The Dissertation Committee for Jangwon Lee

certifies that this is the approved version of the following dissertation:

Cooperative Resource Discovery and Sharing in Group Communications

Committee:

Gustavo de Veciana, Supervisor

Vijay K. Garg

Edward J. Powers

Scott M. Nettles

Harrick M. Vin

Cooperative Resource Discovery and Sharing in Group Communications

by

Jangwon Lee, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2003

Dedicated to my parents,

Jimin and Hanoom

Acknowledgments

I owe a special gratitude to my advisor; Professor Gustavo de Veciana. His invaluable advice and feedback made me advance throughout my research career at UT-AUSTIN. Furthermore, he always found time whenever I needed an intelligent discussion. He exemplified one of the idealized advisors, which I will pursue. I thank Dr. Scott Nettles for providing valuable suggestions on my work and helping me in a number of implementation problems. I also would like to thank Dr. Edward Powers, Dr. Harrick Vin and Dr. Vijay Garg for serving on my committee and sharing their wisdoms.

I also thank my former advisor, Dr. Jin-heon Seo for his continuous support. I am indebted to my colleagues Xiangying Yang, Steven Weber, Xun Su, Jay Yang, Sangkyu Park, Philippe Girolami and Seung Jun Baek for their helpful comments and discussions. Special thanks go to Sanghoon Lee, Sungwook Yu and Tae-jin Lee for helping me in my early stage of graduate study.

My special thanks go to Sungju Lee at HP lab for providing helpful suggestions and helping me learn the importance of networking with people. Special acknowledgment also goes to my colleagues working with me when I was an intern at Cisco and HP lab. Their help and support made me learn and achieve more practical perspectives in my study.

I am also grateful of Kyoungsu Kwak, Sungjoo Lee and Hosung Choo for having

fun with me in my early graduate time. My gratitude also goes to Kyoil Kim, Joonhyuk Kang, Sungkyu Song, Youngmoon Choi, Seokjin Lee and other SNU SoEE alumni for their comments and help when needed. Kudos to my friends back home in Korea and those whom I met in the United States. The friendship I have with them is what kept me going on during the tough times. Most of all, I would like to express my sincere thanks to my parents, my sisters, Jimin and Hanoom for their everlasting love, support and encouragement. This dissertation is dedicated to them.

JANGWON LEE

The University of Texas at Austin

August 2003

Cooperative Resource Discovery and Sharing in Group Communications

Publication No. _____

Jangwon Lee, Ph.D.

The University of Texas at Austin, 2003

Supervisor: Gustavo de Veciana

In this dissertation we study four areas where members' cooperative behavior in discovering and sharing network resources can be beneficial to achieving various objectives in group communication. We propose a framework for discovering the topology of a shared multicast tree based on a novel fan-out decrement mechanism analogous to time-to-live (TTL) decrementing in IP. The proposed algorithm for topology discovery is based on the matrix of path/fan-out distances among session members. We exhibit sufficient conditions for topology discovery based on a reduced distance matrix, and propose a practical protocol to acquire this information. Additionally, we show how the same approach permits nodes to discover the multicast distribution tree associated with members within their fan-out/TTL scoped neighborhoods. This permits one to reduce the computational costs while making the communication costs proportional to the size of neighborhoods.

We then present a novel distributed and scalable framework to support on-demand filtering and tracing services to defeat distributed Denial of Service attacks. Our filtering mechanism is designed to quickly identify a set of *boundary* filter locations so that attack packets might be dropped as close as possible to their origin(s). We argue that precisely identifying the origins of an attack is infeasible when there is only a partial deployment of tracing nodes - as is likely to be the case in practice. Thus we present a tracing mechanism which can identify sets of candidate nodes containing attack origins. Both mechanisms use multicasting services to achieve scalable, responsive and robust operation.

Next we propose a topology-sensitive subgroup communication (TSC) mechanism to support efficient subgroup communications in large-scale multicast applications. Our TSC mechanism exploits spatial locality among members communications within a given subgroup, and enables members to autonomously build a TSC forwarding structure consisting of multiple unicast and scoped multicast connections. This can completely eliminate the need to create additional multicast sessions while minimizing the exposure of receivers to unnecessary packets. Simulations of this approach suggest that TSC mechanisms perform well for diverse densities and distributions for a subgroup's nodes.

Finally we propose a method to quickly distribute large files across distributed nodes. Our Adaptive FastReplica (AFR) mechanism exploits path diversity among the origin and receivers and adaptively balances loads across overlay paths. Since our approach uses a fixed overlay structure but then adapts the loads across paths, the control overhead associated with constructing and maintaining overlay structure, typical of application-level multicasting solution, is reduced. Based on our experiments with a prototype implementation over the Internet, we demonstrate its efficiency at minimizing the overall replication time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

There is an old adage saying

Two heads are better than one.

This proverb suggests that if a group of people get together and share their intellectual assets, an abundance of knowledge will be amassed and the solution to many problems found. In this dissertation, we draw on this concept in the context of computer networking. Our goal is to exhibit cases where *group communications* can be useful to achieve various objectives. Furthermore, we want to gain insights on exactly which components or characteristics of group communications are being exploited in accomplishing those objectives.

While traditional network and transport protocols support only *point-to-point* communication (unicasting) service, there has been a growing need for efficient *multipoint* communication (multicasting) to support a plethora of multi-party applications, including audio and video conferencing tools, shared electronic white boards, distributed interactive simulation and multi-player games. Due to the significant amounts of bandwidth that such applications may require, it would not be efficient to develop them over traditional point-

to-point communication. The terms “multicasting” and “group communication” are often used interchangeably among network researchers. However, “group communication” needs to be distinguished from “multicasting.” In a group communication, there are multiple entities sharing common goals, and members in a group may communicate with each other via multicasting or unicasting. That is, multicasting or unicasting simply specify communication methods on how data or content are exchanged among members in a group.

In the last decade, there has been, and there still is, a large amount of research effort on support for efficient multipoint communication. The work includes a network-level solution, i.e., IP multicast [1] as well as application-level multicasting [2, 3, 4]. In these problem settings, the existence of multiple entities (most likely hosts or nodes in networks) are assumed, and the main objective is to devise efficient data delivery mechanisms for communication among members in a group. The work in this dissertation is basically in alignment with this research effort *but* our focus is on highlighting (1) a variety of contexts where member nodes cooperate, and (2) the advantageous features to be exploited by group communications.

An example of how group communication is effectively used to achieve a special goal is that of a Distributed Denial of Service (DDoS) attack, which will be dealt with in Chapter 3. From the perspective of victims, a DDoS attack is one of the most difficult security problems to address and one of the greatest threats to today’s Internet. However, ironically, DDoS attack is one of the greatest products which can effectively attack the target victim from the perspective of the attacker. In DDoS attacks, the attacker plants attack tools on a number of computers by exploiting security vulnerabilities – turning them into “zombies.” To generate a flood of network traffic to the victim’s site, the attacker issues commands to “handler” computers, and each in turn, sends commands to zombie comput-

ers. Finally, a group of zombie computers forces a flood on the victim site as shown in Figure 1.1. As can be seen, the critical innovation in DDoS attacks is exploiting the fact that there are multiple distributed zombie computers. Even though the small number of attack packets from each zombie may be negligible, when this is multiplied by the large number of zombie computers, their impact can be significant and hard to mitigate. Eventually, in this scenario, the attackers' goal of shutting down target systems can be effectively accomplished with the coordination of members that unwittingly join the group of zombies. As will be seen in the sequel, we propose a mechanism to defeat DDoS attacks based on the same spirit in which they are realized, i.e., a group of distributed components in networks will perform filtering and tracing of attack packets in a cooperative manner.

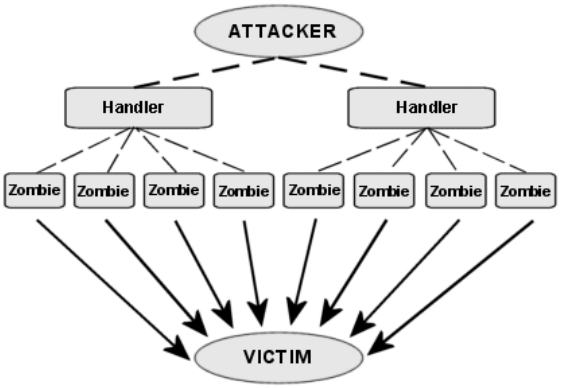


Figure 1.1: Architecture of DDoS attack.

In group communications, there are many challenges and issues that need to be resolved, such as scalability and heterogeneity. The difficulty in handling these problems comes from the fact that group communication deals with interaction among *multiple* entities, often resulting in more complex situations, versus traditional point-to-point type communication. In this dissertation, we are interested in how such diversity can be turned into

an advantage to build various applications/tools drawing on group communication. The key point here is that in group communication environments, there may be ample opportunities to exploit diversity, share resources, and collaborate among distributed entities. For this purpose, we studied four different problems and associated solutions where multicasting methods range from IP multicast to application-level multicast.

In Chapter 2, we study the topology and resource discovery problem for IP multicast. Knowing other members' locations and capabilities is a critical starting point for members to cooperate and perform a common task in group communication environments. Thus, discovering multicast tree topologies is an important component in various areas, such as multicast network management, reliable multicast and multicast congestion control protocols. In this chapter, we propose a new approach to shared tree discovery based on path and fan-out distance information between end nodes in a tree. The fan-out distance information is obtained via a novel fan-out decrement mechanism, which is analogous to the time-to-live (TTL) decrement mechanism in IP.

In Chapter 3, as mentioned the above, we study how to effectively thwart DDoS attacks. Unfortunately, the stateless nature of IP protocols make it difficult to identify the true source of packets if the sources wish to conceal it. Thus, we propose a tracing mechanism which can identify sets of candidate nodes containing attack origin(s). Furthermore, we propose an on-demand filtering mechanism which enables a host to request unwanted packets to be dropped early on, before they reach the victim. Since they are based on IP multicast service, both mechanisms could achieve scalable, responsive and robust operation.

In Chapter 4 we deal with the *preference heterogeneity* problem in large-scale multicast sessions. Abundant content, data type and diverse members' interests naturally lead to preference heterogeneity within large multicast sessions, requiring communication among

subgroups of members sharing common interests/requirements. Thus, we propose a new approach to support efficient subgroup communication. The key idea is to use a combination of unicast and scoped multicast so as to exploit the spatial locality of members within a subgroup. That is, scoped multicast is used if members are well clustered, otherwise unicast is used. Here, finding nearby neighborhoods' preferences and cooperating to forward data are key elements of our solution.

In Chapter 5, we consider a content delivery problem across geographically distributed nodes. Our focus is on distributing large files, such as software packages, stored streaming media files or data associated with distributed simulations, and the objective is to minimize the overall replication time. Our basic idea is to partition a large file into multiple chunks, which each get transmitted to an associated receiver. In turn, each receiver relays its chunk to the other receivers. By contrast with more complex application-level multicasting strategies, our approach does not require optimal construction of paths or probing. Instead, it uses a fixed but large collection of unicast paths among the receivers. This permits the traffic to be spread across the network so as to exploit path diversity. This approach is extended to support adaptive balancing of the loads across paths by creating non-uniform partitions of the file.

We conclude the dissertation in Chapter 6, where we summarize the key findings of our work and highlight the benefits we have exhibited for collaborative resource discovery and sharing in applications based on distributed group communications.

Chapter 2

IP multicast topology discovery

2.1 Introduction

Due to its bandwidth efficiency, IP Multicast is the preferred data delivery method for large one-to-many communication scenarios. Another advantage associated with IP multicast service, is as an abstraction for group communication, that is, users can join and leave a multicast session without requiring explicit knowledge of the membership or of the structure of the distribution tree. However, despite this clean abstraction, if the use of IP multicast sessions becomes widespread, we observe that the potential downside from hiding topological information on multicast distribution trees may be heightened.

Depending on the scope of interest, IP multicast resource and topology discovery problems can be classified into two categories: *global*, where one is interested in discovering all the members in a multicast session and *local*, where one is interested in finding relationships among a subset of members in a session and their associated topology.

From the perspective of IP multicast service user (e.g., movie distributor, advertiser) the number of subscribers in a session, their location, and their density in a specific region

may be useful information. From the perspective of a network service provider, the extent to which network resources are being used (e.g. number of links and routers) by a given multicast session may be important to assess usage costs. In both cases knowledge of the *global* multicast topology would significantly facilitate resource management.[1]

In a large scale multicast session, it is not uncommon for nearby members to co-operate and perform a common task, such as distributed computation and data sharing. In this case, the *local* topology and membership information for a neighborhood of a given node would be useful. A typical use of local resource and topology discovery is in building schemes for loss recovery and congestion control in the context of multicast sessions supporting heterogeneous receivers. While a variety of approaches have been proposed to tackle this problem, e.g., [5], [6], [7], [8], [9], a common thread is to recognize that performance can be enhanced by either implicitly or explicitly exploiting the structure of the multicast distribution tree. OTERS [6], Tracer [5] and GFP [10] are examples of research efforts making use of explicit topology information via MTRACE [11] and an inference technique [12] for local loss recovery. Further motivation for exposing the multicast distribution tree is given in [12], [10] and [13].

Despite its potential usefulness, there has been surprisingly little research, e.g., [14], [12], concerning global multicast topology discovery and even less, to our knowledge, concerning local multicast topology discovery. A large amount of work has however been devoted to Internet topology discovery, see e.g., [15], [16], [17], [18]. By contrast with multicast topology discovery, Internet topology information can be collected during long time scales (e.g., several days or even several weeks) [15], or by passive probing [19], since the physical topology remains stable over reasonably long time periods. In the case of multi-

[1]Throughout this chapter a multicast topology refers to the multicast distribution tree constructed by multicast routing protocols.

cast service the character of the distribution tree is only of interest when the session is active and may change dynamically throughout that period. Thus multicast topology discovery algorithms should be able to operate online and serve as practical protocol building blocks which dynamically track membership changes. As will be discussed below, these and other requirements make proposed approaches based on end-to-end measurements, [14], [12] fall short as practical solutions.

The following are some desirable characteristics that a multicast topology discovery mechanism should have.

Accuracy: Topology information should be "reliable" since potentially critical decisions will be based on it.

Adaptability: A mechanism should adapt to changes in group membership or distribution path topology.

Low overheads: Computational requirements at end hosts or servers and communication overheads should be low.

Distributed: From the perspective of robustness, it is preferable that discovery be performed in a distributed manner rather than relying on central points.

With these in mind, in this chapter we propose a new approach to multicast topology discovery. It is based on introducing a novel fan-out decrement mechanism to IP multicast service, which is analogous to the time-to-live (TTL), or hop count, decrement mechanism currently supported in IP. As discussed in the sequel, the proposed scheme achieves all of the desirable characteristics posed above but *only* for the case where multicast service is based on *shared tree*, e.g., Core Based Trees(CBT) [20], [21], versus source tree routing.

Additionally, we propose both concepts and practical issues for *local* resource and topology discovery which enable further 'scalability' for large scale multicast applications.

The chapter is organized as follows. Section 2.2 introduces the proposed fan-out decrement mechanism, briefly indicating some of its uses for resource and topology discovery. In Section 2.3 we propose and analyze an algorithm for global multicast tree discovery. Section 2.4 includes comments on implementation and information exchange, and is followed by Section 2.5 wherein we discuss a framework for partial (i.e., local) topology discovery of multicast trees. In Section 2.6, additional use of the work is proposed and Section 2.7 discusses the advantages and shortcomings of previous work and contrasts these with our work. Section 2.8 concludes the chapter.

2.2 Fan-out decrement mechanism

We propose a fan-out decrement mechanism for IP Multicast service, which supports the following three elements/behaviors:

1. A fan-out field in a multicast packet;

2. When a multicast packet traverses a router, corresponding to a fan-out point where the packet is replicated and forked out, the router decrements the fan-out field by one.

3. Multicast routers at fan-out points discard incoming packets whose fan-out fields have reached 0.

Note that these components are entirely analogous to those of the current TTL decrement mechanism. The main difference is the location where decrementing occurs: every router along the path of a packet for the TTL field while only routers corresponding to fan-out points in multicast distribution tree for fan-out field.

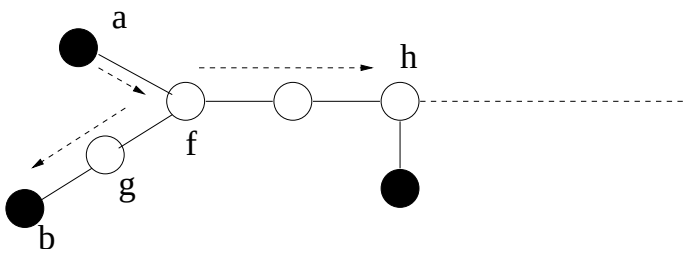


Figure 2.1: Fan-out decrement mechanism illustration.

Table 2.1: Parallels between IP and IP Multicast.

IP	IP Multicast
ICMP	IGMP
Traceroute	MTRACE
TTL decrement	Fan-out decrement

Consider the example shown in Figure 2.1. Suppose member *a* multicasts a packet with its fan-out field set to 1. When the packet reaches fan-out node f, the fan-out field becomes 0 but the packet is duplicated and forwarded and will reach member *b*. Another duplicate packet will be forwarded in the other direction but is discarded at fan-out node *h*. Note that routers that are not fan-out points in the distribution tree, e.g., *g*, do not decrement the fan-out field or discard packets whose fan-out field is 0.

Clearly this mechanism serves as an intuitive and natural counterpart to the TTL decrement mechanism in IP. Table 2.1 summarizes parallels between IP and IP Multicast components. Also note that this new feature is simple to implement and will not incur large overheads at routers. We envisage implementing fan-out decrementing in two ways: 1) changing native IP packet header and router functionality or 2) perhaps more realistically providing this as a service supported by IGMP [22] – see Section 2.4 for details.

The original purpose for the TTL decrement mechanism was to bound the life of packets in the network to circumvent the adverse effects of forwarding loops during routing transients. However, due to its simplicity and usefulness, the TTL decrement mechanism also can be used for scoping IP multicast packets as well as other applications associated with resource and path discovery, e.g., expanding ring search, traceroute [23]. We believe that, in the context of multicast service, the proposed fan-out decrement mechanism can play a similar role.

First, TTL scoping is to constrain how far a multicast packet can traverse within a multicast session by carefully choosing the TTL value. To see how the multicast scoping can be enhanced with the addition of the fan-out decrement mechanism, consider the case in Figure 2.2 (a) where member a wishes to send packets only to a set of node, $A = \{b, c, d\}$. Unless a sends repeated unicast transmissions to A, a can perform TTL scoping by setting TTL value to its maximum distance from a to A, i.e., 5. However, the packets will eventually reach the other members, $\{e, f, g, h, i, j\}$. In addition to TTL scoping, setting fan-out value to its maximum fan-out distance from a to A, i.e., 2, turns out to be more efficient scoping since the packets will arrive only at A.

Second, suppose a member in a multicast session wishes to discover the existence of another one with a given attribute but close by. Currently, it may do so using expanding ring search: i.e., multicasting a sequence of query packets with increasing TTL until an appropriate reply is received. Note that we can save time and resources by using the fan-out field to perform an expanding ring search. The possible increase in efficiency for such a search, can be seen by considering the following of two members that are only one fan-out away but a large hop count distant from each other, shown in Figure 2.2 (b). Member a performs an expanding ring search based on the fan-out field, that is, sending a query packet

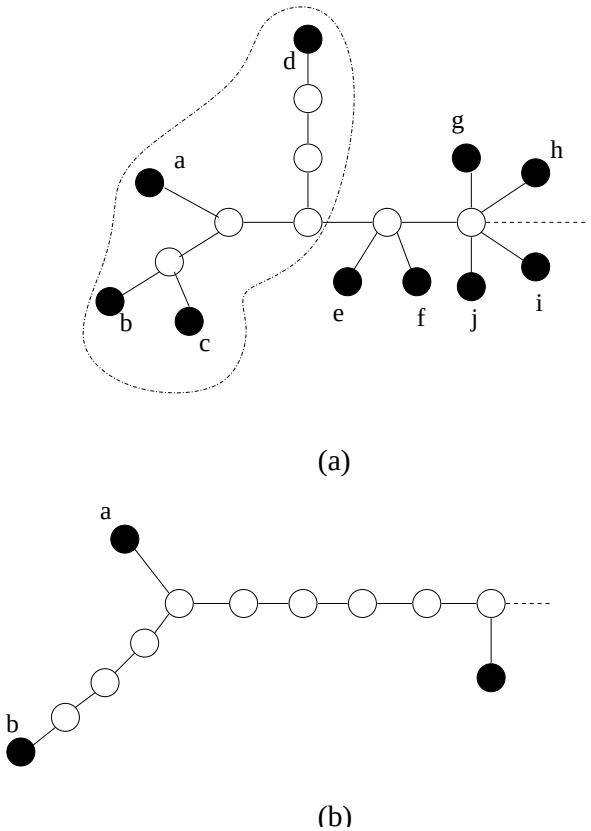


Figure 2.2: Fan-out decrement mechanism usage illustrations.

with fan-out field set to 1. In this scenario, which might not be infrequent for sparse large-scale multicast sessions, member *a* can quickly identify a close member, *b* by the first query. Note that this type of resource discovery is applicable to *both* source and shared tree routing protocols. Also note that we are not arguing for the superiority of the fan-out decrement mechanism over the TTL one but proposing potential benefits when both mechanisms are being used together in IP multicast context.

Finally, as another useful application, we propose the discovery of shared multicast trees based on the proposed fan-out decrement mechanism. Our algorithm requires that each node acquire a *distance matrix* for the current session members, which is the path

and fan-out distances of pairs of members. In order to do so, packets will need to carry two additional pieces of information, Initial_TTL and Initial_fan-out, corresponding to the initial values of the TTL and fan-out fields. Clearly with this information in hand, a receiver can immediately compute its path distance and fan-out distance, i.e., number of fan-out nodes traversed, from the source. In the next section we shall develop a tree discovery algorithm based on full and reduced distance matrices. In Section 2.4 we will discuss practical issues in efficiently acquiring and distributing the required distance information.

2.3 Tree discovery algorithm

We will consider several variations of the following basic problem: given the *distance matrix* associated with the members (i.e., end hosts) of a multicast session using a shared distribution tree, determine its physical topology.

2.3.1 Model and Notation

We will use the physical multicast tree illustrated in Figure 2.3 as a reference in discussing our model.[2] The end nodes, shown as solid black circles, correspond to members of the multicast session, while internal nodes, corresponding to network routers, are shown as white circles.[3] In the sequel we will refer to internal nodes where multiple copies of a multicast packet are created as *fan-out nodes.*

We define two types of distances between nodes on a tree. The *path distance* $d_p(m,n)$ between two nodes, m and n, corresponds to the number of links along the path

[2]Throughout this chapter, a multicast tree or a tree means a shared multicast tree, unless explicitly mentioned.

[3]In a multi-access LAN environment, an end node can be considered as a representative of all multicast members on the LAN, e.g., the one with the lowest IP address among members on the LAN.

between them. The *fan-out distance* $d_f(m,n)$ between two nodes, *m* and *n*, corresponds to the number of fan-out nodes on the path between them. Note that in the case where *m* or *n* are themselves fan-out nodes in the tree, the fan-out distance does not include *m* or *n*. For example, $d_f(f_1, f_2) = 1$ in Figure 2.3. We denote such path and fan-out distance as a tuple $d(m,n) = (d_p(m,n), d_f(m,n))$, e.g., for our example we have $d(e_2, e_6) = (8,4)$. Table 2.2 exhibits the *full distance matrix*, which contains the distances among all pairs of members in the multicast session shown in Figure 2.3. Note that this table is symmetric.

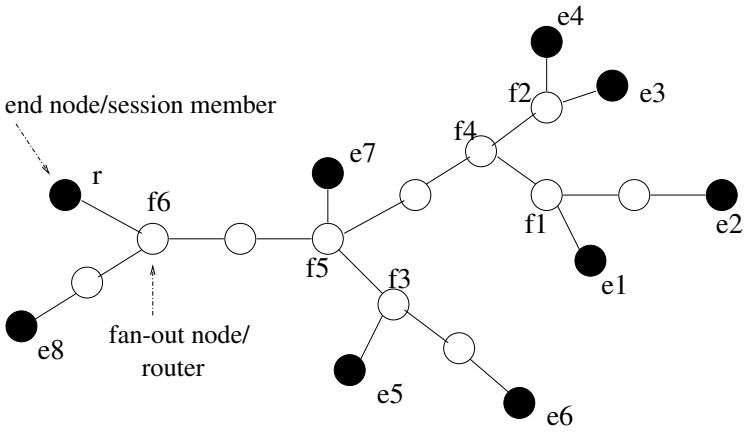


Figure 2.3: Example of a physical shared multicast tree.

Table 2.2: Full distance matrix for tree in Figure 2.3.

	r	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
r		(7,4)	(8,4)	(7,4)	(7,4)	(5,3)	(6,3)	(4,2)	(3,1)
e_1			(3,1)	(4,3)	(4,3)	(6,4)	(7,4)	(5,3)	(8,4)
e_2				(5,3)	(5,3)	(7,4)	(8,4)	(6,3)	(9,4)
e_3					(2,1)	(6,4)	(7,4)	(5,3)	(8,4)
e_4						(6,4)	(7,4)	(5,3)	(8,4)
e_5							(3,1)	(3,2)	(6,3)
e_6								(4,2)	(7,3)
e_7									(5,2)
e_8									

When a node *m* is connected to a link *l*, *m* and *l* are said to be *incident* on each

other. The number of links incident on a node *m* is called the *degree* of *m*. We say node *n* is *adjacent* to a node *m* if the nodes share a link.

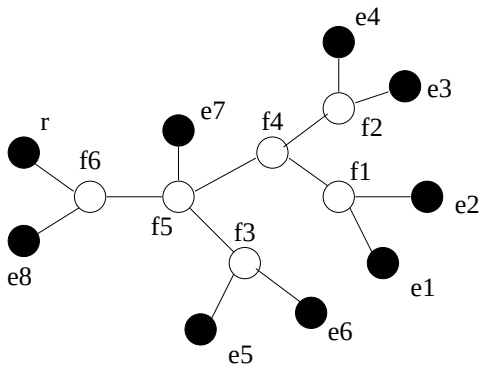


Figure 2.4: The logical tree for our example.

The *logical tree* associated with a physical tree is obtained by eliminating internal nodes whose degree is 2. For example, Figure 2.4 depicts the logical tree corresponding to the physical tree in Figure 2.3. The nodes in a logical tree can be partitioned into *end nodes* E, whose degree is 1, and *fan-out nodes* F, whose degree is at least 3. In the sequel we let $|A|$ denote the cardinality of a set A. For a fan-out node $f \in F$ we let AE_f denote the set of its adjacent end nodes in the logical tree. Thus in our example, $AE_{f_1} = \{e_1, e_2\}$. Fan-out nodes which have at least 2 adjacent end nodes and only 1 adjacent fan-out node in a logical tree, is said to be a *border* fan-out nodes. We let BF denote the set of border fan-out nodes in the logical tree. For example, in Figure 2.4, $BF = \{f_1, f_2, f_3, f_6\}$. The notion of a border fan-out node will be useful when we consider "reduced" distance matrices in 2.3.4.

Theorem 2.1 *A logical tree with at least two fan-out nodes has at least two border fan-out nodes, i.e., if* $|F| \geq 2$ *then* $|BF| \geq 2$.

