Flat random graphs GT-ITM provides a variety of flat random graphs used to model internetworks, such as Pure Random, Waxman [28], Doar-Leslie [29]. These models all distribute vertices at random locations in a plane. The difference exists in how to

decide the edge probability of pairs of vertices. Pure Random uses a constant α as edge probability. The other models define a probability function in terms of Euclidean distance of each pair of vertices, maximum distance between any two nodes and other parameters.

N-level hierarchical graphs This model constructs a hierarchical topology recursively. It starts with a connected graph, at each step in the recursion the nodes in the current topology is replaced by a connected graph. The nodes which are replaced by graphs are selected at random.

Transit-Stub graphs This is a hybrid graph generation method, capable of creating large graphs by composing smaller random graphs. We have explained definition of this model in section 2.1.1. By imposing a domain structure resembling that of the Internet, the Transit-Stub model allows creation of large random graphs having realistic average node degree. Moreover, by generating Transit-Transit, Transit-Stub and Stub-Stub edges in a controllable manner, it can add intra- and interdomain paths in the graph.

2.4.3 NS-2

Ns-2 [25] is the second version of a discrete event simulator targeted at networking research. The simulator has a list of events scheduled in advance and uses single thread of control. NS-2 is designed for research and education in network protocols design, traffic studies, routing and queueing techniques.

NS-2 is written in C++ and OTcl. C++ code is fast to run and used for data ma-

nipulating such as packet processing. It is the core of ns. OTcl code is fast to write and easy to change and understand. It is used for control purpose such as simulation scenario configuration, event scheduling, manipulating existing C++ objects. In Ns-2, OTcl and C++ share class hierarchy and each of them can be mapped onto the other. NS-2 also provides a linkage between C++ and OTcl.

A project implemented with NS-2 has the following components:

- Pre-processing: topology and traffic generators.
- Ns: the simulator itself.
- Post-processing: Simple trace analysis, often written in Awk, Perl or Tcl.
- Nam: the network animator to visualize ns output.

The typical steps for programming are: create network topology (using GT-ITM), set up routing, create the event scheduler, turn on tracing, create transport connection and create traffic.

2.5 Related Work

There are many application-layer multicast protocols published. The general classification has two categories: tree-first (ALMI [4], HMTP [8], TBCP [30], NICE [7]) and mesh-first protocols (Narada [5], Gossamer [6]).

In the mesh-first approach, overlay nodes first distribute organize themselves into the overlay mesh topology. This mesh is used as a control topology maintaining group

membership. A source-specific tree is built on top of the mesh. In contrast, protocols based on the tree-first approach distribute construct the data delivery tree directly.

2.5.1 Narada

Narada [5], or End-system multicast builds the overlay tree on top of a pre-built mesh. The mesh is built with node degree constrains so as the node degree reflects the outgoing bandwidth of each node. When building the tree, Narada runs existing DVMRP [13] on top of the mesh. Hence, the resulting tree is a sender-specific shortest path tree on the underlying mesh. Since the tree is built absolutely on top of this overlay mesh, the mesh's quality is crucial to maintain the tree's efficiency. Narada improves the mesh's quality in a local way by each member randomly selecting an edge either inside or outside the mesh, computing the utility gain and deciding whether to drop or accept it after comparing with a certain threshold. Narada approach has limited scalability because of each member must maintain states for all other members.

2.5.2 ALMI

ALMI, an Application Level Multicast Infrastructure, provides many-to-many multicast for large number of communication groups with small number of members (tens of nodes) [4]. It is a Java based implementation of multicast middleware above the sockets layer. Unlike distributed protocols such as Narada, ALMI uses a centralized scheduler to compute its multicast trees. The participated group members are connected via a virtual minimum spanning tree using application level round-trip delay between them as the

cost metric.

Since ALMI is a centralized approach, the overlay nodes can be classified as *session controller* and *session member*. Session controller handles member registration and maintains the multicast tree by ensuring connectivity when network or host failures occur and by ensuring the efficiency of the multicast tree through periodically calculating a minimum spanning tree. The tree is built on the measurement updates collected from all session members. Session members monitor the performance of unicast paths to and from its neighbors and report to the session controller in order to serve as the cost used to calculate the minimum spanning tree. After calculation, the controller disseminates the results in the forms of a (*parent*, *children*) list to all members. Obviously, a centralized control may lead to a single point of failure for all control operations related to the group at the controller site. ALMI solves this problem in a way that it introduces multiple *backup* controllers, operating in "stand-by" mode.

2.5.3 NICE

NICE[7] is a hierarchical clustering-based protocol which is more scalable in average. The sender sends data to its cluster peers in the basic layer. The leader of this cluster then forwards data to its cluster peers in upper layer and this action continues until all the members receive data. For robustness reason, NICE also provides multiple paths by sending data to a Rendezvous Point(which is the leader of the single cluster in the highest layer in their implementation). RP then forwards data layer by layer down to the lowest layer. This scheme is useful in fault tolerance but will cause much link stress and

resource usage also. NICE is designed for low bandwidth application with large group size. The main problem of NICE is the dependence on some special nodes, such as RP and cluster leaders of each layer. Failure of these hot spots will damage the tree a lot. Another problem with NICE is that if group membership changes frequently, the performance will suffer from no longer center-location of cluster leaders. The cluster grouping is based on network locality, if the group membership changes with new members joining and old members leaving, after a certain time the original leader of each cluster may not be the graph center, which will cause much delay penalty.

2.5.4 Priority-Based Distribution Trees

Priority-Based Distribution Trees (PBDT) [9] is an application-layer multicast protocol aimed at trading minimum spanning tree with shortest path tree in its resulting data delivery tree. In PBDT, the sender assigns a priority to each receiver with respect to their application-level features and then uses this priority to balance between end-to-end delay and resource usage. The metric function in calculating the tree is in terms of MST and SPT cost with each of them having a priority coefficient. For example: $C = (1-p) * C_{MST} + p * C_{SPT}$, where C_{MST} is the MST cost and C_{SPT} is the SPT cost. By defining the priority value p, PBDT can easily adjust the tree to be close to MST or SPT. The two extremes exist when $p = 0$ or $p = 1$, where the tree is absolute MST or SPT. Like ALMI, PBDT is a centralized protocol and designed for small group size also such as network computer games.

2.5.5 LAST on Hierarchical Overlay

Light Approximate Shortest-path Trees (LAST) [31] is another algorithm that trades off latency with cost. This approach traverses a MST in a depth-first fashion, whenever it encounters a node with MST delay larger than SPT delay by a factor of α, it adds links from the node's shortest path to the current tree. LAST is evaluated on a hierarchical overlay in [32]. The authors find that LAST allow application developers to flexibly trade resource usage with delay.

2.5.6 Distributed MST

The basic Distributed MST algorithms [14] constructs a spanning tree consisting of rooted sub-trees, each subtree being a fragment with a label indicating its level. Each node is initially a fragment. When two fragments find a "best" edge among themselves and want to unite through this edge, they follows a rule that only a higher-level fragment can "absorb" the lower level one to avoid forming cycles. This algorithm also uses delaying response to certain kind of messages to reduce number of messages required. The subsequent improved algorithms [15, 16] balance between number of messages exchanged and building time consumed. But they did not reduce the orders of magnitude of communication complexity, which remains $O(E + N \log(N))$. These algorithms require more complicate message content and extra processing at each node.

2.6 Conclusion

We have described the Internet infrastructure and the various techniques for multicast. For better understanding of multicast routing and laying a foundation for the description of our protocol in next chapter, we consider the network as a graph and look at some theory of spanning trees supporting multicast service. Next, we introduced two common network techniques which will be used in our design. This is followed by introduction to the software tools we used for simulation. Finally, we gave an overview of related work on application-layer multicast and address some of its strength and also weakness.

Chapter 3

RESMO Protocol Design

In this chapter, we present the design of our protocol – RESMO (Resource Efficient Scalable Multicast Overlay). We first overview the protocol with an example and compare the resulting tree's quality with MST and SPT to see how RESMO achieves a compromise between them. Next, we describe the control and data topology in our design, including neighbor searching and tree construction. The full protocol description is followed by protocol analysis in which we introduce the mechanism behind our protocol and explain why RESMO achieves a better performance. Finally, we will describe the procedures in building the example tree step by step.

3.1 Protocol Overview

RESMO is a mesh-first protocol – nodes in RESMO maintains a mesh and the overlay tree is build on top of the mesh. The mesh can be regarded as control topology above which overlay nodes exchange information for group membership management. Data topology in RESMO is a source-specific multicast tree used for data delivery from sender to each recipient. We use source-specific tree instead of shared tree is to balance the

traffic load when several sessions are concurrently active. Moreover, each source with an individual tree is the best solution for achieving low end-to-end latency.

