Application-layer versus Network-layer Multicast: Networking Load, Link Stress, and Distribution Delay

by

Seyedmahyar Hosseinimotlagh

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Communications

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Multicast is the task of disseminating a message from a source to a set of destinations. If supported by the network and switches, multicast can be performed at the network layer. The alternative solution is application-layer multicast (ALM) which disseminates the message through a set of unicast communications. ALM is simple to implement and does not require switches to support multicast. This, however, comes at some price including higher networking load, and slower message dissemination compared to network-layer multicast (NLM). This work analyzes some of these costs/penalties. We show that when ALM is done carefully, its networking load can be bounded to three times that of any NLM, irrespective of the network topology, the number of destination nodes, and the message size. In addition, it can be ensured that ALM does not put significant stress/pressure on any physical link in any network topology. We also analyze how slower ALM is compared to NLM. We implement an ALM algorithm in Amazon EC2 and show that the distribution delay increases slightly as the number of destination nodes increases. Deciding on what layer to use for multicasting depends on the trade-offs one is willing to make. The results presented in this work helps in making such a decision.

"If everything seems under control, you're not going fast enough."

- - Mario Andretti

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Dr. Majid Khabbazian, whose advice always steered me to the right direction, I also want to thank him for his continuous support in my research, his patience, motivation, and immense knowledge.

Second, I would like to thank the rest of my thesis committee: Dr. Masoud Ardakani and Dr. Petr Musilek, for dedicating their time and energy regarding my thesis, their insightful comments and ideas.

Then, every single lovely person who shared their passion with me and encouraged me to peruse my dreams. I am also grateful to the individuals who support me through my research. I would also like to thank my fellow labmates and friends for accepting nothing less than excellence from me. I appreciate their invaluable feedback and support, and all the fun that we have had in the last two years. I must express my very profound gratitude to my parents and to my brothers and sisters for their unfailing support in every aspect of my life.

Last, but not the least, I would like to thank all the people and the teachers who helped me to be the one who I am today. In this regards, I quote Thomas Carruthers

"A teacher is one who makes himself progressively unnecessary."

Contents

List of Figures

List of Abbreviations

List of commonly used abbreviations

ALM	Application Layer Multicast
NLM	Network Layer Multicast
VM	Virtual Machine
AWS	Amazon Web Services
ARP	Address Resolution Protocol
NTP	Network Time Protocol
CLI	Command Line Interface
CSSH	Cluster SSH
MiB	Mebibyte
GiB	Gibibyte

List of Symbols

List of Symbols

G	Underlay network graph
V	Underlay network vertices representing switches and hosts
E	Underlay physical link
T	Multicast tree
s	the single source node
D	Set of destination nodes
R	Set of switches that shape the multicast tree
L	Set of links that shapes the multicast tree
m	Multicast group size (including source s)
n	Number of the destination nodes
$\mathcal{O}$	Asymptotic notations
K	Complete overlay graph
V_T	Set of hosts (including the source) in the underlay network
E_K	Set of overlay path constructing minimum spanning tree over K
$w_{u,v}$	Weight of the edge connecting every host u, v
$\mathcal{A}$	ALM algorithm in which destination nodes receive every packet only once
g	Length of the longest shortest path between any pair of destination nodes
k	Size of a message in terms of its number of packets
h	Height of a multicast tree
t_proc	processing time of a node
t	Distribution delay/time

Chapter 1

Introduction

Data distribution is frequently used in scientific experiments, enterprise operations, parallel computing tasks, or machine learning tasks in parallel computing clusters [2]. Today's ever increasing generated or collected massive amount of data from clusters of tens of thousands machines (like in Google, Facebook, and Yahoo and etc.) highlights the need for computing cluster frameworks (such as *MapReduce* [3], *Spark* [4], *Dryad* [5], and *CIRL* [6]) to analyse data [7]. As claimed in [7], the time of data transfer in computing clusters can have a significant impact in job performance (sometimes, data transfer accounts for more than 50% of the total job completion time). Therefore, even a slight reduction in data distribution time can have a great impact on job performance.

1.1 Multicasting

In computer networks, multicasting refers to one-to-many group communication where information is distributed among a specific set of destination hosts. In other words, multicasting is the task of distributing a message from a source to a set of destinations. Multicast has many applications in computer networks. For example, in data center networks, multicast is used in publish-subscribe services for data dissemination [8], system monitoring [9], and web cache updates [10].

Multicast can be implemented either in the network layer or the application layer. In *Network Layer Multicast* (NLM), switches cooperate in packet dissemination by placing a copy of each packet on all the output ports towards the destinations. This support from switches can lead to efficient use of network resources since each physical link on the way from source to the destinations is used only once per same packet.

1.2 Thesis Motivations and Contributions

Implementation of IP Multicast, the IP-specific version of NLM, was designed without considering the commercial services in mind [11]. As stated in [11], this is perhaps one of the reasons for the slow growth of NLM in commercial environments. In addition, IP Multicast suffers from management and security issues regarding its deployment [12], severe scalability issues in terms of the number of supported multicast groups, and lack of robustness against network failures [13]. Moreover, several concerns prevent network layer multicasting to be widely used in today's networks. These concerns include group management, router migration, distributed multicast address allocation, security, and support for network management [11]. In addition, IP Multicast services are widely disabled in current Internet routers [14, 15, 16].

In cloud networks, in particular, enabling multicast services introduces a lot of complexities. In cloud networks, low-end switches which are not expected to carry much intelligence are used [17, 18]. Also, due to problems with technology, modern data centers are rarely enabled with IP Multicast [19].

For the reasons mentioned above, industries and researches have sought alternative solutions in the application level [12]. *Application layer multicast* (ALM), also referred to as overlay multicast, builds an overlay network on top of the existing physical (underlay) network. Each overlay link between two hosts in the overlay network is a virtual link abstracting a path in the physical network between the two hosts, and corresponds to a unicast session. Using the overlay links, hosts collab-

orate by forwarding the message to each other in order to deliver the message to everyone. Overlay networks have gain attention in multicasting [20, 21, 22, 23, 24], content distribution [25], and content sharing [26].

ALM overcomes some of the main shortcomings of NLM. This, however, comes at some performance penalties in distribution delay and use of networking resources. For example, in NLM, a physical link is used only once per the same packet, while in ALM a physical link can be part of multiple overlay links, hence used multiple times to transfer copies of the same packet. In experimental studies, such as the one carried out in [27], a physical link has been reported to being used 7 times in an overlay network with 100 hosts.

One of the main objectives of this work is to analyze and quantify some of these penalties, and study how they can be minimized. We analyze the networking load (i.e., the total number of times that physical links are used) of ALM and NLM. Another metric we study is link stress, which is defined for a physical link as the number of identical packets passed through the link. This is equivalent to the number of times the physical link is used in the overlay network.

A summary of our main contributions are:

1. We analyze the minimum networking load of ALM, and compare it with that of NLM. Computing the minimum networking load of NLM is NP-hard [28]. Nevertheless, we show that ALM can achieve a networking load of at most three times the minimum networking load of NLM, irrespective of network topology, the number of destination nodes, and the message size.

2. We present an ALM algorithm that, in every network topology, imposes a maximum link stress of at most three, while in the algorithm nodes forward packets to each other concurrently for fast delivery.

3. We propose an ALM algorithm with asymptotically optimal distribution delay. In analyzing the distribution delay, unlike many existing works, we consider the queueing delay at switches. Using the proposed optimal algorithm, we

show that the penalty in distribution delay when ALM is used instead of NLM is a factor of $\theta(\log n)$ in general networks, where n denotes the number of destination nodes.

4. We implement an ALM algorithm in Amazon EC2, and show how the distribution delay of the algorithm grows relatively slowly with the number of destination nodes.

1.3 System Model and Definitions

An underlay network, as depicted in Figure 1.1, consists of a set of physical machines (also referred to as nodes, or hosts) and switches that are connected via physical links. In contrast, an overlay network consists of virtual/physical machines, virtual switches in hypervisors or physical switches and, virtual links. As shown in Figure 1.1, any overlay network is built on top of an underlay network. The projection of an overlay link (virtual link) is a path in the underlay network. Communications over an overlay link is enabled by a unicast connection between the two virtual/physical machines located at the two ends of the overlay link.

1.3.1 Graph Model

We model an underlay network by a graph $G = (V, E)$, where V refers to vertices representing switches and nodes, and E is the set of edges. Each $\eta \in V$ is either a switch or a node (i.e., a host), and each $e \in E$ represents a physical link connecting two neighbor nodes/switches in the underlying network.

Links and switches traversed by a packet in NLM form a Steiner tree of G, called an underlay multicast tree, in which leaves are the destination nodes and the source while other vertices are switches. We represent this multicast tree $T = (s, D, L, R)$, where s is the source, $D \subset V$ is a set of destination nodes (receiver nodes), $R \subset V$, and $L \subset E$ are respectively the set of all switches and all links in T.

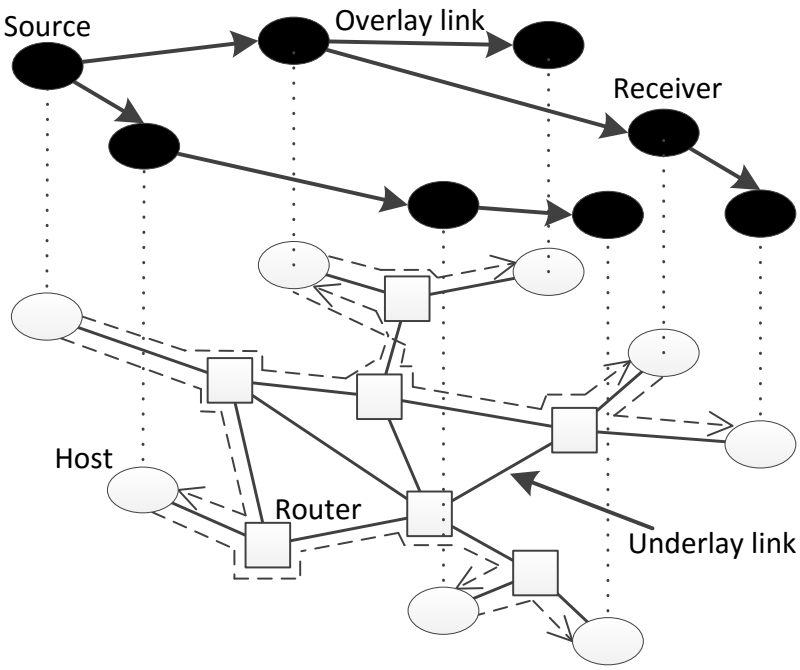

Figure 1.1: An example of overlay multicast tree on top of an underlay network.