Proof Consider one of the longest paths in the logical tree. Since $|F| \geq 2$, such a path must include at least two fan-out nodes. We argue that the nodes adjacent to the the end nodes of

the path must be border fan-out nodes. Suppose one of them is not a border fan-out node. Then there is another adjacent fan-out node which is not currently on the path. This means a longer path than the current one could be constructed and leads to a contradiction. ∎

Theorem 2.2 *A logical tree with $|E|$ end nodes has at most $|E|-2$ fan-out nodes.*

Proof This can be proven by constructing a tree which has a maximal number of fan-out nodes. First, note that a logical tree with $|E|+|F|$ nodes including end and fan-out nodes, has $|E|+|F|-1$ links. Thus total degree sum of all nodes in the tree becomes $2(|E|+|F|-1)$ since each link contributes 2 degrees. If we wish to construct a tree which has the maximal number of fan-out nodes in a tree, the degree of each fan-out node should be as small as possible, i.e., 3. The total degree sum of the tree will be then $3|F|+|E|$. Equating $3|F|+|E|$ with $2(|E|+|F|-1)$ gives $|F|=|E|-2$. ∎

Given an end node $r \in E$ we can consider the r-rooted logical multicast tree associated with a multicast session. We shall exhibit such trees with the root is at the top, and nodes that are equally distant from the root horizontally aligned at levels below it. Figure 2.5 depicts the r-rooted logical tree for physical tree in Figure 2.3.

With the introduction of the root, we can further partition the end nodes, E, and the fan-out nodes, F, according to their fan-out distances from the root. We let E_i represent the set of end nodes whose fan-out distance from the root is i. Similarly F_i denotes a set of fan-out nodes whose fan-out distance from the root is i. E_i and F_i are said to be at *level* i. Note that $i = 0, 1, \ldots, b$ where $b = \max_{m \in E} d_f(r,m)$. We define E_0 as $\{r\}$ and $F_b = \emptyset$. Figure 2.5 shows the example of such a partition of end and fan-out nodes.

If a node p immediately precedes node c on the path from the root to c, then p is the *parent* of c and c is the *child* of p. Nodes having the same parent are said to be *siblings*. We

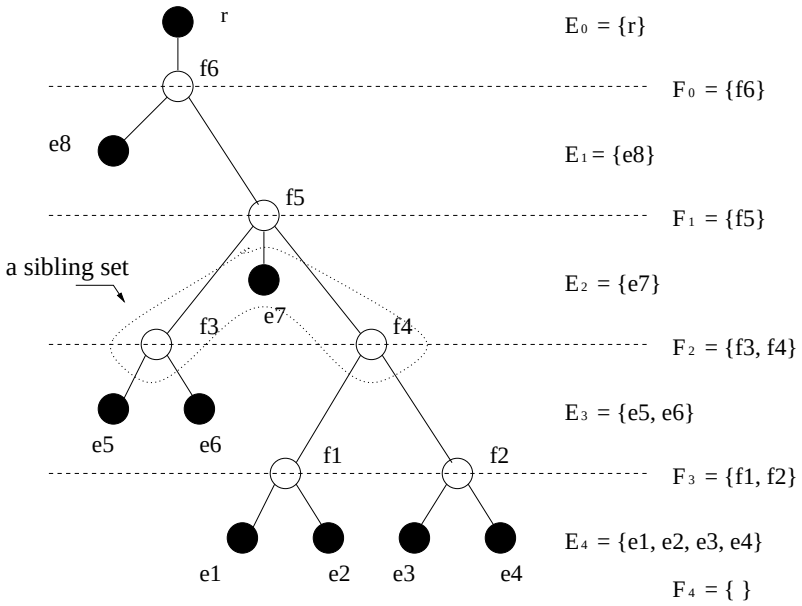


Figure 2.5: The *r*-rooted logical tree for our example.

let a *sibling set* denote an *exhaustive* collection of siblings sharing the same parent. Note that for a given rooted logical tree there are several types of siblings:

- Type 1 (*Mixed Siblings*): An end node *e* at level *i* can be the sibling of a fan-out node at level *i*.
- Type 2 (*Fan-out node Siblings*): Fan-out nodes at the same level can be siblings.
- Type 3 (*End node Siblings*): End nodes at the same level can be siblings.

In Figure 2.5, the sibling sets $\{f_3, e_7, f_4\}$, $\{f_1, f_2\}$ and $\{e_5, e_6\}$ exemplify these types of relations respectively.

A node *d* is said to be a *descendant* of a node *n*, if *n* is on the path from the root to *d*. Note that from the above definition, *n* can be its own descendant. Given a fan-out node $f \in F$, we define a *reference* node of f, denoted by $r(f)$ to be any end node which is

a descendant of f. Reference nodes will be used in checking sibling relations for fan-out nodes, since there is no explicit information for fan-out nodes in the distance matrix.

Note that the level ordering and filial relationships discussed above are always with respect to a given rooted logical tree. However, for simplicity we have not included the specified root in our notation.

2.3.2 Algorithm using the full distance matrix

In this section we discuss an algorithm to discover a tree given the full distance matrix. The algorithm includes two parts. Based on fan-out distances, one first discovers the logical tree, and then based on path distances, one determines the hop count lengths associated with links in the logical tree. The steps of the algorithm can be summarized as follows:

1. Logical tree discovery:
    - Select a root.
    - Perform a level ordering on end nodes, $E_i, i = 0, \ldots b$.
    - Perform bottom up discovery of $F_i, i = 0, \ldots b$ and sibling/parent relationships among nodes.
2. Physical tree discovery:
    - Perform bottom up discovery of path distances associated with the logical tree's links.

Below we outline the details associated with these steps.

Logical tree discovery

The first task is to select a root for the logical tree. In general any node could be selected, however since we intend the discovery algorithm to be carried out in a distributed fashion at each end node we shall assume without loss of generality that each end node considers itself to be the root of the tree. We let $r \in E$ be the root for our ongoing example. Next, we partition the end-nodes into sets $E_i, i = 0, \ldots b$, based on their fan-out distances from the root. This is done by checking r's row in the distance matrix.

The key task in the logical tree discovery step is to progressively identify complete sibling sets in a bottom up fashion. Note that each sibling set is associated with a unique, previously unknown, parent fan-out node at a higher level of the logical tree. Thus we can progressively determine not only $F_i, i = 0, \ldots b-1$ but the filial relations among the rooted tree's nodes. We shall start at the bottom, setting $i = b$. The key step will be at each level i, to discover complete sibling sets among E_i and F_i and create the associated set of parent fan-out nodes, F_{i-1}, at the next level. The following lemma will enable us to check whether two nodes in $E_i \cup F_i$ are siblings.

Lemma 2.1 Sibling Checking Lemma

1. *Suppose $e \in E_i$ and $f \in F_i$ then they are siblings iff*

$$d_f(e, r(f)) - d_f(f, r(f)) = 2.$$

2. *Suppose $f_a, f_b \in F_i$ then they are siblings iff*

$$d_f(r(f_a), r(f_b)) - d_f(f_a, r(f_a)) - d_f(f_b, r(f_b)) = 3.$$

3. *Suppose $e_a, e_b \in E_i$ then they are siblings iff $d_f(e_a, e_b) = 1$.*

The proof of the lemma is straightforward. In the first case, e and f are siblings iff $d_f(e,f) = 1$, so the lemma follows by noting that we can compute $d_f(e,f)$ based on f's reference node $r(f)$ as $d_f(e,r(f)) - d_f(f,r(f)) - 1$. In Figure 2.6 sibling nodes f_4 and e_7 exemplify this case. For the second case, note that f_a and f_b are siblings iff $d_f(f_a,f_b) = 1$. The lemma follows by computing this distance based on reference nodes for associated fan-out nodes, i.e.,

$$d_f(f_a,f_b) =$$
$$d_f(r(f_a),r(f_b)) - d_f(f_a,r(f_a)) - d_f(f_b,r(f_b)) - 2.$$

Siblings f_3 and f_4 in Figure 2.6 exemplify the second case. The final case is clear and can be easily checked using e_a's (or e_b's) row in the distance matrix.

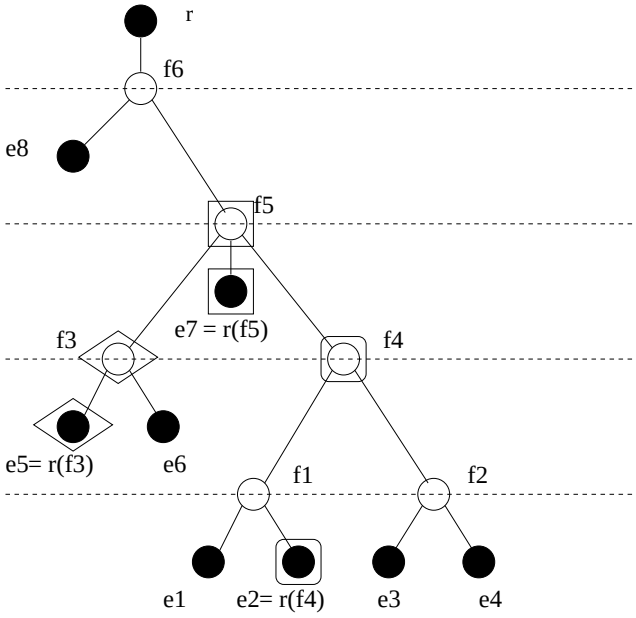


Figure 2.6: Illustration of sibling checking criteria.

In order to discover complete sibling sets among the nodes we let $C = E_i \cup F_i$ denote the set of nodes that need to be considered. Select any node $c_1 \in C$ and determine the set

of *all* of its siblings S_1, including c_1, by checking each of the remaining nodes in C using Lemma 2.1. Now let $C := C \setminus S_1$ and proceed iteratively until there are no more nodes in C. Suppose this process terminates after k steps, then k disjoint sibling sets $S_1, \ldots S_k$ are obtained. For each of these, generate a parent node f_j, $j = 1, \ldots, k$ and place it in the set F_{i-1} of fan-out nodes at the next level up. Also define the reference node $r(f_j)$ for each parent, f_j, to be any end node which descends from f_j. At this point one can proceed in discovering siblings and parents at the next level up. This procedure continues until the logical tree topology is determined.

Discovery of path distances of logical links

Once we have identified the logical tree, we need only to find path lengths associated with its logical links to determine the physical tree. The key idea is captured by the following lemma, which determines path distances of logical links between a border fan-out node $f \in BF$ and its adjacent end nodes AE_f.

Lemma 2.2 *Suppose* $f \in BF$, $m,n \in AE_f$ *and* $k \in E, k \neq m,n$ *then*

$$\begin{aligned} d_p(m,f) &= [d_p(m,n) + d_p(k,m) - d_p(k,n)]/2, \\ d_p(n,f) &= [d_p(m,n) - d_p(k,m) + d_p(k,n)]/2. \end{aligned}$$

The proof of this lemma follows directly by decomposing path lengths into their constituent components – consider Figure 2.7. Moreover for any additional node, $e \in AE_f \setminus \{m,n\}$, the path distance $d_p(e,f)$ can be computed to be $d_p(m,e) - d_p(m,f)$. Observe that to determine the lengths of the logical links from a border fan-out node f to all its adjacent end nodes AE_f we only require two rows of the distance matrix, where at least one is associated with one node in AE_f.

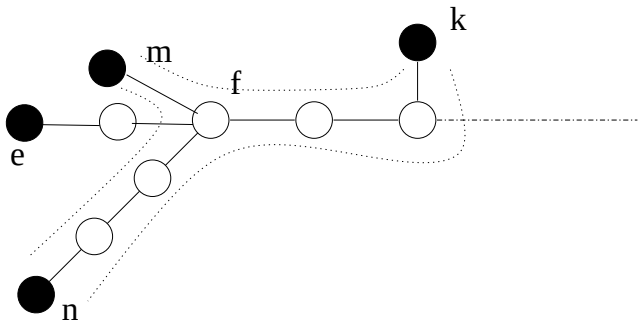


Figure 2.7: Path distance calculation at a border fan-out node f.

Note that for any rooted logical tree, if $f \in F_{b-1}$ then $f \in BF$. Thus by Lemma 2.2 all the lengths for logical links at the bottom level can be computed. In order to proceed systematically in a bottom up fashion, we propose to prune the tree and update the path distance matrix. At level i, all links and end nodes E_i whose distance to their parents have been computed are pruned. Then all fan-out nodes at level $i-1$, i.e., F_{i-1}, became end nodes at level $i-1$. In this pruned tree, all $f \in F_{i-2}$ are border fan-out nodes, which guarantees that the path distance calculation step can again be performed for level $i-1$.

As a result of pruning, the path distance matrix for the new tree must be generated. This is done by eliminating entries associated with all the pruned end nodes, and adding a new entry, for each fan-out node f that becomes an end node of the new tree. Table 2.3 is the path distance matrix for the pruned tree in Figure 2.8.

Table 2.3: Path distance matrix for the tree in Figure 2.8.

	r	f_1	f_2	e_5	e_6	e_7	e_8
r		6	6	5	6	4	3
f_1			2	5	6	4	7
f_2				5	6	4	7
e_5					3	3	6
e_6						4	7
e_7							5
e_8							

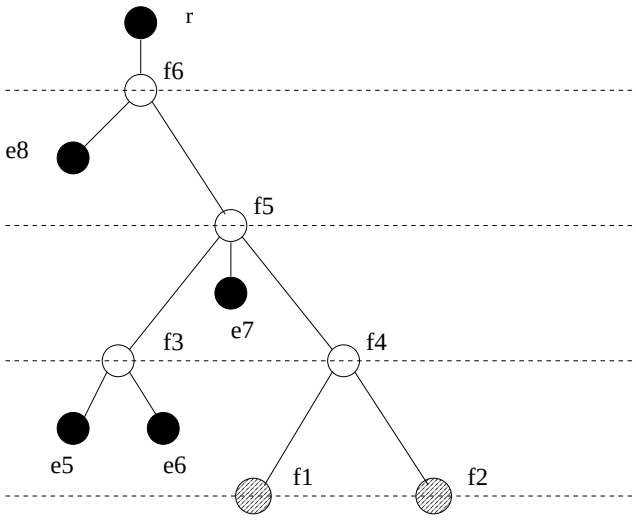


Figure 2.8: The pruned tree of Figure 2.5 at Level 4.

2.3.3 Computational complexity

The computational complexity for the proposed algorithm can be roughly evaluated as follows. The level ordering step is $O(|E|)$. The bottom up step in the logical topology discovery phase can be shown to be $O(|E|^2)$. Indeed there are at most $|E|-2$ fan-out nodes in the tree by Theorem 2.2 and determining siblings associated with each parent fan-out node has a cost of at most $|E|$. Path distance computations to obtain the physical topology are also quadratic. So the overall computational cost is $O(|E|^2)$.

2.3.4 Reducing the required distance information

There is in fact a large amount of redundant information in the distance matrix. This motivates us to ask the following question: What is the minimal required distance information in order to discover a tree? To answer this question, we will define our unit of information as an end node's entire row table which includes path/fan-out distances from the end node to all other end nodes in a tree. Let NE denote the set of end nodes whose row tables are

available when performing topology discovery. Our goal is to find a reduced set NE such that the topology of the multicast session can still be determined. Note that the algorithm described in Section 2.3.2 requires the full distance matrix, i.e., $NE = E$.

Theorem 2.3 *Given a shared multicast tree with $|F|$ fan-out nodes the following conditions on the set NE of available rows in the distance matrix are sufficient to allow topology discovery:*

1. *If $|F| = 1$ then $|NE| \geq 2$.*
2. *If $|F| \geq 2$ then NE should include at least one node in the set of end nodes AE_f associated with each border fan-out node $f \in BF$.*

Proof Consider the first case. If $|F| = 1$, the discovery of the logical topology is straightforward, i.e., all nodes are 1 fan-out distant from each other. This can be determined based on a single row table. Note that by Lemma 2.2 if two row tables are available, one can compute all path distances from a fan-out node to its adjacent end nodes. This establishes the condition for the first case.

Now suppose that NE includes one node from each set AE_f associated with border fan-out nodes $f \in BF$. We show that the logical topology can be determined as follows. Select any node $r \in NE$ as the root and perform a level ordering on end nodes based on r's row table. Note that during our bottom up phase, we will be able to assign a reference node in NE to each generated fan-out node, since every fan-out node in a rooted logical tree, has at least one border fan-out node as its descendant. This guarantees that all the required information is available to use Lemma 2.1 for sibling checking.

Next we show that subject to given conditions, the physical topology can also be discovered. Note that by Theorem 2.1, if $|F| \geq 2$ then $|NE| \geq 2$. Recall that by Lemma 2.2,

in order to know the path lengths associated with logical links from a border fan-out node, e.g., f, to its adjacent end-nodes, we only need two row tables of which at least one node should be in AE_f. Since NE contains at least one in AE_f, and $|NE| \geq 2$, all path distances to f can be computed. The path length computation can once again be carried out by pruning, starting from the bottom level to the top. ∎

Note that the computational complexity of topology discovery based on the reduced distance matrix remains $O(|E|^2)$.

2.4 Obtaining distance information

In this section we discuss implementation issues concerning how members of the multicast session can selectively acquire sufficient distance information to discover the topology of the multicast tree. The elements necessary in our proposed framework are:

1. Fan-out decrement mechanism.

2. Initial path/fan-out field in packets for allowing a receiving host to obtain distance information from the sender to itself.

3. *Bidirectional* shared multicast routing protocols, e.g., CBT and Border Gateway Multicast Protocol (BGMP) [24] for preserving path symmetry between members.

Note that TTL decrement mechanism operates on every IP packet. Similarly, we can envisage that the fan-out decrement mechanism could be applied to every multicast packet. However, this would need an additional fan-out field in the IP packet header while requiring modifications to all routers. Alternatively, the fan-out decrement mechanism can be implemented as a special feature in IGMP [22]. In this case, applications wishing to

use fan-out decrementing, will encapsulate their packets within IGMP packets. Then, the fan-out decrementing would be performed only when desired, i.e., not for every multicast packet. This new feature would be simple to implement and will incur fairly low overheads at routers.

To create shared multicast trees, *unidirectional* multicast routing protocol such as PIM-SM [25] might be used. However, note that PIM-SM is not applicable to our model since in *unidirectional* multicast protocols the sender's packet goes to the core first and then the core multicasts it to the others. Thus there is no way for each member to acquire other members' distance information. In contrast, in *bidirectional* multicast routing protocols, members can communicate with each other without going through the core since packets can travel both up toward the core and down from the core [21].

Assuming that the above requirements are satisfied, first, we discuss how each member can obtain the full distance matrix. Suppose every member periodically multicasts a *heartbeat* packet to the whole group. The role of the heartbeat packet is two-fold: 1) it serves as an indication of the liveness of the sending host, which is necessary if the algorithm is to adapt to changing membership or topologies; and 2) it enables receiving members to obtain their fan-out/path distances from the sender. Note that senders which persistently multicast data packets to the session may not need to send heartbeat packets, as long as initial values for the TTL and fan-out fields are included in the IP multicast packet's header. Whenever a member receives a heartbeat from other members, the member can build/update its row in the session's distance matrix, where each member is identified by its IP address. In addition to periodically sending heartbeat packets, each member becomes a *reporter* and periodically multicasts a *report* packet to the session which contains its own row table. Thus, eventually each session member would have access to the full distance

matrix.

Theorem 2.3 suggests that it would suffice for only one node among adjacent members of each border fan-out node to generate report packets. The above approach has two advantages over the full distance distribution method. First, it reduces the number of reporters in a session, which results in significant reduction of communication overheads since report packets can be large relative to heartbeat packets. Second, it can also reduce memory storage space required at end-hosts. In order to enable this type of reporting, one must however identify border fan-out nodes, and then select a unique reporter for such a node. This in itself requires that the network topology be known a priori, which is not practical.

As a compromise between full distance matrix distribution and the impractical second approach discussed above, we propose the following rules to determine which end hosts should serve as reporters:

Rule 1: A member will serve as a reporter if there is at least one other member which is 1 fan-out distant from it and it has the smallest IP address among members within 1 fan-out distance.

Rule 2: A member will serve as a reporter if all other members in a session are 1 fan-out distant from it and it has the largest IP address among members within 1 fan-out distance.

Note that the first rule guarantees that there will be a reporter selected from set of adjacent members to a border fan-out node – there may also be some additional reporters. The second rule ensures that if the tree has but one fan-out node, there will be at least 2 reporters. Thus with these two rules enforced, the sufficient conditions stated in Theorem 2.3 will be satisfied.

Note that these rules can be applied by nodes in a decentralized fashion in that they need only to check their own row table without any computation. This approach would of course reduce network traffic to acquire the required distance information. Also note that in this context the minimum number of reporters is 2 while the the maximum number of reporters is $\lfloor |E|/2 \rfloor$.[4] In general the communication complexity to acquire the distance matrix would be $2|E|$ multicast messages, i.e., a heartbeat and report packet per session member, where the size of heartbeat packets is $O(1)$ while that of reports is $O(|E|)$.

2.5 Local topology discovery framework

If a multicast session involves a huge number of members, the proposed global topology discovery scheme may not be workable. In particular, the communication, computation and storage overheads may be unwarranted.

Note also that since each row in the distance matrix contains each members' IP address, for large multicast trees this may include a lot of data, eventually requiring reports to be partitioned across several packets.[5] Moreover, in a large scale multicast session, members may not be interested in discovering the entire distribution tree. Instead they may only be interested in a local view of the multicast tree's structure. This is, for example, the case in the context of applications for local loss recovery where members only wish to identify other members within a given neighborhood. Thus it would be advantageous if the proposed framework could also be used to discover a restricted local topology while reducing the overheads associated with acquiring this information.

[4] The notation, $\lfloor \rfloor$, is a floor operator.

[5] In order to reduce communication overheads, one might consider reports that include only incremental changes in data. This must, however, be done with care in a dynamic scenario as new members need to eventually acquire sufficient information to discover the tree.

2.5.1 Concept

Let us consider an instance of this problem for a session member $r \in E$. Let a *neighborhood* N_r be the set of members that share a particular attribute, including r itself. Note that there is quite a bit flexibility in defining N_r. For example the neighborhood could correspond to FN_r^k, the set of members within the k fan-out scope from r including r itself, or the set of members that serve as DNS servers and are in FN_r^k. Given such a neighborhood, we define the *induced physical and logical trees* as follows.

Definition 2.1 *Given a neighborhood $N_r \subset E$ of a node $r \in E$ in a multicast tree, we let the* N_r induced physical tree *be the subtree connecting r to the members of its neighborhood N_r. We define the* N_r induced logical tree *as the logical tree associated with the N_r induced physical tree.*

For example, consider the neighborhood $N_r = \{r, e_5, e_8\}$ of r in a physical multicast tree shown in Figure 2.9. The region that has been outlined corresponds to the physical tree induced by N_r while Figure 2.10 depicts the N_r induced logical tree.

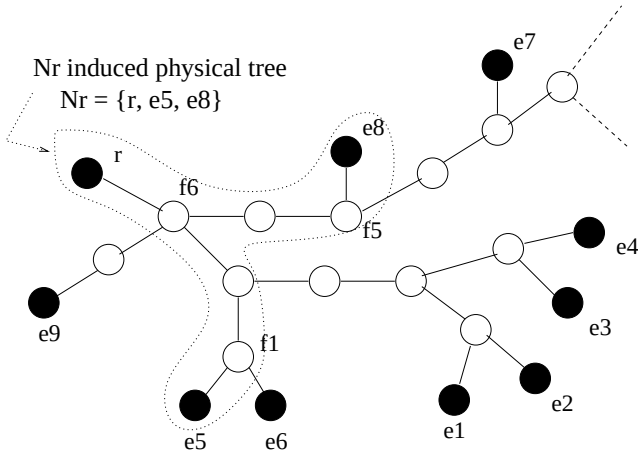


Figure 2.9: A physical multicast tree.

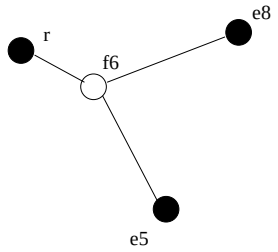


Figure 2.10: The N_r induced logical tree. ($N_r = \{r, e_5, e_8\}$)

Note that an N_r induced logical tree simply shows the logical relationship among members in N_r, and it might include logical links that hide fan-out nodes in the global multicast tree. For example, the logical link from f_6 to e_5 in Figure 2.10 actually represents 3 physical links and 2 fan-out nodes.

Definition 2.2 *Given a neighborhood $N_r \subset E$ of a node $r \in E$ in a multicast tree, the* local multicast topology discovery of N_r *is defined as determining the N_r induced logical tree topology, as well as path/fan-out distances for its logical links.*

Local topology discovery can be based on an N_r *restricted* distance matrix including only row and column entries associated with the nodes in N_r. This problem can be viewed as a restricted version of the global topology discovery problem presented in Section 2.3. It is relatively easy to see that one can, with some care, apply the same methods developed for global topology discovery in this context.

We propose to perform local topology discovery by first determining the N_r induced logical topology applying the algorithm in Section 2.3.2. In this step, we in fact determine the subtree induced by N_r on the *global logical topology*. This is illustrated in Figure 2.11 for the local topology discovery problem associated with FN_r^3 in the multicast session Figure 2.9. Note that the subtree enclosed in the dashed line need not be the desired N_r induced logical tree. In particular, the subtree obtained by using our previous algorithm on

the restricted set may include fan-out nodes, e.g., f_3 and f_4 in the above example, which would not be part of the N_r induced logical topology, see Figure 2.12. Once such nodes are pruned, the structure of the N_r induced logical tree has been discovered along with the fan-out distances associated with its logical links.

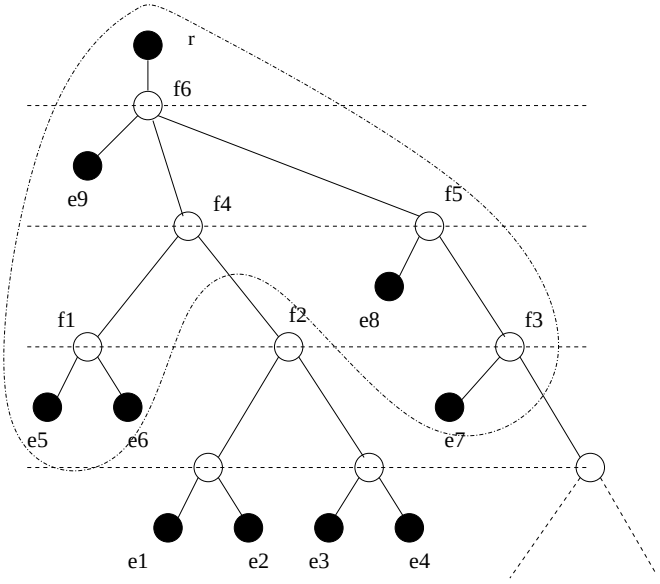


Figure 2.11: The *r* rooted logical tree of Figure 2.9.

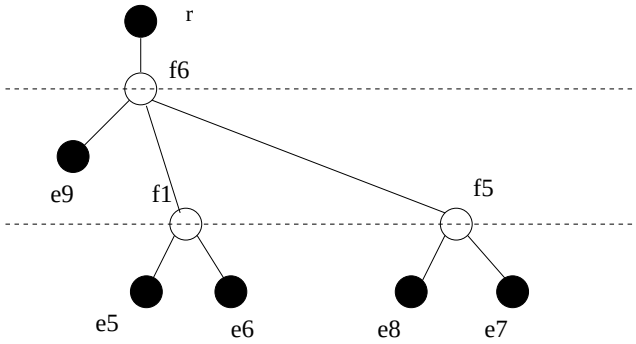


Figure 2.12: The FN_r^3 induced logical tree.

Next, based on Lemma 2.2, one can identify the path distances of the logical links in the N_r induced logical subtree. Note that certain path metrics would not, and in fact can not, be identified based on the N_r restricted distance matrix. For example, node f_4 is not present in the induced logical subtree, and thus the path lengths f_1 to f_4 and f_4 to f_6 would not be determined, however the overall path metric associated with the logical link from f_1 to f_6, can be identified.

In summary, discovering an N_r induced logical tree's topology and the associated logical links' distances requires basically the same steps as discussed for the global case. It should be clear that the computational complexity of local topology discovery is quadratic in the size of the neighborhood, and storage requirements would also depend on the size of the the neighborhood.