To construct the control topology, we use expanding ring search (we have introduced in Chapter 2) to discover neighboring nodes and form the mesh. We will give further explanation in Section 3.2.1. RESMO constructs the overlay tree step-by-step, "growing" the tree from the sender. At each step, the current leaf nodes in the tree are actively involved in constructing the tree by sending out invitations to its neighbors to join the tree. This process stops when a leaf node has no neighbors that is interested in the given session. The involved states and messages will be described in Section 3.3.1.

RESMO considers link latency only during the tree construction process, and does not consider link bandwidths. Links that do not meet the bandwidth requirement of a session are filtered out when selecting neighbours. In other words, only links that are "fat enough" are considered by RESMO.

The resulting tree is a compromise between minimum spanning tree and shortest path tree. As a result, all aspects of the tree properties are in-between MST and SPT, avoiding the limitations of the two extremes. We will see this clearly in the following example.

3.1.1 RESMO by Example

In order to give an clear view of the tree built by RESMO, we simulate a small topology with 8 overlay nodes out of totally 35 nodes using GT-ITM and NS-2. The underlying topology with involved nodes and links only is represented in Figure 3.1. The edge weight in the figure is physical link delay. On this underlying topology, we build MST,

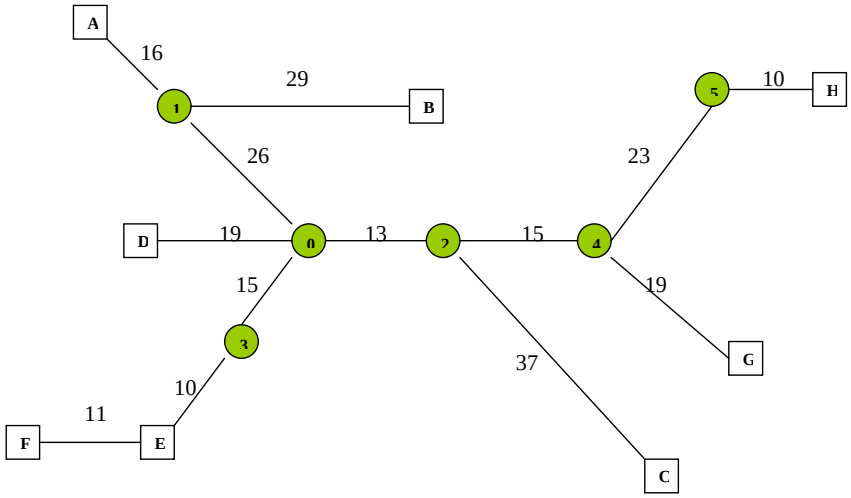

Figure 3.1: Underlying Topology

SPT and RESMO trees respectively. The corresponding data delivery paths are indicated in Figure 3.2, 3.4, 3.6.

From these figures, we can see that the three overlay trees have occupied the same physical links, but the link load distributions are different. MST and RESMO have more balanced traffic load distribution on physical links than SPT (in SPT, links near the source encounter higher traffic load). The data delivery path in RESMO has much similarity to MST than to SPT. From all of the recipients, only C and H have different paths from the sender in RESMO and SPT. This confirms our original design to approximate MST.

The virtual trees of above three approaches are abstracted in Figure 3.3, 3.5, 3.7. We can see that SPT has largest node degree – 6 at the sender whereas MST and RESMO both have 3 as the maximum node degree. Therefore, MST and RESMO excel SPT in terms of node stress.