To compare ALM with NLM, we constrain ALM to use only the links in T. This ensures that the comparison is fair since this restricts ALM to the same set of networking resources used in NLM. Note that, when the underlay network is a tree, any overlay link has a unique projection, a path in the underlay network of form $p = (n_s, r_0.r_1, ..., r_k, n_d)$, where $n_s \in \{s\} \cup D$, $n_d \in D$, $r_i \in R$, $0 \leq i \leq k, k \leq 2h$ and h is the height of T.

1.4 Metrics

Application layer multicast protocols can be evaluated by metrics such as *networking load*, *link stress*, *distribution delay*, *scalability* and *robustness*. The focus of this work is mainly on the first three metrics.

The term *networking load* (total bandwidth usage) for a multicast algorithm is defined as the total number of times that underlay links are used by the algorithm to deliver a message to all destinations. In other words, transmitting a single packet

over a single link equals to one unit of network load. The second metric, *link stress*, as defined in [29], is the number of times that identical packets are passed through the underlay link. This is equivalent to the number of times the underlay link appears in all overlay links.

The third metric is *distribution delay* which consists of three components: *communication delay, processing delay, and queueing delay*. Communication delay is the time needed for a packet to pass through a physical link. Processing delay refers to the duration from the time a host fully receives a packet, to the time right before the first bit of the packet is placed on an output port of the host. Queueing delay is the time that a packet should wait in a port's queue of a switch before its turn to be forwarded. This happens, for example, when more than one incoming packets in a switch need to be simultaneously forwarded to the same output port of the switch. In this work, we assume that switches are non-blocking, so queueing can only occur at the output ports. In general, queuing can also occur at input ports of switch if the switch fabric is not fast enough.

1.5 Brief Description of the Problem

The objective of multicast is to disseminate a message from a source node to a set of n destination nodes in a network. The multicast group refers to the set of size $m = n + 1$ consisting of the destination nodes as well as the source node. At the source, the message is divided into packets; then packets are forwarded. In ALM, a destination node can also forward a received packet. Our goal is to compare ALM and NLM performances with regard to networking load, link stress and distribution delay.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, we briefly describe the asymptotic notations and some preliminaries. We also summarize related work in network layer multicasting and application layer multicasting in the Internet and data centers. Our proposed algorithms and their analysis are presented in Chapter 3. We describe the implementation details of a basic ALM algorithm, and discuss our evaluation results in Chapter 4. Finally, we conclude in Chapter 5

Chapter 2

Background & Related Work

2.1 Asymptotic Notations

Asymptotic notations are used to evaluate the behaviour of a mathematical function as its input size increases. In other words, it shows how fast a mathematical function grows with its input size. Running time (processing time) of every algorithm can be mapped to a mathematical function which is called *time complexity* of an algorithm. Similarly, the efficiency of using storage locations by an algorithm is called *space complexity* of an algorithm. In this thesis, we focus on time complexity of algorithms. The growth of the running time is studied in terms of the input size which is also known as an algorithm's growth rate. Formally, we describe the limiting behaviour of any given mathematical functions $f(n)$ and $g(n)$ with a natural number variable n as follows:

$$f(n) \sim g(n) \quad (as \;\; n \to \infty) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

In general, an algorithm can be described with an asymptotic notation by its best case, worst case, or equivalent case performance. In this thesis, we only discuss the worst case performance of an algorithm.

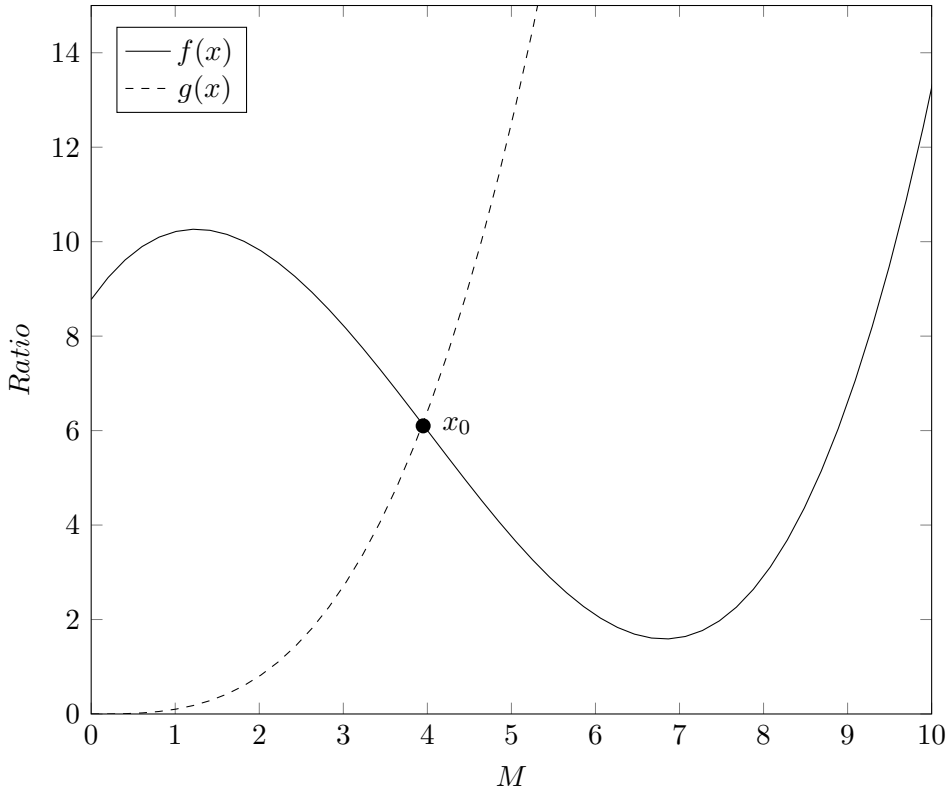

Figure 2.1: As shown in this figure, $f(x) = \mathcal{O}(g(x))$ since for all $x \geq x_0 = 4$, $g(x)$ will be greater than $f(x)$ (figure from [1]).

2.1.1 Big-O Notation

Performance of algorithms is often defined by its worst case performance. The main notation commonly used is $\mathcal{O}(.)$ which shows the upper bound on the asymptotic growth rate of a function. Different functions with the same growth rate can be bounded with a similar upper-bound function. In other words, algorithms with the same growth rate may be represented with the same $\mathcal{O}(.)$ notation.

Formally speaking, for given functions $f(n)$ and $g(n)$:

$$f(x) = \mathcal{O}(g(x)) \Longleftrightarrow \exists\ c\ and\ x_0 : \forall_{x \geq x_0} |f(x)| \leq c \times |g(x)| \tag{2.1}$$

where c is a positive real number, x_0 is a real number. The Equation 2.1 is depicted in the Figure 2.1.

2.2 Preliminaries

In this section, we briefly explain *application layer multicast* and *network layer multicast* approaches.

2.2.1 Network Layer Multicast

In one-to-many and many-to-many communications, a sender needs to send same message to multiple destinations. In network layer multicast, a sender sends a packet to a group of destinations only once. In other words, the sender hands out the packet to the network layer and the network layer delivers the packet to all specified destinations by replicating the packet at each switch's port to the destinations. This approach uses the network resources efficiently because it sends the message only once on each link on its path from the source to the destinations. However, this approach requires the network layer multicast protocol to act in a smart way to decide whether or not it is needed to replicate the packet on each output port of every switch on the path from the source to the destinations.

Although multicasting can be done efficiently at the network layer, several issues have prevented NLM from being widely used on a global Internet level. One of these issues is that IP Multicast-capable switches are needed to be installed in from the backbone level to the edge level of the network. This not only increases the deployment costs but also results in having lower speed in backbone switches compared with simple and unintelligent switches. Deployment of IP Multicast is also suffering from management and security issues [12]. In addition, IP Multicast protocols have severe scalability issues in terms of number of supported multicast groups and their lack of robustness against network failures [13]. The scalability of multicast is highly affected by the forwarding table capacity of a single switch. The lack of multicast support at network layer has led researchers and industrial entities to exploit the application layer capabilities. The difficulty of deployment and its costs along with scalability, management and security issues has led cloud providers

to sought alternative ways of multicasting at the application layer.

2.2.2 Application Layer Multicast

Application layer multicasting is an application service implemented at the end systems that provides multicast functionality of the network layer. In application layer multicast (ALM), the source and destination nodes collaborate, at the application layer, to disseminate the message; any node that receives a packet can forward it to another node. Clearly, this way, a physical link may be used multiple times for transferring the same packet, as the physical link may be on the way between multiple different pair of nodes. As mentioned earlier, in network-layer multicast, every physical link is used at most once per packet. This is one advantage of implementing the multicast at the network layer rather than the application layer A simple ALM algorithm commonly used in practice is the one in which the source establishes connections to all the destination nodes and sends/unicasts the message to every destination node either sequentially or simultaneously. Some of the advantages of ALM over NLM are 1) easier and immediate ability of ALM to be deployed on a network, 2) easier maintainability of the algorithm and 3) providing the ability to write an application specific algorithm [12].

2.2.3 Cloud Network Structure

Fat-trees

A common cloud network structure is fat-tree. Fat-trees were originally proposed by Leiserson in [30], then Greenberg and Leiserson presented a fine structure of fat-trees in [31], called *butterfly fat-tree* in Leiserson's thesis [32].

Referring to [33], a fat-tree has two types of structure a *coarse structure* (Figure 2.2) and a *fine structure* (Figure 2.3), which are both explained next.

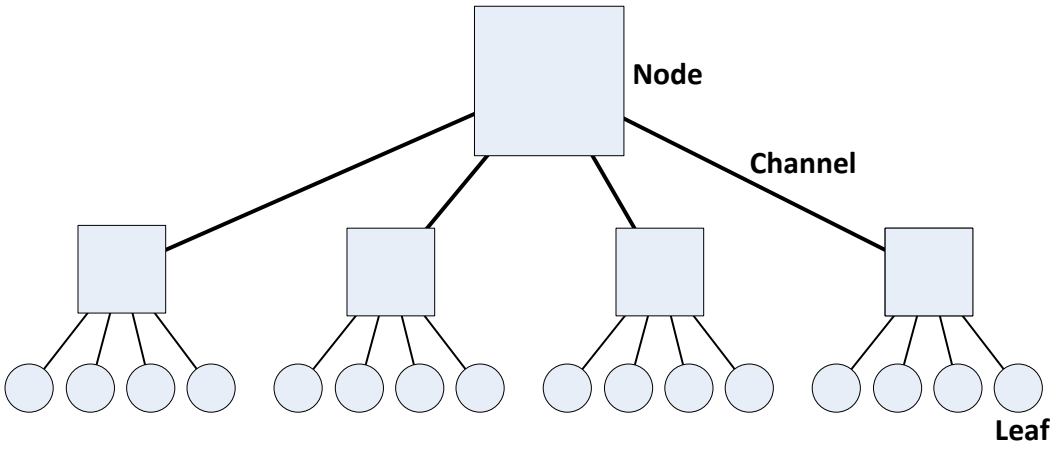

Figure 2.2: An example of a coarse structure of 4-ary fat-tree.