In principle a neighborhood can be any set of members sharing a particular attribute. However, below we will focus on local topology discovery, i.e., that associated with neighborhoods having spatial proximity on the multicast tree. Thus we will define both fan-out and TTL scoped neighborhoods for a given node. We let TN_r^l denote the set of members in a multicast group that are within an l limited TTL scope from r including r itself. In general one can define a jointly scoped neighborhood, e.g., $N_r = FN_r^k \cap TN_r^l$, for each node in a network and proceed to discover the induced logical trees based on restricted distance matrices.

2.5.2 Obtaining restricted distance matrices

The remaining question is how each node would acquire the restricted distance matrix associated with its k fan-out and l TTL scoped neighborhood. Depending on the application, we can envisage the following two cases for local topology discovery. For some collaborative

applications, *every* member in a session may need to have a local view of its neighborhood, each with the same uniform k fan-out and l TTL scope. By contrast, other applications might only require *some* nodes to acquire their own local topology associated with possibly heterogeneous fan-out/path scopes. Considering the above two cases, here we propose two schemes for acquiring the restricted distance matrix.

Uniform local topology discovery

The goal of the first scheme is to enable *each* member, say r, to discover its neighborhood, $N_r = FN_r^k \cap TN_r^l$ with a uniform k and l. The following simple protocol suffices:

1. Each member periodically sends heartbeat packets with the fan-out scope set to $2k-1$ and the TTL scope set to $2l-2$.

2. Each member periodically sends a report packet with k and l set as the fan-out and TTL scopes respectively.

The idea underlying this scheme is quite simple. First, each node r should receive reports from all members of its neighborhood, thus report packets should be scoped as indicated above. Second, since an N_r restricted distance matrix contains path and fan-out distances among all pairs of members in N_r, they have to know of each other's existence and the associated distances. Note that $2k-1$ and $2l-2$ are the maximum possible fan-out and TTL distances between members in $N_r = FN_r^k \cap TN_r^l$. Thus it should be clear that the proposed fan-out and TTL scopes on heartbeat packets ensure that the N_r restricted distance matrix acquired by a node r is complete. Note that we have assumed an a priori uniform selection of k and l for all nodes. This poses the question of how they might be 'optimally' chosen and whether they might be selected in a non-homogeneous decentralized fashion.

This would of course depend on applications.

Assuming nodes share information in this fashion, one can significantly reduce the communication overhead associated with topology discovery, in terms of the number of heartbeat and report packets seen on *any* link in the multicast tree and the size of the report packets. Indeed, although the same total number of packets, $2|E|$, will be sent as in the global discovery case, these packets are scoped and hence will not be seen by all links and members. In particular, a rough estimate for the number of messages seen by a member would be the size of its $2k-1$ fan-out and $2l-2$ TTL scoped neighborhood. Similarly the size of report packets would is no longer be $|E|$ but proportional to the size of the neighborhoods.

Non-uniform local topology discovery

The above scheme may incur heavy communication overhead in the case where topology information is not frequently required and not necessary for all nodes in a session. In such cases, we propose the following scheme which allows a single node to discover its local topology within a predefined fan-out/path distance when desired. To do so, we introduce a 32 bit *requesterID field* in heartbeat packets. Depending on the content of the field, we can classify heartbeat packets into two types: a *request heartbeat* or a *normal heartbeat*. In a request heartbeat, the requesterID field is set to 0 by a *requester* which wants to discover its local topology. A normal heartbeat is generated by a *responder*, i.e., a node that receives a request heartbeat. When a responder generates a normal heartbeat, it places the requester's IP address in the requesterID field.[6]

[6]Note that when a node receives a heartbeat packet, it can determine if the heartbeat packet comes from the requester by checking if the requesterID field is set to 0, and if so, it also can extract the requester's IP address from the source address of the heartbeat packet.

Let r be a requester and suppose its aim is to acquire its restricted $N_r = FN_r^k \cap TN_r^l$ distance matrix. We propose the following mechanism:

1. A requester, r, multicasts a request heartbeat setting k and l as scoping fan-out and path distance parameters respectively.

2. Each responder of r, say a, multicasts a normal heartbeat with its fan-out scope set to $d_f(r,a)+k-1$, TTL scope set to $d_p(r,a)+l-2$, and r's IP address in its requesterID field.

3. Based on the requesterID fields of the received heartbeat packets, each responder of r builds its own row table whose entries are composed of r and responders of r.

4. Each responder of r unicasts a report packet to r.

In this scheme when there is no requester, no traffic is injected into the network, which in turn, may enormously reduce communication overheads. Note that the scoping parameters in Step 1, suppress packet injection from other members but only r's responders, which indeed are members of r's neighborhood. For example, consider the case where there is one requester, r, in Figure 2.13. Only a, q and r will generate packets while b and o will discard the normal heartbeat packets from r's responders, e.g., a or q.

In Step 2, the selection of scoping parameters of each responder is such that fan-out/path distance information among all pairs of members in N_r is eventually obtained. It is clear that by setting the fan-out scope to $d_f(r,a)+k-1$ and the path scope to $d_p(r,a)+l-2$, a's heartbeat packets can reach all members in N_r - see Figure 2.14 for the fan-out distance case. This choice of scoping parameters instead of $2k-1$ and $2l-2$ further suppresses the scope of the normal heartbeat and thus minimizes the communication overheads.

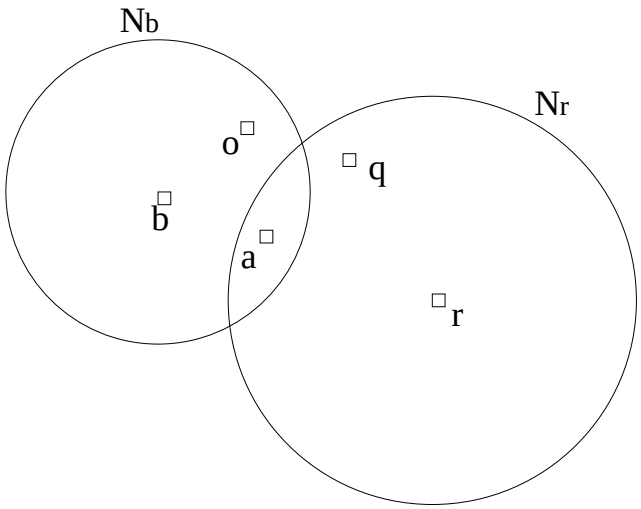


Figure 2.13: Illustration of multiple requesters.

Note that in Step 3, each responder of *r* would include only data associated with nodes in *r*'s neighborhood. In particular, the requester field enables a responder to distinguish more than one ongoing topology discovery attempts and makes our proposed scheme work well even in the presence of multiple requesters whose neighborhoods overlap. For example, suppose that there are 2 concurrent requesters *r* and *b* in Figure 2.13. In this case *a* will multicast two differently configured normal heartbeats associated with *r* and *b*. Also note that when *a* builds a report packet to *b* , it would not include *q* entry since *q* is not a responder of *b*. This feature keeps the size of a report packet proportional to the size of the requester's neighborhood.

Lastly, the report will be sent to *r* via unicast further reducing communication overheads.

In addition to the basic mechanism described above, some more detailed issues need to be addressed. In the above scheme we make two assumptions: 1) there is no packet loss, e.g., heartbeats or reporters and 2) each responder knows when it has received all the

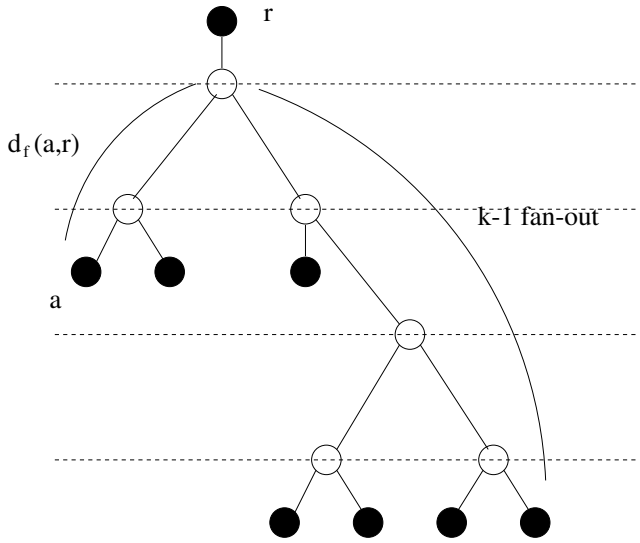


Figure 2.14: Fan-out scoping parameter selection.

required information to build its own row table. A simple way to handle these problems is to repeat the above scheme several times. Then eventually a requester can obtain its restricted distance matrix.

2.6 Annotated Trees

So far, the main role of heartbeat packets is to enable each host to obtain fan-out/path distance information from the others. In this section, we briefly describe how additional information in a heartbeat can be beneficial for some one-to-many multicast applications, e.g., local loss recovery or locating approximate problematic links.

Consider the case that one sender persistently multicasts packets and each receiver can evaluate its own performance metrics, e.g., packet loss rate or bottleneck bandwidth [12]. By combining these performance values exchanged through heartbeats with the proposed topology discovery scheme, one can obtain an *annotated tree*, i.e., a tree whose leaves

have associated performance metrics. Note that obtaining annotated trees only slightly increases the size of heartbeat packets with no additional packet exchanges. Figure 2.15 exhibits an example of a tree annotated with packet loss rate.

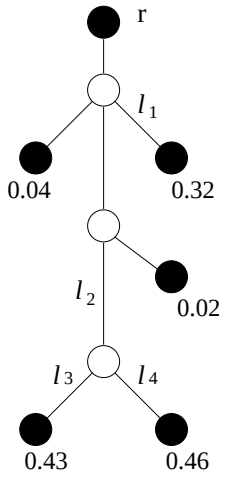


Figure 2.15: A tree annotated with packet loss rate.

This tree might be useful to a local loss recovery mechanism in determining a "good"(close and capable) *helper* from which a node can obtain lost packets [5]. Furthermore, this tree can also help to approximately locate the problematic links [12].[7]

For example, based on the annotated tree in Figure 2.15, it is easy to determine that a link l_1 is seeing a high degree of packet loss. However, it is not clear how to differentiate the quality of links, l_2, l_3, l_4 since there are several scenarios are possible, e.g., 1) only l_2 is bad or 2) both l_3, l_4 is bad etc. If accurate estimation of each links' packet loss rate is desired, the approach in [26] could be used with the topology information provided by our framework.

[7]Here note that considered links are logical.

2.7 Related Work

In this section, we discuss the pros and cons of existing work on multicast distribution tree discovery and the approach proposed in this chapter. Our intent is to find in which environment each approach fits best by identifying its advantages and shortcomings rather than arguing the superiority of our approach over existing ones. Existing approaches to multicast distribution tree discovery can be classified into two types: those based on end-to-end measurements [14], [12], and those requiring the help of intervening network nodes [11].

The key idea underlying the first approach is that receivers sharing common paths on the multicast tree associated with a given source will see correlations in their packet losses. Thus based on the shared loss statistics for transmitted probe packets one can attempt to infer the multicast tree. This elegant approach to the problem is particularly advantageous in that it requires no support from internal nodes. However, since this approach is based on the loss of packets, a source needs to send a large number of *probe* multicast packets even if the goal is to discover the topology of a small scale multicast tree. The lower the packet loss rate for links is, the larger the number of probe packets is needed. Furthermore, it potentially suffers from significant communication overheads required to periodically gather large amounts of loss data so as to adapt to changing memberships or topology, and processing overheads to assemble and perform the inference step. This is currently conceived as a centralized approach whose accuracy is unlikely to scale nicely. The approach assumes network links have steady state loss characteristics, which may or may not be realistic on the time-scales during which loss data are collected. A final point is that the approach permits identification of the logical multicast topology rather than the actual physical topology. This means that a session member that is at the end of a long path

with no intervening fan-out points, would see this section of its path collapsed to a single logical link. In practice this may or may not be an appropriate abstraction of the actual topology. The key advantage of this approach lies in its applicability to inferring multicast trees without requiring modifications to, or the help from, internal nodes.

Compared to the first approach, our approach has a number of advantages. To name a few, the communication overhead is low since it requires at most $2|E|$ multicast packets of size $O(|E|)$. Its computation complexity is low as much as $O(|E|^2)$ and it is quickly adaptable to tree changes since distance information will be immediately seen by heartbeat packets.

The second approach to multicast topology discovery which has the above-mentioned desirable characteristics is based on using the MTRACE feature currently implemented in the IGMP protocol [22]. MTRACE enables tracing the path from a source to a destination on a given multicast distribution tree [11]. A query packet is sent from the requester to the last multicast router (on the distribution tree) prior to a given destination. This query is then forwarded hop-by-hop along the reverse path from the "last-hop" router to "first-hop" router, i.e., that to which the source is attached. While the query packet traverses the tree, each router adds a response data block containing its interface addresses and packet statistics. When the query packet reaches the first-hop router it is sent back to the requester via unicasting or multicasting.

Note that an MTRACE query provides full information, i.e., interface addresses and performance characteristics, but *only* for *one* path from a multicast source to a given destination. Thus if all members wish to know the full multicast topology for a given source, each receiver would send a query packet to its last-hop router, and query responses should be *multicast* to the entire group. Then the reconstruction of the full multicast topology is

achievable since each packet includes a stack of interface addresses for nodes along the path from the source and the destination. Note that all query traffic would visit the first-hop router which would in turn generate multicast responses. Due to this focussed load, in a large-scale multicast session, this approach may not scale. By comparison, in our approach, there is no single focussed, or central point, which leads to a more decentralized mechanism. Key advantages of the second approach are that it provides full information on the multicast topology based on currently available IGMP features.

In contrast, our approach is based on introducing a new fan-out decrement mechanism in IP multicast, which is not currently available. However, as pointed out in Section 2.2, it is simple to implement and provides a generic service which has broad applicabilities, i.e., not only topology discovery but also efficient scoping within IP multicast context. Furthermore, by implementing this as a special feature of IGMP, as proposed in Section 2.4, fan-out decrementing need only be supported when needed, thus incurring low overheads at routers. Note that this overhead at routers may be 'lighter' than that of MTRACE since MTRACE inserts each interface's address as well as packet loss statistics.

Note that while the first approach is strictly based on using end-to-end measurements, the second relies heavily on special services at routers, thus from the perspective of required network support these are two extremes of the spectrum. Also the first approach identifies the logical topology while the second determines the physical topology including interface addresses of routers. Note that our approach lies somewhere in their midst, requiring light weight cooperation from multicast capable routers (i.e., fan-out decrementing) and cooperation among members in the session to identify the physical topology (without internal interface addresses).

One limitation of our approach lies in its narrow applicability to bidirectional shared

multicast routing protocols since it requires a path symmetry property among members. However, bidirectional shared multicast routing protocols are likely to become increasingly crucial, as a number of large scale multicast applications are emerging. First, it is generally considered that shared tree routing is more efficient than source tree routing for large scale multicast applications such as distributed interactive simulations(DIS) [27] where each member is both a sender and a receiver. This is because source tree routing maintains source as well as group specific state information at routers. Second, once shared tree routing is determined to be used, unidirectional routing protocols are inefficient for multicast scoping and communications among neighborhoods since every multicast packet should visit the core in first. The larger are multicast sessions and the more is the demand for local resource discovery, the larger communication overheads will be incurred in unidirectional shared multicast routing protocols. Reflecting these observations, the long term inter-domain routing solution, Border Gateway Multicast Protocol(BGMP) [24] currently under development, constructs bidirectional shared trees.

Finally note that while existing work focus on *global* multicast *topology* discovery, our approach provides a general framework for *resource* discovery within a session and its associated *topology* discovery, which allows not only *global* but also *local* topology discovery. This comes from the fact that our approach is based on interactions among members.

2.8 Conclusion

In this chapter, first we propose an algorithm, which can discover the topology of the shared multicast tree based on a full distance matrix, with the analysis of computational complexity. Second, we provide sufficient conditions to achieve the same result with a reduced

distance matrix. Third, we show how reduced distance information could be acquired efficiently by exchanging a small number of multicast packets with an analysis of explicit communication overheads, i.e., the minimal number of packets injected in the network and the size of the packets. Forth, we consider concepts in the context of local topology discovery enabling nodes to discover the distribution tree within their fan-out and TTL scoped neighborhoods. Furthermore, we discuss practical issues for acquiring distance information in both uniform and non-uniform manner. Finally, we present an annotated tree concept for possible applications, e.g., identification of congested links.

Chapter 3

Filtering and Tracing Service for Defeating DDoS attacks

3.1 Introduction

Denial of Service (DoS) attacks are one of the greatest threats to today's Internet. They not only degrade performance but deprive legitimate users of basic access to network services. As seen in frequent news headlines attacks are becoming increasingly prevalent and evolving since the first spectacular attack on high-profile web sites Feb 2000 [28].

Despite their diverse character, such attacks share a common feature: they exploit defects or weaknesses of various network components ranging from applications, operating systems to protocols. Attacks using implementation defects or bugs in network components, can be prevented by frequent system updates and software patch work. However, it is much harder to thwart attacks which exploit intrinsic vulnerabilities and characteristics of the existing IP infrastructure.

For example, in IP, irrespective of a receiving side's intent, any host can basically send packets to any other host provided that those hosts are connected to the Internet. This simple feature of IP allows attackers to launch DoS attacks by simply inundating victims with large amounts of useless traffic. In turn, such traffic consumes network resources along the way to victims and eventually degrades performance for other users sharing these network resources. In a distributed DoS (DDoS) attack, i.e., one which is carried out by multiple compromised hosts, the damage can become exceedingly detrimental.

Moreover, IP has no mechanism for checking or controlling the correctness of the sender's address. This facilitates *spoofing*, i.e., concealing the true origins of packets by placing incorrect source IP addresses in them. Furthermore, it is difficult to identify the physical locations of attacks in an IP network due to its stateless nature, i.e., routers forward packets based on destination addresses alone and maintain no state information on traffic flows. The identification of the origins of attacks is even more difficult in the case of a DDoS attack where attackers may inject multiple, identical packets at multiple locations.

One way to deal with such attacks is *proactive* filtering [29], [30]. The key idea is to configure routers to drop spoofed packets whose source IP addresses are inconsistent with the network topology. Note that the strength of this approach is its proactiveness, i.e., attacks can be eliminated before they affect victims. However, if DoS attacks are infrequent, most resources allocated to proactive filtering are wasted except when spoofed packets are actually dropped. Furthermore, attackers may evade proactive filtering by forging IP addresses using hundreds or thousands of legitimate host addresses within a given domain.

By contrast, in this chapter our focus is on two *reactive* approaches: *on-demand filtering*[1] and *tracing*. Our approach is *reactive* in that actions are initiated after an attack

[1]Throughout the chapter, we omit 'on-demand' when there is no confusion with proactive filtering.

reaches a victim. Both tracing and on-demand filtering mechanisms can make up for the above-mentioned deficiencies of IP networks. That is, a tracing mechanism can identify the true origins of an attack and a filtering mechanism can enable a host to request unwanted packets to be dropped early on, before they reach the victim.

The critical innovation in DDoS attacks is its distributed nature. Even a small number of attack packets from each compromised host can eventually become a large traffic flow, inundating a target system. We argue that the solutions to thwart DDoS attacks should be also *distributed* to be effective. The *local* solutions on the victim computer or in its local network without outsider's cooperation, can neither identify where the packets are coming from nor effectively mitigate the possibly large volume of attack traffic.

The following are some desirable characteristics that filtering and tracing mechanisms should have.

- *Scalability:* Mechanisms should be scalable to benefit a *global* cooperation across a number of different administrative domains.
- *Promptness:* Filtering and tracing should be performed quickly before the victim is seriously damaged or there no longer exists a trail of information.
- *Flexibility:* Mechanisms should allow heterogeneous equipment and proprietary operation across different administrative domains.
- *Distributed:* From the perspective of robustness, it is preferable that mechanisms be implemented in a distributed manner rather than relying on central points.

In this chapter we consider two separate goals, filtering and tracing, and propose a framework to aid in thwarting DDoS attacks. For filtering, the objective is to block attack packets as close as possible to attack origins by way of filtering components that are distributed over the Internet. Under a *partial* deployment environment where only a subset of

routers are tracing-enabled, it becomes impossible to pinpoint the precise origins of attack packets. Thus our objective for tracing is to identify sets of candidate nodes containing attack origins.

The proposed solutions are based on IP multicast service to achieve a number of desirable characteristics, e.g., scalability, distributedness, quick response and robustness. Furthermore, they rely on existing monitoring and filtering mechanisms, allowing heterogeneity from different network domains.

The chapter is organized as follows. Section 3.2 introduces components and attack models used throughout the chapter. In Section 3.3 and 3.4, we discuss our objectives and our solutions for filtering and tracing mechanisms respectively. Section 3.5 includes a performance evaluation for the proposed framework and is followed by Section 3.6 wherein we include comments on implementation and various possible modes of operation. In Section 3.7, we discuss the advantages and shortcomings of previous approaches to defeating and mitigating attacks and contrast these with our work. Section 3.8 concludes the chapter.

3.2 Models

3.2.1 Attack Model

Consider an attack whose target is a node, v, referred to as the *victim*. A victim can be an end host, a router or a network border device such as a firewall. In the sequel, we will refer to the set of *attack nodes* A as those participating in the attack. We will leave the initiation time of the attack unspecified. Thus, the tuple of the set of attack nodes and the victim, i.e., (A, v) represents an attack incidence. In the case of a distributed attack, $|A|$ is greater than 1 and the locations of attack nodes are potentially widespread. For each

attack node, say, $a_i \in A$, the *attack path* for a_i is an ordered list of nodes traversed by attack packets from a_i to v, excluding a_i itself. An *attack graph* induced by (A,v), denoted by $AG_{(A,v)}$, consists of all links and nodes traversed by attack packets associated with nodes in A and the victim v.[2] For example, in Figure 3.1, (r_3, r_4, r_5, r_6, v) is a_3's attack path and the dotted lines represent an example of the attack graph for attack incidence $(\{a_1, a_2, a_3\}, v)$. Throughout this chapter, an *attack origin* is referred to as a local router to which the attack node is attached. For example, r_1 is the attack origin of a_1 in Figure 3.1. We will refer to an *attack signature* $AS_{(A,v)}$ as a common feature shared by attack packets generated from A. For example, for a smurf DoS attack case, an attack signature could be 1) ICMP echo protocol and 2) a range of source IP addresses [31].[3]

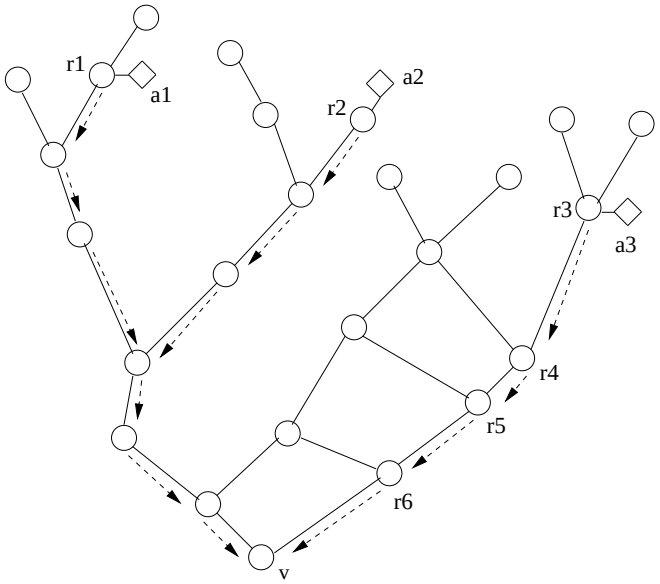


Figure 3.1: An example of an attack incidence.

[2] Attack paths and graphs may vary during the attack period due to routing instabilities and dynamic attack patterns.

[3] A smurf attacker sends a stream of ICMP echo requests to the broadcast address of the reflector subnet. Since the source addresses of these packets are falsified to be the address of the target, many hosts on the reflector subnet will respond, flooding the target. Thus, the source addresses of the echo reply packets are clustered in a few address prefixes.

3.2.2 Framework Model

Our framework includes the following components.

• *Detecting component:* Reactive approaches to DoS attacks require a component which can detect incidences of attacks and generate an attack signature by extracting a feature shared by attack packets. This attack signature is further used in the tracing and filtering processes.

• *On-demand filtering component:* An on-demand filtering component drops packets conforming to a rule set derived from an attack signature.

• *Tracing component:* When a tracing component is queried for a given attack signature, it can check for the existence of packets with the attack signature in the past or current traffic traversing the component.

There are various available, or proposed, implementation methods for each component, which will be described in Section 3.6.1. Note that it is not our intent to propose specific or new implementation methods for the above components. Our focus is on allowing heterogeneous mechanisms to cooperate so as to provide a flexible global filtering and tracing service.

We refer to entities which provide detecting, filtering and tracing services to victims as *detector*, *filter* and *tracer* which are equipped with detecting, on-demand filtering and tracing components respectively.[4] Note that each entity is associated with a location of interest. That location may be an end host, a router or a link depending on the implementation. Without loss of generality, throughout the chapter, we assume that filters and tracers are located at internal nodes (i.e., routers), and detectors are co-located at victim nodes.[5] In

[4]Multiple functionalities can be co-located and perform multiple roles.

[5]Usually detector nodes, D, will be located at highly-defended and secured strong points such as high-profile web servers or network entry points. Filter nodes, F, and tracer nodes, T, could be located at network entry points or distributed all over the Internet.

our framework, we assume a *partial* deployment of components over the Internet, i.e., only a subset of nodes are equipped with detecting, filtering and tracing functionalities – this is deemed a realistic environment.

Given a particular attack incidence (A, v), any set of nodes S can be partitioned into two subsets: *positive* and *negative* nodes, where positive nodes are on the attack graph of (A, v) and negative nodes are not. That is, positive nodes, denoted by $S_{(A,v)}$, are given by $S \cap AG_{(A,v)}$. For a given set of nodes S and $a_i \in A$, we define a *boundary* node, $b(a_i)$ as the node in S that is first encountered by attack packets making their way towards a victim v. We refer to a *boundary interface* as the interface where attack packets enter in a boundary node. We denote by $BS_{(A,v)}$ the collection of boundary nodes associated with an attack incidence (A, v). Letting S be a set of filter nodes, F, or a set of tracer nodes, T, we can obtain sets of corresponding to positive, negative and boundary filter (tracer) nodes associated with a given attack incidence.

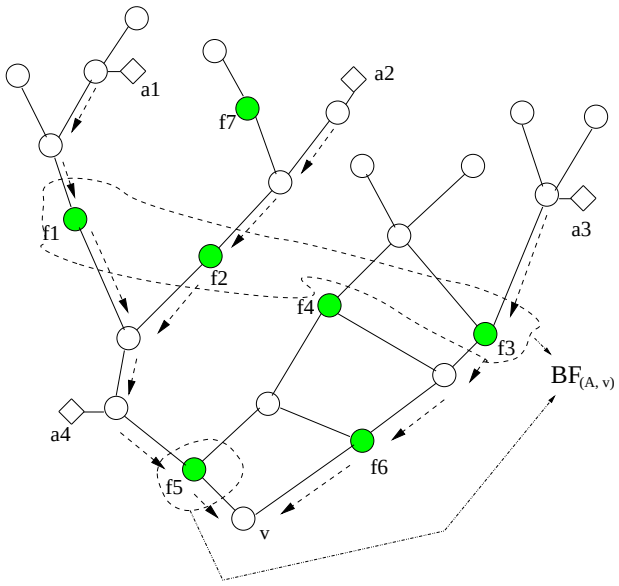


Figure 3.2: An example of filter deployed network.

For example, in Figure 3.2 the filter nodes (shadowed nodes) have been deployed in the network shown in Figure 3.1. For an attack incidence $(A, v) = (\{a_1, a_2, a_3, a_4\}, v)$, $F_{(A,v)}$ = $\{f_1, f_2, f_3, f_5, f_6\}$, $b(a_1) = f_1$ and $BF_{(A,v)}$ = $\{f_1, f_2, f_3, f_5\}$.