Till now we have compared the three trees glancingly. We then give quantitative

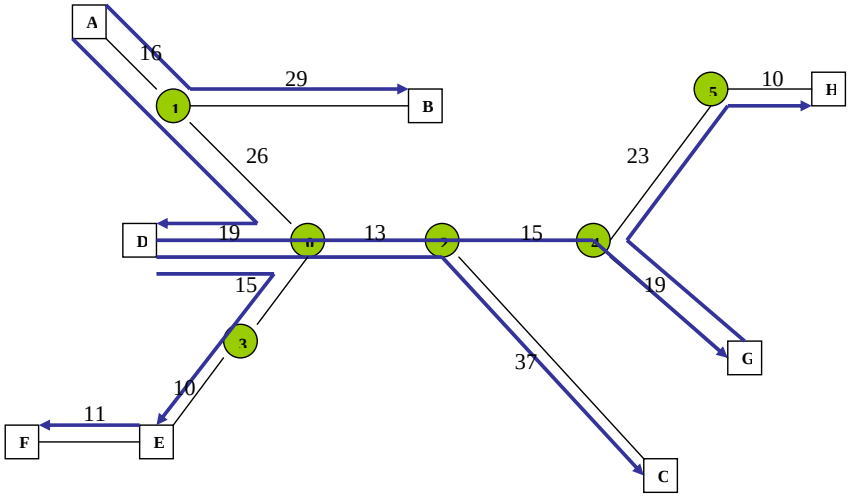

Figure 3.2: MST Path

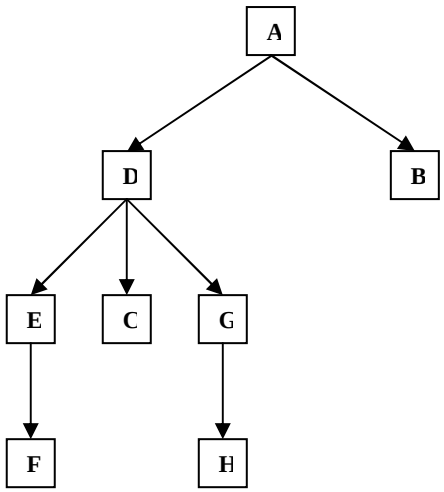

Figure 3.3: MST Tree

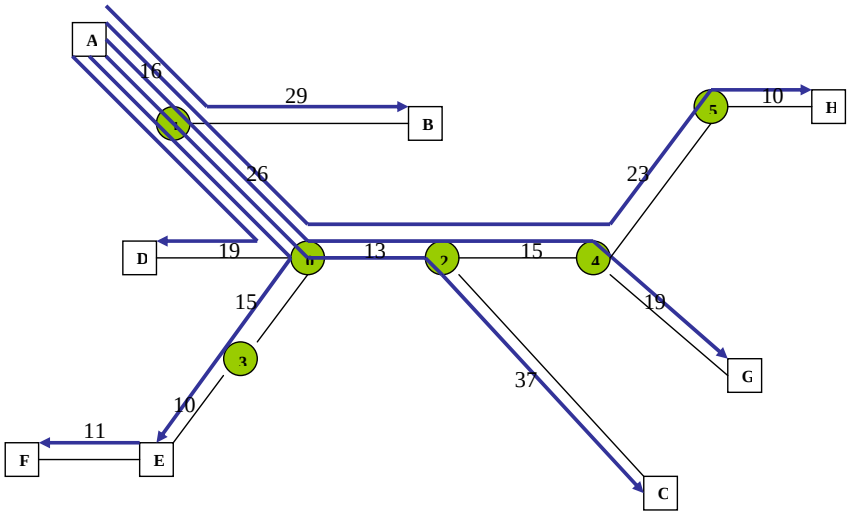

Figure 3.4: SPT Path

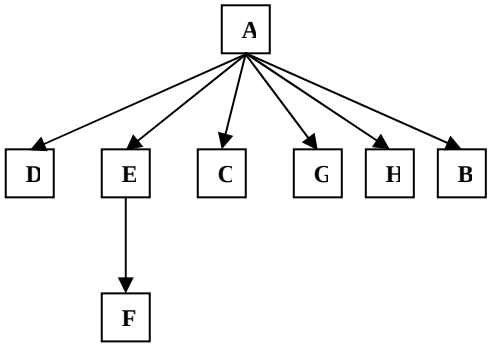

Figure 3.5: SPT Tree

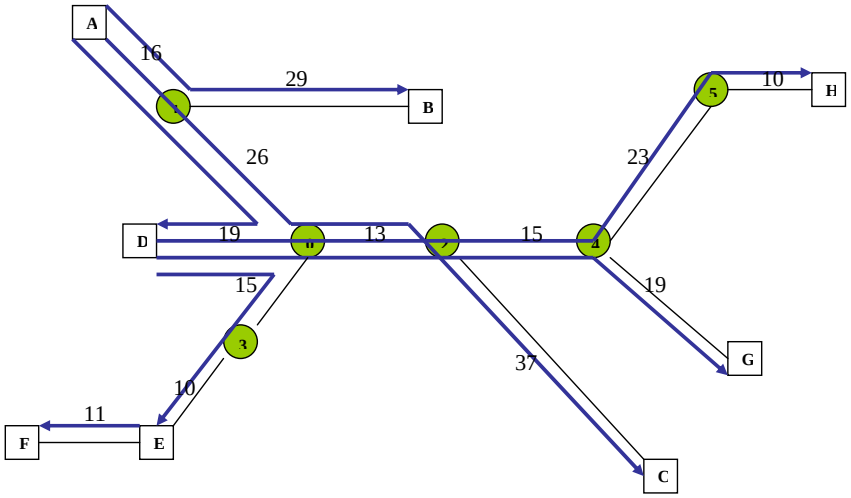

Figure 3.6: RESMO Path

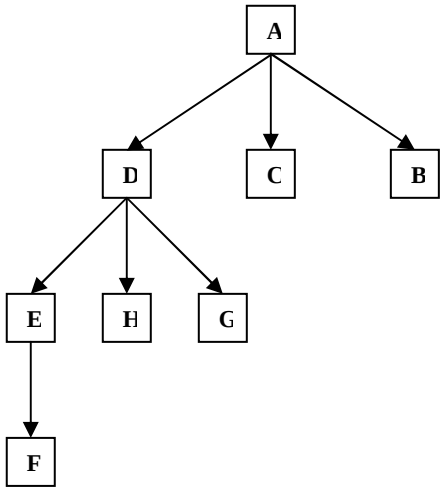

Figure 3.7: RESMO Tree

Table 3.1: Comparison between MST, SPT, RESMO

Method	**RU**	**Max link stress**	**Mean link stress**	**Max RDP**	**Mean RDP**
MST	348	4	1.46	1.75	1.38
SPT	468	6	1.92	1	1
RESMO	399	4	1.69	1.57	1.27

analysis of the tree qualities according to the performance metrics introduced in Chapter 1. The results are summarized in Table 3.1. From this table, we can see that RESMO has a compromise performance between MST and SPT in all metrics. It has low resource usage comparable to MST (15% over MST) whereas SPT has 35% excess resource usage over MST. The mean link stress for RESMO is 1.69, which is between 1.46 (MST) and 1.92 (SPT). In this example, SPT has the same paths with unicast, therefore, relative delay penalties (RDP) in SPT are all unit. MST has the largest maximum and mean RDP. RESMO is in-between. Recipient experiences different delays from the source in each multicast tree, the differences are listed in Table 3.2 with minimum delay highlighted. It is clear that SPT always has the minimum end-to-end delays. As for RESMO, the delay is improved much from MST for receiver C and H.

Now we have an overview of RESMO, its resulting tree and the tree's quality. In the following sections we will describe the control topology and data topology followed by a description of the procedures in constructing the example tree.

Table 3.2: Delays in MST, SPT, RESMO

Receiver	**delay in MST**	**delay in SPT**	**delay in RESMO**
B	**45**	**45**	**45**
C	130	**92**	**92**
D	**61**	**61**	**61**
E	105	**67**	105
F	116	**78**	116
G	127	**89**	127
H	179	**103**	141

3.2 Control Topology

Control topology is used for group membership management and also the mesh for tree building. Each overlay node exchanges **alive** message with its neighbours on the control topology periodically to make their neighbours up to date. Unlike existing mesh-first protocols such as Narada [5]and Gossamer [6] which make efforts to maintain a mesh with good quality and run existing multicast protocols such as DVMRP to build a tree, our focus is the construction of an efficient tree on any overlay mesh. A mesh with minimum function to serve as a control topology for exchanging neighboring information is enough. Hence, a simple method for building mesh can be used here. In RESMO, we use expanding ring search to discover neighboring nodes so as to form the mesh.

3.2.1 Expanding Ring Search

It is desirable if each participant reaches all of its neighbors and get full information of the group membership. This is feasible when group and topology size is small but may not scalable when group size increases. The long searching time is also intolerable when topology is huge. In RESMO, every overlay node performs an expanding ring search [22] by broadcasting query to increasingly larger concentric circles. The searching scope is constrained by a so called *time-to-live* (TTL) value. For example: if $TTL = 10$, one only searches within 10 hops from itself. Only those members within this range can be regarded as neighbors. The TTL value should be configured for different network scenario.

After neighbor searching, each overlay node obtains a list of their neighbors on the overlay mesh and begin to exchange **alive** messages on the control topology. The neighboring information is maintained in soft states.

3.3 Data Topology

Data topology in RESMO is the multicast data delivery tree built on top of the control topology. The tree is built in a distribute manner and based on messages sending/listening mechanism. During the tree building process, each participant will experience a sequence of states before it goes into stable. The states they are in at one time reflect the current progress in their way to being part of the tree.

In this section, we first introduced the involved messages and states. Then we de-

Table 3.3: A Summary of Message Types in RESMO

phase	type	description
construction	invite	Invitation to join the session
	thanks	Response to an invite message
	be-my-child	Asking the recipient to be its child
	be-my-parent	Asking the recipient to be its parent
maintenance	join	Request from a new joining member
	welcome	Response to a join message
	bye	Farewell from a leaving member
	alive	Periodic heart beat message

scribe our tree building protocol in three parts: (a) **Tree construction** covers a full description of the process and state transitions involved in building a multicast tree. (b) **Tree Maintenance** will talks about group membership management with respect to member joining, leaving and failure. (c) **Tree Improvement** gives solution to adapting the tree to the network conditions.

3.3.1 States and Messages

There are eight messages involved in RESMO. They are from two phases, namely tree construction and tree maintenance. The messages are listed in Table 3.3.

A RESMO node can be in one of the following five states at one time: (a) *sleep*: a node is in the *sleep* state (with respect to a session) if it has not been invited to join the session. (b) *awake*: a node is *awake* if it has received one invitation to join the tree,

but has not yet become part of the tree. (c) *weighing*: a node is in the *weighing* state if it has received more than one invitations to join the tree, and is deciding which inviter should be its parent. (d) *inviter*: a node is an *inviter* if it has a parent but has no children. Overlay nodes in this state will send invitations to its neighbors to expand the overlay tree, constructing the tree recursively. (e) *parent*: a node is said to be in the *parent* state when it has one or more children.

3.3.2 Tree Construction

For convenience of description, we present the symbols and terminology used in our description:

- S is the sender of given session. $G_i, (1 < i \leq m)$ is any overlay node wants to join the session except the sender.
- $neighborlist$ is a list of neighbors after expanding ring search from each node.
- $rtttable$ is a table storing round trip time values from neighbors every overlay node should maintain.
- $prev - hop$ and $next - hops$ store the node's parent and children's list in the tree.

Tree Building and State Transition

RESMO is a self-organized, distributed tree building protocol initiated at the sender. Like other distributed application-level protocols design (such as AGLP[33]) does, our protocol adopts *soft state* to store information for robustness and scale.

All overlay participants will traverse all of or part of the five states before it's be linked to the tree and go to stable. The sequence of state transitions depends on the network topology and the mechanism behind RESMO (discussed later). Figure 3.8 gives the state diagram where each node will follow a certain path before going stable. The stable states can be either *inviter* or *parent*. If a node has *inviter* as its final stable state, it is a leaf node in the resulting tree. On the contrary, a node with *parent* as its final state will be an intermediate node in the tree. The state transition is triggered when overlay node receives a message or one of the timers it maintains timeouts. We will give the detailed explanation in the following paragraphs.

Sender S is different from other nodes in that when initiating the tree construction it goes directly to *inviter*, other nodes have to follow the path *sleep* $\Rightarrow$ *awake* first to get to either *inviter* or *weighing*. At the beginning of tree construction, all nodes except S are in *sleep*. We describe how the state transitions happen:

sleep $\Rightarrow$ ***awake***

The sender S sends **invite** message to neighbors $G_i, G_j, ...$ according to its $neighborlist$. If a participant in *sleep* state G_i receives it, it replies the sender right away with a **thanks** message and goes into *awake* state. It then computes the delay $d_{S,i}$ by deducting the timestamp in the message content from current time and record this one-way rtt into $rtttable$. G_i also sets a timer for this first incoming invitation. If the timer timeouts and G_i is still in *awake*, G_i sends **be-my-parent** to S. We will explain why setting this timer later in Section 3.4.