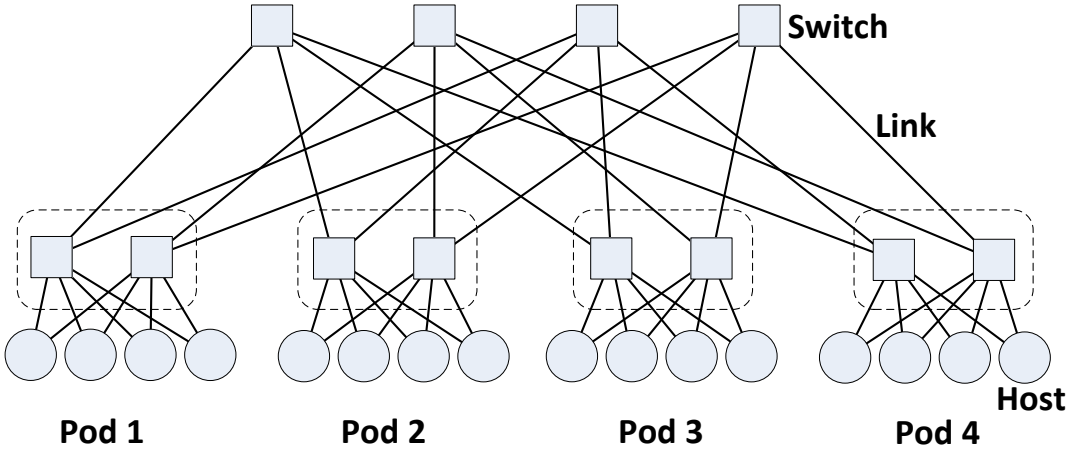

Figure 2.3: A fine structure corresponding to the coarse structure shown in Figure 2.2.

Fine Structure of Butterfly Fat-trees

As shown in Figure 2.2 and Figure 2.3 a butterfly fat-tree has a tree-like coarse structure, but it contains cycles in its fine structure.

In the fine structure, each link in the tree connects two devices that are in successive levels. The top switches are connected to exactly one switch of every pod. Besides, no link connects any two devices from different pods. There is only a sin-

gle link between every parent and its child, and no link between any node at the same level of the tree. In general, to construct a butterfly fat-tree, each switch can have different number of parents and children [33]. However, it is unusual to have a switch with more parents than children since children cannot fully utilize all links to parents.

Every path between any two hosts first proceeds upwards in a fat tree, then goes down to the destination host. On the way to the top, any switch can be chosen, which results in having multiple path up in a fat-tree. However, there is only a single unique path on the way down to a particular destination host. Having multiple paths between every two hosts is one of the important features of fat-trees. This characteristic enables fat-trees to not only perform load balancing among the fat-tree links but also be more fault tolerant in the case of link failure.

Coarse Structure of Butterfly Fat-trees

Each leaf in the coarse structure is a processor, server, or other device in the fine structure. Every internal node (i.e, switches) in the coarse structure maps to a set of switches in the fine structure. Every channel in the coarse structure maps to a set of links in the fine structure, that each of them connects a port of the parent switch to a port of the child switch. The bandwidth allocated to each channel connecting a pair of nodes (in the coarse structure) is sum of the bandwidth of the corresponding links (in the fine structure) between the equivalent nodes. The number of switches in each coarse node, the number of ports on each switch and, the away that the ports connect to each other are all determined by the designer of the network. Two important but conflicting design goals in multicasting are $1)$ minimizing the delay experienced by a single destination $2)$ minimizing the total time needed to deliver the message to all destinations. [12]"

2.2.4 Multicast in Cloud Networks

There are many applications in cloud environments that require broadcasting or multicasting messages. Some of these applications are:

- User applications drive their host virtual machine (VM) to send broadcast or multicast messages into the cloud network. These packets can be generated to satisfy the needs of distributed databases, file sharing services, audio/video streaming, or audio/video conferencing.

- A VM sends out a broadcast or multicast message by flooding it through the network to get information about how to map the destination's logical IP address to the corresponding physical IP address.

- A VM sends a broadcast or multicast message into the network to support standard protocols, e.g. Address Resolution Protocol (ARP). For instance, some data center virtualization standards such as VXLAN [34] and NVGRE [35] convert broadcasting in the virtualized subnet into multicasting in the physical network [13].

Currently, the common approach to multicast in cloud environment is the one-to-all method, in which the source establishes connections to all the destination nodes and sends/unicasts the message to them either one by one or simultaneously. This approach does not use the upload capacity of the destination nodes to reduce the distribution delay. Also, the approach puts lots of pressure on some physical links such as the one connected to the network interfacer of the source node (several copies of the message have to go through that link). However, the above approach is attractive because of its simplicity, as there is only one node (the source node) which sends packets (destination nodes do not participate in sending/forwarding packets).

2.3 Related Work

Whether multicast related services should be implemented in the network layer or the application layer was revisited by Chu. et al. in [27]. The authors compare the performance of their ALM solution with that of IP Multicast using both simulations and experiments on the Internet. Their results indicate that ALM can achieve low link stress, in small to medium sized multicast groups. The maximum link stress in their ALM algorithm called Narada was reported 5 for multicast group size of 16, and 7 for the multicast group size of 100. Our results show that, when ALM is done carefully, the maximum link stress can always be capped at three, no matter how large the the multicast group is. This implies that ALM is scalable with regards to link stress.

In this thesis, we focus on routing algorithm in ALM, which is about what node should forward the message to what other node. In other words, routing algorithms are about building an overlay network. Several ALM protocols are classified in [12] based on their routing algorithms. For example, TAG [23] builds an overlay network in a distributed manner by using a path overlap information among members.

Chuang and Sirbu [36] were among the first who consider networking load as a metric to compare NLM with unicast services. They found that the ratio between the networking load of NLM and the average length of unicast routing path has a power-law of $m^{0.8}$, where m is the size of multicast group. They validated their evaluation on both real and generated network topologies. This ratio was argued to be closer to $m^{0.7}$ than $m^{0.8}$ in a later work [37]. In our work, we compare the networking load of NLM with the networking load of ALM instead of the average networking load of unicast (i.e., the average length of unicast path). If we take the average networking load of unicast in the whole network as an estimate to the average networking load of unicast in the overlay tree used in ALM, we get that the ratio between the networking load of ALM and that of NLM grows as $(m-1)/m^{0.7} \approx m^{0.3}$ where m is the size of multicat group. It is because the overlay tree corresponding to ALM has $m-1$

overlay links, each corresponding to a unicast path. In this work, we show that, when ALM is done carefully, the ratio between the networking loads of ALM and NLM is at most three, irrespective of the multicast group size.

Radoslavov et al. [38] considered two metrics of link stress and, tree stretch[1] in overlay multicast to that of network layer multicast in evaluating the impact of different network topology on two heuristic overlay multicast method. Using both Internet and simulation experiments, Chu et al. [27] claimed that ALM can achieve low performance penalty in comparison with IP Multicast in terms of link stress and tree stretch.

With regard to distribution delay, Mokhtarian and Jacobsen [39] prove that minimizing average or maximum delays are both NP-hard problems, and, further, cannot be approximated with any reasonable approximation ratio in polynomial time. In [40], Brosh, Levi and Shavitt present approximation algorithms with approximation factor of $\mathcal{O}(\log n)$ to the problem of minimizing the maximum delay. Our work on the distribution delay differs from [40] in two ways. i) unlike [40] and the majority of related papers such [39, 41], we consider queueing delays in computing the distribution delay of multicast. When queueing delay is considered, the simple model used in [40] in which fixed delay costs are assigned to overlay links is not applicable, because a switch may be used in more than one overlay link. In other words, two packets transmitted over two overlay links that share a switch may get queued at the switch, causing extra delay on one of those overlay links. ii) We compare the distribution delay of our proposed ALM algorithm with that of NLM, and show that they are at most a factor of $\mathcal{O}(\log n)$ away. Similar factor is proven in [40]. However, the distribution delays of their algorithms are compared to the minimum distribution delay of ALM, which is higher than that of NLM. Objective of authors in [41] is finding minimum-delay multicast distribution tree while considering the nodal processing delay. They also introduce a delay measure called reception-and-processing

[1]Tree stretch is defined as the ratio of the number of the links in the overlay multicast tree to that of underlay multicast tree.

delay. Overlay multicast algorithm usually aim to minimize the link-by-link distance among the receiver nodes [18, 27, 42, 43, 44], however, authors in [39] consider the nodal processing delay as well.

Finally, there has been some recent works on implementing ALM in cloud infrastructure [17, 19]. As pointed out in [17], solving multicast problem in cloud networks is easier than in the Internet due to the special architecture of cloud networks. For example, in a cloud network with fat-tree topology, the shortest path distance between two nodes can be computed using the IP addresses of the nodes [17]. For instance, the shortest distance between two nodes with the IPs $10.x_1.y_1.z_1$ and $10.x_2.y_2.z_2$ is six if $x_1 \neq x_2$, and the shortest distance is four if $x_1 = x_2$ and $y_1 \neq y_2$. This property can be very useful in some multicast algorithms such as the ALM algorithm with minimum networking load presented later in this work.

Chapter 3

Analytical Results

In this section, we compare the performance of ALM with NLM in terms of networking load, link stress and distribution delay. Majority of our analytical results apply to general network topologies, while some others are tuned for data center networks. In Chapter 4, we implement an algorithm in Amazon EC2 to evaluate distribution delay of ALM.

3.1 Networking Load

At first glance, it may seem that ALM imposes significantly higher networking load than NLM when the size of multicast group or the message is large. In this section, we prove that this is not the case if ALM is performed wisely. In fact, we show that the networking load of ALM can be as low as (up to a constant factor) NLM.

Following, we start by highlighting that, unlike NLM, finding an ALM algorithm with minimum networking load is straightforward. Then, we answer two interesting and challenging questions: i) how far is the minimum networking load of ALM from that of NLM? ii) is there any ALM algorithm with maximum link stress of constant with respect to the size of multicast group?

3.1.1 Minimizing Networking Load

Finding an NLM algorithm with minimum networking load is NP-hard since it is equivalent to the minimum Steiner tree problem [28]. In contrasts, computing the minimum networking load of ALM, and finding an ALM algorithm that achieves it is relatively straightforward. What is challenging is how large that number is compared to the minimum networking load of NLM.

Without loss of generality, we assume that the message is a single packet. Any ALM algorithm can then be translated into a weighted overlay graph with m vertices, each vertex representing a node in the multicast group, and an edge between two vertices u, and v if and only if the packet is transferred between u and v in the ALM algorithm.