3.3 On-demand Filtering

3.3.1 Objective

The objective for on-demand filtering considered in the chapter is to block attack packets as close as possible to the attack nodes by using a set F of cooperative filter nodes distributed over the Internet. One might consider a centralized solution wherein each detector maintains and queries all the filters, F, over the Internet. However, this approach may incur high overheads and seems to scale poorly when there is a large number of filter or attack nodes. We observe that boundary filter nodes play a key role in achieving the objective, since they are the filtering nodes first met by attack packets from A to v. That is, performing filtering only at boundary filter interfaces is resource-efficient while blocking attack packets as early as possible given the available set of filters F. Thus, the filtering objective becomes a resource discovery problem, i.e., finding boundary interfaces given a set of filter nodes, F, and a particular attack (A, v).

3.3.2 Our solution

Filter multicast session

The key idea in our filtering solution is that all filter nodes subscribe to a *filter multicast session*. As with the '911' telephone number allotted to serve police, fire and emergency situations in the United States, in our approach, a filter multicast session is dedicated to a

1: { For each interface, i, except the one receives the request packet}
2: { t : filtering initiation time, $d_i = \alpha d_v$ $(0 < \alpha \leq 0.5)$ }
3: { install a filtering rule set conforming to $AS_{(A,v)}$ at t}
4: **if** no attack packets dropped in $[t, t + d_i]$ **then**
5: stop filtering
6: make i state OFF
7: **else**
8: keep filtering
9: **if** no attack packets dropped in $[t + d_v - d_i, t + d_v]$ **then**
10: make i state OFF
11: **else**
12: make i state ON
13: **end if**
14: **end if**

Figure 3.3: Interface state decision algorithm.

communication channel to support on-demand filtering service.

Once an attack, (A, v), is launched and detected by a victim v,[6] it joins the filter multicast session and requests filtering service by multicasting a *filter request packet*. This packet contains an attack signature, $AS_{(A,v)}$, used to generate appropriate filtering rule sets, and a *filtering period* d_v, the desired duration over which filtering is to be performed. Once each filter receives an *initial* request packet, it associates its interfaces with either the ON or OFF state based on the following decision rule:

The state of each interface is basically determined according to whether it carries the attack packets over two equal-sized intervals at the beginning and the end of filtering period: $[t, t + d_i]$ and $[t + d_v - d_i, t + d_v]$. Note that the range for α in Line 2 guarantees that there is no overlap between two intervals.

Given an attack incidence, (A, v), interfaces at filter nodes can be classified into three types: 1) negative: whose interface state should be OFF after the first interval (Line

[6] In the case where a detector and a victim is not co-located, the detector will act as an agent on behalf of the victim.

6), 2) boundary: whose interface state should be ON after the second interval (Line 12), and 3) positive but not boundary: whose interface state should be OFF after the second interval (Line 10). The third type may happen because filtering requests are handled in a distributed and asynchronous manner at each filter node. Note that in the above decision rule, even a single packet matching with the attack signature enables an interface to be put in the ON state. One may consider a threshold method, i.e., only when the number of packets matching the attack signature is larger than some specified threshold value, the interface enter the ON state.

Each filter node which has at least one interface in the ON state (referred to as ON filter node), sends a *filtering report* packet to the victim containing some filtering statistics, e.g., the number of packets dropped on each of its ON interfaces. Based on report packets(whether the attack persists or not), the victim can renew its filtering request by multicasting another filtering request packet. As the state of each interface at filter nodes has been decided after the initial filtering request packet, OFF filters simply ignore the request, while ON filter nodes keep filtering at ON interfaces until either 1) a filtering timeout(d_v) expires, or 2) a renewal arrives, in which case the filtering period is restarted and again a filtering report packet is sent to the victim.

Note that state information should be preserved long enough that the next filtering request packet arrives, i.e., next renewal. However, it should be eventually eliminated if no renewals arrive. Thus, once a node gets a filtering request packet, it restarts the *state elimination timer*, (e.g., $3 * d_v$). After the timer expires (i.e, no renewal packet prior to timeout), it eliminates state information associated with the attack signature, $AS_{(A,v)}$.

Adaptiveness

In the above protocol, filter states are determined upon receipt of an initial request packet, and remain fixed before they expire. However, even for the same attack incidence, the boundary filter nodes may change due to 1) dynamic attack patterns, e.g., an attacker may have a strategy that periodically turns on and off some attack nodes and 2) routing instability, i.e., packets injected by the same host to the same destination may travel different paths. This requires a filtering mechanism to adapt to dynamic changes in the set of boundary filter nodes (interfaces).

To this end, we include a *reset flag* in the filtering request packet and add the following behavior at filter nodes: 1) if the received request packet's reset flag is 0, then perform the state decision procedure for only `ON` interfaces, and, 2) if the reset flag is 1, then perform the state decision procedure for all the interfaces ignoring previously determined states. These rules ensure a transition from `ON` to `OFF` and a transition from `OFF` to `ON` respectively. A failure in this state transition might eventually be detected by the victim, since attack packets may reach the victim.[7] Thus, the victim node will periodically send filter request packets with the reset flag set to either 1 or 0 depending on whether it is seeing attack packets or not until the attack is suppressed or ends.

[7]Receiving attack packets at the victim side does not necessary indicate the failure of state transitions. Consider the case where there is no filter node along the way from the attack nodes to the victim. However, we do not assume such a case.

3.4 Tracing

3.4.1 Objective

The ultimate goal of a tracing mechanism is to identify attack nodes, e.g., $\{a_1, a_2, a_3\}$ for the example in Figure 3.1. However, the goal is usually relaxed to determining the origins of the attack, i.e., $\{r_1, r_2, r_3\}$. This is because 1) tracing-enabled functionality is usually associated with routers and 2) MAC address spoofing is possible. Most existing tracing approaches [32], [33], [34] focus on developing mechanisms which can discover an attack graph. Once the attack graph is discovered, the origins of the attack can be pinpointed. However, precisely pinpointing the origins of an attack is not achievable when there is only a partial deployment of tracing nodes – as is likely to be the case in practice. Thus, rather than identifying the exact attack graph, we set up our tracing objective as that of localizing attack nodes by providing sets of *candidate* nodes, possible attack origins.

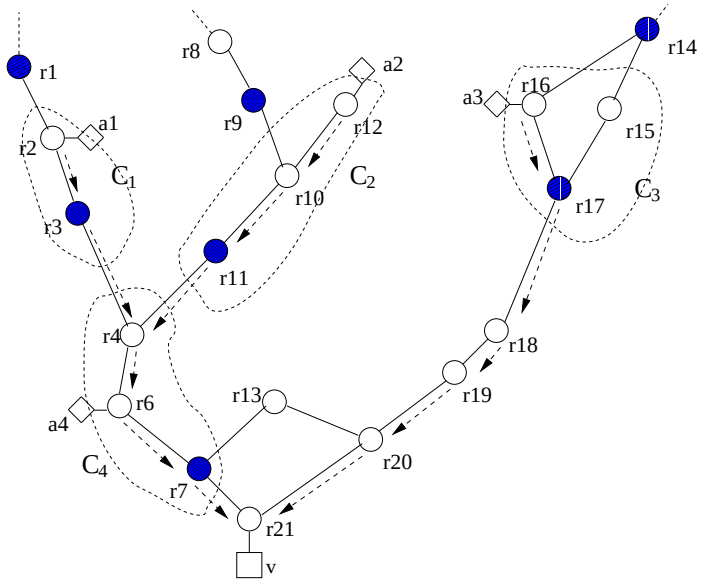


Figure 3.4: An example of tracer deployed network.

3.4.2 Our solution

Localization

First, we define the notion of 'candidate nodes' used in this chapter. Candidate node sets are determined based on the set of boundary tracer nodes. Recall that boundary tracer nodes $BT_{(A,v)}$ are simply tracer nodes which are first met on the path of attack packets associated with given attack incidence (A, v).

Let M_v denote a network map (tree) of upstream nodes rooted at the victim, v. For any node $m \in M_v$, let M_m denote the subtree rooted at m. For each boundary tracer node $n \in BT_{(A,v)}$, let the set of *candidate nodes* of n, C_n, be

$$C_n = M_n \setminus \bigcup_{x \in M_n \cap T} M_x.$$

Note that C_n is obtained by subtracting all subtrees rooted at descendant tracer nodes of n from the subtree rooted at n. For example, in Figure 3.4 where tracer nodes are shown in black, boundary tracer nodes are $\{r_3, r_{11}, r_{17}, r_7\}$ and their sets of candidate nodes are $C_1 = \{r_3, r_2\}, C_2 = \{r_{11}, r_{10}, r_{12}\}, C_3 = \{r_{17}, r_{15}, r_{16}\}, C_4 = \{r_7, r_6, r_4\}$ respectively.

We further define a node c to be a *boundary negative tracer* node with respect to $n \in T_{(A,v)}$ if

1. c is a negative tracer node,

2. n is on the path from the root, v to c, and

3. there are no other nodes in T on the path from n to c.

For example, r_3's boundary negative node is r_1 while r_7 has no boundary negative tracer nodes in Figure 3.4.

The key idea underlying our tracing approach is that a set of candidate nodes contains at least one origin of an attack. For example, consider the attack node a_1 in the network

shown in Figure 3.4 where tracer nodes are shown in black. With the following information: 1) r_3 is a boundary tracer node, 2) r_1 is a negative tracer node, and 3) the network topology, one can conclude that $\{r_2, r_3\}$ is a set of nodes which contain an attack origin(s).

A *positive* attack graph is defined as the collection of all links and nodes traversed by packets from each node in $T_{(A,v)}$(i.e., positive tracer nodes), to the victim v. We define an *expanded* attack graph as the positive attach graph plus the collection of all links and nodes traversed by packets from boundary negative nodes to the victim. For the attack incidence in Figure 3.4, its expanded attack graph is depicted in Figure 3.5. Thick links and nodes from the victim to positive tracer nodes comprise the positive attack graph.

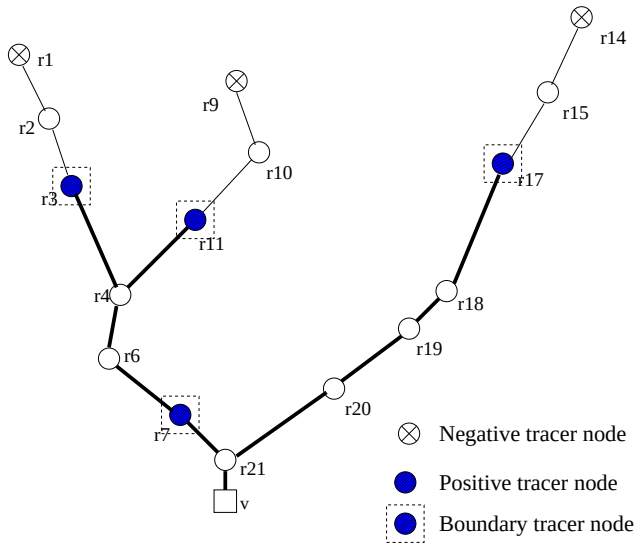


Figure 3.5: An expanded attack graph.

Note that sets of candidate nodes can be obtained with the following information: 1) a network map of upstream nodes to the victim, 2) an *expanded* attack graph, and 3) boundary tracer nodes. By overlapping the two graphs, i.e., network map and expanded attack graph, we can identify candidate node sets. The network map of upstream nodes to the victim can be obtained using a tool such as Skitter [35] or that developed in [36]. Thus

the remaining task is to obtain an expanded attack graph and identify the boundary tracer nodes.

Obtaining expanded attack graphs

Note that obtaining a positive attack graph is the goal of existing tracing approaches [34], [33], [32]. In this section, we propose a new approach to obtaining a positive attack graph and then extend it to determining the expanded attack graph.

As with our filtering approach, we propose to have a multicast session support tracing services, referred to as a *tracer multicast session*. This session is joined by the set of tracer nodes, T. Once an attack incidence, (A, v) is detected at the victim, then v joins the tracer multicast session and sends a *tracing request packet* containing a signature, $AS_{(A,v)}$.[8] Upon receiving the tracing request packet, each tracer node checks whether it has carried attack packets conforming to the attack signature.

In order to obtain path information from tracer nodes to a victim, we propose to use the *traceroute* program. Traceroute provides an executing node with a forward path information from it to a destination. After checking whether it is seeing any attack traffic, each positive tracer node n in $T_{(A,v)}$, performs a traceroute toward the victim, which enables it to identify the forward path from n to v. Then n sends a *tracing report* packet including this path information to the victim. By collecting paths reported by positive tracer nodes, the victim can construct a positive attack graph – a subset of an attack graph. A simple approach to obtain an expanded attack graph is to have every negative tracer node also perform a traceroute to the victim and send a tracing report packet. However, this will not scale when there is a large number of (negative) tracer nodes in a network. Thus, we

[8]Note that tracing request packets may contain the attack incidence time for post-mortem tracing if it is supported.

propose the following mechanism. Once a tracer node is determined to be positive, after some time, the node performs a *scoped* multicast by sending a *tracing solicit packet* to its neighborhood with Time-to-Live(TTL) set to some value, *k*. The TTL value is first specified in the original tracing request sent by the victim. Upon receiving the tracing solicit packet, a positive tracer node simply ignores it, but a negative tracer node performs a traceroute to the victim and sends a tracing report packet to the victim. Tracing report packets require a flag representing whether they were generated by a positive or negative tracer node. Note that this mechanism makes negative tracer nodes within *k* TTL distance from positive tracer nodes participate in the tracing operation. However, this mechanism may only produce an *approximate* expanded attack graph. This depends on the scope of the tracing solicit packet and the locations of tracer nodes. For example, if TTL value is too large or tracer nodes are closely located, unnecessary negative nodes may be included in the graph. However, in this case, one can still identify candidate nodes. In the other case, i.e., where the TTL value is too small or tracer nodes are too far away, one fails to obtain a set of candidate nodes. Reflecting this practical situation, we classify sets of candidate nodes into two classes: *closed* or *open*, i.e., identified or unidentified candidate nodes respectively from the perspective of a victim. This classification depends on the deployment of tracer nodes, i.e., how many and where, as well as which TTL value is used. To reduce the number of open sets, we can envisage the following scheme: after the victim recognizes existence of sets of candidate nodes which are open, it may perform another tracing operation with a larger TTL value. However, even though this can reduce open sets to closed ones, too large number of nodes in a closed set requires lots of search effort within the set, which in turn makes localization expensive.

Identification of boundary tracer nodes

Once one obtains an expanded attack graph, it is clear how to identify boundary tracer nodes residing at the end of a positive attack graph, e.g., r_3, r_{11} and r_{17} in Figure 3.5. However, it is difficult to find boundary tracer nodes residing at internal nodes of a positive attack graph, e.g., r_7 in Figure 3.5. This is because the packets generated at attack nodes associated with a boundary tracer node can not be differentiated from packets injected at upstream attack nodes. Note that this problem exists even with networks in which tracers are fully deployed. For example, in Figure 3.4, a_4's attack origin is invisible unless tracing components have an ability to not only check if an attack packet has passed but also acquire information about which interface(s) attack packets are flowing through. This problem can be handled by the following methods: 1) filtering and tracing can be performed jointly or 2) packets from different attack nodes can be further differentiated, e.g., instead of using attack signature shared by all attack nodes, using packets themselves may differentiate those attack packets.

3.5 Performance evaluation

We conducted three sets of simulations varying topologies and placement strategies. The objective was to explore two questions: how many and where should filters and tracers be deployed in the network to be effective for blocking and localizing attacks. We do not claim these experiments are comprehensive. Instead, our goal is to provide insights on how the proposed framework performs in several representative settings. Below we first discuss performance metrics for the proposed filtering and tracing framework, and then present our simulation results.

3.5.1 Filtering performance metrics

We define the *cost* of an attack node, a_i, as the product of two quantities: the amount of attack traffic injected at a_i, $w(a_i)$ and the distance from the attack node to the victim, $d(a_i, v)$. This can be roughly considered as a measure of network resources (e.g., bandwidth, routers' CPU cycles, etc.) that are wasted in processing attack packets from a_i to v. For an attack incidence (A, v), the *total attack cost* $c(A, v)$ is

$$c(A, v) = \sum_{a_i \in A} w(a_i) d(a_i, v).$$

When filter nodes F are deployed and filtering is conducted at the boundary, the total attack cost $c(A, v, F)$ is reduced to

$$c(A, v, F) = \sum_{a_i \in A} w(a_i) d(a_i, bf(a_i)).$$

where $bf(a_i)$ is the boundary filter node associated with attack node a_i. To demonstrate the amount of traffic that can be blocked using filtering, we define the *relative attack cost* γ as the ratio of the cost with filters deployed to the cost without, i.e., $\gamma = c(A, v, F)/c(A, v)$. Note that this cost metric is from the perspective of the network rather than the victim. Although the proposed metric is simple, it captures well features associated attack traffic aggregation. For example, consider a local solution where there is a filter installed at an ingress point only a few hops away from the victim. According to this cost function, γ will be high, capturing the situation where the attack is still effectively disrupting the victim's local network.

3.5.2 Tracing performance metrics

Let C_i be a closed set of candidate nodes and A_{C_i} be the set of attack nodes in C_i. For a given attack incidence (A, v) and set of tracer nodes T, suppose k of closed sets of candidate nodes

are obtained after tracing. In this case, the victim will have to search for attack nodes within $C_1, \ldots, C_k$. We shall define two metrics. The first one, Φ_1, is given by

$$\Phi_1 = \frac{\sum_{i=1}^{k} |A_{C_i}|}{|A|}.$$

Note that Φ_1 is the ratio of the number of identified attack nodes to the total number of attack nodes. Here, (1-Φ_1) is the portion of attack nodes that lie in open candidate sets, and are assumed remain undetected. To capture how 'localized' the portion of identified attack nodes is, we define a second metric, Φ_2 as

$$\Phi_2 = \frac{1}{k} \sum_{i=1}^{k} \frac{|C_i|}{|A_{C_i}|}.$$

Note that $|C_i|/|A_{C_i}|$ represents the *search effort*, i.e., the average number of nodes to be searched in order to determine attack node(s) in C_i. Thus, Φ_2 is the average search effort over the closed of candidate nodes. In summary, Φ_1 represents how many attack nodes are identified and Φ_2 captures the search effort or degree of localization achieved by the tracing mechanism.

3.5.3 Simulation Results

For an attack incidence (A, v), we randomly select $|A|$ nodes (excluding the victim) in a given topology to be attack nodes. We assume that attack nodes generate the same amount of attack traffic, i.e., $w(a_i) = c, i = 1, \ldots, |A|$. We have built a tool to estimate the performance measures proposed in Section **??** and **??**. For all of our results, each performance metric value is the average value of 100 simulation runs (attack incidences). For the results associated with tracing, we set 5 as the TTL value for the solicit tracing packets. For simplicity, we assume that filtering is also performed at boundary tracer nodes, which can

identify all boundary tracer nodes. In the simulation, we do not consider dynamic changes in the configuration of attack nodes during each attack incidence.

Simulation I

In this simulation we use a real tree topology from [37]. The tree was obtained by performing traceroutes at the server, `www.bell-labs.com`, to its clients and consists of around 23,000 distinct nodes. We considered a scenario where randomly placed nodes launch an attack to the server. For the placement of tracer or filter nodes in the network, we choose a random strategy for a given *coverage ratio* β, i.e., β is the portion of total nodes in the network that are filter or tracer nodes.

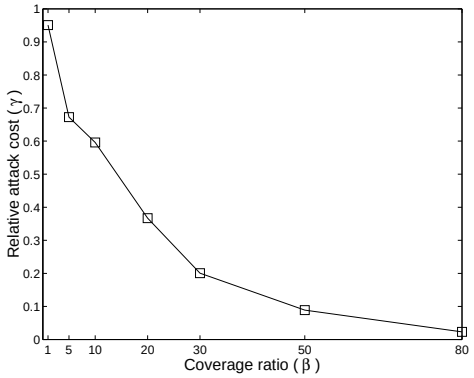


Figure 3.6: Relative attack cost ($|A| = 25$) in simulation I.

Figure 3.6 shows the relative attack cost, i.e., γ, resulting from an attack involving 25 nodes and varying filter coverage ratios. Note that it has a convex shape, i.e., a small increase in coverage ratio can cause a large reduction in the attack traffic when the coverage ratio is small. We observe that one can reduce the attack traffic by 80% (relative attack cost is 0.2), with a filter coverage of 30%.

Figure 3.7 shows γ for several coverage ratios and a varying number of attack nodes

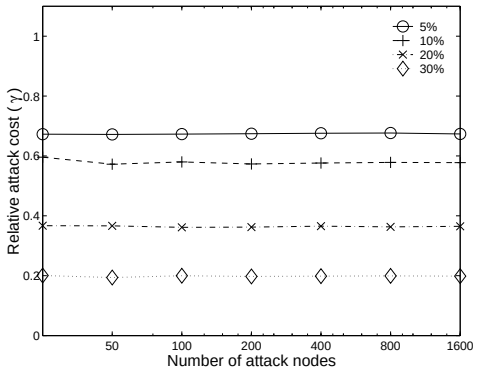


Figure 3.7: Relative attack cost in simulation I.

from 25 to 1600. We observe that γ is independent of the number of attack nodes. To explain this result, we notice that $c(A,v,F)$ is roughly given by $|A|w(a_i)E[d^*]$ where $E[d^*]$ is the expected distance from attack nodes to their boundary filter nodes. Likewise $c(A,v)$ becomes $|A|w(a_i)E[d]$ where $E[d]$ is an expected distance from attack nodes to the victim. Since we assume $w(a_i)$ is constant, γ simply depends on the ratio of the above two distances.

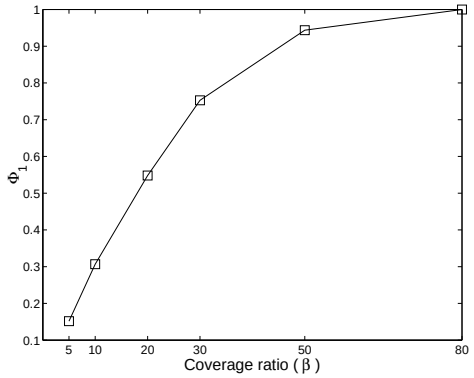


Figure 3.8: $\Phi_1(|A| = 25)$ in simulation I.

Figures 3.8 and 3.9 show Φ_1 and Φ_2 respectively for an attack of 25 nodes. At a 30% coverage ratio, around 75% attack nodes can be detected and on average attack nodes can be localized to within 9 nodes. Figures 3.10 and 3.11 show results for Φ_1 and Φ_2

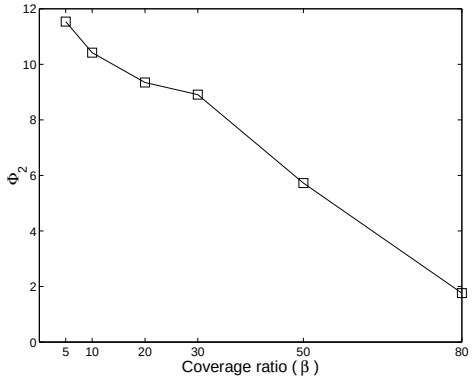


Figure 3.9: $\Phi_2(|A| = 25)$ in simulation I.

respectively for various coverage ratios varying the number of attack nodes. We make the following observations. First, Φ_1 is independent of the number of attack nodes. Second, Φ_2 is decreasing as the number of attack nodes increases. These can be explained as follows. As the number of attack nodes increase, one can see more attack nodes will be placed in closed sets of candidate nodes, which ensures more attack nodes will be detected. Thus Φ_1 is likely to be independent of the number of attack nodes. Furthermore, Φ_2 will decrease since there are more attack nodes found in the same closed candidate sets as the number of attack nodes increases.

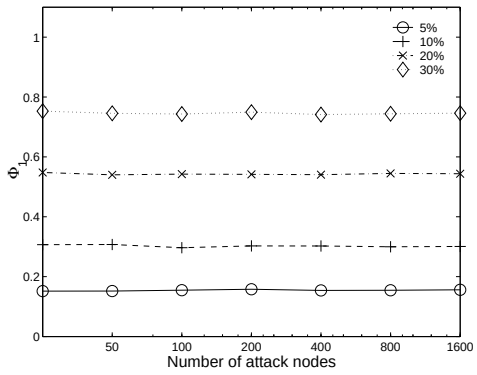


Figure 3.10: Φ_1 in simulation I.

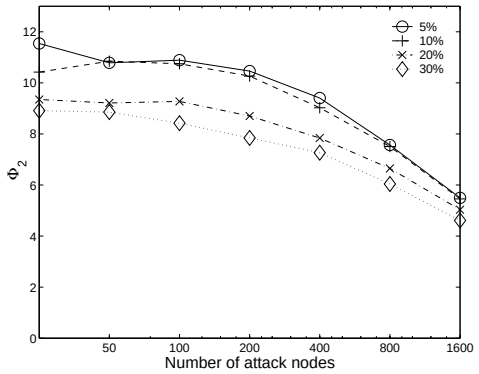


Figure 3.11: Φ_2 in simulation I.

Simulation II

For the second set of simulations, we generated a 2000-node random transit-stub graph using GT-ITM [38]. This topology is composed of interconnected transit and stub domains where domains are assumed autonomous. In this setup, we randomly choose a victim node for each attack incidence. To investigate the performance impact of filter or tracer location, we explore a *border* placement strategy, i.e., randomly choose the location of filter or tracer nodes among border nodes of domains versus randomizing over all possible locations. This reflects the case where a network administrator of a domain decides to provide services, the administrator is likely to place filter or tracer nodes at border nodes rather than at random locations within the domain.

The results for different numbers of attack nodes show the same qualitative behavior as those in the previous experiments. We exhibit the results for a 50 node attack. Figure 3.12 shows γ for various placements. For the same coverage ratio 5% and 10%, border placement performs better than random, and border placement with 15% coverage outperforms random placement with 30% coverage. Figures 3.13 and 3.14 show results for Φ_1 and Φ_2 respectively. Here, we obtained an encouraging result that even a 15% deployment of tracer

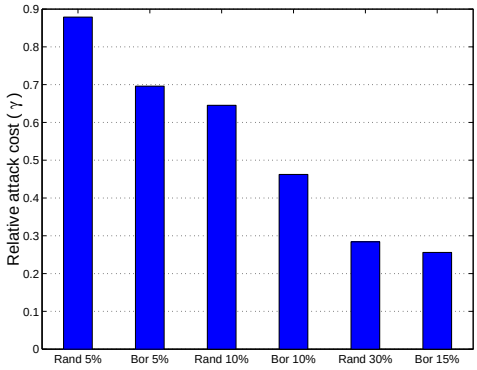


Figure 3.12: Relative attack cost ($|A| = 50$) in simulation II.

nodes at border nodes can detect more than 90% of the attack nodes and on average attack nodes are localized to within 10 nodes. Additionally, we made the following observations. First, border placement can detect more attack nodes than random. Second, all random placements have lower Φ_2 value than border placement. Indeed even a small coverage ratio of border placement can create a well-balanced number of candidate sets. This reduces the number of open candidate sets, which achieves high Φ_1. Compared to this, with random placement, the number of nodes in candidate nodes set vary more dramatically. This results in small Φ_1, which identifies the smaller number of open candidate sets. However, once it is identified to be open, its search effort becomes less, leading to a smaller Φ_2 value.

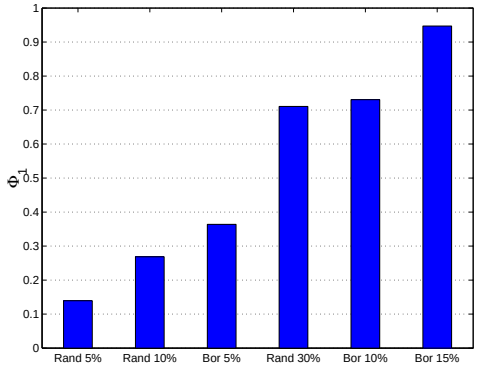


Figure 3.13: $\Phi_1(|A| = 50)$ in simulation II.