***inviter* ⇒ *parent* (I)**

When the first **thanks** which we assume to be from G_j reaches sender S, S accept G_j as its child and goes into *parent* state. At the same time it sends **be-my-child** to G_j to inform it and add G_j to its $next - hops$. There are other **thanks** messages arriving at S consequently, S just ignore them. If S receives **be-my-parent** from G_i, it just appends G_i to its $next - hops$.

awake* ⇒ *inviter

G_j should be in *awake* state now. Once upon receiving **be-my-child** from S it sets $prev - hop$ to S and goes into *inviter* state. Whenever a participant goes into *inviter*, it carries on the inviting task. G_j therefore sends **invite** messages to its *silent neighbors*(neighbors they haven't received any messages except **alive** before) as the sender S does and waits for any incoming message.

awake* ⇒ *weighing

It is possible that a node receives multiple invitations. For example, G_k first receives **invite** from S and turns into *awake* state, during its stay at *awake* ,it may receive another **invite** from G_j since G_j is already been linked to the tree and is G_k's neighbor. G_k then should decide which one to be its parent. It goes into *weighing* state. The weighing time is also controlled by a timer $T_{wgh,j}$. If G_k stays in this state and keeps receiving **invite** from any other neighbors: $G_m, G_n, ...$, it will set timers for all of them as following: $T_{wgh,m}, T_{wgh,n}...$ and keep *weighing*.

weighing* $\Rightarrow$ *inviter

If G_k is in *weighing* and receives **be-my-child** from G_b, it adds G_b as parent and goes into *inviter*. All the other pending timers in G_k will be discarded.

This state transition can be done in another situation: G_w is assumed to be in *weighing* and one of its T_{wgh} timer timeouts, G_w then searches through its $rtttable$ for a nearest neighbor G_n (neighbor has minimum one-way rtt) which has sent invitation to it before. G_w adds G_n as its parent and sends **be-my-parent** to G_n. Note that $rtttable$ has indications for the neighbors from which it receives **invite**. Once G_w goes into *inviter*, it continues on inviting others to join the tree by sending **invite** to its *silent neighbors*.

***inviter* $\Rightarrow$ *parent* (II)**

This state transition is a little different from (I). It is for other nodes except sender S. We assume G_j is an *inviter*. Upon receiving either first **thanks** or **be-my-parent**, it becomes *parent*. If the first incoming message is **thanks** from G_t, G_j appends G_t to its $next - hops$ and sends **be-my-child** to G_t. If the first incoming message is **be-my-parent** from G_r, it just appends G_r to its $next - hops$. The difference between (I) and (II) is that (I) is special for sender S which could only receives **thanks** first.

For ease of understanding, we provide a state diagram (Figure 3.8) and give a summary of the state transition rules.

Summary

- An overlay node must pass by *inviter* to reach *parent*.

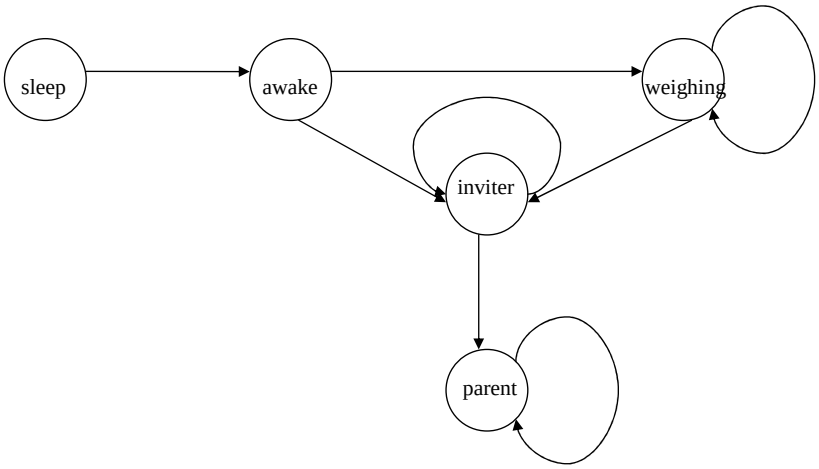

Figure 3.8: State Diagram for RESMO

- If an overlay node goes into *inviter* either from *awake* or *weighing*, it must carry on inviting new members. If it has no neighbors to invite or none of the invitation is accepted, it will be a leaf node in the tree.

- If a node has ever been in *inviter*, upon receiving **invite** it will record the invitation to $rtttable$ regardless of its current state. All of the invitation will be treated at parent candidates information.

- $rtttable$ has indication for those neighbors who have sent **invite** to it. This information is very useful for tree partition recovery. We will talk about it in section 3.3.3.

- One accepts those who send **be-my-parent** to it as children regardless of its current state.

Besides those so called “tree building” messages, note that overlay nodes also exchange **alive** periodically. Once upon receiving **alive**, overlay nodes refresh their corre-

sponding $prev - hop$, $next - hops$, and update the $rtttable$.

3.3.3 Tree Maintenance

Member Join

When a new node G_{new} wants to join the session, it must have finished expanding ring search for the neighbors. Our aim is quick connection to new joiner. G_{new} sends **join** to everyone in its $neighorlist$ and waits for the first **welcome** response. If the first response is from G_{offer}, it takes G_{offer} as parent and informs G_{offer} also by sending **be-my-parent** to it. Once G_{offer} receives this message, it adds G_{new} to its $next - hops$ and starts forwarding packet to G_{new}. After joining the tree, G_new will start exchanging **alive** messages with its neighbours also.

Member Leave and Failure Recovery

If a node wants to leave the session, it must inform its parent and children by sending **bye** to them. The parent removes the leaving node from its children list. The children of the leaving node will look through $rtttable$ and send **be-my-parent** to the nearest parent candidate (which sent invitation to them before).

Neighbor failures can be detected by not receiving **alive** messages for a certain time. Parent failure is treated as if the parent has left the session.

3.3.4 Tree Improvement

As network conditions and group memberships change continuously, RESMO must adapt since the overlay tree may not be efficient anymore. Our solution is to reconstruct the tree periodically. During reconstructions, nodes continue to receive and forward data along the current tree, and switch over to the new tree when reconstructions is completed. This procedure should occur rarely.

3.4 Protocol Analysis

After describing the involved states and messages, we now explain how our algorithm can produce an efficient tree by balancing between MST and SPT. The subtleties lies in the two kinds of timer mentioned above.

Firstly, G_i sets a $T_{ivt,s}$ timer when it first receives **invite** from G_s. Since G_i is in *awake*, if its **thanks** message wasn't the first one to reach G_s and it hasn't any other neighbors from which it can receive **be-my-child**, it may never be linked to the tree. Another consideration is that: if we set the timer to a bigger value, during this time period there is no other **be-my-child** arrives, we can deduce that the delay between G_i and G_s though other route is very long and in order to achieve a lower delay we let G_i be linked to G_s directly by sending **be-my-parent** to G_s. Usually we set $T_{ivt,i} = k \times d_{s,i}$, where k is a constant.

Secondly, nodes set several T_{wgh} timers when they are in *weighing* and receiving multiple **invite** messages. The key to approximating MST lies in the setting of this kind

of timers. An efficient tree should take tradeoffs between resource usage and end-to-end delay into consideration. Bearing this in mind we define a value as following: the time period that is sufficient for overlay node, say G_k to get necessary delay information from its neighbors and select the "best" one to be its parent. Here we define *best* as *nearest* is to approximate MST.

The selection of parent with smallest RTT is crucial to the approximation of MST in RESMO. The value of $T_{wgh,*}$ can be tuned to trade-off resource usage with RDP of the resulting overlay tree. A large $T_{wgh,*}$ allows G to wait for more invitations, hence increasing the chance of selecting a parent with lower RTT. While selecting parent with lowest RTT reduces resource usage, it does not guarantee that the end-to-end delay from S to G is also small. The key to reducing end-to-end delay is to use smaller $T_{wgh,*}$. Since RESMO construct the tree in a stepwise fashion, a node that is closer to the source is more likely to become an inviter and send out invitations sooner. By using smaller $T_{wgh,*}$, a node will select those who becomes inviters sooner, thus reducing the end-to-end delay from S to itself.

We set $T_{wgh,p}$ to be a random value in $[a \times d_{p,k}, b \times d_{p,k}]$, where a, b are both constants. By adjusting the value of the timer, we can tune the tolerable end-to-end delays in the tree.

For a better understanding of the timers, messages and states in RESMO protocol, we will describe the building procedures in the example tree (Figure 3.6 and 3.7).