The following two properties hold for any ALM algorithm with minimum networking load: 1) the overlay graph corresponding to the ALM algorithm must be a tree. It is because, otherwise, a node will receive the packet more than once. In other words, one of the overlay links can be safely removed. 2) The networking load of the ALM algorithm is equal to the total weight of its corresponding overlay graph, as each overlay link is used exactly once. Also, observe that any overlay spanning tree can represent an ALM algorithm. By the above two observations, the overlay graph corresponding to the optimal ALM algorithm must be a minimum spanning tree of the complete overlay graph $K = (V_T, E_K, w)$ where, $V_T = \{s\} \cup D$ and $E_K \subseteq P$ and the weight of each edge connecting every pair u, $v \in V_T$ is set to $w_{u,v} = dist(u, v)$. Therefore, an ALM algorithm with minimum networking load, and its networking load can be computed in polynomial time. In general, the link stress of the above algorithm could grow with the size of the multicast group. For example, suppose that the underlay network is a star topology with a switch at the center, and every node in the multicast groups directly connected to the switch. An ALM algorithm that achieves the minimum networking load is the one in which the source unicasts the packet to every n destination nodes. This simple ALM algorithm, however, puts the maximum possible stress of n on the physical link connecting the

source to the switch. So an interesting question is whether there is an ALM algorithm with constant link stress. Another important question is how far the minimum networking load of ALM is from that of NLM. We answer both those questions next.

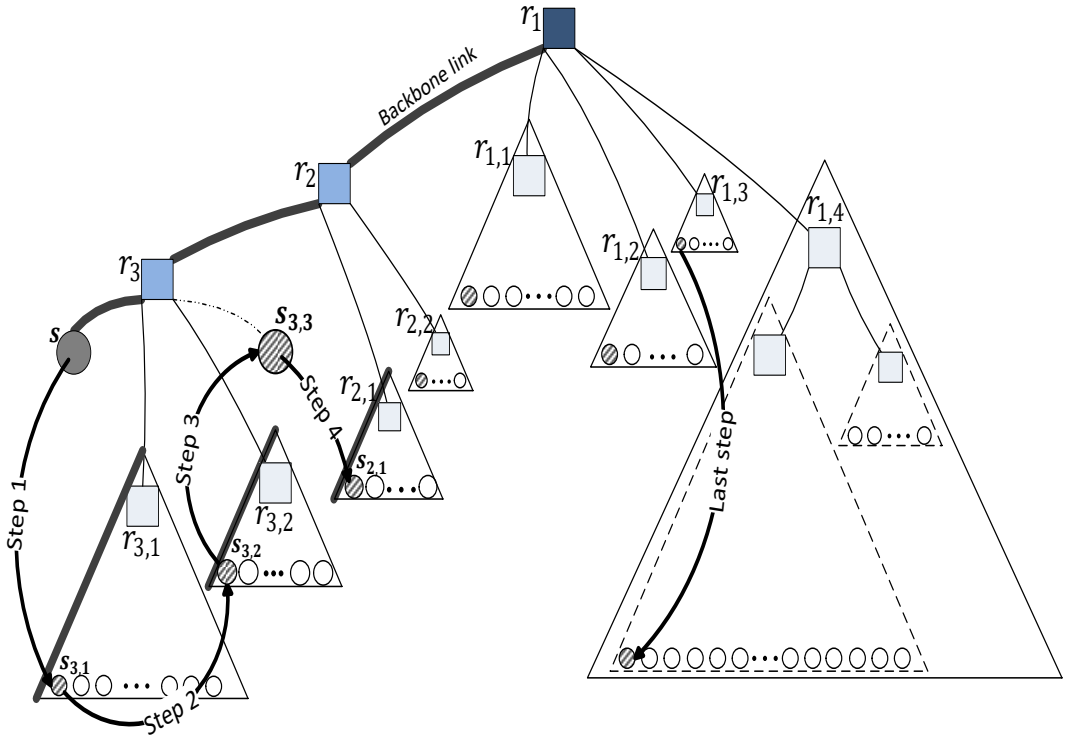

Figure 3.1: Illustrating the recursive algorithm proposed in Section 3.2.1.

3.2 Link Stress

3.2.1 Bounded Link Stress

The algorithm introduced in this section aims for limiting the number of times that each underlay link is used in order to lower the pressure borne by every link. This also somewhat balances the overall traffic load on all the available links in L.

Let us set an arbitrary switch $r_1 \in R$ as the root of the NLM tree T. Our ALM algorithm is a recursive algorithm, and works as follows. Let us call the path between

the source and the root of a tree as the backbone of the tree. Let $r_m, r_{m-1}, \ldots, r_1$ denote the sequence of switches on the backbone of T from the source to the root r_1. As shown in Figure 3.1, each switch $r_{i,j}$ adjacent to a switch r_i on the backbone of T is a root of a subtree. All subtrees whose roots are adjacent to the same switch on the backbone of T are called sibling subtrees. In the first iteration of the algorithm, the source sends the message to an arbitrary node $s_{m,1}$ in a sibling subtree (i.e, a subtree connected to switch r_m), which in turn forwards the message to another arbitrary node in the next sibling subtree $s_{m,2}$. This continues until one node in every sibling subtree receives the message (Steps 1, 2, and 3 in Figure 3.1). The last node that receives the message forwards the message to a node $s_{m-1,1}$ in a subtree connected to r_{m-1}, the next switch on the backbone (Step 4 in Figure 3.1). This continues, until a node in every subtree receives the message. The algorithm is then run recursively on each subtree with the node $s_{i,j}$ that has the message as the source and the switch $r_{i,j}$ as the root.

Proposition 1. *The proposed algorithm has the maximum link stress of three.*

Proof. A link on the backbone of T is used exactly once. For example, the link (r_3, r_2) on the backbone of the tree T in Figure 3.1 is used only once in Step 4 when $s_{3,3}$ sends the message to $s_{2,1}$. A link on the backbone of a subtree is used three times. For example, in Figure 3.1, consider a link on the backbone of the subtree rooted at the switch $r_{3,1}$, that is a link on the path from $s_{3,1}$ to $r_{3,1}$. The link is used once when the source s sends the packet to $s_{3,1}$. Next time it is used when $s_{3,1}$ forwards the packet to $s_{3,2}$, the source of the adjacent subtree. The link is used one last time when the algorithm is recursively called on the subtree rooted at $r_{3,1}$. It is because the link is on the backbone of the subtree. Links that are not on the backbone of any subtree are those that connect a subtree to the backbone of the higher level subtree (e.g., link $(r_{3,1}, r_3)$ in Figure 3.1). Those links are used at most two times; once when the message is forwarded to the source of the subtree, and once when the source of the subtree forwards the message to the node in the next subtree. □

The following corollary follows directly from Proposition 1.

Corollary 1. *The networking load of the proposed algorithm is at most three times that of the NLM algorithm corresponding to the tree T.*

Now by Corollary 2, we get that the minimum networking load of ALM must be within a factor of three of that of NLM.

Corollary 2. *The minimum networking load of ALM is at most three times of the minimum networking load of NLM.*

Proof. Finding an NLM algorithm with minimum networking load is NP-hard. Nevertheless, by our recursive ALM algorithm, we know that for any NLM algorithm (including one with minimum networking load), there is an ALM algorithm whose networking load is at most three times higher than that of the NLM algorithm. The optimal MST-based ALM algorithm presented earlier has the minimum networking load among all ALM algorithms, hence its networking load must be within a factor of three of the networking load of any NLM algorithm. □

Remark 1. *In defining networking load, we assumed that transmitting a single packet over a single underlay (physical) link contributes one unit to the networking load. In general, to each link we can assign a non-negative weight indicating how much transmission of a single packet on that link will add to the networking load. Our results still holds under this generalized assumption. In particular, the MST-based ALM algorithm is still optimal (i.e., has minimum networking load), and its networking load can be similarly shown to be at most three times that of any NLM algorithm.*

Remark 2. *As stated in the proof of Proposition 1, non-backbone links have a stress of at most two. Also, links on the backbone of the tree T have stress of one. Therefore, the average stress in our recursive ALM algorithm is strictly less than three. As a result, the networking load of the MST-based ALM algorithm is indeed strictly less than three times that of any NLM algorithm. This number is at most 2.5 in multicast*

trees where the number of backbone links in all subtrees is not more than the number of non-backbone links. This holds if every switch on the backbone is connected to at least one subtree, as in this case every backbone link can be corresponded to a distinct non-backbone link. For example, in Figure 3.1, each link on the backbone of T can be corresponded to a distinct non-backbone link. To get the factor of 2.5, this must hold for all the subtrees too.

The ALM algorithms described so far require some knowledge about the underlay network topology. For example, the MST-based ALM algorithm needs to know the length of the shortest path between any pair of nodes in the multicast group, where the multicast group refers to the set of nodes consisting of the source and the hosts that are supposed to receive the message. Even in the absence of any information about the network topology, under certain conditions, the ratio of the networking load of ALM to the networking load of NLM can be bounded as stated in the next remark.

Remark 3. *Let g, k and n respectively denote the length of the longest shortest path between any pair of nodes in the multicast group, the size of message in packets, and the number of destination nodes. Let $\mathcal{A}$ be an ALM algorithm that guarantees every node receives every packet only once. Then, the networking load of $\mathcal{A}$ is at most g times that of any NLM algorithm. This is because the networking load of any NLM is at least $k \cdot n$, as each of the n nodes in the multicast group must receive every packet at least once (each of those packets imposes at least one unit of load at the link connected to the receiving host). Also, the networking load of $\mathcal{A}$ is at most $g \cdot k \cdot n$, as each node receives every packet exactly once, and each of those packets impose at most g units of load on the network.*

Remark 3 comes in handy for networks with small diameters. For example, today's commodity data center's network architectures are trees (or tree-like) with either two or three levels of switches [45], hence they have diameters of either four or six. Therefore, by Remark 3, in such networks, we can assure that the networking

load of ALM is bounded by a constant number (irrespective of the size of message, and the group size) if the ALM algorithm used does not send multiple copies of the same packet to a node. This allows us to focus mainly on distribution delay in Chapter 4 where we design a fast ALM algorithm which we implement in Amazon EC2.

3.3 Distribution Delay

Here, we study two different scenarios based on the size of message. In the first scenario, the size of message is small (a single packet), and in the second scenario which is described in Section 4.1, the size of message in packets is larger than the size of the multicast group. The second scenario applies in cases where a large file needs to be distributed, while the former holds for multicasting, say, control packets. As our theoretical results indicate, multicasting a single packet can be considerably slower in application layer than in network layer. However, the distribution overheads of ALM become negligible when the message is large. In fact, our implementation on Amazaon EC2 shows that multicasting a large file to a large group can be done in a reasonable amount of time in comparison with unicasting the file to a single node.

3.3.1 Single Packet Multicast

Let T be the multicast tree used in NLM, and h denote its height. Suppose a packet transmission on every underlay link takes one unit of time. For now let us ignore node's processing time. Then, NLM requires at least $h+1$ units of time to complete, as the distance of the farthest destination node to the source is at least $h + 1$.