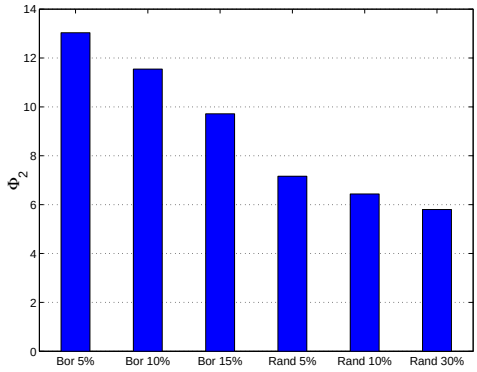


Figure 3.14: $\Phi_2(|A| = 50)$ in simulation II.

Simulation III

Finally, we performed filtering simulations on a 145-node routing tree rooted at the server, `www.bell-labs.com` in [37]. The motivation for this set of simulations is to explore the case where the victim (the server) has full control over the placement of filter nodes on a mid-size network.

We considered an *optimal* placement to be one that minimizes the total attack cost under the assumption that each node in the network can be an attack node and generate the same amount of attack packets. This problem is equivalent to the cache location problem studied in [37], [39], thus we used their dynamic programming formulation to find optimal placements.

Figure 3.15 shows the relative attack cost results of 25 attack nodes for random versus optimal placements. As can be seen, optimal placements outperform random. A 10% coverage with optimal placement can reduce attack traffic by 80%.

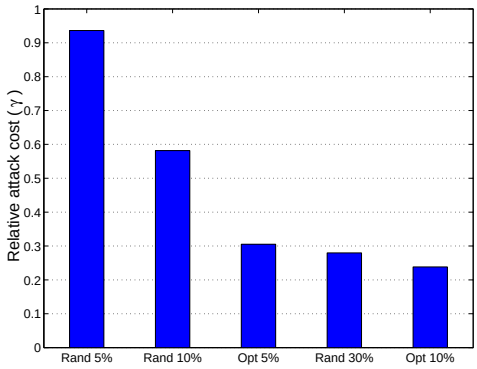


Figure 3.15: Optimal vs. Random ($|A| = 25$) in simulation III.

3.6 Discussion

3.6.1 Implementation issues

Implementation of components

In this section, we briefly present various implementation methods for the components described in Section 3.2.2. A number of intrusion detection systems(IDS), or identification algorithms [40], can serve as detecting components. Note that the ability to obtain attack signatures from attack packets is a critical requirement for our framework and is itself a significant on-going topic of a number of intrusion detection systems. For tracing components,[9] input-debugging [41] can check tracing results only for on-going traffic. By contrast, logging [41] and hash-based logging [34] methods can provide a *postmortem* tracing service which can check if even past traffic (before a query) contains attack packets. Furthermore one can envisage that stand-alone data capture devices (e.g., RMON probes [42]) or sniffers (e.g., tcpdump) can be used to provide on-going tracing services in a non-intrusive way, i.e., not affecting routing performance. Also note that there are various filtering services avail-

[9] See Section 4.4 for more detailed explanations for each tracing method.

able which operate at various layers of the protocol stack - e.g., IP-level, TCP-level and application level.

Security issue

To be a viable solution to defeat DDoS attacks, our framework should itself not be misused by an attacker. For example, we can consider the following possible security concerns: 1) a compromised tracer provides incorrect information, and 2) an attacker generates faked tracing or filtering requests. Note that the critical solution for the above problems is to authenticate the source of the packets (i.e., who is sending the request or report packets). A comprehensive solution addressing security concerns in our framework, we would use a secure multicast service. However, secure multicast service is still being developed [43]. Instead, we observe that a standard digital signature scheme will suffice for source authentication in our framework. As pointed in [33], if the authentication of packets is required too frequently, a digital signature is an expensive solution incurring lots of computational and space overhead. However, note that in our scheme, the packets are exchanged between a victim and a subset of tracer or filter nodes only when required.

Inaccurate information

As seen in Section 3.4, our tracing mechanism relies on 1) a map of upstream routers and 2) traceroute results. In this section, we consider the impact of inaccurate information on our tracing mechanism. Even though the network map can be obtained using the tools described in Section 3.4.2, we observe that it is difficult to obtain an updated accurate Internet topology. However, since our goal is not to pinpoint the exact attack origins, such a map does not have to be perfect. Furthermore, note that even without a map, the expanded

attack graph itself can be a useful tool for identifying the regions where attacks originated. Also, a map can be obtained in a post-mortem manner after an attack ends.

Note that traceroute may not work as desired due to the manner in which routers are configured along given paths. In addition, traceroute may not provide an accurate attack path due to changes in routing and attack patterns on parallel routes. However, as mentioned above, our goal is to obtain a set of candidate nodes not to identify the exact attack origins. Thus, inaccurate path information can be still useful. Furthermore, if traceroute is performed while an attack continues, attack packets and traceroute packets will be forwarded based on the same unicast routing table, which produces a more accurate approximate attack graph.

Multicast routing protocol

In this section, we consider which multicast routing protocol will be suitable for our framework. Current implementations of multicast routing service, can be classified into two types: source tree and shared tree routing. In source tree routing, the distribution tree is a *reverse shortest-path tree* which is formed by overlaying shortest path from each member to a source. In shared tree routing protocols, e.g., CBT [20] or PIM-SM [25], the distribution tree is commonly shared by all members irrespective of the sources. Note that in our approach, during a time without attacks, there are no data packets injected into a multicast session. This is because the multicast session is not used for data distribution among members but only for tracing or filtering request distributions from unpredictable victims. Therefore, the cost for maintaining the multicast session becomes an important issue when selecting a multicast routing protocol. In source tree routing, the distribution tree is maintained by periodic reverse-path forwarding and pruning, which incur large overheads. Thus

we conclude that a shared tree routing protocol is more suitable than a source tree for our approach.

Shared multicast routing protocols can be further classified into two types: *unidirectional* and *bidirectional* shared tree routing protocols. In a unidirectional shared tree protocol, the sender's packets go to the core first and the core multicasts them to others. By contrast, in bidirectional shared tree routing protocols, members can communicate with each other without going through the core since packets can travel both up toward the core and down from the core. Thus, once a shared tree routing is used, unidirectional routing protocols are inefficient for scoped multicast and communications among neighborhoods. The larger the multicast session and the more demands for scoped multicast, the larger the communication overheads will be in unidirectional shared multicast routing protocols. Reflecting these observations, the long term inter-domain routing solution, Border Gateway Multicast Protocol(BGMP) [24] constructs bidirectional shared trees. Since our tracing mechanism uses scoped multicast to find negative tracer nodes, bidirectional shared tree routing protocols will be more efficient in our framework.

3.6.2 Economic Incentives

Irrespective of the existence of feasible solutions to mitigate DDoS attacks, a significant hurdle may be the lack of viable economic incentives [44]. For example, installing ingress filters in a domain consumes valuable router resources and reduces the overall routing performance. However, its beneficiaries are likely to be other domains rather than the domain performing ingress filtering. In our framework, we can envisage the following economic model: victims pay a fee for the services provided by tracer and filter nodes. Clearly this gives an incentive to provide tracing and filtering services to victims in other domains. The

payment may be dependent on the number of attack packets dropped, the number of tracing or filtering report packets and so on. One can further consider that victims pay a fee for detecting services. We leave the more detailed pricing mechanisms as future work.

3.6.3 Differentiated Services

As described in Section 3.6.1, various implementation methods for tracing and filtering components are available with different characteristics. To allow such heterogeneity in implementation in our framework, we can consider providing differentiated services. Instead of having a single multicast session for tracing (or filtering) service, we create different multicast sessions for different services. For example, there could be a multicast session for postmortem tracing or one for application level filtering.

Note that this scheme allows 1) flexible service requests for victims and 2) heterogeneity in different implementation methods from different domains. Depending on the attack scenarios, a victim can request different services. For example, if a victim detects an attack very late and the attack has already ended, the victim can ask for a postmortem tracing service rather than tracing for an on-going one. Given its service requirements, each domain may use its own methods. This requires less standardization across network domains, allowing more heterogeneity.

3.7 Related Work

In this section, we discuss the pros and cons of existing research efforts on defeating DDoS attacks and the proposed approach in this chapter.

3.7.1 Detection and mitigating approach

There are two types of mitigation mechanisms: host-based and router-based. Host-based approaches [45], [46] try to detect and mitigate the impact of attacks by an efficient control of resources from the perspective of the operating system on the victim side. Even though this helps sustain a victim longer, the victim will eventually give in to attacks. By contrast, in the router-based approach [40], detection and mitigation of attacks are performed at routers. This work defines an *aggregate* as a particular set of packets causing the overload and proposes an identification algorithm for detection and control mechanisms which can reduce such aggregates. A SYN flooding detection mechanism is recently proposed in [47]. It is based on discrepancies between SYN and FIN packets. It is stateless and requires low computational overhead to detect SYN flooding attacks.

3.7.2 Proactive filtering approach

Ingress filtering [29] and route-based filtering [30] [10] proactively prevent attacks employing spoofing. Routers are configured to drop packets whose source IP addresses are illegitimate based on routing and network topology information. Ingress filtering uses simple direct connectivity information, so it is usually performed at border routers in stub networks [29]. However, in transit networks, it lacks the ability to distinguish between legitimate and illegitimate packets and its effectiveness can only be guaranteed via wide deployment. To overcome these weaknesses, a route-based mechanism [30] performs filtering using source reachability information imposed by routing and network topology. By using additional network topology information, route-based filtering requires less coverage than ingress filtering to be effective.

[10] Route-based filtering approach also has a tracing by-product, i.e., attack origins can be localized to a set of AS (Autonomous System) sites.

3.7.3 Tracing approach

As indicated by recent work on tracing mechanisms [34], [32], [33], tracing can be an effective way of discouraging attackers. The identified compromised hosts intentionally or unwillingly participating in attacks can be isolated from the Internet or can provide clues to the real attacker. Existing tracing approaches can be classified into the following two types.

The first type is a query-based approach where trail information is queried to tracing components in the network. The query is usually performed in the reverse direction of the attack packets, i.e., from the victim toward the source(s) of the attack. Checking whether attack packets have been forwarded by given routers can be done by several currently available techniques: logging packets using monitoring tools [41] or input debugging which can identify which ingress port was used by packets departing on a given egress port. However, there may be a high storage overhead for logging methods and some adverse effects on routing functionality when input debugging is used. To overcome these problems, in the hash-based logging approach [34], routers store packet digests generated by a hash function rather than the packets themselves. Upstream routers to the victim are successively queried for attack packets in a reverse path flooding manner. A key advantage of this approach is that it is capable of tracing a single recently forwarded packet while keeping privacy. However, tracing queries should be initiated early enough that appropriate digest entries have not been overwritten by more recent packets. Note that since queries are sequentially processed in the query-based approach, the malfunctioning of some tracing components may not deliver a query to upstream routers, which result in the failure of tracing operation.

Another tracing alternative is based on partial path information which is proactively sent to end hosts when packets are forwarded. Once attacks are detected, the victim can reconstruct the routes attack packets took based on stored trail information. In the iTrace

method [48], [49], with a low probability routers send extra ICMP messages including their own addresses to the end host. By contrast, IP marking schemes [32],[33] can eliminate the extra ICMP messages used in iTrace by having routers probabilistically inscribe *edge* information (represented by two routers at the end of a link) onto a traversing packet. The advantage of this approach is that it enables incremental deployment while keeping router's overhead low. However, as pointed out in [50], in the presence of multiple attack nodes, the approach suffers from a scalability problem, i.e., uncertainty in identifying origins of attack packets increases proportionally with the number of nodes in a distributed DoS attack. Furthermore, due to its probabilistic character, the solution is confined to tracing attacks associated with large volumes of traffic. Note that in this second approach, end hosts need to proactively save incoming packets irrespective of whether an attack is ongoing, requiring large storage overheads at end hosts.

Schnackenberg et al. [51] propose an IDIP (Intruder Detection and Isolation Protocol) for automated intrusion response systems that can detect a DoS attack and request upstream network elements to block the traffic. Their work focuses on standardization of a set of protocols for interaction among infrastructure components to realize a simple query-based tracing idea.

3.7.4 Characteristics of our solution

In a proactive filtering approach, filters need to maintain a large number of filtering rule sets and examine every single packet. We envisage a large scale filtering framework, which suffers from the complexity of managing a large number of filtering rule sets. Furthermore, if attacks are infrequent, valuable resources may be wasted if irrelevant filtering rule sets are applied to packet flows. By contrast, our filtering mechanism has soft state properties, e.g.,

renewal and expiration of filtering, which significantly reduces complexity of managing filtering rule sets.

The conventional reactive filtering approach confined to a single administrative domain containing the victim has its limitation. Once attack packets (possibly generated at multiple locations) have been aggregated into large traffic flows, attack traffic can overwhelm the local domain and make local filters inoperable; as a result, the filters are victimized. Note that our filtering mechanism is designed to quickly identify a set of boundary filter locations so that attack packets might be dropped as close as possible to their origins before they are aggregated.

The key requirement for reactive approaches – promptness – can be met by the use of multicast communication in our framework. This is the case when a request packet is distributed in real time via multicast, so filter/tracer nodes can respond concurrently and quickly. The use of IP multicast provides the following additional advantages. First, there is no overhead in managing a list of cooperative filter/tracer nodes on detector or filter/tracer nodes, leading to improved scalability. This feature comes from the member abstraction property of multicast service, i.e., members can join and leave a multicast session without explicit knowledge of its membership. Second, the proposed mechanism is robust even in the presence of some malfunctioning filter nodes or packet losses. In a sequential query-based approach, the query process may be terminated due to packet losses or malfunctioning of some filter nodes. In such situations, our approach may not result in optimal filtering/tracing operation, i.e., filtering is performed in non-boundary filter nodes(or attack graph is not accurate), but it may still work well. Finally, multicast ensures an efficient use of network resources – a single packet traverses each link in the multicast distribution tree and is replicated at fan-out points.

3.8 Conclusion

In this chapter, we presented a new multicast-based filtering and tracing service framework to defeat DDoS attacks. The proposed filtering mechanism pushes filtering operation to boundary nodes so that attack packets might be dropped as close as possible to their origin(s). If we assume that attacks are infrequent, our filtering mechanism can achieve more efficient use of network resources versus proactive solutions. Indeed when no attacks are ongoing, only a multicast session needs to be maintained, without overheads associated with filtering operation. In this chapter, we consider the goal of determining sets of candidate nodes for localizing attack origins under a partial deployment of tracing components, and propose a mechanism to achieve this end.

A significant challenge of large scale deployment of both mechanisms is handled by a novel use of IP multicast and soft-state. Furthermore the use of IP multicast provides a number of desirable characteristics, e.g., fast response - one of key requirements for reactive solutions, and robustness. Additional contributions of our work include a number of practical considerations: 1) addressing economic incentives, 2) using currently available equipment and technologies without major router modifications, and 3) allowing incremental and partial deployment.

The performance evaluation for the proposed framework shows that a small coverage ratio of well-placed filter or tracer nodes can achieve efficient blocking and localizing of attacks.

Chapter 4

Topology-sensitive Subgroup Communication

4.1 Introduction

Due to its bandwidth efficiency, IP Multicast is the preferred data delivery method for large-scale interactive applications such as distributed interactive simulations (DIS), video conferencing tools and multi-player games. Although the members in such applications join a multicast session for some common goal, abundant content, data type and heterogeneity in members' interests naturally lead to *preference heterogeneity* within sessions [52], requiring frequent communication within *subgroups* of members sharing common interests/requirements. A multicast session shared by all members (referred to as the global multicast session) can be used to support subgroup communication. However, this may lead to inefficiency, i.e., packets are delivered to the entire tree, which results in wasted bandwidth and CPU processing power to transmit and handle unnecessary packets. This is referred

to as the *exposure* problem. The exposure problem can be completely eliminated if data is only forwarded along a tree induced by the members of each subgroup, as required. This can be achieved by creating a new multicast session for each subgroup. However, this requires routers to store multicast forwarding state information for each subgroup, which can cause a significant scalability problem as the number of subgroups increases [53, 54]. Thus, mechanisms to handle preference heterogeneity should consider both the exposure problem and the scalability of increasing multicast forwarding state. Most existing approaches to the preference heterogeneity problem focus on developing *clustering* frameworks, i.e., given a limited number of multicast sessions, determine how to best cluster multiple subgroups into multicast sessions based on a preference matrix [52] or players' positions in a virtual cell [55].

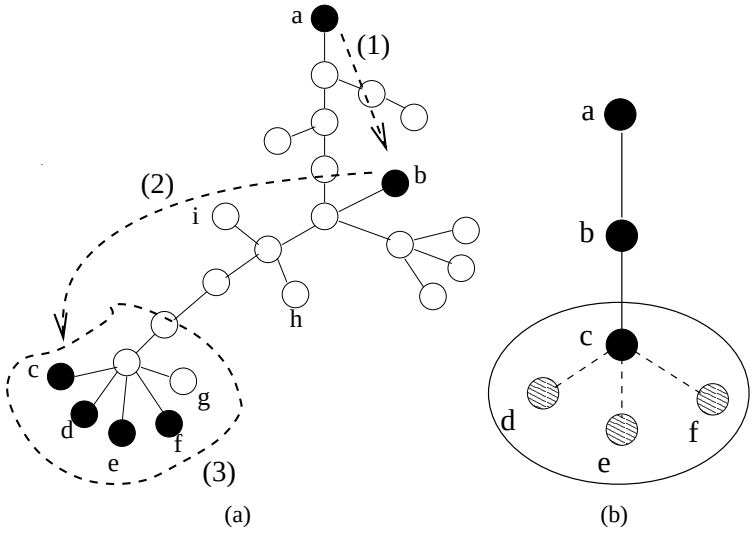


Figure 4.1: An example of TSC forwarding structure

In this chapter, we propose a topology-sensitive subgroup communication (TSC) mechanism to support efficient subgroup communication in large-scale multicast applications. Our TSC mechanism allows members in a subgroup to autonomously build a TSC

forwarding structure consisting of multiple unicast and scoped multicast connections. For example, consider a distribution tree for a multicast session, G, in Figure 4.1(a). All end nodes are members of G and the black end nodes are members of a subgroup, S. In our scheme, when a wishes to send packets to other members in S, packets will be delivered as follows: (1) $a \longrightarrow b$ via unicast; (2) $b \longrightarrow c$ via unicast; and, (3) $c \longrightarrow \{d, e, f\}$ via multicasting with a TTL scope of 2 as shown in Figure 4.1(a). We assume that the multicast tree for G is a bidirectional shared one.[1] Note that in the example, the use of unicast can suppress the exposure and the use of scoped multicast can reduce duplicate packets traversing the same link. Our approach does not require the creation of new multicast sessions, which can completely eliminate any additional multicast forwarding state except those of the global session. We attempt to minimize exposure by exploiting spatial locality among members within a given subgroup.

This chapter is organized as follows. In Section 4.2 we discuss how to construct and maintain a TSC forwarding structure. In Section 4.3, we evaluate and compare the proposed TSC mechanism with other schemes in various environments. We discuss related work in Section 4.4 and conclude this chapter in Section 4.5.

[1]Our mechanism targets many-to-many large-scale multicast applications where each member can be a sender and/or receiver. For such applications, it is generally agreed that shared multicast routing protocols are more efficient than source based ones. Even though PIM-SM, widely deployed for shared multicast routing, takes a unidirectional forwarding mechanism, we argue that bidirectional forwarding mechanisms are more efficient. The larger the multicast session and the more the demand for local communication, the larger the overhead incurred by using a unidirectional tree. Reflecting these observations, the long term inter-domain routing solution, Border Gateway Multicast Protocol(BGMP) [24] currently under development, constructs bidirectional shared trees.

4.2 TSC forwarding structure

Given G and $S = \{a,\ b,\ c,\ d,\ e,\ f\}$ as shown in Figure 4.1(a), Figure 4.1(b) depicts an example of a TSC forwarding structure exhibiting an overlay structure among members in subgroup S. The solid and dashed lines represent unicast and TTL scoped multicast respectively. Subgroup members in a TSC scheme are classified into two types: *normal* or *head* members. A normal member is associated with a head member. Head members, denoted by a set H_S, communicate with each other via unicast connections in a TSC forwarding structure. The role of the head members is two-fold: 1) they participate in constructing a unicast overlay structure, and 2) they perform scoped multicast forwarding to their associated normal members. We define an *island* as a set of nodes consisting of a head and its normal nodes. Note that it is possible to have one member island where there are no normal nodes associated with the head node. For example, $H_S = \{a,\ b,\ c\}$ and $\{a\}$,$\{b\}$, $\{c,\ d,\ e,\ f\}$ are islands in Figure 4.1(b).

Note that if there are only one-member islands in a TSC forwarding structure, this completely eliminates the *member exposure* problem. However, this will introduce performance penalties, i.e., duplicate packets on the same physical links. The use of TTL scoped multicast may reduce such bandwidth wastes in the case where subgroup members are clustered with each other. Thus, our goal is to build a TSC forwarding structure which minimizes wasted bandwidth while limiting the exposure of non-subgroup members given a multicast session G and subgroup preferences for each member in G. Building TSC structures involves two stages: constructing islands and then connecting islands.

4.2.1 Constructing islands

Let the set $N(a,t)$ be the set of neighbors of a, i.e., nodes within a TTL distance of t of a node $a \in G$, excluding a itself. Note that if a performs a scoped multicast with a TTL scope of t, packets will be delivered to all the nodes in $N(a,t)$. Let $N_S(a,t)$ denote the set of subgroup S members in $N(a,t)$, i.e., $\{n|n \in S \cap N(a,t)\}$. For example, in Figure 4.1, $N(c,2) = \{d,e,f,g\}$ and $N_S(c,2) = \{d,e,f\}$. For $a \in S$ and a given t, the *exposure ratio* $\beta_S(a,t)$, is defined as follows:[2]

$$\beta_S(a,t) = \begin{cases} 1 & \text{if } |N(a,t)| = 0, \\ \frac{|N(a,t) \setminus N_S(a,t)|}{|N(a,t)|} & \text{, otherwise.} \end{cases}$$

Before proceeding, we briefly describe how each member can compute an exposure ratio for a given TTL scope. Let L_a be a set of subgroups which a member a wishes to join. Each member, $a \in G$, periodically multicasts a *subgroup advertisement* packet containing L_a with a fixed TTL distance k. Then each member can maintain a *TTL-neighbor profile* storing tuples of all neighboring members and their subgroup lists along with TTL distance up to k. [3] Figure 4.2 shows an example of the TTL-neighbor profile of member c in Figure 4.1 when k is 5. With the TTL-neighbor profile, each node can easily obtain exposure ratios up to TTL scopes of k. Note that the scope k value should be large enough to create an efficient and large island, but also should be small enough not to incur too much traffic.

Note that the exposure ratio $\beta_S(a,t)$ can indicate whether a scoped multicast performed by a with a TTL scope of t is efficient or not. That is, when the exposure ratio is low, scoped multicast can be considered an efficient delivery method.

[2] $A \setminus B$ represents A minus B, i.e., elements from A that are not in B. $|A|$ denotes the cardinality of a set A.

[3] Distance information between members can be obtained by senders inserting initial TTL value in packets. This enables a receiver node to compute its TTL(path) distance from the sender by simply subtracting the value in TTL field from initial TTL value.

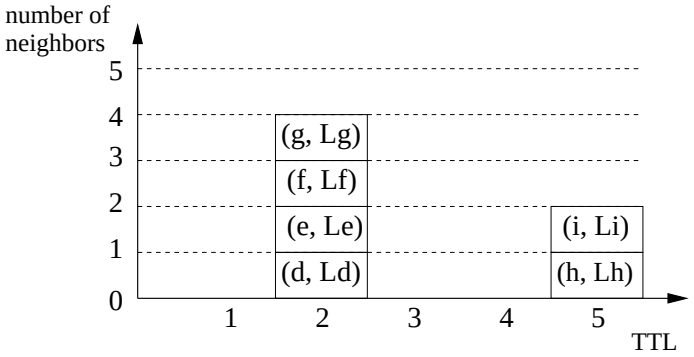


Figure 4.2: TTL-neighbor profile of *c* in Figure 4.1

Since we envisage that a head node forwards packets to its normal nodes via a TTL scoped multicast for subgroup communication, an island can be specified by a head node and an associated TTL scope. Note that constructing islands requires (1) each member in S to decide its role between head and normal, (2) a normal node to decide its head node, and (3) a head node, $h \in H_S$, to decide its TTL scope, called *radius*, $r_S(h)$. Note that there is an important trade-off in selecting the radius. If it is too large, there may be a high exposure, and if it is too small, we underutilize the use of scoped multicasts. To make a trade-off, we introduce an *exposure threshold* ε to control the degree of exposure. Thus, the goal is to make the radius as large as possible for a given allowable exposure threshold. Since each node, $a \in S$, initially considers itself to be a head candidate, each node computes its radius as shown in Figure 4.3.

If there are no TTL scope values with exposure ratios that are less than ε for a node *a* (line 2), *a* sets its radius and exposure ratio to 0 and 1 respectively (line 3). Line 5 indicates that each node chooses as large a radius as possible given an exposure threshold. Line 6 and 7 try to reduce the radius if there are unnecessary bandwidth wastes. For example, consider two cases where a node *c* chooses its radius as 2 or 4 respectively in Figure 4.1 (a). Even though the exposure ratios of both cases are the same, with the selection of a larger radius

1: Let $T = \{t|\beta_S(a,t) < \varepsilon, 0 \leqq t \leqq k\}$ where $0 \leqq \varepsilon \leqq 1$
2: **if** $|T| = 0$ **then**
3: $r_S(a) = 0$ and $\beta_S(a,0) = 1$
4: **else**
5: $t_m = \max\{t|t \in T\}$
6: **if** there exists $t_n \in T$ such that $N_S(a,t_n - 1) \neq N_S(a,t_n) = N_S(a,t_n + 1) = \ldots = N_S(a,t_m - 1) = N_S(a,t_m)$ **then**
7: $r_S(a) = t_n$
8: **else**
9: $r_S(a) = t_m$
10: **end if**
11: **end if**

Figure 4.3: Radius selection algorithm

i.e, 4, packets will traverse more links than with a radius of 2. That is, the goal of the radius selection algorithm is to minimize bandwidth waste while satisfying the exposure threshold constraint. Figure 4.4 shows an example of the radius decision rule for node *c* in Figure 4.1 where an exposure threshold, ε, is 0.4. In this example, t_m and t_n are 4 and 2 respectively.

A *weight vector* of *a*, $w_S(a)$, is defined as a 3-tuple including the exposure ratio, radius and node ID, i.e., $< \beta_S(a, r_S(a)), r_S(a), ID(a) >$. An IP address can be used as an ID of a node. The elements of the weight vector are ordered in a lexicographical manner. That is, a weight vector for a node *a* is *less* than that of a node *b*, $w_S(a) < w_S(b)$, if

1) $\beta_S(a, r_S(a)) < \beta_S(b, r_S(b))$ or

2) $\beta_S(a, r_S(a)) = \beta_S(b, r_S(b))$ and $r_S(a) < r_S(b)$ or

3) $\beta_S(a, r_S(a)) = \beta_S(b, r_S(b))$ and $r_S(a) = r_S(b)$ and $ID(a) < ID(b)$.