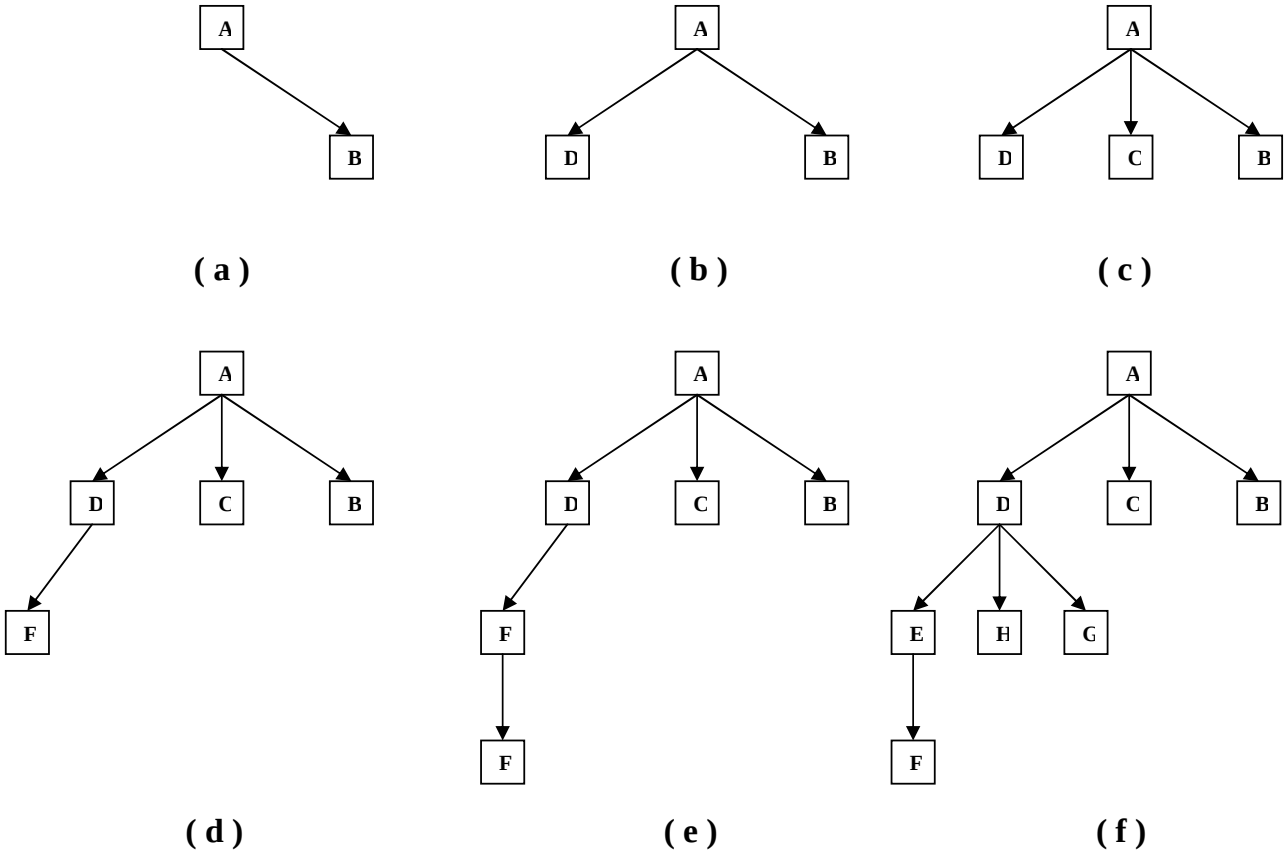

Figure 3.9: An Example Tree Building Steps

3.5 A Tree Construction Example

In Section 3.1.1, we give an example network topology and RESMO tree built on top of it. Now we explain how the tree is built step-by-step. Since this is a small topology, after expanding ring search, all of the overlay nodes have the others as neighbors except F, which has only one neighbor E. In our following description, we assume the time used in message processing at each node is 0.

(1) At time 0, sender A as an *inviter* sends **invite** to its neighbours: B, C, D, E, G and H. These neighbours respond with **thanks** when they receive invitations from A and changes their states to *awake*. Since B is the nearest neighbbour of A, **thanks** from B

arrives first at A at time 90. A takes B as its first child, informing B by sending **be-my-child** to it. A is a *parent* now and ignores the lately arriving **thanks** messages from C, D, E, G and H.

(2) At time 135, B receives **be-my-child** from A. It goes into *inviter* and sends invitations to neighbours: C, D, E, G and H (B has not received messages from them). Now, B has been part of the tree (see Figure 3.9 (a)).

(3) D, E, G, C and H receive **invite** from B at time 209, 215, 237, 240 and 251 respectively. Since they are already in *awake*, they will all set T_{wgh} timer for B and go into *weighing*.

(4) At time 399, timer T_{wgh} at D timeouts. D looks through its $rtttable$ and selects an inviter with minimum rtt – A as its parent by sending **be-my-parent** to A. It goes into *inviter* state and sends invitations to the left neighbours: C, E, G and H. After A receiving request **be-my-parent** from D, it adds D as one of its children. Now the tree is expanded to three nodes: A, B and D (see Figure 3.9 (b)).

(5) At time 459, one of the T_{wgh} timer at C timeouts. C selects a parent as D does before and becomes an *inviter*. This time also A is selected and requested by a **be-my-parent** message. This selection is different from MST in which C is linked to D instead of A. The difference is caused by the timer used in RESMO. Note that D's **invite** message will arrive at time 528, long time after the timer timeouts. C has no information about D when it must make a decision, hence cannot request D as its parent although D is the nearest neighbour. We sacrifice resource usage at this point, but the delay from source A to C is improved from 130 to 92. Here is the trade-off in RESMO. The current

tree is in Figure 3.9 (c).

(6) E, G and H are in *weighing* now and they just keep setting the weighing timers for incoming **invite** messages from D and C.

(7) At time 468, E finishes its weighing. Now he has information from A, B and D. Since D is the nearest neighbour, it sends **be-my-parent** request to D and goes into *inviter*. After D takes E as its child, the tree has five nodes now (see Figure 3.9 (d)).

(8) F is invited by E and its **thanks** is the first response arrived at E. E will sends **be-my-child** to F and becomes a *parent*, see Figure 3.9 (e).

(9) G and H at last finish their weighing task and are linked to the tree. Note that H is not a child of G as in MST, this is also due to the timer mechanism (G's invitation has not arrived when H timeouts). The complete tree is in Figure 3.9 (f).

3.6 Conclusion

We have present the design of RESMO, including control and data topology construction. In order to provide a quick view of the compromise between MST and SPT in RESMO, we gave an example tree built by RESMO as well as MST and SPT. A detailed comparison with respect to some tree efficiency metrics among these trees is also provided. The key to compromise between MST and SPT is the settings of timers during tree construction process. We elaborate on the timers setting mechanism which makes RESMO tree achieve a better performance. For easy understanding of the state transitions involved in RESMO tree building, we listed the procedures in building an example tree.

Chapter 4

Simulation and Evaluation

In order to evaluate the performance of RESMO, we simulate our protocol using NS-2 [25] network simulator. The topologies are generated with GT-ITM toolkit [26]. This chapter will first talk about the methodology and implementation matters in the simulation. Next, we give the simulation results and discussion.

4.1 Simulation Methodology

We wrote Otcl codes to simulate RESMO in NS-2. The experiment topologies were generated in GT-ITM and reformatted to *.tcl as NS-2 will accept using `sgb2ns`.

The unicast delay between two nodes in NS-2 is determined by the underlying routing algorithm which may give suboptimal route. In our overlay tree building we ignore this and assume unicast delay is the minimal delay between any two nodes when calculating round trip time.

Table 4.1: Combination of Flags and State

F1 (invited)	**F2 (be child)**	**F3 (be parent)**	**State**
0	0	0	*sleep*
1	0	0	*awake*
1	0	0	*weighing*
1	1	0	*inviter*
1	1	1	*parent*

Table 4.2: RESMO Message Format

resmo	message type	sender address	timestamp

4.1.1 Design Matters

In order to specify the current state of a node, we use three kinds of flags, namely *invited*, *be child* and *be parent*. The combinations of these flags can determine the node's current state. Table 4.1 gives a list of the flags combinations and their corresponding states. Note that *awake* and *weighing* have the same group of flags. This would not be problem since we can easily distinguish them by checking the type of pending timers.

In our simulation, we use a simple format of the messages. All of the message packets have a common format described in Table 4.2: **resmo** states that the message is from RESMO protocol; **message type** is one of the type we have introduced in Chapter 3; **sender address** contains the message sender's address; **timestamp** is the sending time.

Using simulations, we (i) determined the effects of weighting timer on the con-

structed tree, (ii) compared the performance of RESMO with five other schemes, namely NICE, Narada, MST, SPT and Unicast, using RDP, resource usage, link stress and maximum node degree as comparison metrics. Simulations of NICE and Narada are based on the *myns* simulator provided by the authors of NICE, while the rest of the schemes are simulated in ns-2.