We argue that ALM can be done in $h \cdot \mathcal{O}(\log n)$ units of time, where n denotes the number of destination nodes. A simple argument is that the number of nodes that have the packet can be doubled in a time phase of length $2h$ units. This is because in a phase, each node that has received the packet can forward it to any other node in the multicast group. This argument neglects the possibility that packets forwarded in

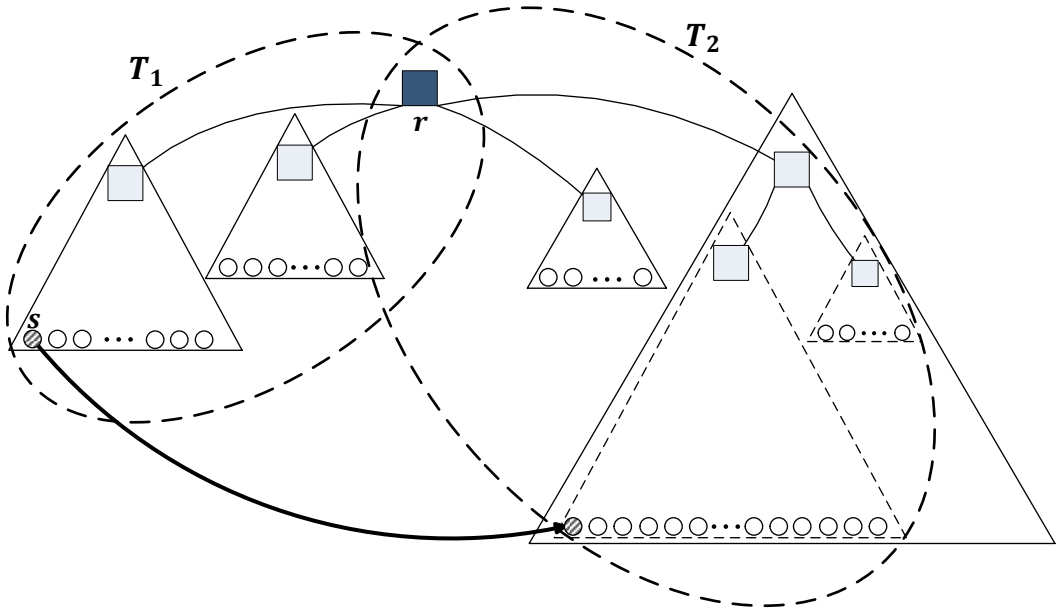

Figure 3.2: An example tree to illustrate how the queueing-aware algorithm works.

a phase can be simultaneously placed on the same output port of a switch, hence get queued. This potential queueing, however, can be avoided due to our next recursive ALM algorithm, called the *queueing-aware algorithm*.

The queueing-aware algorithm starts with selecting a switch as the root of T. We call a switch pivot if each subtree connected to the switch contains at most half of the nodes in the multicast group. Suppose that the multicast tree T has a pivot r. In this case, the queueing-aware algorithm selects r as the root of tree T. Figure 3.2 shows the switch r and the subtrees connected to r, each placed in a triangle. By Lemma 1, we can divide the subtrees connected to r into two groups g_1, and g_2, such that the number of nodes in each group is at most two third of the total number of nodes in the multicast group.

Lemma 1. *Let $\mathcal{S} = \{a_1, a_2, \ldots, a_l\}$ be a set of l positive numbers. Suppose $\sum_{i=1}^{l} a_i = A$, and $a_i \leq \frac{A}{2}$, for every $1 \leq i \leq l$. Then, S can be partitioned into two non-empty subsets S_1, and S_2 such that*

$$\sum_{a \in S_1} a \leq \sum_{a \in S_2} a \leq \frac{2A}{3}.$$

Proof. See Appendix 6.3 □

Let T_1 be the tree obtained by removing all the subtrees in g_2 from T (see Figure 3.2). Similarly, let T_2 be the part of T that includes the root r and all the subtrees in g_2 (i.e, T_2 is the tree obtained from T by removing all the subtrees in g_1). Without loss of generality, suppose the source is located in T_1. In the the queueing-aware algorithm, the source sends the packet to an arbitrary node in T_2 (node s_2 in Figure 3.2). This takes at most $2h$ units of time. Then, the queueing-aware algorithm is simultaneously called on both T_1 and T_2. Note that T_1 and T_2 share the switch r. However, this does not causes the packets transmitted in the two runs of the algorithm on T_1 and T_2 to block each other, as those packets are never forwarded to the same output port of switch r. Therefore, assuming that a pivot can be found, we get

$$T(m) \leq 2h + T(2m/3), \tag{3.1}$$

where $T(m)$ denotes the time complexity of the queueing-aware algorithm, and m denotes the number of nodes in the multicast group (i.e. $m = n + 1$, where n is the number of destination nodes). If T does not have a pivot, then by Lemma 2 it must have a switch r that directly connects to more than $m/2$ nodes.

Lemma 2. *Let T be a multicast tree that does not contain any pivot. Then, T contains a switch r with more than $m/2$ neighbouring nodes, where m denotes the number of nodes (i.e., leaves) of the multicast tree.*

Proof. See Appendix 6.4 □

Let T_1 be the tree consisting the switch r and all the nodes directly connected to it, and T_2 be the tree obtained by removing nodes in T_1 from T. In the queueing-aware algorithm, if the source is located in T_1, it forwards the packet to an arbitrary node in tree T_2; otherwise, it forwards the packet to an arbitrary node in T_1. This takes at most $2h$ units of time. The queueing-aware algorithm is then run simultaneously on T_1, and T_2. On tree T_1, the algorithm simply doubles the number of nodes

that have the packet every two units of time. Therefore, on T_1, the algorithm needs at most $2\lceil \log m \rceil$ units of time to finish. Hence, if a pivot is not found, we get

$$T(m) \leq 2h + \max\{2\lceil \log m \rceil + T(m/2)\}. \tag{3.2}$$

Combining (3.1) and (3.2) yields

$$\begin{aligned} T(m) &\leq \max\{2h + T(2m/3), \\ &\qquad 2h + \max\{2\lceil \log m \rceil + T(m/2)\}\} \\ &= 2h + \max\{2\lceil \log m \rceil + T(2m/3)\}, \end{aligned}$$

solving which we get $T(m) \in \mathcal{O}(h \log m)$. Consequently, ALM is at most $\mathcal{O}(\log n)$ times slower than NLM, and this is tight as it holds in star topologies.

The above bound of $\mathcal{O}(h \log m)$ or equivalently $\mathcal{O}(h \log n)$ applies to general multicast trees. In some trees this bound can be improved. For example, as stated in the next proposition, in balanced binary trees, the time complexity of ALM is $\mathcal{O}(h \log n / \log \log n)$ instead of $\mathcal{O}(h \log n)$.

Proposition 2. *In a full binary tree with height h, ALM can be performed in $\mathcal{O}(h. \log n / \log \log n)$ units of time.*

Proof. See Appendix 6.1. □

If we assume that the complete binary tree has oversubscription of 1:1, then , as stated in the next proposition, ALM can be done in $\mathcal{O}(h \log \log n)$ instead of $\mathcal{O}(h \log n)$. Referring to [45], oversubscription is "the ratio of the worst-case achievable aggregate bandwidth among the end hosts to the total bisection bandwidth of a particular communication topology".

Proposition 3. *Consider a balanced binary tree with oversubscription 1:1. Then ALM can be done in time $\mathcal{O}(h \log \log n)$, where h denotes the height of the tree.*

Proof. See Appendix 6.2. □

Remark 4. *Considering the nodes processing time, denoted by t_{proc}, the distribution delays of NLM will be at least $h + 1 + t_{proc}$. The distribution delay of the proposed queueing-aware ALM, in this case, is upper bounded by $O(h \log n) + (n + 1)t_{proc}$ instead of $O(h \log n)$.*

As stated in [39], the processing time t_{pros} is typically very small compared to other delay components defined in the distribution delay.

Chapter 4

Implementation and Performance Evaluation

This section describes the details of the implementation of a tuned version of BitTorrent (described in Section 4.1) in Amazon EC2, and the evaluation of its distribution delay. In practice, distribution delay of an ALM algorithm can be affected by factors that may not be easily captured in a theoretical model. Some of these factors are:

1. policies and routing algorithms set by the data center provides,

2. packet queuing caused by other applications running in the data center,

3. the presence of other virtual machines in the same physical machines that our virtual machines are placed in (each virtual machine uses the share resources in a different way that cause different nodal delays),

4. traffic congestion in physical links,

5. different hop distance of virtual machines from each other and etc.

4.1 A simple ALM Algorithm

As discussed earlier, single packet multicast can be a factor of $O(\log n)$ slower in application layer than in network layer. Our implementation, however, shows that ALM can be fairly fast when the message is large (e.g., when the message is a large file). The short answer to why ALM is fast when the message is large is efficient pipelining of packets. The algorithm we use for the implementation is a tuned version of BitTorrent [46], in which the number of connections of each peer (node) is set to the maximum (i.e., it is set to n in the source node, and $n-1$ in the destination nodes), and the message is divided into n equal chunks. The algorithm starts by the source connecting to all n destination nodes, and sending packets from a distinct chunk to everyone in parallel. Upon receiving the first packet of the chunk from the source, a destination node opens connections to all the remaining destination nodes, and starts forwarding the packets received from the source to all those nodes. The above algorithm does not require information about the network topology. However, by Remark 3, the algorithm's networking load is at most six times that of any NLM, as the algorithm ensures that only one copy of a packet is sent to any destination node.

4.2 A Simple Analysis of the ALM Algorithm

Let $size_f$ denote the size of the file at the source node. Let us consider a simple model in which sending the file from any node to any other node takes t seconds. Also, suppose that transferring a file of size $size_f/n$ from one node to any other node can be done in t/n seconds. In a real environment, the above assumptions may not be accurate as different links may have different capacities and different congestions. The nodal delay can vary from one node to another node. Also, the hop distance between a pair of nodes may be different than that of another pair.