Once each member, *a*, has a weight vector, the choice of a head node is based on the weight associated with each node: the lower the weight of a node, the higher its priority to assume the role of head. This idea is similar to the algorithm for organizing mobile nodes into clusters proposed in [56]. Each node, *a*, advertises its weight vector only to

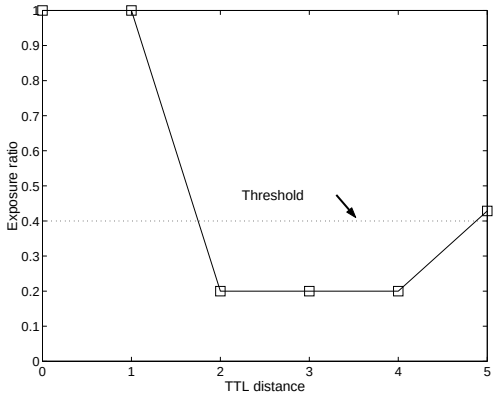


Figure 4.4: Radius selection of c in Figure 4.1

$N(a,r_S(a))$, i.e., performs a scoped multicast with scope $r_S(a)$. After gathering neighbors' weight vectors, each node nominates the one with the lowest weight vector to be its head and notifies its head of its decision. Once a node, a, is elected by at least one normal member, a becomes a head member. Note that a node can nominate itself if there is no other node with lower weight vector.[4] Also note that this algorithm may generate overlapping islands.

4.2.2 Connecting islands

Once each member decides its role, the head members, H_S, are responsible for connecting islands, i.e., building an overlay structure consisting of unicast connections among head nodes. This problem is similar to recent research being conducted on application level multicasting [2, 3, 4]. Our solution is to build filial relationships among head members based on TTL distance information. Each head member finds its parent head member. If a head node nominates itself as a parent node, it becomes a *root* head node for the subgroup, S. This strategy, i.e., finding its own parent, guarantees that every head member participates

[4]This includes the case where a node is so far away from other nodes that other nodes' weight vectors are unavailable.

1: $C = \{n \in H_S | d(n,r) < d(h,r)\}$ where $d(h,r)$ denotes TTL distance between h and r.
2: **if** $|C| \neq 0$ **then**
3: $D = \underset{n \in C}{\text{argmin}}\, d(n,h)$
4: $p(h) = \underset{n \in D}{\text{argmin}}\, ID(n)$
5: **else**
6: $\bar{C} = \{n \in H_S | d(n,r) = d(h,r)\}$.
7: $p(h) = \underset{n \in \bar{C}}{\text{argmin}}\, ID(n)$
8: **end if**

Figure 4.5: Parent head node selection algorithm

in constructing the overlay structure.

Suppose a *reference* node, $r \in G$, periodically sends *reference* packets to the entire session.[5] When receiving reference packets, each member can obtain the distance between itself and the node r. This distance information is used to build parent-child relationships among head members in the session. The parent node, $p(h)$ of a node $h \in H_S$, is selected based on the following rules: Choose the one with the closer distance to r than that of itself. If multiple nodes satisfy this condition, then choose the node which is the closest to itself. If there are again multiple nodes, then choose the one with the lowest ID. Figure 4.5 and 4.6 present the detailed algorithm and an example of the parent selection process respectively. In Figure 4.6, the number in parenthesis is the ID of the node. Note that node c would select its parent in Step 4 of Figure 4.5 while nodes $a,\ b$ would select their parents on Step 7.

The only required information in the above algorithm are TTL distances among head members. Thus, each head member, $h \in H_S$, puts two additional pieces of information in its subgroup advertisement packets: 1) $d(h,r)$ and 2) the fact that it is a head node. However, in the case where its parent head node may be further than k (the scope of subgroup advertisement packets) hops away, an expanding ring search [23] can be used to find the

[5]Note that the reference node is not dependent on any subgroup.

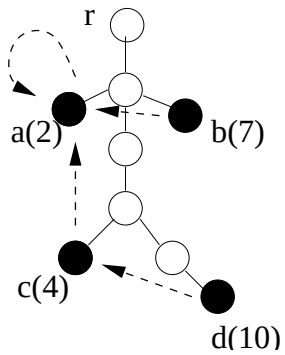


Figure 4.6: An example of parent selection algorithm

parent head node. Once each node, h, sends a *parent nomination* packet, then its parent node, $p(h)$, sends back a *parent confirmation* packet, which sets up a filial relationship.

4.2.3 Forwarding and maintenance of a TSC mechanism

While each node, $a \in S$, sets up a relationship among members in S, it needs to build a routing table, $T_S(a)$ for subgroup S communication. A normal node simply maintains an entry for its head node. A head node stores its radius and entries of nodes which have filial relationships, i.e., one parent and its children, if any. The radius entry in the routing table represents a scoped multicast in the G session with the TTL scope of the radius. Then the following two rules suffice for subgroup S communication: (1) if a node, a, is a source node, broadcast packets over entries in $T_S(a)$ and (2) if a node is a relay node, broadcast packets over entries in $T_S(a)$ except the one from which packets are received.

Note that during the multicast session, the interests of members may change and members may leave or join the global multicast session. Such dynamics are handled by periodic subgroup advertisement packets injected by each member. A change of interest or membership will produce different TTL-neighbor profile, leading to a change in the weight vectors. If a normal node wishes to change its head node, it notifies the previous

head node of its intention, so that the head which no longer has any normal members, can become a normal member. When a head node leaves or becomes a normal node, it notifies its parent and children of the event, so that they update their routing tables and find other parents. Consider the case where nodes abruptly fail or the network is partitioned, wherein explicit notification is impossible. In this case, periodic subgroup advertisement packets indicate the liveness of members, thus our mechanism can dynamically adapt to the situation. However, since there is a scope limit to subgroup advertisement packets (k in Section 4.2.1), parent and child nodes whose distance is farther than k, need to periodically exchange acknowledgment packets.

4.3 Evaluation

We conduct simulations to study various issues and trade-offs in applying the proposed TSC mechanism in multicast applications. Our goal is to investigate in which environments it is advantageous to apply the proposed mechanism. For comparison, we examine the performance of the following schemes for subgroup communication.

• *Global Multicast:* This represents a scheme that simply uses the original global multicast group G for subgroup communications.

• *Unicast-Only:* This scheme constructs unicast overlay trees among subgroup members. Though there have been numerous overlay schemes presented, we use our methods for constructing overlay structures. That is, this scheme can be considered as a TSC mechanism with 0 exposure ratio.

• *TSC-ε:* This represents our proposed scheme with an ε exposure threshold.

4.3.1 Performance Metrics

To evaluate our TSC mechanism, we use the following metrics.

• *Cost ratio*: Let us define the *cost* of a subgroup communication using scheme f by $C_f(S)$, the total number of links traversed by packets generated to distribute a unit amount of data for S subgroup communication.[6] For example, in the case where a new multicast session is created, the cost of subgroup S communication, denoted by $C_{new}(S)$, is simply the number of links in the tree induced by subgroup members. Since $C_{new}(S)$ can be considered optimal, we define a *cost ratio* γ_f as the ratio of $C_f(S)$ to $C_{new}(S)$, i.e., $\gamma_f = C_f(S)/C_{new}(S)$. A value close to 1 for the cost ratio metric represents an efficient use of bandwidth.

• *Global member exposure ratio*: Let $E_f(S)$ be a set of members in G exposed while applying a scheme f for subgroup communication among a set of users. The *global member exposure ratio*, β_f, is defined as $\frac{|E_f(S) \setminus S|}{|E_f(S)|}$. The higher the value of β_f, the more members are exposed.

4.3.2 Methodologies

We have built a simulation tool to compute the above two metrics. We measure them by varying the following elements.[7]

• *Topologies:* We use real multicast trees gathered in [57]. Note that unicast packets will follow the same path taken by multicast packets in our simulation environment, which may not be the case in real world. However, as shown in [57] there is a topological closeness between unicast paths and multicast paths. Thus, we believe that the performance results are valid.

[6] For simplicity, we assume that a link cost is symmetric and unit cost. However, the cost can be generalized with inclusion of asymmetric and variable link costs.

[7] In the simulation, we do not consider a dynamic membership change.

• *Subgroup density:* We vary density of subgroup members from sparse, to mid-range, to dense.

• *Subgroup membership distribution:* We follow the same methodology as proposed in [54] to model topological correlation within a subgroup, i.e., as with random, affinity/disaffinity and distributed clusters. Affinity mode emulates subgroup distributions with members that tend to cluster together and the disaffinity mode is for the subgroup member distribution that tends to be spread out.

We create a subgroup S with m nodes for affinity and disaffinity modes as follows: initially S has no members and we choose subgroup members one by one from the global session G until $|S| = m$. The first node is randomly selected. For k^{th} node selection, we assign a probability $p_i = \frac{\alpha}{g_i^{\theta}}$ to each node $n_i \in G \setminus S$, where $g_i = \min_{n_j \in S} d(n_i, n_j)$ and α is calculated such that $\sum_{n_i \in G \setminus S} p_i = 1$. Then, we randomly select a subgroup member among $C = \{n_i | n_i \in G \setminus S, p_i \geq p\}$ where p is a random value from 0 to 1.[8] We use $\theta = 15$ and $\theta = -15$ for affinity and disaffinity respectively.

In the distributed clusters mode, a few clusters are randomly scattered in the tree and each cluster is modeled according to the affinity mode. We modeled a number of clusters that was linearly increasing as a density of subgroup increases, i.e., $0.3 * density + 2$.

• *Scope of subgroup advertisement packet:* We vary scope of subgroup advertisement packet to investigate its impact on the performance of our TSC mechanism.

4.3.3 Results

Since our results for the various topologies in [57] show similar trends, we only present results for the real multicast tree shown in Figure 4.7. It consists of 2359 nodes and 1487

[8] If $|C| = 0$, then choose a different p value.

end nodes(members).

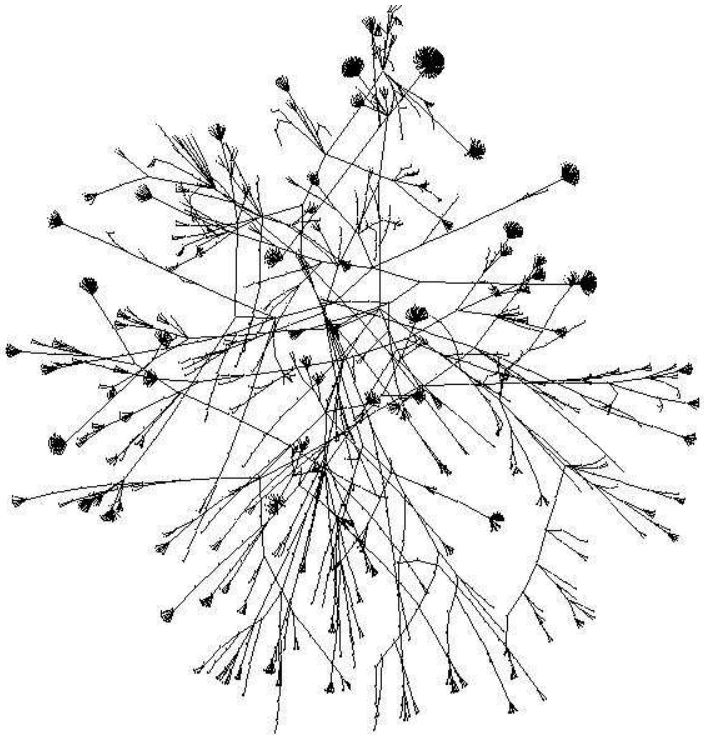
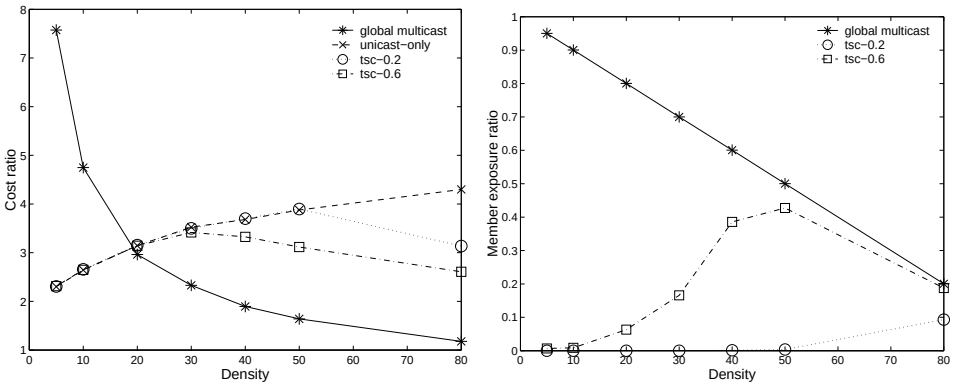


Figure 4.7: A global multicast tree topology

The following figures show the cost ratio and member exposure ratio results varying subgroup densities for the various node distributions. We set 7 as the TTL scope for subgroup advertisement packets. In each figure, we present performance metrics for global multicast, unicast-only, TSC-0.2 and TSC-0.6 schemes. Each point in the figures represents an average over 100 different subgroup distributions for given distribution mode and density. We do not include the member exposure ratio of the unicast-only scheme since it is always 0.

Figure 4.8 shows the results for random node distributions. We observe that the cost ratio of global multicast scheme heavily depends on the density of subgroups: the larger the

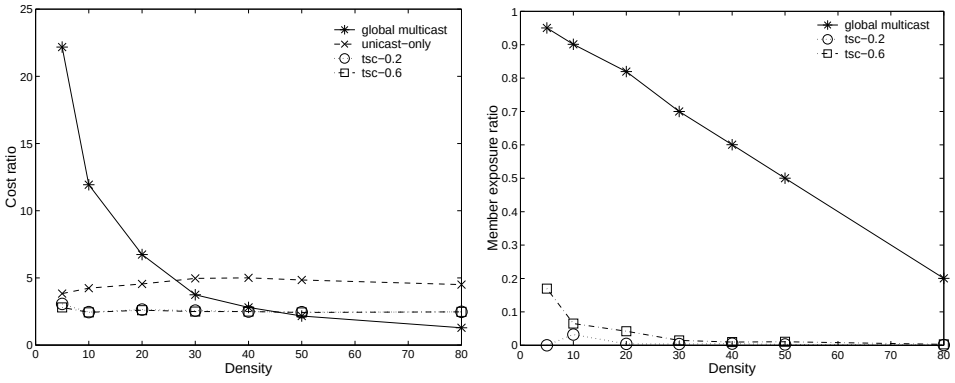


(a) Cost ratio (b) Member exposure ratio

Figure 4.8: Random mode

density of a subgroup is, the lower cost ratio of global multicast is; above 20% density global multicast beat all the other schemes in terms of the cost ratio. However, the member exposure ratio of global multicast scheme is higher than other schemes for all densities. Also note that the member exposure ratio of global multicast is constant regardless of subgroup members' distributions. For low density regimes, we observe that unicast-only and TSC schemes show similar results. This is because members are randomly distributed and the density is low, TSC generates one-member islands in most cases. As subgroup density increases, TSC outperforms unicast-only by using scoped multicast. TSC-0.6 achieves better cost ratio than TSC-0.2 as density increases since TSC-0.6 scheme aggressively forms non-one member islands. However, better cost ratio performance is at the expense of more member exposure ratio as shown in Figure 4.8 (b).

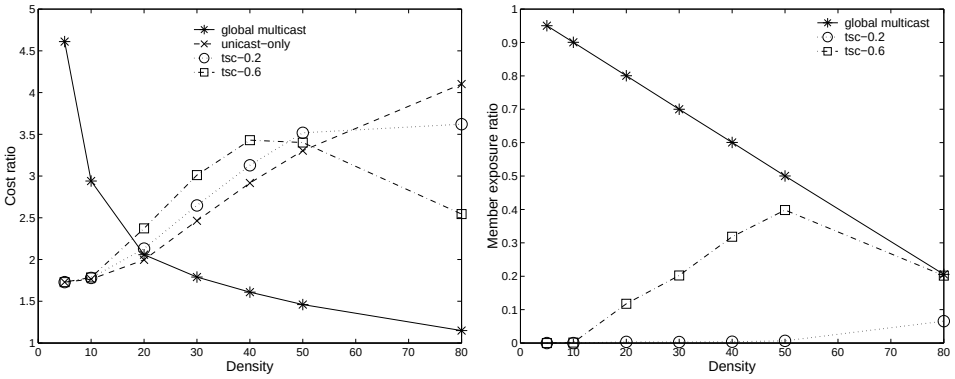
Figure 4.9 shows the performance results for subgroups with nodes placed based on the affinity distribution. Note that since in the affinity mode, members are spatially clustered together, a global multicast scheme causes an excessive cost ratio, e.g., γ= 23 at 5% density. The cost ratio of TSC mechanism is almost two times lower than unicast-only scheme for



(a) Cost ratio (b) Member exposure ratio

Figure 4.9: Affinity mode

the range of densities. TSC-0.2 and TSC-0.6 have almost the same cost ratio results since members are clustered so that the exposure threshold value is not a major factor for creating islands any more. Also note that TSC mechanisms achieve fairly low member exposure ratios.

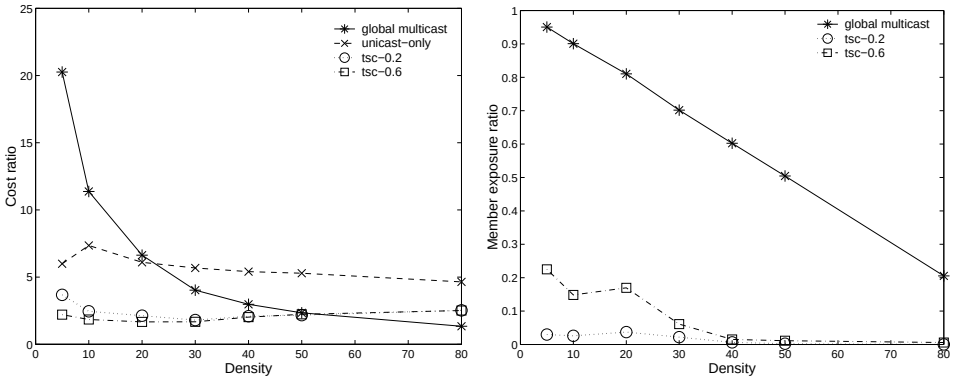


(a) Cost ratio (b) Member exposure ratio

Figure 4.10: Disaffinity mode

Figure 4.10 depicts the results for the disaffinity distribution mode. Both cost ratio and member exposure ratio results show similar trends to those for the random case except

that in the mid-range of densities, the cost ratio of TSC mechanisms is slightly higher than that of the unicast-only scheme. This can be explained since members are spread out from each other in a disaffinity mode, the effort to form islands in a TSC mechanism leads to more link exposure by scoped multicasts. However, for high densities, the cost ratio eventually benefits TSC mechanisms.

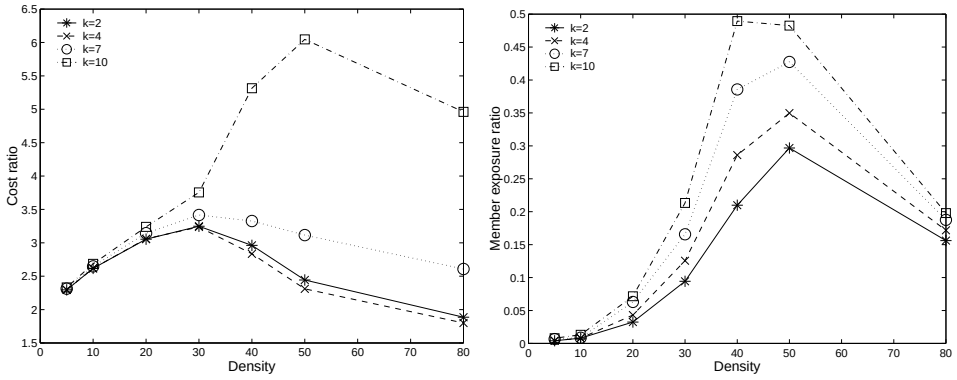


(a) Cost ratio (b) Member exposure ratio

Figure 4.11: Distributed clusters mode

Figure 4.11 shows the performance results for the distributed clusters distribution mode. We observe that the results are similar to the affinity mode.

Figures 4.12, 4.13, 4.14 and 4.15 show the performance results for random, affinity, diaffinity and distributed clusters respectively varying the scope (k) of subgroup advertisement packets, i.e., $k = 2, 4, 7, 10$ with TSC-0.6 scheme. Note that the scope k provides a hard limit for the radius of islands, i.e., the radius cannot be larger than k. We observed that all four distribution modes show similar results for varying scopes of subgroup advertisement packet as follows. First, as the scope becomes smaller, member exposure ratio decreases. This is intuitive since a smaller scope does not allow large islands, which can reduce member exposure. At the extreme case where $k = 0$, the member exposure ratio is 0.

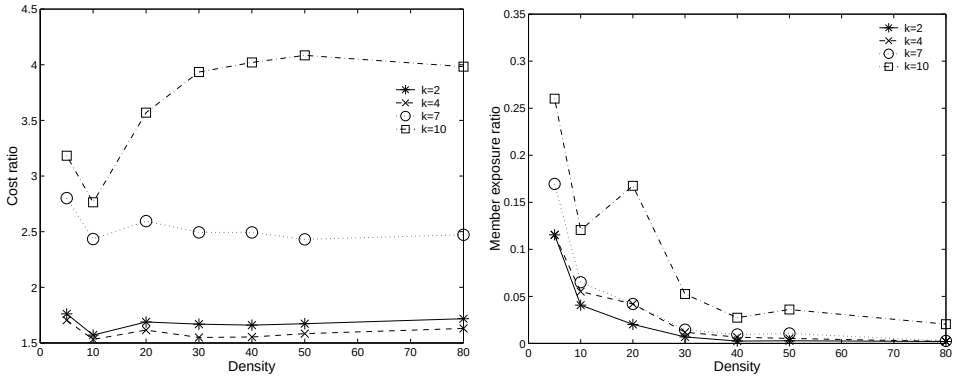


(a) Cost ratio (b) Member exposure ratio

Figure 4.12: Random mode (TSC-0.6)

Second, $k = 4$ is the best choice for the cost ratio metric for all distribution modes. Though smaller scopes generate smaller member exposure ratio, it may underutilize scoped multicast to reduce the cost ratio. Also, if TTL scopes are too large, they generate high cost ratios due to large islands. Thus, an intermediate scope value can produce the lowest scope value, which is $k = 4$ for our simulation results. Third, even a small scope can generate pretty low cost ratio for all distribution modes. For example, $k = 2$ achieves slightly higher cost ratio compared to $k = 4$. This result demonstrates that most benefit from scoped multicast can be achieved with even small scopes. This is an encouraging result since the overhead for control messages for TSC mechanism can be significantly reduced by using subgroup advertisement packets with small scopes.

Through the simulation studies, we observed that different subgroup membership distributions and varying subgroup densities heavily influence the performance of the schemes for subgroup communication. As expected, the TSC mechanism benefits greatly from clustered distributions (affinity and distributed clusters modes). The TSC mechanism also



(a) Cost ratio (b) Member exposure ratio

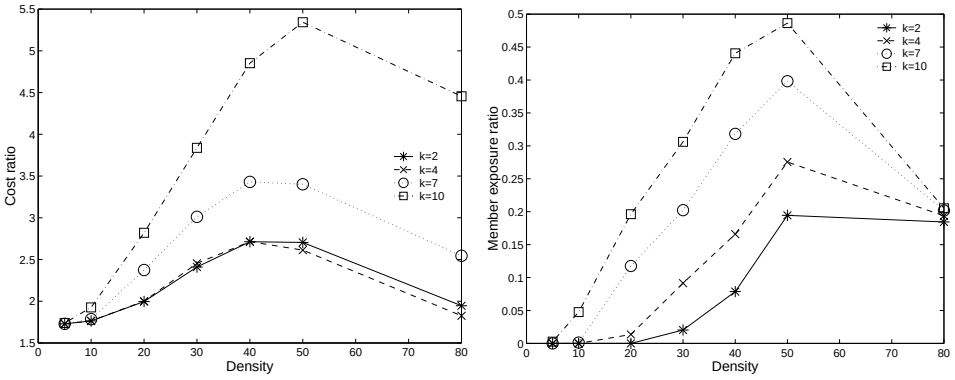
Figure 4.13: Affinity mode (TSC-0.6)

achieves fairly stable cost ratios (from 1.5 to 4) irrespective of variations in density or distribution modes.

4.4 Related work

The scalability of state associated with multicast forwarding by routers has been one of the significant issues for the wide deployment of IP multicast. Reduction of multicast forwarding state at routers can be achieved through aggregation or elimination of non-branching approaches. In [53], multiple multicast forwarding entries are aggregated if entries have adjacent group address prefixes and matching incoming and outgoing interfaces. The goal of dynamic tunnel multicast [58] and REUNITE [59] is to reduce multicast states by eliminating non-branching point. That is, only fan-out (branching) points keep state information, which is mostly beneficial in a sparse distribution of members. Note that the above two approaches require the modification of routers, which may take a long time to be deployed.

The clustering schemes aim to efficiently cluster members into a limited number of multicast sessions based on a preference matrix [52] or players' position in a virtual

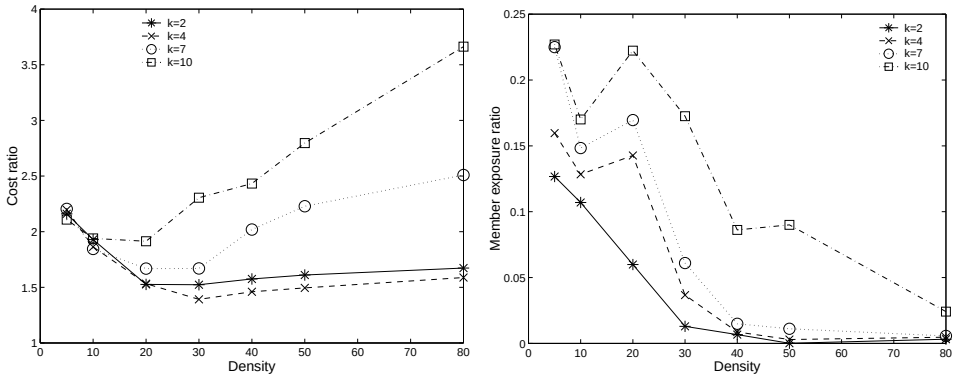


(a) Cost ratio (b) Member exposure ratio

Figure 4.14: Disaffinity mode (TSC-0.6)

cell [55]. Note that the first two approaches (aggregation and non-branching elimination) are at the routing level, that is, trying to eliminate multicast forwarding state at each router. However, the clustering schemes are at the application level, i.e., aim at reducing the number of multicast groups using application specific information. Thus, the first two approaches can be applied to any single multicast group and the clustering schemes are for large-scale multicast applications requiring lots of subgroup communication, which is our aim in this chapter. Note that in clustering schemes, there is a central point where all the member's preference information should be gathered. Due to not only processing overhead but also communication overhead this is unlikely to scale nicely when there is dynamic change of membership or preference in a large-scale multicast session. In contrast, our approach is a distributed algorithm wherein the forwarding decision is made at each node.

Our approach should be contrasted with a number of recent application-level multicast studies, e.g., [2, 3, 4]. First, the goal of application-level multicast is the replacement of IP multicast due to a number of challenges such as infrastructure modification, reliability, flow and congestion control. However, we use the end-to-end approach for reduction



(a) Cost ratio (b) Member exposure ratio

Figure 4.15: Distributed clusters mode (TSC-0.6)

of multicast forwarding state in large-scale multicast applications. In our view IP multicast and application-level multicast may coexist and IP multicast will survive as an important delivery mechanism to serve large-sized groups. Second, our TSC mechanism is not a simple adaptation of unicast overlay solution to the preference heterogeneity problem. It uses scoped multicast by exploiting spatial locality among members. By varying the exposure threshold, it can position itself in the middle of two extreme points: a global multicast and unicast overlay solution.

4.5 Conclusion

In this chapter, we designed and evaluated a topology-sensitive subgroup communication mechanism to handle the preference heterogeneity problem in large-scale multicast applications. Our TSC mechanism takes a complete end-to-end approach which eliminates additional creation of multicast groups. Depending on the local density of subgroup members, members in the session self-configure into islands and forwarding structures. Within islands, scoped multicast is used to derive benefit from clustered membership distribution

and between islands, unicast is used to reduce unnecessary exposure. Throughout our simulations, we observe that our TSC mechanism performs in a consistent way over diverse densities and distribution modes of the subgroup.