4.1.2 Simulation Scenario

We randomly generated ten 1000-nodes topologies using one set of parameters. The results are averaged over these ten. The topologies all use transit-stub method with 0.42 as the edge connection probability within stub domains. There is no extra transit-stub or stub-stub edges. Since overlay nodes are end hosts which are located only on the edge of the network, in our simulation we randomly locate the nodes at stub domains. The sender is randomly selected from gateways. In each experiment, group size is varied between 25 and 150.

4.1.3 Performance Metrics

Through comparison we want to confirm that RESMO excels other schemes in following properties:

- *Relative Delay Penalty*
- *resource usage*
- *link stress*

- *maximum node degree*

Relative Delay Penalty (RDP), resource usage and link stress is as defined in Chapter 1. Node degree is the fan-out of the overlay node. Since lots of nodes in the tree are leaf nodes, it is not feasible to calculate the mean node degree. We will use maximum node degree to evaluate this metric. Moreover, maximum node degree will reflect the load of an overlay node, this is also one aspect we want to discover.

4.2 Simulation Results and Discussion

It is crucial to set the values of the timers in RESMO since the value can be tuned to trade-off resource usage with RDP of the resulting overlay tree. A large $T_{wgh,*}$ allows G to wait for more invitations, hence increasing the chance of selecting a parent with lower RTT. While selecting parent with lowest RTT reduces resource usage, it does not guarantee that the end-to-end delay from S to G is also small. The key to reducing end-to-end delay is to use smaller $T_{wgh,*}$. Since RESMO construct the tree in a stepwise fashion, a node that is closer to the source is more likely to become an inviter and send out invitations sooner.

4.2.1 Effects of Weighting timer

In our experiments, we set the value of T_{ivt} to $8d$, where d is the estimated one-way delay (obtained from *rtttable* between the inviter and the receiving node). $T_{wgh,*}$ is set to $k \times d$, for some constant k. To find reasonable values of k, we vary $T_{wgh,*}$ between $0.2d$ and $6d$

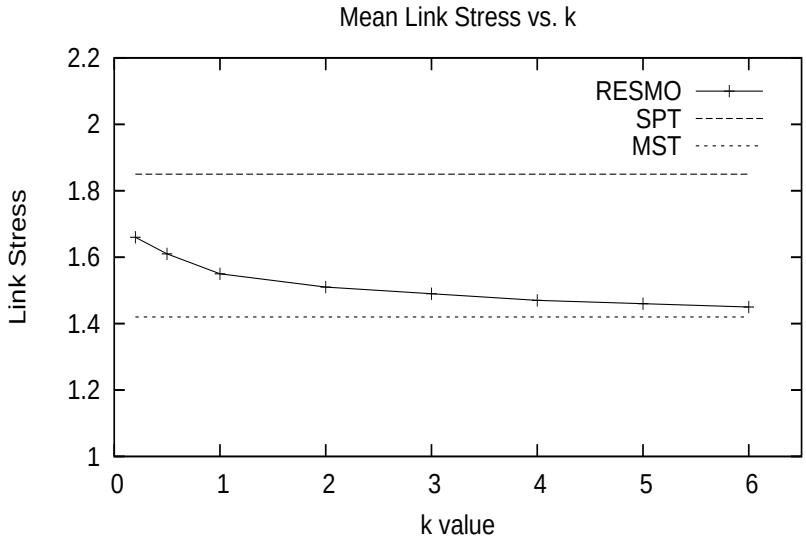

Figure 4.1: Mean Link Stress with 90% Confidence Interval vs. k.

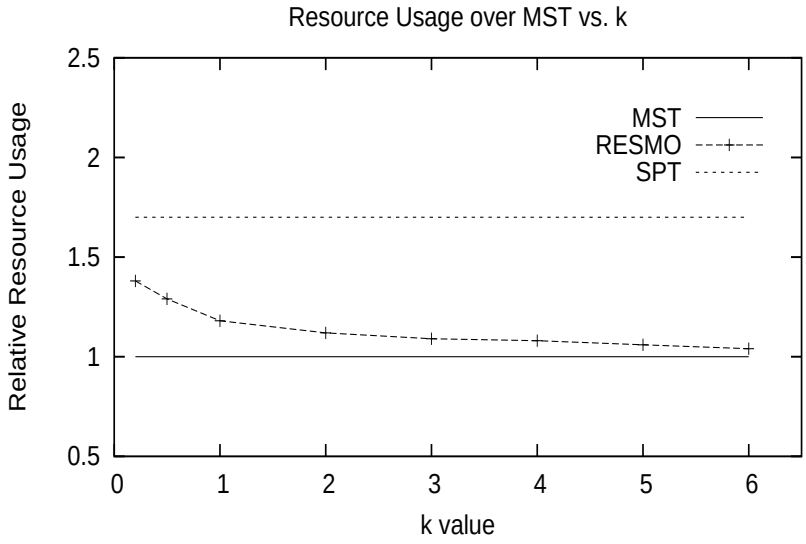

Figure 4.2: Resource Usage over MST vs. k.

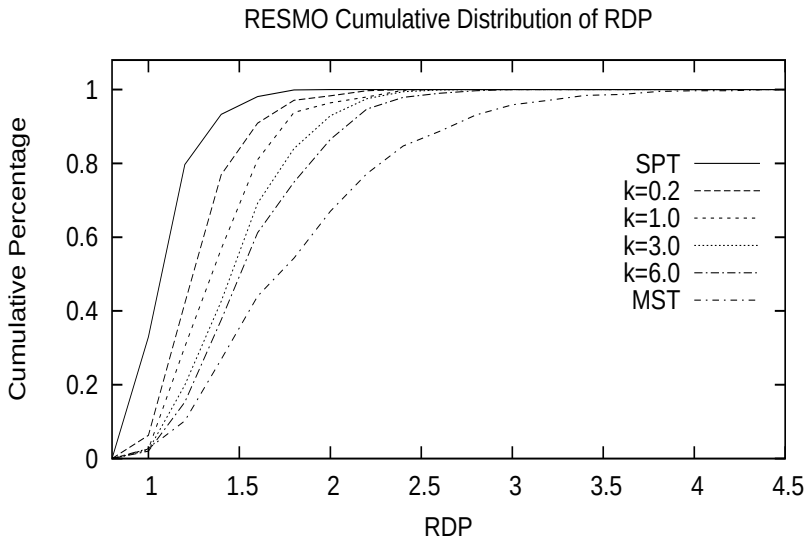

Figure 4.3: Cumulative RDP vs. k.

and plot the different performance metrics for group size 100 (we found similar curves for other group sizes). As expected, increasing k reduces average link stress (Figure 4.1), resource usage (Figure 4.2), but it also deteriorates RDP (Figure 4.3) and increases the tree construction time. Values of k between 2 and 4 shows a good compromise between the various performance metrics. We therefore set $T_{wgh,*}$ to a uniform random value between $[2d, 4d]$.

4.2.2 Comparing with Other Schemes

Now we comparing RESMO with other schemes of related work. Our findings can be summarized as follows:

(a) Link stress of RESMO is significant lower than those of NICE and Narada and is comparable to MST. Figure 4.4 shows that for each group size, RESMO has around 1.46 mean stress, which is about 50% of NICE (2.33 - 3.30) on average although they have

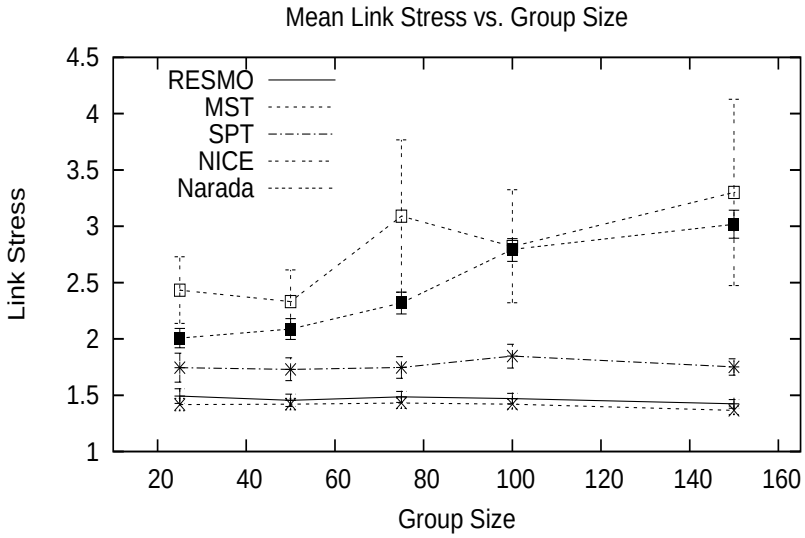

Figure 4.4: Mean Link Stress with 90% Confidence Interval.