By the above assumptions, sending a chunk of size $size_f/n$ to each destination node takes t/n seconds, as shown in Figure 4.1. Therefore, after t seconds, every

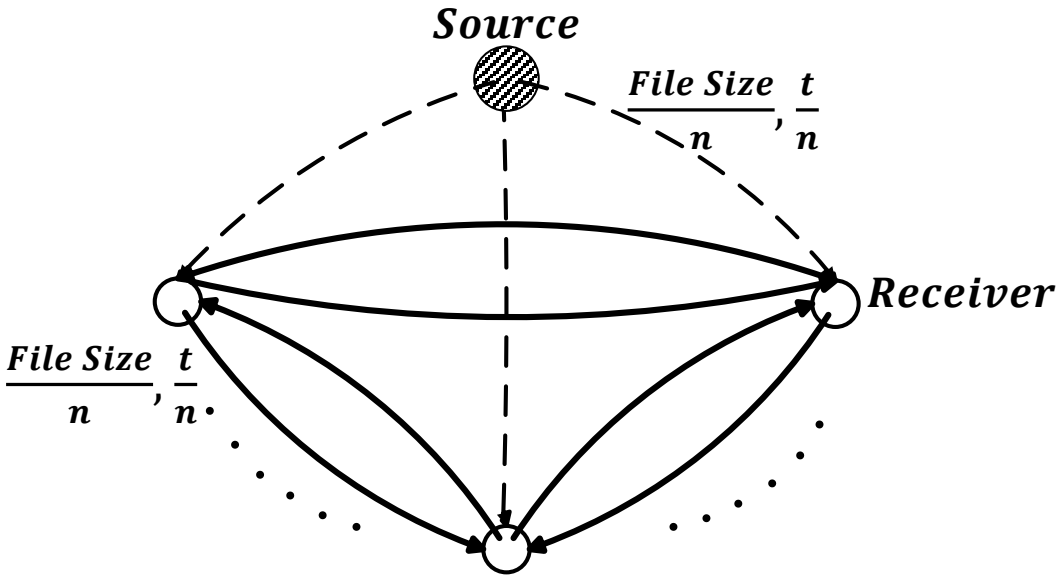

Figure 4.1: The implemented tuned version of BitTorrent

destination node has received all the packets of a distinct chunk of the file. A destination node starts forwarding packets from the source to all other destination nodes as soon as it receives the first packet from the source. Every destination node needs to forward a chunk of size $size_f$ to $n-1$ other nodes; this takes about t seconds. Therefore, the whole process of disseminating the file would take about t seconds as the source and destination nodes send packets simultaneously. In other words, by the simplified model, the tuned-version of BitTorrent needs about t seconds to distribute the file, which is about the time needed to send the file to a single destination node!

By the above argument, when the file size is fixed, one may expect the distribution delay of the algorithm not to grow with the number of destination nodes. In our implementation, however, we see that the distribution delay slightly increases with the number of destination nodes (i.e., with the number of VMs in the Amazon EC2). This is because, as stated earlier, the distribution delay is a function of many parameters some of which may not be captures appropriately in a theoretical model.

4.3 Implementation Challenges

Although the BitTorrent-like algorithm described in Section 4.1 is very simple in theory, its implementation and evaluation in a real data center imposes some challenges. In fact, validating and implementing a multi-threaded version of the algorithm from scratch for a distributed system took the majority of this research work.

One of the challenges we faced in evaluating the algorithm was to measure the distribution delay with high accuracy. The distribution delay of the data dissemination procedure starts when the first packet is sent out from the source s and finishes as soon as all of the nodes receives all of the packets. In other words, the duration of distribution time depends on the last node that receives the last packet of the source file. The challenge in measuring the distribution delay is because the virtual machines in data centers are not tightly synchronized. More specifically, the distribution delay of the multicast algorithm may be comparable to the synchronization error when the file size is not very large.

Our first attempt to overcome this issue was to use the Network Time Protocol (NTP) to synchronize our virtual machines as much as possible. We set the Amazon's NTP servers as the reference server since using a reference server located out of the data center leads to higher synchronization errors between our EC2 virtual machine instances. After synchronizing VMs, we measured the distribution delay by comparing the time the source sent the first packet to the time when the last node received the whole file. This way, we achieved an error of about 0.1 second in measuring the distribution delay.

The above approach works well when the distribution delay is considerably higher than 0.1 second. To make the error in measuring the distribution delay even lower, in our second approach, we had every destination node notify the completion of file reception to the source node. The source node then estimated the distribution delay by comparing the time it sent the first packet to the time it received the last notification from the destination nodes.

We implemented the ALM algorithm using multi-threaded programming in Python. The program is distributed as the same code is run on all the VMs. The single VM/node that has the source file takes the leader/source role and the other nodes take the receiver role. The source begins with creating two main threads: *Send thread* and *Control thread*. Receiver nodes, similarly, start with creating two main threads: *Receive thread* and *Relay thread*.

The receivers open a number of sockets to communicate with other nodes and assign each socket to a new thread to mange forwarding and receiving of packets. The source's *Send thread* sends data to all receivers in the same manner. Each receiver node has a stack (i.e., a last-in-first-out buffer), where incoming data received from the source node is placed in. It is also used to send out the data to all other receiver nodes. Each of the *Receive threads* puts the arriving packets into the stack. Moreover, every receiver starts forwarding packets received from the source as soon as they are available. Note that only packets from the chunk received from the source node are forwarded to other nodes. To do the forwarding, the *Relay thread* constantly checks the stack and picks a packet received from the source and sends it to all other receiver nodes. One approach to forward a packet from one node to other nodes is to send the packet sequentially in a round robin fashion. To be more time efficient, however, we used multiple threads to forward the packets in the stack to all the other receiver/destination nodes. To avoid costs of creating and destroying threads, the *Relay thread* uses a thread pool with threads each given one packet to send to all other destination nodes. In addition, each receiver starts writing the incoming packets of its corresponding chunk of data on the node's hard disk as soon as the reception of the first packet in the receiver node.

Some other implementation challenges are:

1. Before sending or forwarding packets to a destination node, the sending node must know that the code (the TCP server socket in particular) is up and running at the destination node. In other words, the algorithm cannot start before the TCP server socket at every node is up.

2. excluding the time required for creating sockets and threads from measuring the distribution delay.

3. Setting up and managing a large number of virtual machines.

We approached the first two challenges in the above list as follows. We wake up the source node before other nodes. The source's *Send thread* loads the source file into the memory and initializes the variables then sleeps and waits to be waken up again by the *Control thread*. When a destination node is up and ready, it sends a *NodeIsOn* signal to the source's *Control thread*. When all the *NodeIsOn* signals are received, the *Control thread* unicasts a *Connect* signal to every destination node, informing them that everyone is up and ready for receiving packets. It also notifies the *Send thread* to wake up and start establishing connections to all the destination nodes. Every time the source node creates a socket to connect to a destination node, it creates a new thread to manage the communications to/from the socket. When all the sockets and their corresponding threads are created, the *Send thread* sleeps again and waits to be notified by the *Control thread* for the right time to start the process of data dissemination.

So far, we have excluded the time of establishing sockets at the source node from the distribution delay. We also exclude the time of creating sockets at the destination nodes. To that end, the *Receive thread* at each destination node starts establishing connections to other destination nodes as soon as it receives the *Connect* signal from the source. If a connection is refused by the other side, the *Receive thread* keeps trying until it creates a connection. As in the source node, a thread is assigned to each established socket at destination nodes. By assigning each socket to a single thread, the receiver sends a *RecvReady* signal to the source. When all the connections are established at a destination node the Relay thread sends a *RelayReady* signal to the source. When the Control thread at the source receives all the *RecvReady* and *RelayReady* signals, it wakes up the *Send thread* to start sending packets.

To address the third challenge, i.e., to setup and monitor a large number of VMs

(up to 100 VMs in our case), we used Amazon Web Services' (AWS) Command Line Interface (CLI) [47] and Cluster SSH (CSSH) [48].

Using CLI, we can run VM instances by writing a script. Since different instances take different amount of time to boot up, we first wait until all instances are up. Then, using CLI we get all of the internal IPs and the public IPs. We copy the internal IPs in each instance since it is needed by the virtual machines to communicate with each other in their own virtual private cloud network. Finally, all the public IPs are given to CSSH, which opens a different terminal window for every single virtual machines. The terminal windows allow us to monitor and manage each of the virtual machines individually.

4.4 Test-bed Specification

The same program is run on each Amazon's virtual machine (VM). We utilize Amazon EC2 instances. The type of each instance is *m4.xlarge*. The *m4.xlarge* instance has 4 vCPU, 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) processors, 16 GiB memory, EBS-only SSD Storage (GB) and, 750 dedicated EBS bandwidth (Mbps) with high networking performance. We attached 8 GiB (gibibyte) of a general purpose SSD (GP2) with baseline of 100 I/O operations per second (IOPS) per GiB (burstable to 3000 IOPS). The Ubuntu server 14.04.4 LTS (GNU/Linux 3.13.0-91-generic x86_64) (HVM) is running on every VM instances as the operating system. The VMs are connected through a virtual private cloud (VPC) network in Amazon data centers. We choose our VMs to run in Amazon's data centers in US East (N. Virginia) region.

4.5 Performance Evaluation

We tested our ALM algorithm with a message/file of size 85 MiB on up to 100 Amazon EC2 VMs. As shown in Figure 4.2, the average distribution delay ranges

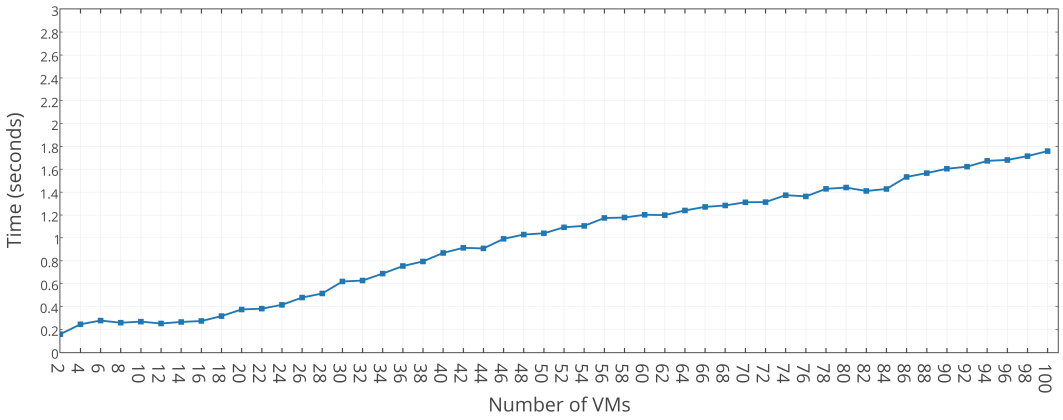

Figure 4.2: Message distribution among 100 Amazon EC2 VMs with message size of 85 MiB.

from 0.16 to 1.76 seconds when the number of destination nodes varies from 1 to 99 (Figure 4.2 shows the total number of nodes, that is the number of destination nodes plus the source node). This increase corresponds to a growth rate of about 0.016 seconds per virtual machine. The ALM algorithm most commonly used in clouds is the one in which the source sends the whole file to every other node one by one [17]. Since the source requires about 0.16 seconds to send the file to another node, the growth rate of this common ALM algorithm will be about 0.16 seconds per virtual machine, which is an order of magnitude higher than that of the ALM implemented in this work.