Chapter 5

Fast Content Replication

5.1 Introduction

Content delivery networks (CDNs) are deployed to improve network, system and end-user performance by caching popular content on edge servers located close to clients. Since content is delivered from the closest edge server to a user, CDNs can save network bandwidth, overcome server overload problems, and reduce delays to end clients.

CDN edge servers were originally intended for static web content, e.g., web documents and images. Thus, if the requested content was not available or out-of-date, the local server would contact the original server, refresh its local copy, and send it to the client. This *pull* type of operation works reasonably well for small to medium size web content, since the performance penalties for a cache miss, e.g., additional network traffic from the original server to the local server and higher delay to the client, are not significant. However, CDNs have recently been used to the deliver large files, e.g., digital movies, streaming media and software download packages. For large files, it is desirable to operate in a *push* model, i.e., replicating files at edge servers in advance of user's requests, since their distribution

requires significant amounts of bandwidth. The download times may be high, e.g., 20 min media file encoded at 1 Mbit/s results in a 150 MBytes file, or a high quality digital movie may be around 700 MBytes. Such push style file replication across distributed machines is also required for web mirror services.

In this chapter we consider the problem of content distribution across geographically distributed nodes. Our focus is on distributing large files such as software packages or stored media files, and our objective is to minimize the overall replication time, i.e., minimizing the worst case download time to a set of receivers.

5.1.1 Related work

IP multicast [1] is an efficient one-to-many delivery method which can provide a number of operational advantages for content and network providers by reducing the overall resources consumed to achieve such distribution. A single packet transmitted by the source traverses each link in the multicast distribution tree to all receivers in the multicast group. However, the deployment of IP multicast has been hampered by a number of challenges including the need to modify infrastructure and the need to support reliability, flow, and congestion control.

Limited network layer support for multicast in the Internet today, has led to active research on end-system approaches [60], [61], [3], [62], [4], [63], [64], which do not require such infrastructure support, i.e., all multicast related functionalities, including group management and packet forwarding, are implemented at end systems. In this architecture, hosts in the group cooperate to construct an *overlay* structure of unicast connections. One advantage of this approach over IP multicast is that participating hosts have the flexibility to choose which overlay structure is constructed. Thus there is a possibility of routing around

congested links. Furthermore, participating hosts may reconfigure their overlay structure to adapt to dynamic changes in network congestion. Thus, many existing end-system solutions [60], [61], [3], [62], [4], focus on constructing or reconfiguring a "good" overlay structure to optimize the performance according to the application's requirements.

However, the above-mentioned flexibility in building overlay structures comes at a price. That is, building optimized overlay structures requires path quality information among hosts, which in turn must be obtained through probing. Since overlay paths may share common physical links, *sequential* probing to estimate available bandwidth on end-to-end paths (i.e., without the presence of other overlay path probing) may result in poor choice. If this is the case, *joint* probing over a large number of combinations should be performed, which may lead to huge overheads. After building an overlay structure, participants need to maintain it by exchanging control signals. To adapt to dynamic network situations, additional monitoring of alternative paths may be required. Furthermore, while restructuring happens, further overheads may be incurred due to lost packets or reconfiguration.

5.1.2 Contributions

To improve the delivery time in distributing large files in the context of a content delivery network, we proposed *FastReplica*, which is also an application level approach [65]. FastReplica uses two key ideas: (1) file splitting and (2) multiple concurrent connections. That is, the source divides the original file into *m* (the number of receivers) chunks, and concurrently transmits a different chunk to each receiver. Meanwhile each receiver relays its chunk to the remaining nodes so that each node ends up with all *m* chunks.[1] To support a larger number of receivers in the group, the above basic algorithm can be applied iteratively

[1]This will be described in detail in Section 5.2.

using a hierarchical k-ary tree structure where k is chosen small enough to support efficient file distribution using the FastReplica algorithm.

In contrast with most existing end-system approaches, FastReplica does not attempt to build a "good" overlay structure, but simply uses all available paths, i.e., fixed m^2 overlay paths among the source and m receivers. This reduces flexibility, but also reduces the overheads incurred in probing, building and reconfiguring the overlay structure. Experiments on a wide-area testbed showed the potential of this approach to reduce the overall replication time [65].

Despite these encouraging results, there are inherent weaknesses with the FastReplica scheme. FastReplica is oblivious to heterogeneity in its m^2 overlay paths, i.e., some paths are "good" and others "bad". FastReplica simply puts an equal amount of data on each chunk. In this case, the overall replication time depends on the completion time of the chunk traversing the worst path. Heterogeneity in the overlay paths may arise due to at least three factors. First, it is inherent in network resources. Infrastructure-based CDNs or web server replica networks are equipped with a dedicated set of machines and/or network links, which are engineered to provide a high level of performance, however there are inherent heterogeneities among network resources, e.g., different capabilities of servers or available capacity on network links. Second, even with homogeneous capabilities on network resources (e.g., nodes' capabilities and capacities of links between all end points are uniform), each chunk transfer may not achieve the same throughput since multiple overlay paths may be mapped onto the common physical links. A link shared by multiple flows generated by FastReplica becomes bottleneck point, which will dominate the transfer time. Third, Internet traffic is variable. The available bandwidth on each path may vary with time possibly even during the transfer of a given file. In summary, although FastReplica

exploits path diversity, it is unaware of heterogeneous resources, path sharing and dynamic environments.

To overcome the above-mentioned limitations, in this chapter we propose the *Adaptive FastReplica (AFR)* mechanism. The key idea in AFR is to have chunks of different sizes depending on network conditions. That is, the source node sends a smaller portion of the file on "bad" paths and larger one on "good" paths. Several questions need to be answered to make this type of application level load balancing practical.

- Which criterion should be used to decide which paths are "good" or "bad"?
- How does the source obtain such information?
- How large should the source make each chunk?

We propose an analytical network model, and study the problem of determining the optimal partition. Based on the insights from the analysis and considering a practical context, we propose the AFR mechanism to expedite file transfers. We implemented the proposed scheme, and experimented with it on the Internet. Our performance evaluation shows that there are significant performance gains for AFR over FastReplica, and that the gain becomes more significant if network state is changing dynamically.

5.1.3 Organization

The rest of the chapter is organized as follows. Section 5.2 introduces our framework and network model. In Section 5.3 we formulate and solve the optimization problem associated with minimizing the overall replication time, and discuss our practical approach. Section 5.4 discusses an implementation of the Adaptive FastReplica mechanism. This is followed by Section 5.5 wherein we present our experimental results over a wide-area network environment. In Section 5.6 we discuss additional related work and Section 5.7 concludes the

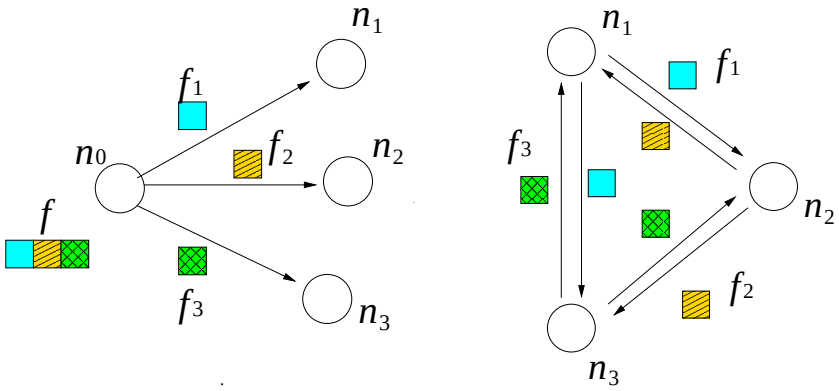
chapter.

5.2 Framework & Network model

In this section, we will present our framework, notation, *and* network and bandwidth allocation model used throughout the chapter.

5.2.1 Replication framework

Consider the problem of replicating a file f originating at node n_0 across a set of receiver nodes, $R = \{n_i | i \in N\}$, where $N = \{1, \ldots, m\}$ is the index set for receivers. Throughout this chapter f will denote not only a file but also its size in bytes. The file f is divided into m chunks, $f_1, \ldots, f_m$, such that $\sum_{i=1}^{m} f_i = f$ and $f_i \geq 0$, $i \in N$. To represent the portion of the original file that goes to each chunk, we define a *partition ratio vector*, $\mathbf{x} = (x_i = \frac{f_i}{f}, i \in N)$. Note that $0 \leq x_i \leq 1$ and $\sum_{i=1}^{m} x_i = 1$. The mechanism underlying FastReplica includes two basic activities.



(a) At the source. (b) At the receivers.

Figure 5.1: An illustration of the FastReplica framework.

- *At the source n_0:* Node n_0 opens m concurrent connections to the replicating set, $n_1, \ldots, n_m$,

and sends to each node n_k the following two items: 1) a list of replicating nodes, $\{n_i | i \in N\}$ and 2) a chunk f_k. This procedure is shown in Figure 5.1(a) in the case where $m = 3$.

• *At each node n_k:* After receiving the list of replicating nodes, each n_k opens $m-1$ concurrent connections to the remaining nodes, $R \setminus \{n_k\}$, and relays its chunk, f_k to each of them. This procedure is depicted in Figure 5.1(b). Here, ideally a *cut-through* operation is used to relay data, i.e., relaying nodes forward data on the fly as they receive it, instead of waiting for arrival of the entire chunk.[2]

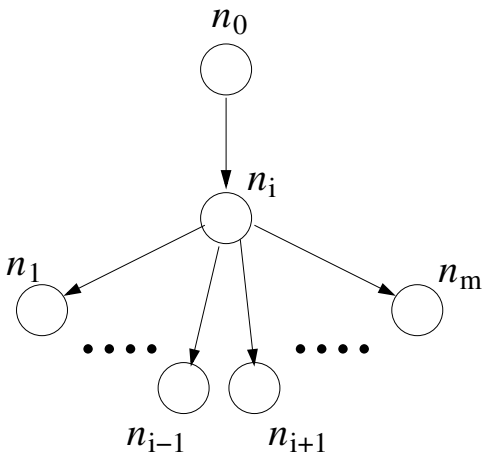


Figure 5.2: Overlay tree of the i^{th} chunk.

Figure 5.2 shows the *overlay tree* associated with chunk i. As can be seen, the tree includes m *overlay links*. Since there are m chunks and thus m such trees, FastReplica uses m^2 overlay links to realize the file transfer. Note that an overlay link between two nodes may consist of multiple physical links and routers, i.e., it corresponds to a unicast path in the underlying physical network. Also, note that multiple overlay paths may share the same physical link.

We shall let $t_{ij}(\mathbf{x})$, $i, j \in N$, denote the *transfer time* of the i^{th} chunk from node n_0

[2]Note that the cut-through operation was not assumed nor implemented in the original implementation of FastReplica [65].

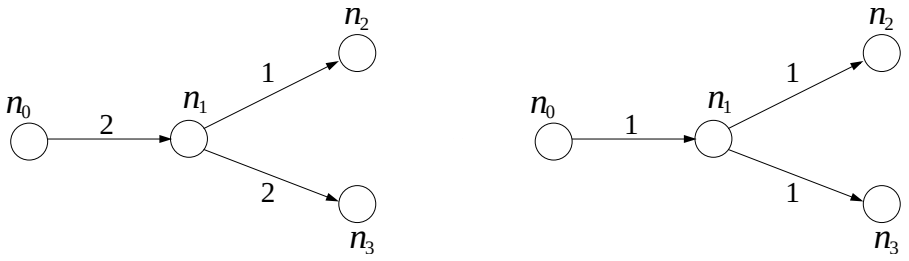
to node n_j, when the partition vector $\mathbf{x}$ is used. We further define the *download time* to the j^{th} receiver, $d_j(\mathbf{x}) = \max_{i \in N} [t_{ij}(\mathbf{x})]$ and the *worst case transfer time* for the i^{th} chunk, $t_i(\mathbf{x}) = \max_{j \in N} [t_{ij}(\mathbf{x})]$. Note that these times all depend on the partition ratio vector $\mathbf{x}$. In the sequel we will occasionally suppress $\mathbf{x}$ in these notations.

5.2.2 Network and Bandwidth allocation Model

In our analytical network model, we assume that file transfer time is governed by the available bandwidth from a source to a destination node. This is reasonable when large files are being transferred [66].

We consider two models for bandwidth availability: *point-to-point* sessions and *tree* sessions. In the point-to-point session model, the overlay tree associated with each chunk consists of m independent point-to-point sessions. A chunk may be delivered at different rates along different overlay paths. Figure 5.3(a) shows an overlay tree for the 1^{st} chunk where $m = 3$. In the tree, there are three point-to-point sessions from n_0 to n_1, from n_1 to n_2, and from n_1 to n_3. The values next to the overlay links in Figure 5.3(a) represent the available bandwidth to each session. A transfer can achieve such transmission rates on each session if the available bandwidths are decreasing along paths in the tree. If there is a strict decrease, then an intermediate node needs to buffer the data to be transmitted by the downstream session. By contrast, in our tree session model, we assume a chunk is transferred at the same rate along the entire overlay tree. This can be achieved by coupling transmissions from the source and relay nodes (e.g., through backpressure or flow control), i.e., n_0 sends each chunk at the minimum rate associated with its overlay tree. This is shown in Figure 5.3(b).

In our subsequent analysis, we will model bandwidth availability based on tree



(a) Point-to-point session model. (b) Tree session model.

Figure 5.3: An example of bandwidth allocation to the overlay tree for the 1^{st} chunk when $m = 3$.

sessions. This not only simplifies our analysis but also is consistent with our objective of minimizing the overall replication time. Indeed, note that in a point-to-point session model, fast transmission rate from n_0 to n_1 or from n_1 to n_3 in Figure 5.3(a) does not help to minimize the *overall* replication time as compared to the strategy in Figure 5.3(b). This is because the overlay link with the minimum available bandwidth is the bottleneck to the chunk's overall replication at all receivers.

We consider a network consisting of a set of links $\mathcal{L}$ with capacity $\mathbf{c} = (c_l, l \in \mathcal{L})$. We associate a *tree* session with each chunk. Let $\mathcal{S}$ denote the set of m tree sessions sharing the network. Bandwidth is allocated among tree sessions, according to an appropriate criterion, e.g., max-min fair allocation [67], proportional fair allocation [68], max-throughput allocation [69], or that realized by coupled TCP sessions.

For $s_i \in A \subset \mathcal{S}$, we let $a^*_{s_i}(A)$ denote the bandwidth allocated to session s_i when only the tree sessions in A persist on the network. Subsequently, we will call $A \subset \mathcal{S}$ the *active set*, i.e., the set of tree sessions which still have a backlog to send. For $s_i \in \mathcal{S}$, we let $a^*_{s_i}$ denote the bandwidth allocated to s_i when all m tree sessions are active in the system, i.e., $a^*_{s_i} = a^*_{s_i}(\mathcal{S})$.

Since the same bandwidth is allocated along all paths of the tree session for each

chunk, the completion times under this model will satisfy $t_{ij} = t_i, \forall j \in N$, i.e., all receivers will get each chunk at the same time. Each tree session $s_i \in \mathcal{S}$ traverses a set of physical links $\mathcal{L}_{s_i}$ associated with the overlay tree for i^{th} chunk. Recall that there may be multiple-crossings of the same link by a given tree session, and such multiplicities can be easily accounted for.

Since there is a one-to-one mapping between a chunk and a tree session, we will use these interchangeably in our notation, i.e., we will equivocate the i^{th} chunk with s_i session. Thus, we have $a_i^* = a_{s_i}^*$ and $N = \mathcal{S}$ etc.

5.3 Analysis

5.3.1 Optimal partition ratio

Note that by making partition ratios the same for all chunks, i.e., $x_i = 1/m$, $i \in N$, the scheme in Section 5.2 corresponds to the original FastReplica algorithm [65]. In this section, we formulate an optimization problem whose objective is to minimize the overall replication time by controlling the partition vector $\mathbf{x}$ under the tree session model introduced in Section 5.2.2.

Problem 5.1 *Suppose we are given a file f at a source node n_0 and a receiver set R. Under the tree session bandwidth allocation model, determine a partition ratio vector, $\mathbf{x} = (x_1, \ldots, x_m)$ which minimizes the overall replication time $r(\mathbf{x})$, given by*

$$r(\mathbf{x}) = \max_{j \in N} [d_j(\mathbf{x})] = \max_{i,j \in N} [t_{ij}(\mathbf{x})] = \max_{i \in N} [t_i(\mathbf{x})].$$

By controlling the partition ratio vector, the source can determine how much data is injected into each tree session, i.e., fx_i will be delivered through the i^{th} tree session. By making $x_i = 0$, the source can decide not to use the i^{th} tree session at all. Depending on the partition ratio vector **x** and the network capacity, the bandwidths allocated for tree sessions may change dynamically over time. This is because if a session leaves the system (i.e., a chunk is successfully delivered), the network resources will be shared among the remaining sessions, possibly resulting in a new bandwidth allocation.

Let us consider an example to understand dynamics associated with Problem 5.1, i.e., how the transfer time of each chunk and the available bandwidth for each session are dynamically determined. We will suppose for simplicity that bandwidths among trees are allocated according to max-min fair criterion[3] in our examples.

Suppose the file size is $f = 4$ and the number of receivers is $m = 3$. Sessions s_1 and s_2 share a physical bottleneck link l_1 whose capacity is 2. Sessions s_2 and s_3 share a bottleneck link l_2 whose capacity is 3. The remaining links in the network are unconstrained and not shown in Figure 5.4.

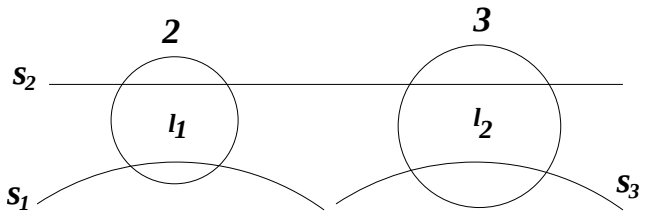


Figure 5.4: An example network.

Figure 5.5 shows each session's transfer time and assigned available bandwidth as time evolves when the partition ratio vector is $\mathbf{x} = (0.1, 0.3, 0.6)$. Times t_1, t_2, and t_3 in

[3]In max-min fair bandwidth allocation method, each session crossing a link should get as much as other such sessions sharing the link unless they are constrained elsewhere. Thus, it has the following characteristics: (1) each session has a bottleneck link, and (2) unconstrained sessions at a given link are given an equal share of the available capacity [67], [70].

Figure 5.5 represent the transfer time of chunk 1, 2 and 3 respectively. Over the time, we have the following active sets and bandwidth allocations:

$$
\begin{aligned}
0 \le t \le t_1, \quad & A = \{s_1, s_2, s_3\}, \\
& a_1^*(A) = a_2^*(A) = 1, a_3^*(A) = 2 \\
t_1 < t \le t_2, \quad & A = \{s_2, s_3\}, a_2^*(A) = a_3^*(A) = 1.5 \\
t_2 < t \le t_3, \quad & A = \{s_3\}, a_3^*(A) = 3
\end{aligned}
$$

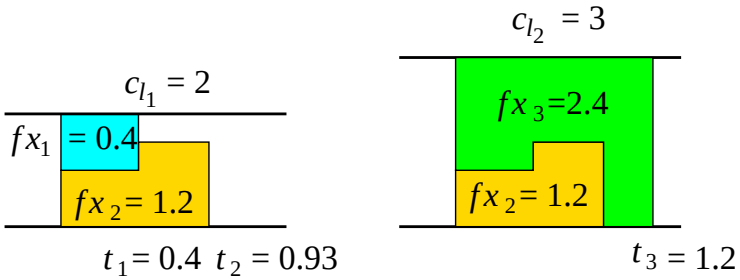


Figure 5.5: Max-min fair bandwidth allocation for the network in Figure 5.4 with $\mathbf{x} = (0.1, 0.3, 0.6)$ and $f = 4$.

For a given active set A, we let $g(A) = \sum_{i \in A} a_i^*(A)$ denote the *aggregate bandwidth*, i.e., the sum of bandwidths allocated to tree sessions in A.

Theorem 5.1 *Suppose that A^* maximizes the aggregate bandwidth among all possible active sets in $\mathcal{S}$, i.e.,*

$$A^* \in \underset{A \subset \mathcal{S}}{argmax}\, g(A) = \underset{A \subset \mathcal{S}}{argmax} \sum_{i \in A} a_i^*(A). \tag{5.1}$$

Then, x_i^* given by

$$x_i^* = \begin{cases} \frac{a_i^*(A^*)}{\sum_{k\in A^*} a_k^*(A^*)}, & i \in A^* \\ 0 & i \in \mathcal{S} \setminus A^* \end{cases}, \tag{5.2}$$

is an optimal solution to Problem 5.1.

Proof Note that under the partition ratio x_i^*, all sessions in A^* complete at the same time, i.e., $\frac{fx_i^*}{a_i^*(A^*)} = \frac{f}{\sum_{k\in A^*} a_k^*(A^*)}$, for all $i \in A^*$. By definition, A^* offers the maximum instantaneous aggregate bandwidth rate to the receivers throughout the entire transfer. Thus, the proposed partition ratio vector must be optimal, though not necessarily unique. ∎

Note that from the perspective of the receivers, $g(A)$ is the amount of data per unit time they will get when all sessions in A are being used for file transfer. A higher aggregate bandwidth will result in the lower overall replication time. Thus, Theorem 5.1 says that 1) we need to find which active set (tree session configuration) provides us with the maximal aggregate bandwidth, and 2) given such a set, say A^*, the partition ratio x_i^* satisfying Eq.(5.2) guarantees that the maximal aggregate bandwidth will be achieved throughout the file transfer. For example, for the network in Figure 5.4, we obtain $A^* = \{s_1, s_3\}$, $g(A^*) = 5$, $\mathbf{x}^* = (2/5, 0, 3/5)$, $r(\mathbf{x}^*) = \frac{f}{5}$ and for the network in Figure 5.6(a), we have $A^* = \{s_1, s_2, s_3\}$, $g(A^*) = 4$, $\mathbf{x}^* = (1/4, 1/4, 1/2)$, $r(\mathbf{x}^*) = \frac{f}{4}$.

Note that Theorem 5.1 does not assert that the solution is unique, i.e., there may be multiple optimal solutions not satisfying the conditions in Theorem 5.1. For example, in Figure 5.6(b), all tree sessions are constrained at a single bottleneck, and any partition ratio vector will be an optimal solution.

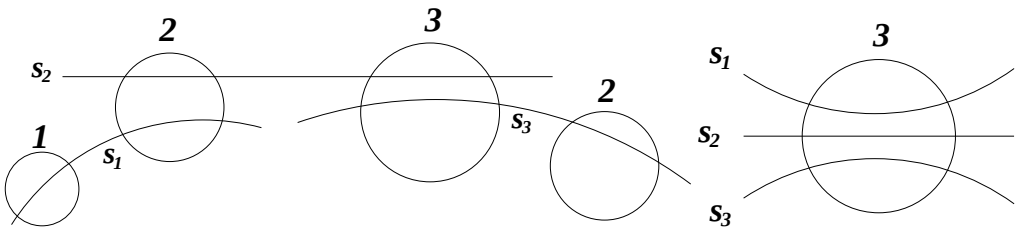


Figure 5.6: Example networks.

5.3.2 Practical solution to obtaining good partitions

Theorem 5.1 provides an optimal strategy based on the available bandwidth along overlay trees which minimizes the overall file replication time. However, there are critical limitations to applying this results in practice. One of them is obtaining the active set A^* which maximizes the aggregate bandwidth. This is a complex combinatorial problem requiring either detailed knowledge of the network and tree structures, or possibly a brute force search over $2^m - 1$ possible solutions.

Instead of searching for an optimal active set, we propose to use all tree sessions (m overlay trees) for the file transfer, i.e., $A = \mathcal{S}$, and use the partition ratio vector suggested by Theorem 5.1:

$$x_i^+ = \frac{a_i^*}{\sum_{k \in \mathcal{S}} a_k^*}, \quad i \in \mathcal{S} \ . \tag{5.3}$$

Recall that $a_i^* = a_i^*(\mathcal{S})$ in Section 5.2.2. Note that this solution may or may not be an optimal solution, e.g., for the cases in Figure 5.4 and Figure 5.6(a), this approach results in sub-optimal and optimal solutions respectively. Although hard to prove, we believe that multiple concurrent overlay tree sessions are likely to produce higher aggregate bandwidth in practice. In part because concurrency delivers higher throughput, and the impact on allocations among multiple trees might be small.

Note that the partition ratio for the i^{th} chunk, x_i^+ is proportional to the bandwidth a_i^*. It is intuitive that for a 'bad' route (small a_i^*) a small chunk should be selected, and for a 'good' route (large a_i^*) a large size chunk should be selected in order to reduce the overall replication time. Also recall that this partition ratio will make the transfer times for each chunk identical.

Under our model, the overall replication times for FastReplica (FR) and our approach (AFR) would be[4]

$$r^{FR} \approx \frac{f}{m \min_{i \in \mathcal{S}} a_i^*} , \tag{5.4}$$

$$r^{AFR} = \frac{f}{\sum_{i \in \mathcal{S}} a_i^*} . \tag{5.5}$$

Since $\sum_{i \in \mathcal{S}} a_i^* > m \min_{i \in \mathcal{S}} a_i^*$, our approach will always beat FastReplica in terms of minimizing overall replication time. While the performance of FastReplica is limited by a single worst bottleneck link, the performance of our approach is limited by the sum of the m tree bottleneck links.[5]

5.3.3 Throughput-based approach

Once we choose to use all overlay trees for the file transfer, one more hurdle remains. In order to determine a partition ratio vector, a source needs prior knowledge of all available bandwidth for each overlay tree, which is not easy to measure [71]. Moreover, extra measurement traffic may need to be injected into the network. To overcome this limitation and reduce the overheads to obtain path quality information, we propose the following *throughput-based* approach.

[4]Note that r^{FR} is an approximation in Eq.(5.4). This is because all chunks may not complete at the same time.

[5]Even though the performance of FastReplica is limited by a single bottleneck, only $\frac{1}{m}th$ of the entire file will be transferred on each overlay path. This can alleviate the congestion level.

Consider a case where a source has a large number of files (or blocks of the same file), $f(1),\ldots,f(n),\ldots$ to be replicated across a receiver set R where $f(n)$ denotes the n^{th} file (block) to be transferred. Throughout the rest of the chapter, the number inside parenthesis represents an iteration step, e.g., $\mathbf{x}(n)$ is a partition ratio vector used for n^{th} file (block) transfer and $t_i(n)$ is the worst transfer time for the i^{th} chunk for n^{th} file (block) transfer.[6]

The following describes our throughput-based approach:

- For the first file transfer, the source uses an arbitrary (possibly cached based on previous transfers) partition ratio vector $x_i(1) > 0$, $i \in N$.
- At the $(n-1)^{th}$ iteration step ($n \geq 2$), each receiver n_j measures $a_{ij}(n-1) \equiv \frac{f(n-1)x_i(n-1)}{t_{ij}(n-1)}$, $i \in N$ defined as an *average throughput* achieved by the i^{th} chunk to receiver n_j and sends it to the source.
- At the n^{th} iteration step, the source updates the partition ratio vector $\mathbf{x}(n)$:

$$x_i(n) = \frac{a_i(n-1)}{\sum_{k=1}^{m} a_k(n-1)},\ i \in N \tag{5.6}$$

where $a_i(n-1) = \min_{j \in N} [a_{ij}(n-1)]$.

In the above approach, we replace the theoretical available bandwidths with receiver estimates for average throughputs in Eq.(5.3). The intuition for this strategy is that we consider a route 'good' if it achieved a high throughput and a route 'bad' if it saw a low throughput.