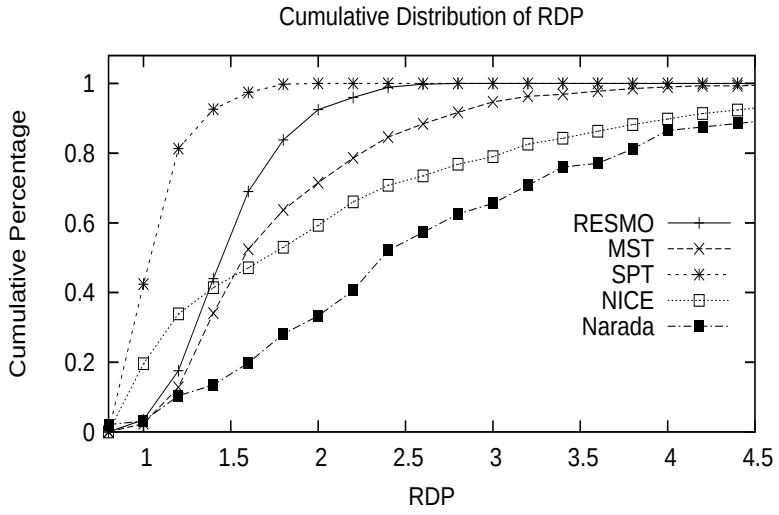

Figure 4.5: Cumulative RDP of Group Size 100.

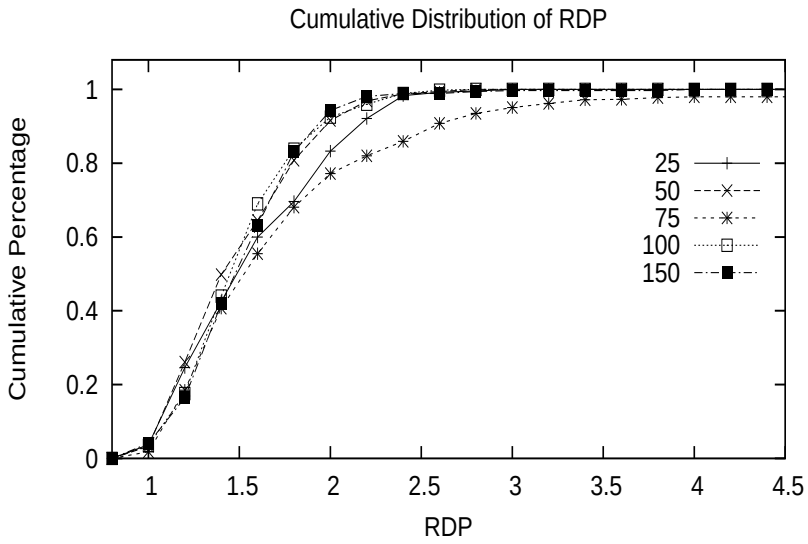

Figure 4.6: Cumulative RDP with Different Group Size.

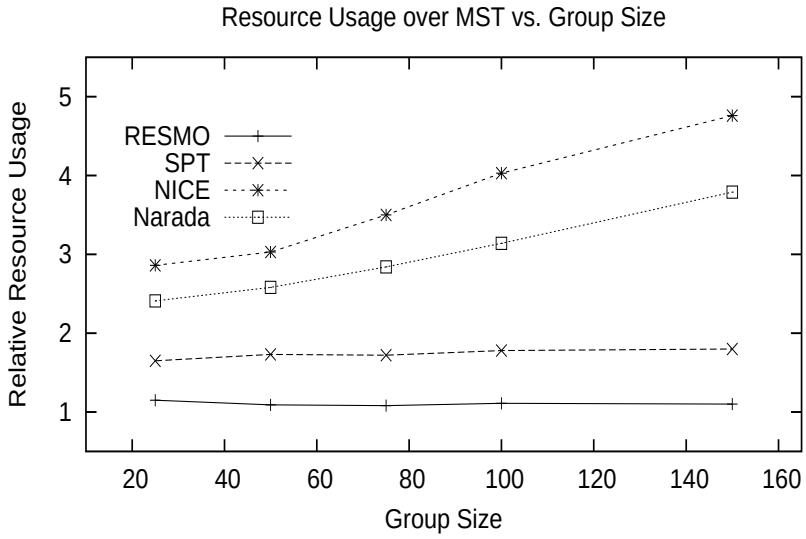

Figure 4.7: Resource Usage penalty over MST.

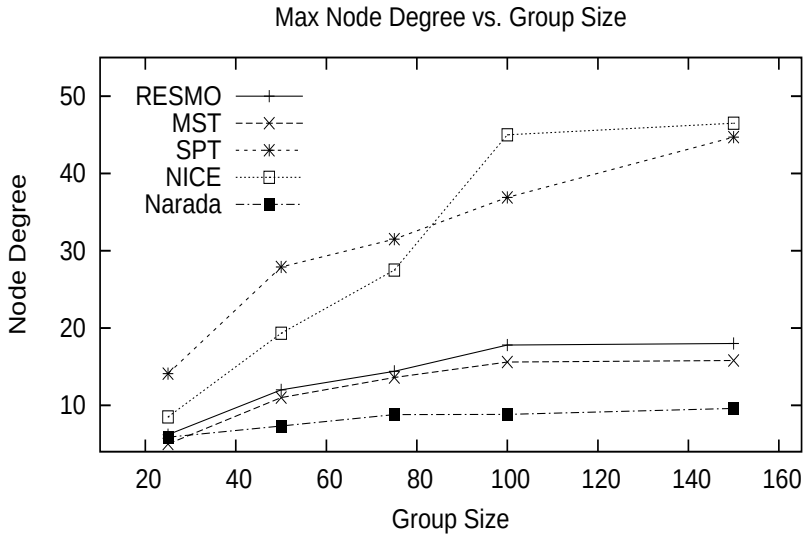

Figure 4.8: Maximum Node Degree.

similar number of active physical links. This is because RESMO is a fully distributed protocol where each node shares the forwarding task evenly; whereas NICE is a hierarchical protocol and leaders at each level will forward more packets. The large confidence interval of NICE supports our analysis. The mean link stress of RESMO is about 60% of Narada (2.01 - 3.02) on average. Narada uses DVMRP to produce lower delay, thus it uses fewer number of physical links which in turn introduces more link stress.

(b) RESMO has RDP values between those of MST and SPT. For 100 nodes topology, the 90% RDP for these trees are: 2.0 (RESMO), 2.8 (MST), and 1.4 (SPT) respectively (see Figure 4.5). We also compare the RDP with NICE and Narada: NICE has more overlay nodes with RDP less than 1.4 than other schemes, but the 90% RDP is up to 4.0 which is twice of RESMO. Furthermore, increasing the group size from 25 to 150 does not degrade RDP of RESMO (Figure 4.6).

(c) RESMO also has resource usage comparable to MST (9%-15% more, see Figure

Table 4.3: Penalty Reducing with Group Size Increasing

Group Size	**Mean States**	**States Ratio**
25	11.08	0.4432
50	21.62	0.4324
75	28.62	0.3816
100	22.26	0.2236
150	14.03	0.0935

4.7). NICE and Narada have much higher resource usage, possibly due to the existence of multiple paths in both protocols. When the group size increases, their relative resource usage increases also while the relative resource usage of RESMO remains stable.

(d) The maximum node degree of RESMO is close to MST and slightly increases with the group size (Figure 4.8). Narada has lowerest maximum node degree due to the degree constrains in its protocol.

We also evaluate other penalties such as average number of states maintained at each gateway. It is natural that when the group size increases, the topology is denser, hence the number of searched neighbors will also increases(from 11 to 28). We can see from Table 4.3 that the mean number of states decreases after the group size is over 75, this is due to smaller TTL value we applied. In order to evaluate the scalability of RESMO with number of states maintained at each member, we calculate the ratio of them over group size. The ratio is decreasing from 0.44 to 0.09. Therefore, we can conclude that in RESMO the number of states maintained at each member is scalable.

4.3 Conclusion

In this chapter, we presented the simulation of RESMO from simulation scenario generating to design implementation. We also simulate MST, SPT, NICE and Narada protocols to compare with RESMO. Our simulation results support that RESMO can achieve a comparable low resource usage to MST. The relative delay penalty is the second lowest among these five approaches. The most exciting result is that RESMO shows a good scalability when group size increases.