Chapter 5

Conclusion & Future work

In this thesis, we proposed application layer algorithms to compare performances of ALM and NLM with regards to networking load, link stress and distribution delay. We showed that ALM can preform the same (up to a constant factor of at most three) to NLM with respect to both networking load and link stress. We also proposed an ALM algorithm with asymptotically minimum distribution delay. In general networks, we proved that the distribution delay of the proposed ALM algorithm is at most a factor of $\mathcal{O}(\log n)$ away from that of the optimal NLM, where n denotes the number of destination nodes. We also implemented an ALM algorithm on Amazon EC2 and evaluated its distribution delay. Our results show that the distribution delay of the implemented ALM algorithm slowly increases with the number of destination nodes. The focus of our work was on networking load, link stress and distribution delay. There are other measures such as robustness against packet loss that can be used to compare ALM and NLM. An interesting theoretical work is to investigate the existence of ALM algorithms with simultaneous asymptotic optimality with regards to link stress, networking load and distribution delay. Finally, we believe that better (perhaps customized) models that reflect the complications in practice are needed to better analyze distribution delay and to propose fast ALM algorithms.

Bibliography

[1] Afshin Arefi. On the optimal set of channels to sense in cognitive radio networks. Master's thesis, University of Alberta, 2015. https://era.library.ualberta.ca/files/td96k5371.

[2] Yan Liu. Cooper: Expedite batch data dissemination in computer clusters with coded permutation gossips. Master's thesis, University of Alberta, 2015. https://era.library.ualberta.ca/files/w66346008.

[3] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[5] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 59–72. ACM, 2007.

[6] Derek G Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. Ciel: a universal execution engine for distributed data-flow computing. In *Proc. 8th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 113–126, 2011.

[7] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 98–109. ACM, 2011.

[8] Object Management Group. Data distribution service. `http://portals.omg.org/dds/`.

[9] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[10] Oracle coherence. `http://coherence.oracle.com/display/COH35UG/Network+Protocols`.

[11] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassem, and Doug Balensiefen. Deployment issues for the ip multicast service and architecture. *Network, IEEE*, 14(1):78–88, 2000.

[12] Mahmood Hosseini, Dewan T Ahmed, Shervin Shirmohammadi, and Nicolas D Georganas. A survey of application-layer multicast protocols. *Communications Surveys & Tutorials, IEEE*, 9(3):58–74, 2007.

[13] Xiaozhou Li and Michael J Freedman. Scaling ip multicast on datacenter topologies. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 61–72. ACM, 2013.

[14] Kai-Wei Ke and Chia-Hui Huang. Performance evaluation of multisource application layer multicast (alm): Theoretical and simulative aspects. *Computer Networks*, 57(6):1408–1424, 2013.

[15] Pekka Savola. Overview of the internet multicast routing architecture. `http://www.ietf.org/rfc/rfc5110.txt`, January 2008. RFC 5110.

[16] Bob Quinn and Kevin Almeroth. Ip multicast applications: Challenges and solutions. `http://www.ietf.org/rfc/rfc3170.txt`, September 2001. RFC 3170.

[17] Jessie Hui Wang, Jeffrey Cai, Jerry Lu, Kevin Yin, and Jiahai Yang. Solving multicast problem in cloud networks using overlay routing. *Computer Communications*, 70:1–14, 2015.

[18] Dan Li, Mingwei Xu, Ying Liu, Xia Xie, Yong Cui, Jingyi Wang, and Guihai Chen. Reliable multicast in data center networks. *IEEE Transactions on Computers*, 63(8):2011–2024, 2014.

[19] Ymir Vigfusson, Hussam Abu-Libdeh, Mahesh Balakrishnan, Ken Birman, Robert Burgess, Gregory Chockler, Haoyuan Li, and Yoav Tock. Dr. multicast: Rx for data center communication scalability. In *Proceedings of the 5th European conference on Computer systems*, pages 349–362. ACM, 2010.

[20] Yang Chu, Sanjay Rao, Srinivasan Seshan, and Hui Zhang. Enabling conferencing applications on the internet using an overlay muilticast architecture. *ACM SIGCOMM computer communication review*, 31(4):55–67, 2001.

[21] John Jannotti David K Gifford, Kirk L Johnson, M Frans Kaashoek, and James W OToole Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. Usenix Fourth Symp. Operating System Design and Implementation (OSDI00)*, 2000.

[22] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. *Scalable application layer multicast*, volume 32. ACM, 2002.

[23] Minseok Kwon and Sonia Fahmy. Topology-aware overlay networks for group communication. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, pages 127–136. ACM, 2002.

[24] Sherlia Y Shi, Jonathan S Turner, and Marcel Waldvogel. Dimensioning server access bandwidth and multicast routing in overlay networks. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pages 83–91. ACM, 2001.

[25] John Byers, Jeffrey Considine, Michael Mitzenmacher, and Stanislav Rost. Informed content delivery across adaptive overlay networks. *ACM SIGCOMM Computer Communication Review*, 32(4):47–60, 2002.

[26] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, 2003.

[27] Yang-hua Chu, Sanjay G Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE Journal on selected areas in communications*, 20 (8):1456–1471, 2002.

[28] Michael R Gary and David S Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[29] Sonia Fahmy and Minseok Kwon. Characterizing overlay multicast networks and their costs. *IEEE/ACM Transactions on Networking (TON)*, 15(2):373–386, 2007.

[30] Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10):892–901, 1985.

[31] Ronald Greenberg and Charles E Leiserson. Randomized routing on fat-trees. 1989.

[32] Ronald I Greenberg. Efficient interconnection schemes for vlsi and parallel computation. Technical report, DTIC Document, 1989.

[33] Aditya Akella, Theophilus Benson, Bala Chandrasekaran, Cheng Huang, Bruce Maggs, and David Maltz. A universal approach to data center network design. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, page 41. ACM, 2015.

[34] Mallik Mahalingam, D Dutt, Kenneth Duda, Puneet Agarwal, Lawrence Kreeger, T Sridhar, Mike Bursell, and Chris Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[35] Murari Sridharan, A Greenberg, N Venkataramiah, Y Wang, K Duda, I Ganga, G Lin, M Pearson, P Thaler, and C Tumuluri. Nvgre: Network virtualization using generic routing encapsulation. *IETF draft*, 2011.

[36] John C-I Chuang and Marvin A Sirbu. Pricing multicast communication: A cost-based approach. *Telecommunication Systems*, 17(3):281–297, 2001.

[37] Robert C Chalmers and Kevin C Almeroth. Modeling the branching characteristics and efficiency gains in global multicast trees. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 449–458. IEEE, 2001.

[38] Pavlin Radoslavov, Hongsuda Tangmunarunkit, Haobo Yu, Ramesh Govindan, Scott Shenker, and Deborah Estrin. On characterizing network topologies and analyzing their impact on protocol design. 2000.

[39] Kianoosh Mokhtarian and Hans-Arno Jacobsen. Minimum-delay multicast algorithms for mesh overlays. *IEEE/ACM Transactions on Networking (TON)*, 23(3):973–986, 2015.

[40] Eli Brosh, Asaf Levin, and Yuval Shavitt. Approximation and heuristic algorithms for minimum-delay application-layer multicast trees. *IEEE/ACM Transactions on Networking*, 15(2):473–484, 2007.

[41] Hwa-Chun Lin, Tsung-Ming Lin, and Cheng-Feng Wu. Constructing application-layer multicast trees for minimum-delay message distribution. *Information Sciences*, 279:433–445, 2014.

[42] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. *Resilient overlay networks*, volume 35. ACM, 2001.

[43] Yair Amir, Claudiu Danilov, Stuart Goose, David Hedqvist, and Andreas Terzis. An overlay architecture for high-quality voip streams. *IEEE Transactions on Multimedia*, 8(6):1250–1262, 2006.

[44] Georgios Rodolakis, Anis Laouiti, Philippe Jacquet, and Amina Meraihi Naimi. Multicast overlay spanning trees in ad hoc networks: Capacity bounds, protocol design and performance evaluation. *Computer Communications*, 31 (7):1400–1412, 2008.

[45] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM Computer Communication Review*, 38(4):63–74, 2008.

[46] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[47] Amazon web services command line interface. `https://aws.amazon.com/cli/`.

[48] Cluster ssh. `https://github.com/duncs/clusterssh`.

Chapter 6

Appendix

6.1 Proof of Proposition 2

We prove the proposition by proposing a recursive ALM algorithm, and showing that its time complexity is $O(h \cdot \log n / \log \log n)$. Let T be a binary tree of height h, and T_L and T_R denote its left and right subtrees, each of height $h-1$. Without loss of generality, assume that the source is in T_L. Let T_{init} be a subtree of T_L of height $h' = \lfloor \frac{\log n}{2} \rfloor$, containing the source. As illustrated in Figure 6.1, the subtree T_R is partitioned into $2^{h'}$ subtrees each of height $h - h' - 1$. These trees are called partitioned-subtrees of T_R.

In Step one, the recursive algorithm is run on T_{init}. Step two starts as soon as all the nodes in T_{init} receive the packet. In this step, each node in T_{init}, in order and one by one (to avoid queueing), forwards its packet to a selected distinct node in a partitioned-subtree of T_R. As soon as the last node in T_{init} sends its packet, step three starts by calling the algorithm on T_L. Last step, i.e., Step four, starts as soon as all the selected nodes in the partitioned-subtrees of T_R receives the forwarded packets. In this step, the algorithm is simultaneously called/run on all partitioned subtrees of T_R.

Let $T(h)$ denote the time complexity of the above recursive algorithm on a full binary tree of height h. The time complexities of steps one and two are $T(h')$, and

$2^{h'}$, respectively. The time complexity of Step three is $T(h-1)$, and that of Step four is $T(h-h'-1)$. Note that step four starts $2h-1$ units of time after step three begins. By the above discussion, we get the following recursive equation

$$T(h) = T(h') + 2^{h'} + \max\{T(h-1), 2h-1+T(h-h'-1)\},$$

solving which we get $T(h) = O(h \log n / \log \log n)$.