Chapter 5

Conclusions and Future Work

In this thesis we have presented a new distributed application-layer multicast protocol for building resource-efficient tree by approximating MST. The resulting tree is comparable to MST with respect to resource usage, link stress and node degree. It also improves end-to-end latency as well. However, there is still further room for improvement of RESMO.

5.1 Considering Bandwidth and Other Network Characteristics

In RESMO we only take link delay into consideration when building the tree. However, this routing measure is incomplete. Network condition is measured in terms of various characteristics including link delay, bandwidth, end-to-end reliability, loss rate, node stability and security. Routing in such a complicated network should consider as many characteristics as possible to obtain optimal paths.

Maximizing throughput under high load condition and achieving fairness are the

goals for many transmission application on network. Our future work should consider them also. Since media streaming applications can easily saturate link bandwidth and cause congestion, building an overlay tree with “load-balance” physical links is crucial. By load-balance, we mean: (a) involving more physical links to “fairly” share the traffic load in an effort to reduce link stress implicitly; (b) number of copies carried on each physical link should reflect the the bandwidth of the involved link so as links with higher bandwidth can share more traffic load according to their capacity.

5.2 Heterogeneity of Network

Heterogeneity is the most notable characteristic of today’s Internet. Not only the link bandwidth varies throughout the network, end hosts are also more heterogeneous with respect to CPU and storage capacity as [11] argues. For example a PDA and a high resolution desktop will require different quality of data stream. Transcoding in application-level gateways is under current research to address this problem. Since application-layer multicast is implemented in the user space, encompassing application level transcoding in application-layer multicast is straightforward. We propose a two-tier overlay approach for the future work. The first tier consists of application level media gateways [34] which form a multicast tree first. Clients can select to attach to a set of gateways according to their capacity or requirement and location. This attachment should also consider the global tree efficiency such as network resource usage. It is ideal that clients with similar capacity can be grouped together which in turn increases the efficiency of transcoding at gateways.

5.3 Transience of Overlay Nodes

RESMO, like other protocols in application-layer multicast follows the early proposed approach to reduce the efficiency penalties incurred in migrating multicast functionality from the network layer to the application layer. However, the current performance measures are incomplete since they fail to notice the differences between the overlay nodes (end hosts) and the infrastructure routing units (underlying routers) [35]. End host are less stable than routers, the transience of end hosts caused by their arbitrary joining and leaving the multicast group leads to challenge in group maintenance and tree partition recovery. Moreover, packets transmitted when the tree is being repaired will be lost. Therefore, when evaluating the tree quality, we should measure if the tree can handle such transience gracefully with minimum packets loss and time for tree partition recovery.

Bibliography

[1] J. Apostolopoulos, W. Tan, and S. Wee, “Video streaming: Concepts, algorithms, and systems,” Hewlett Packard Laboratories, Tech. Rep. HPL-2002-260, Sept. 2002.

[2] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, “The PIM architecture for wide-area multicast routing,” *IEEE/ACM Transactions on Networking*, vol. 4(2), pp. 153–162, Apr. 1996.

[3] C. Diot, B. Levine, B. Lyles, H. Kassem, and D. Balensiefe, “Deployment issues for the IP multicast service and architecture,” *IEEE Network*, vol. 14, pp. 88–98, Jan. 2000.

[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, “ALMI: An application level multicast infrastructure,” in *Proc. of 3rd Usenix Symp. on Internet Technologies and Systems*, Mar. 2001.

[5] Y. H. Chu, S. G. Rao, S. Seshan, and H. Zhang, “A case for end system multicast,” in *Proc. of ACM SIGMETRICS*, June 2000.

[6] Y. Chawathe, “Scattercast: An architecture for internet broadcast distribtuion as an infrastructure service,” Ph.D. dissertation, University of California, Berkeley, Dec. 2000.

[7] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, “Scalable application layer multicast,” in *Proc. of ACM SIGCOMM*, Aug. 2002.

[8] B. Zhang, S. Jamin, and L. Zhang, “Host multicast: A framework for delivering multicast to end users,” in *Proc. of IEEE infocom*, June 2002.

[9] J. Vogel, J. Widmer, D. Farin, M. Mauve, and W. Effelsberg, “Priority-based distribution trees for application-level multicast,” in *Proc. of NetGames 2003*, May 2003.

[10] D. Wu, W. Z. T. Hou, Y. Zhang, and J. M. Peha, “Streaming video over internet: Approaches and directions,” *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 282–300, Mar. 2001.

[11] R. van Renesse, K. Birman, A. Bozdog, D. Dumitriu, M. Singh, and W. Vogels, “Heterogeneity-aware peer-to-peer multicast,” in *Proceedings of the17th International Symposium on Distributed Computing (DISC 2003)*, Oct. 2003.

[12] S. McCanne, V. Jacobsen, and M. Vetterli, “Receiver-driven layered multicast,” in *Proc. of ACM SIGCOMM*, Aug. 1996.

[13] S. Deering, “Multicast routing in internetworks and extended lans,” in *Proc. of ACM SIGCOMM*, Aug. 1988.

[14] R. G. Gallager, P. A. Humblet, and P. M. Spira, “A distributed algorithm for minimum-weight spanning trees,” *ACM Trans. on Programming Languages and Systems*, vol. 30(12), Dec. 1983.

[15] B. Awerbuch, “Optimal distributed algorithms for mimimum weight spanning tree, counting, leader election and related problems,” *Theory of Computing*, pp. 230–240, May 1987.

[16] M. Faloutsos and M. Molle, “Optimal distributed algorithm for minimum spanning trees revisited,” in *Proc. of Principles Of Distributed Computing (PODC)*, 1995.

[17] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, “How to model an internetwork,” in *Proc. of IEEE INFOCOM*, Mar. 1996.

[18] L. L. Peterson and B. S. Davie, *Computer Networks: A System Approach*. Morgan Kaufmann, 2000.

[19] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. (1996, Mar.) Prorocol Independent Mulicast, sparse mode protocol: Specification. Internet draft.

[20] S. Sharma, D. Estrin, S. Floyd, and V. jacobson, "Scalable timers for soft state protocols," in *Proc. of IEEE INFOCOM*, Apr. 1997.

[21] D. D. Clark, "The design philosophy of the DARPA internet protocols," in *Proc. of ACM SIGCOMM*, Aug. 1988.

[22] D. R. Boggs, "Internet broadcasting," Ph.D. dissertation, Electrical Engineering Dept.,Stanford University, Jan. 1982.

[23] S. Deering, "Multicast routing in internetworks and extended lans," *ACM SIGCOMM Computer Communication Review*, 1995.

[24] OTcl - Object Tcl Extensions. [Online]. Available: http://bmrc.berkeley.edu/research/cmt/cmtdoc/otcl/

[25] The VINT Project. The Network Simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/index.html

[26] E. W. Zegura, K. Calvert, and S. Bhattacharjee. (2000) GT-ITM: Georgia Tech Internetwork Topology Models. [Online]. Available: http://www.cc.gatech.edu/projects/gtitm/

[27] D. E. Knuth, *The stanford GraphBase: A platform for combinational Computing*. Addison-Wesley, 1994.

[28] B. M. Waxman, "Routing of multipoint connections," *IEEE Jounal on Selected Areas in Communications*, vol. 6(9), pp. 1617–1622, 1988.

[29] M. Doar and I. Leslie, “How bad is naive mulicast routing,” in *Proc. of IEEE INFOCOM*, 1993, pp. 82–89.

[30] L. Mathy, R. Canonico, and D. Hutchison, “An overlay tree building control protocol,” in *3rd Int’l Wksp. Networked Group Commun.*, London, U. K., Nov. 2001.

[31] S. Khuller, B. Raghavachari, and N. Young, “Balancing minumum spanning and shortest path trees,” *Algorithmica*, vol. 14(4), pp. 305–321, 1994.

[32] D. Kostic and A. Vahdat, “Latency versus cost optimizations in hierarchical overlay networks,” Duke University, Tech. Rep., Nov. 2001.

[33] W. T. Ooi and R. van Renesse, “An adaptive protocol for locating media gateways,” in *Proceedings of the 8th ACM International Multimedia Conference*, 2000.

[34] E. Amir, S. McCanne, and H. Zhang, “An application level video gateway,” *Proc. of 3rd ACM Intl. Multimedia Conf. and Exhibitons*, pp. 255–266, Nov. 1995.

[35] M. Bawa, H. Deshpande, and H. Garcia-Molina, “Transience of peers and streaming media,” *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 107–112, 2003.