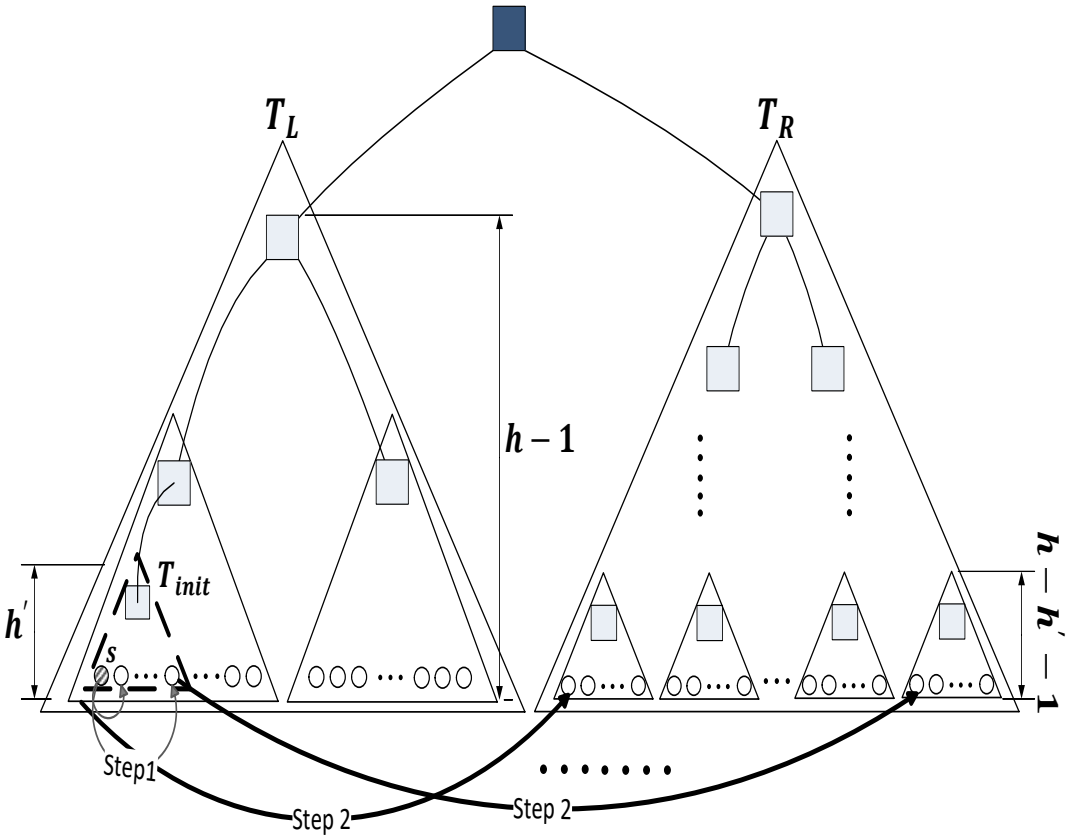

Figure 6.1: A carefully designed ALM can perform message distribution in $O(h \log n / \log \log n)$ in a full binary tree.

6.2 Proof of Proposition 3

To prove the proposition, we first propose a multicast algorithm, and then show that the time complexity of the algorithm is $\mathcal{O}(h \log\log n)$ in a complete binary graph of height h with 2^h leaves.

For simplicity, assume that h is an even number. The proposed multicast algorithm first divides the tree into $2^{\frac{h}{2}}$ subtrees. Then, the algorithm is recursively called on the subtree that includes the source. When the algorithm finishes on that subtree, every node of the subtree sends the packet to a distinct node at the other subtrees. This step requires $2h$ rounds because of the oversubscription of 1:1. By the end of this step, there will be one node in each subtree that has received the packet. In the last step, the algorithm is run in parallel on all the subtrees.

Let $T(h)$ denote the time complexity of the above recursive algorithm. We have

$$T(h) = 2T(h/2) + 2h,$$

solving which yields $T(h) = \mathcal{O}(h \log h)$.

6.3 Proof of Lemma 1

For every set S, $S \in \mathcal{S}$, let $\sigma(S) = \sum_{a \in S} a$, and

$$(S_1^\dagger, S_2^\dagger) \in \underset{\substack{S_1 \cup S_2 = S \\ S_1 \cap S_2 = \emptyset}}{\arg\min} \max(\sigma(S_1), \sigma(S_2)). \tag{6.1}$$

Without loss of generality, assume $\sigma(S_2^\dagger) \geq \sigma(S_1^\dagger)$. Towards showing a contradiction, assume $\sigma(S_2^\dagger) > \frac{2A}{3}$. Suppose

$$\exists a \in S_2^\dagger \quad s.t. \quad a \geq \frac{A}{3}.$$

Then, for the sets $S_1 = \{a\}$, and $S_2 = \mathcal{S} \backslash S_1$, we get

$$\max(\sigma(S_1), \sigma(S_2)) \leq \frac{2A}{3},$$

which is a contradiction by (6.1), and the assumption that $\sigma(S_2^\dagger) > \frac{2A}{3}$. Therefore, we must have

$$\forall a \in S_2^\dagger \qquad a < \frac{A}{3}. \tag{6.2}$$

Let $S_2 = S_2^\dagger \backslash \{a\}$, and $S_1 = S_1^\dagger \cup \{a\}$, where $a > 0$ is an arbitrary member of $S_2^\dagger$. By (6.2), and the fact that $\sigma(S_2^\dagger) > \frac{2A}{3}$, we get

$$\frac{A}{3} < \sigma(S_2) < \sigma(S_2^\dagger),$$

which implies

$$\max(\sigma(S_1), \sigma(S_2)) < \max(\sigma(S_1^\dagger), \sigma(S_2^\dagger)),$$

a contradiction to (6.1).

6.4 Proof of Lemma 2

We call a switch a *terminating* switch if it is directly connected to more than $m/2$ nodes. Note that a tree can have at most one terminating switch.

In the first step, we set an arbitrary switch r_0 as the root of tree T. Since r_0 is not a pivot, it must have a subtree rooted at, say, r_1 that has more than $m/2$ nodes (see Figure 6.2). Therefore, the number of nodes in the other subtrees of r_0 all together cannot be more than $m/2$.

If r_1 is a terminating switch, we are done. Otherwise, we set r_1 as the root of tree T. Again, since r_1 is not a pivot, it must have a subtree rooted at switch, say, r_2, that contains more than half of the nodes. Note that $r_2 \neq r_0$, because, as depicted in Figure 6.3, the subtree of r_1 which is rooted at r_0 (i.e, the subtree in dashed triangle) cannot contain more than half of the nodes. In each step of the above process, we

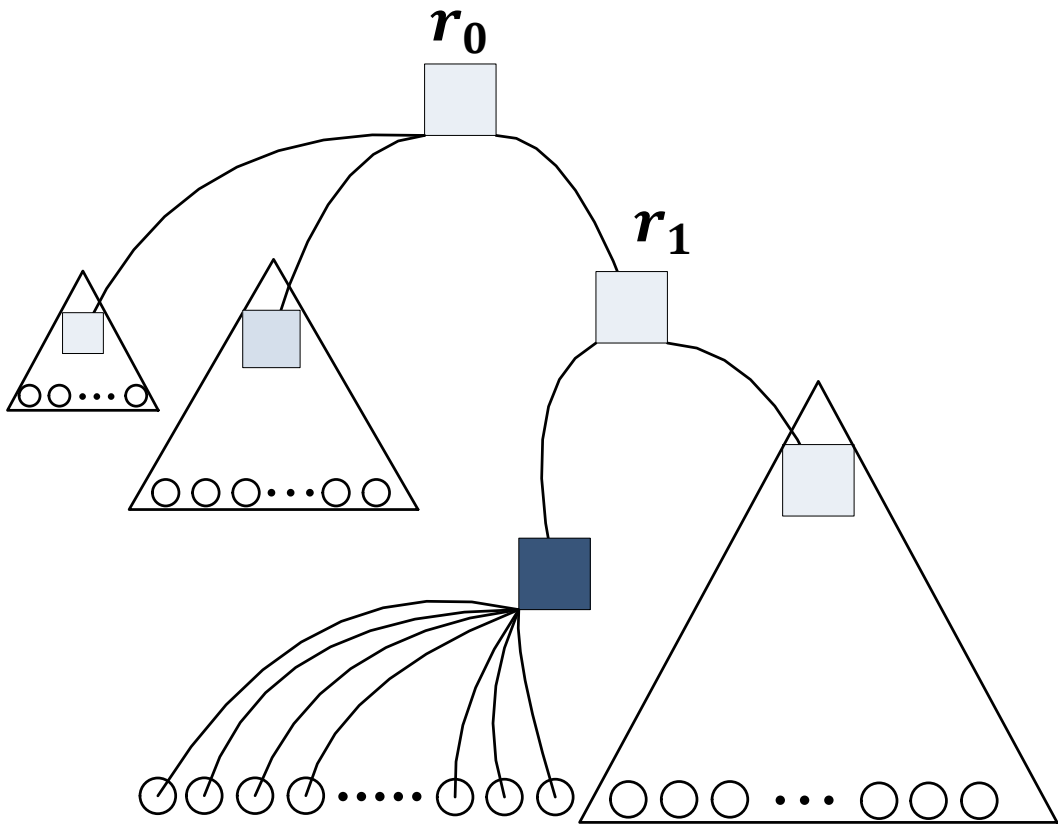

Figure 6.2: r_0 is neither a pivot nor a terminating switch

find a new subtree with smaller height than the one found in the previous step, and the new subtree contains more than half of the nodes. Thus, this process must end with a terminating switch.

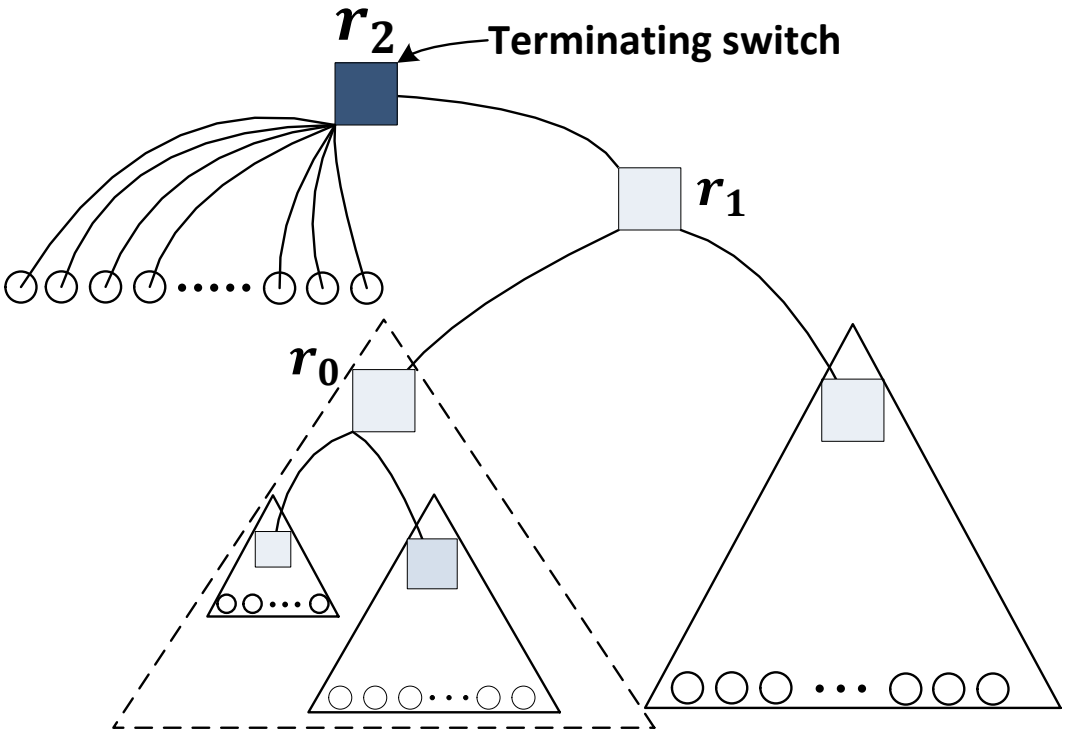

Figure 6.3: r_2 is a terminating switch