This approach has several practical advantages. First, the overhead to obtain path characteristics is low. It is easy to estimate average realized throughput rather than available bandwidth. Furthermore, since it uses an in-band measurement, this approach does not generate any extra traffic except feedback messages from receivers to the source. Second, the approach can provide more accurate path information as compared to *sequential* probing.

[6] Recall that here in time notations, we suppress a partition ratio vector $\mathbf{x}$, i.e., $t_i(\mathbf{x},n) \equiv t_i(n)$.

Note that multiple overlay paths are concurrently being used in this framework. Thus our estimates for the average realized throughputs are obtained in the presence of other active overlay trees. Hence, they will provide fairly accurate path quality information. Finally, in practice, file transfer times between two hosts will depend on a number of factors, e.g., not only available bandwidth but also network latency, probability of packet losses, and overhead to create TCP connections and processes etc. Additionally, the bottleneck might be the relaying node due to limited memory or CPU processing power. Since the various components are coupled in a complicated way, it is extremely difficult to decouple them, and identify which components become bottlenecks. However, we note that irrespective of the cause for the bottleneck, the achieved throughput is a fair indicator for the quality of the path.

5.3.4 Convergence

In the above throughput-based approach, the source initially has no knowledge of the path characteristics. However, it learns path quality information from past file transfers, and updates partition ratios. A natural question to ask is whether this procedure would converge, and if so, how quickly when the network capacity is static, i.e., there is no other interfering traffic. That is, irrespective of the initial chunk sizes, will our throughput-based algorithm converge to the partition ratio vector given in Eq. (5.3)?

The difficulty here is that when the partition ratio is not optimal, different chunks will complete at different times. Thus, one will obtain possibly biased estimates of the available bandwidth for tree sessions which see dynamically changing bandwidth allocations. To this end we will prove the following convergence result.

Theorem 5.2 *Under throughput-based adaptation with max-min fair bandwidth allocation*

for tree sessions, given any initial partition ratio vector $x_i(1) > 0,\ i \in N$*, the partition ratio* $x_i(n)$ *converges geometrically to* x_i^+ *given in Eq.* (5.3).

Proving Theorem 5.2 is equivalent to showing that for each session $s_i \in \mathcal{S}$, the average throughput $a_{s_i}(n)$ converges geometrically to $a_{s_i}^*$. Max-min fair allocation can be characterized in terms of a *hierarchy* of sets of bottleneck links and sessions [72], [70]. We prove this theorem by induction on the bottleneck hierarchy. For each level of bottleneck, we obtain a lower bound and an upper bound for average throughput $a_{s_i}(n)$, and show both bounds converge geometrically to $a_{s_i}^*$ because the iterations are monotonic, or correspond to a contraction mapping.

Before proceeding, we formally define the hierarchy bottlenecks related to max-min fairness that will be useful in the sequel. We define the *fair share* $y_l^1 = c_l/m_l^1$ at a link $l \in \mathcal{L}$ as a fair partition of capacity at the link in the 1^{st} level of the hierarchy, where $m_l^1 = |\mathcal{S}_l|$ is the number of sessions through l. Then, the set of 1^{st} level bottleneck links and sessions are defined as follows:

$$\begin{aligned} \mathcal{L}^{(1)} &= \{l \in \mathcal{L} | \forall s \in \mathcal{S}_l,\ a_s^* = b^{(1)} = \min_{k \in \mathcal{L}} y_k^1\}, \\ \mathcal{S}^{(1)} &= \{s \in \mathcal{S} | s \in \mathcal{S}_l \text{ and } l \in \mathcal{L}^{(1)}\}. \end{aligned}$$

where a_s^* is the bandwidth assigned for the session s. Thus $\mathcal{L}^{(1)}$ is the set of 1^{st} level bottleneck links such that the sessions in $\mathcal{S}^{(1)}$ traversing these links are allocated the minimum bandwidth ('fair share') in the network, denoted by $b^{(1)}$. These two sets make up the 1^{st} level of the bottleneck hierarchy.

The next level of the hierarchy is obtained by applying the same procedure to a reduced network. The reduced network is obtained by removing the sessions in $\mathcal{S}^{(1)}$. The capacity at each link in $\mathcal{L} \setminus \mathcal{L}^{(1)}$ traversed by sessions in $\mathcal{S}^{(1)}$ is reduced by the bandwidth

allocated to these sessions. The bottleneck links $\mathcal{L}^{(1)}$ are also removed from the network. Thus $\mathcal{L}^{(2)}$ and $\mathcal{S}^{(2)}$ are obtained based on a network with fewer links and sessions and adjusted capacities. The set of 2^{nd} level bottleneck links, sessions, and bottleneck rate $b^{(2)}$ are defined as before. This process continues until no sessions remain.

Let $\mathcal{U}^{(h)} = \cup_{k=1}^{h} \mathcal{L}^{(k)}$ and $\mathcal{V}^{(h)} = \cup_{k=1}^{h} \mathcal{S}^{(k)}$. These are defined as the cumulative sets of bottleneck links and sessions, respectively, i.e., for levels 1 to h of the hierarchy. The fair share y_l^h $(h \geq 2)$ of link l in $l \in \mathcal{L} \setminus \mathcal{U}^{(h-1)}$ is defined as the fair share of the available capacity at the link in the h^{th} level of the hierarchy:

$$y_l^h = \frac{c_l - \beta_l^{h*}}{m_l^h} \tag{5.7}$$

where $\beta_l^{h*} = \sum_{s \in \mathcal{S} \cap \mathcal{V}^{(h-1)}} a_s^*$ is the total flow of sessions through l which are constrained by bottleneck links in $\mathcal{U}^{(h-1)}$, and $m_l^h = |\mathcal{S}_l \setminus \mathcal{V}^{(h-1)}|$, where $m_l^h > 0$, is the number of sessions through l which are unconstrained by the links in $\mathcal{U}^{(h-1)}$. Based on the fair share, the set of h^{th} level $(h > 2)$ bottleneck links and sessions can be defined as:

$$\begin{aligned} \mathcal{L}^{(h)} &= \{l \in \mathcal{L} \setminus \mathcal{U}^{(h-1)} | \forall s \in \mathcal{S}_l,\, a_s^* = b^{(h)} = \min_{k \in \mathcal{L} \setminus \mathcal{U}^{(h-1)}} y_k^h\}, \\ \mathcal{S}^{(h)} &= \{s \in \mathcal{S} \setminus \mathcal{V}^{(h-1)} | s \in \mathcal{S}_l \text{ and } l \in \mathcal{L}^{(h)}\}. \end{aligned} \tag{5.8}$$

Here $\mathcal{L}^{(h)}$ is the set of h^{th} level bottleneck links such that the sessions in $\mathcal{S}^{(h)}$ are allocated the minimum fair share in the reduced network. We repeat this procedure until we exhaust all the links and sessions resulting in a hierarchy of bottleneck links and corresponding sessions $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(q)}$ and $\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(q)}$, which is uniquely defined by (5.7) and (5.8), where q is the number of levels in the hierarchy.

In the sequel, when we refer to the hierarchy of bottlenecks we mean the *initial* hierarchy structure when all sessions are active, i.e., $A = \mathcal{S}$. Consider an h^{th} level link $l \in \mathcal{L}^{(h)}$ of this bottleneck hierarchy. Note that the sessions sharing link l can be partitioned

into those in levels 1 to h, i.e., $\mathcal{S}_l = \cup_{j=1}^{h}[\mathcal{S}_l \cap \mathcal{S}^{(j)}]$. For sessions in the j^{th} level of the hierarchy, suppose we order them by their finishing times (i.e., when they depart from the system) on the n^{th} file transfer. We let $s_i^j(n)$ denote the i^{th} session to leave the system among j^{th} level sessions at the n^{th} step.[7] Thus we have that

$$t_{s_1^j}(n) \leq t_{s_2^j}(n) \leq \ldots \leq t_{s_k^j}(n),$$

where $k = m_l^j$. Note that at each iteration step, the order in which sessions complete may change.

For the session $s_i^h(n)$, we let $p^j(s_i^h(n))$ be the number of sessions at the j^{th} level whose departure times are equal or less than that of session $s_i^h(n)$. Then, note that at time $t_{s_i^h}(n)$, the number of remaining j^{th} level flows in the system is $m_l^j - p^j(s_i^h(n))$ since m_l^j is the total number of j^{th} level sessions in link l. Figure 5.7 illustrates the notation we have defined.

Proof of Theorem 5.2. We will prove Theorem 5.2 by induction on the *initial* bottleneck hierarchy which is constructed when all sessions are in the system. Without loss of generality, we let the file size be 1, i.e., $f(n) = 1,\ n \geq 1$.

Step 1: Consider a 1^{st} level bottleneck link $l \in \mathcal{L}^{(1)}$. Note that for any session s in $\mathcal{S}_l$ and at any iteration step n,

$$b^{(1)} \leq a_s(n). \tag{5.9}$$

Indeed the bandwidth rates for all first level sessions are non-decreasing over time. That is, once sessions start to leave the link l, the additional bandwidth available to the remaining sessions can only result in an increase in the bandwidth allocated to a first level bottleneck session.

[7] For simplicity, we may suppress an iteration step index in notations, e.g., $t_{s_i^j(n)}(n) = t_{s_i^j}(n)$.

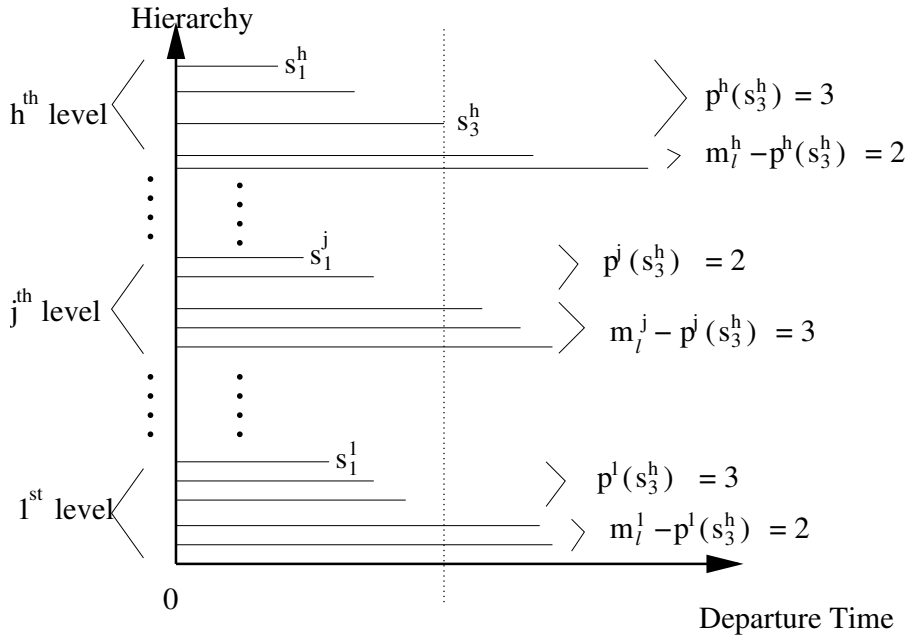


Figure 5.7: Sessions in the h^{th} level link.

For s_i^1, the i^{th} session to leave the system among the first level sessions on link l, we have a lower bound on its transfer time as follows:

$$\underline{t}_{s_i^1}(n) = \frac{\sum_{k=1}^{i} x_{s_k^1}(n)}{c_l - (m_l^1 - i)b^{(1)}} \leq t_{s_i^1}(n). \tag{5.10}$$

The numerator is the amount of work to be done from the first to the i^{th} session. The denominator is the largest amount of bandwidth available for transferring this data. This is because during $t_{s_i^1}(n)$ there are at least $(m_l^1 - i)$ flows each with a bandwidth allocation of at least $b^{(1)}$.

Thus we have that

$$\begin{aligned} b^{(1)} &\leq a_{s_i^1}(n) = \frac{x_{s_i^1}(n)}{t_{s_i^1}(n)} \leq \frac{x_{s_i^1}(n)}{\underline{t}_{s_i^1}(n)} \\ &= \frac{x_{s_i^1}(n)}{\sum_{k=1}^{i} x_{s_k^1}(n)}[c_l - (m_l^1 - 1)b^{(1)}] \qquad (5.11) \\ &= \frac{a_{s_i^1}(n-1)}{\sum_{k=1}^{i} a_{s_k^1}(n-1)}[c_l - (m_l^1 - 1)b^{(1)}] \qquad (5.12) \\ &\leq \frac{a_{s_i^1}(n-1)}{(i-1)b^{(1)} + a_{s_i^1}(n-1)}[c_l - (m_l^1 - 1)b^{(1)}]. \qquad (5.13) \end{aligned}$$

We obtain Eq. (5.12) from Eq. (5.11) by using Eq. (5.6) and we have Eq. (5.13) from Eq. (5.12) by Eq. (5.9). Thus,

$$\begin{aligned} |a_{s_i^1}(n) - b^{(1)}| &\leq \left| \frac{a_{s_i^1}(n-1)}{(i-1)b^{(1)} + a_{s_i^1}(n-1)}[c_l - (m_l^1 - 1)b^{(1)}] - b^{(1)} \right| \qquad (5.14) \\ &= \left| \frac{b^{(1)}(i-1)(a_{s_i^1}(n-1) - b^{(1)})}{(i-1)b^{(1)} + a_{s_i^1}(n-1)} \right| \qquad (5.15) \\ &\leq \frac{i-1}{i}|a_{s_i^1}(n-1) - b^{(1)}|. \qquad (5.16) \end{aligned}$$

Eq. (5.16) is obtained from Eq. (5.15) by using $a_{s_i^1}(n-1) \geq b^{(1)}$.

Let $\xi_l = \frac{m_l^1 - 1}{m_l^1}$. Then, $\forall\, s \in \mathcal{S}_l$ we have that

$$|a_s(n) - b^{(1)}| \leq \xi_l |a_s(n-1) - b^{(1)}|. \qquad (5.17)$$

Since $0 < \xi_l < 1$, $a_s(n)$ converges geometrically to $b^{(1)} = a_s^*$.

Step 2: Suppose that for all $s \in \mathcal{V}^{(h-1)}$, $a_s(n)$ converges geometrically to a_s^*. Consider an h^{th} level link $l \in \mathcal{L}^{(h)}$ and a $s_i^h \in \mathcal{S}_l \cap \mathcal{S}^{(h)}$, which is the i^{th} session to leave the system among h^{th} level sessions in link l at iteration step n. We will show that $a_{s_i^h}(n)$, $i \in \{1, \ldots, m_l^h\}$ will converge to $b^{(h)}$. This can be done by finding a lower bound and an upper bound and showing the bounds converge to $b^{(h)}$.

Convergence of lower bound: Let $t_l^{\min}(n) = \min_{s \in \mathcal{S}_l \cap \mathcal{V}^{(h-1)}} [t_s(n)]$ and $t_l^{\max}(n) = \max_{s \in \mathcal{S}_l \cap \mathcal{V}^{(h-1)}} [t_s(n)]$. Thus, $t_l^{\min}(n)$ and $t_l^{\max}(n)$ are the earliest and the last departure time among sessions from levels 1 to $(h-1)$ at link l respectively. Now, define $\varepsilon_l^h(n)$ as the difference between these two times, i.e, $\varepsilon_l^h(n) = t_l^{\max}(n) - t_l^{\min}(n)$. Consider the time-varying bandwidth allocation for an h^{th} level session depicted in Figure 5.8.

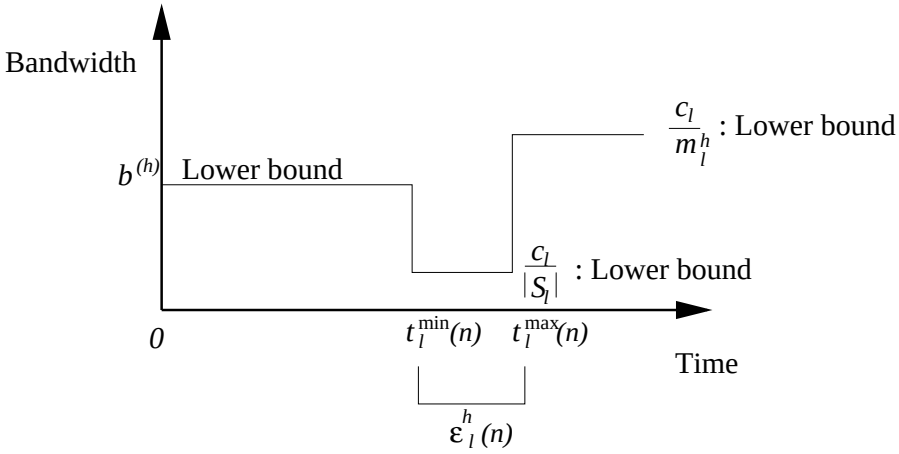


Figure 5.8: Bandwidth allocation for an h^{th} level session.

• From the beginning of the transfer to $t_l^{\min}(n)$, no session in $\mathcal{S}_l \cap \mathcal{V}^{(h-1)}$ leaves link l. Thus, at least $b^{(h)}$ is allocated to an h^{th} level session during this period.

• After $t_l^{\max}(n)$, there are only h^{th} level sessions remaining, which traverse link l. A lower bound on bandwidth allocation for such sessions is c_l/m_l^h since m_l^h is the entire number of h^{th} level sessions at the beginning. Note that the lower bound is larger than $b^{(h)}$ by Eq. (5.7).

• Note that it is possible for the h^{th} level session to be less than $b^{(h)}$ only during the period from $t_l^{\min}(n)$ to $t_l^{\max}(n)$. This may happen if sessions lower than h^{th} level change their bottleneck links to link l during this period. For example, consider Figure 5.5. Initially,

the bottleneck hierarchy is preserved before session s_1 leaves the system. However, once s_1 departs, session s_2 changes its bottleneck link from l_1 to l_2 and equally sharing the capacity of l_2 with s_3. This results in reduction of s_3's bandwidth from 2 to 1.5. During this period of length $\varepsilon^h(n)$, we can have a lower bound on bandwidth for an h^{th} level session by $\frac{c_l}{|\mathcal{S}_l|}$. Note that this lower bound is the first share of the link l, i.e., y_l^1.

Based on the above observations, we have the following lower bound for the average throughput of the any h^{th} level session:

$$\begin{aligned} \underline{a}_l^h(n) &= \frac{b^{(h)} t_l^{\min}(n) + \frac{c_l}{|\mathcal{S}_l|}\varepsilon_l^h(n)}{t_l^{\max}(n)} \\ &= b^{(h)} - \frac{(b^{(h)} - \frac{c_l}{|\mathcal{S}_l|})\varepsilon_l^h(n)}{t_l^{\max}(n)} \equiv b^{(h)} - \delta_l^h(n) \\ &\leq a_s(n),\ s \in \mathcal{S}_l \cap \mathcal{S}^{(h)}. \end{aligned} \tag{5.18}$$

Note that the above lower bound is the average throughput at time $t_l^{\max}$ in Figure 5.8, and it is a worst case scenario including a maximal amount time $\varepsilon_l^h(n)$ at the lower rate $\frac{c_l}{|\mathcal{S}_l|}$.

By our induction hypothesis, for $s \in \mathcal{V}^{(h-1)}$, $a_s(n)$ converges geometrically to a_s^*, so $\varepsilon_l^h(n)$ also converges to 0. Also note that as $\varepsilon_l^h(n)$ goes to 0, $\delta_l^h(n)$ also goes to 0 and eventually the lower bound $\underline{a}_s^h(n)$ converges geometrically to $b^{(h)}$.

Convergence of upper bound: Next, consider the following lower bound for the finishing time of session s_i^h:

$$\underline{t}_{s_i^h}(n) = \frac{\sum_{j=1}^{h} \sum_{k=1}^{p^j(s_i^h)} x_{s_k^j}(n)}{c_l - \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\underline{a}_l^j(n-1)} \leq t_{s_i^h}(n). \tag{5.19}$$

As with Eq. (5.10) in Step 1, the numerator is the amount of work to be done prior to s_i^h's departure and the denominator is an upper bound on the available bandwidth.

In a similar manner to Step 1, we have that

$$\begin{aligned}\underline{a}_l^h(n) &\leq a_{s_i^h}(n) \\ &\leq \frac{a_{s_i^h}(n-1)[c_l - \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\underline{a}_l^j(n-1)]}{\sum_{j=1}^{h-1} p^j(s_i^h)\underline{a}_l^j(n-1) + (p^h(s_i^h)-1)\underline{a}_l^h(n-1) + a_{s_i^h}(n-1)}. \qquad (5.20)\end{aligned}$$

By replacing Eq. (5.18) into Eq. (5.20) and using the fact that $c_l = \sum_{j=1}^{h} m_l^j b^{(j)}$, we have that

$$a_{s_i^h}(n) \leq \frac{a_{s_i^h}(n-1)[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} + P(n-1)]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} - Q(n-1) + a_{s_i^h}(n-1)},$$

where $P(n) = \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\delta_l^j(n)$ and $Q(n) = \sum_{j=1}^{h-1} p^j(s_i^h)\delta_l^j(n) - \delta_l^h(n)$.

Now letting $R(n) = \max[P(n), Q(n)]$, we have an upper bound for an average throughput:

$$\begin{aligned}a_{s_i^h}(n) &\leq \frac{a_{s_i^h}(n-1)[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - R(n-1)]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + R(n-1) + a_{s_i^h}(n-1)} \\ &\equiv \bar{a}_{s_i^h}(n) \equiv T(R(n-1), a_{s_i^h}(n-1)) \qquad (5.21)\end{aligned}$$

Now we will show that $\bar{a}_{s_i^h}(n)$ converges to $b^{(h)}$ geometrically. Consider $T(\cdot, a_{s_i^h})$. We have that

$$\begin{aligned}\max_{R\geq 0}\left|\frac{\partial}{\partial R}T(R, a_{s_i^h})\right| &= \left|\frac{-2a_{s_i^h}\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} + a_{s_i^h}b^{(h)} - (a_{s_i^h})^2}{[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + R(n-1) + a_{s_i^h}(n-1)]^2}\right|_{R=0} \\ &\equiv A(a_{s_i^h}).\end{aligned}$$

For any $\varepsilon > 0$, by our lower bound, there is an n such that $a_{s_i^h}(n) \geq \underline{a}_{s_i^h}(n) \geq b^{(h)} - \varepsilon$. Thus, letting the Lipschitz constant $\bar{K} = A(b^{(h)} - \varepsilon)$, we have that

$$|T(R(n), a_{s_i^h}(n)) - T(0, a_{s_i^h}(n))| \leq \bar{K}|R(n) - 0| \qquad (5.22)$$

Note that $R(n)$ is a linear combination of $\delta_l^j(n)$, so $R(n)$ will converge geometrically to 0.

Furthermore, $T(R,\cdot)$ is a pseudo-contraction [73], which converges to $T(0,b^{(h)}) = b^{(h)}$, i.e.,

$$\begin{aligned} |T(0,a_{s_i^h}(n)) - T(0,b^{(h)})| &\leq \left| \frac{a_{s_i^h}(n)[\sum_{j=1}^{h} p^j(s_i^h)b^{(h)}]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + a_{s_i^h}(n)} - b^{(h)} \right| \\ &\leq \left| \frac{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)}}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + a_{s_i^h}(n)} \right| |a_{s_i^h}(n) - b^{(h)}| \\ &\leq \xi |a_{s_i^h}(n) - b^{(h)}|, \ 0 < \xi < 1 \end{aligned} \tag{5.23}$$

So,

$$|a_{s_i^h}(n+1) - b^{(h)}| = |T(R(n), a_{s_i^h}(n)) - T(0,b^{(h)})| \tag{5.24}$$

$$\leq |T(R(n), a_{s_i^h}(n)) - T(0, a_{s_i^h}(n))| + |T(0, a_{s_i^h}(n)) - T(0,b^{(h)})|$$

$$\leq \bar{K}|R(n) - 0| + \xi |a_{s_i^h}(n) - b^{(h)}|, \ 0 < \xi < 1. \tag{5.25}$$

Thus, $a_{s_i^h}(n)$ converges geometrically to $b^{(h)}$. This completes the proof.

5.4 Adaptive FastReplica and its Implementation

In this section, we discuss our implementation of Adaptive FastReplica (AFR). In our implementation, we used the point-to-point sessions, i.e., TCP connections from source to relays and relays to receivers. One of the main reasons for this choice was ease of implementation. Realization of tree sessions would require tight coupling among m point-to-point sessions. This requires modifications of the TCP layer, which would slow down deployment. By adopting the point-to-point session model our solution could be implemented entirely at the application layer. In addition to ease of implementation, point-to-point sessions may result in a higher average throughput since they decouple the rate of transmission over the different branches of the distribution tree [74], particularly when there are fast

changes in the interfering traffic. Finally we note that the solutions we proposed in Section 5.3 (the use of full overlay trees and throughput-based approach) can be applied as is under point-to-point sessions.

5.4.1 Block-level adaptation

In our analysis in Section 5.3, we assumed that the available network capacity for file transfers was fixed. However, this needs not be the case in practice, i.e., the available bandwidth may change dynamically because of other traffic sharing the network. Even during a single file transfer, the available bandwidths for overlay paths may significantly change due to highly variable Internet traffic.

To be more adaptive to such variations, we propose to divide large files into multiple equal-sized *blocks*, and use throughput-based adaptation of the partition ratio vector on a per block basis. That is, within a single file transfer, the source obtains feedback messages containing routes' average throughput information from past block transfers, and uses these in computing the partitions for subsequent blocks. Note that with this method, the argument n in $x_i(n), a_i(n)$ in Section 5.3.3, is interpreted as referring to the n^{th} block of a given file.

In addition to being adaptive to dynamic environments, there are several practical advantages to the above block-level adaptation. First, even with the assumption of static network capacity, the approach can expedite file transfer. This is because instead of sticking with 'bad' partition ratio vector, it can quickly learn network capacity by way of frequent feedback messages, and use a 'good' partition ratio vector for the file transfer. Second, to add resilience to node failures, heartbeat messages from each receiver to its source need to be introduced, see [65]. However, there is no need for heartbeat messages in our solution since feedback messages play the role of indicating each node's liveness.

1: INPUT: file f, list of receivers, block size B
2: send the list of receivers to R
3: initialize $x_i(1) = \frac{1}{m},\ i \in N$
4: **for** $n = 1$ to $\lceil \frac{f}{B} \rceil$ **do**
5: **if** there is a new latest throughput information ($a_i(j),\ i \in N,\ j < n$) **then**
6: $x_i^{new}(n) \leftarrow \frac{a_i(j)}{\sum_{k=1}^{m} a_k(j)},\ i \in N$
7: $x_i(n) = (1-\alpha)x_i(n-1) + \alpha\, x_i^{new}(n)$, $0 \leq \alpha \leq 1$
8: **else**
9: $x_i(n) = x_i(n-1)$
10: **end if**
11: assign i^{th} chunk size based on $x_i(n)$ and concurrently send it to node n_i, $i \in N$ with size and block index information.
12: **end for**

Figure 5.9: AFR algorithm for a source.

Our solution does not incur a significant amount of control overhead: the total number of feedback message packets generated in our solution for a single file transfer is $m^2\lceil \frac{f}{B} \rceil$ where B denotes the block size (each receiver generates $m\lceil \frac{f}{B} \rceil$ feedback messages). This overhead is further alleviated with TCP's piggyback mechanism [75], where a receiver does not have to send a separate TCP acknowledgment.

5.4.2 Implementation

Figure 5.9 and 5.10 exhibit pseudo-codes describing the behavior of the source and receiver respectively.

We made each receiver ready for a file transfer by having them listen to a specific port number, and let the source initiate file transfer.[8] Once the list of receivers is sent to each receiver via m concurrent TCP connections from the source node, each receiver establishes data connections to the other receivers. We implemented concurrent file transfers using a standard *pthread* multi-thread library [76].

[8]The mechanism for selecting/soliciting receiver nodes, is outside the scope of the chapter.

```
1: receive the list of receivers
2: while file f not completely downloaded do
3:    concurrently keep reading data from n_0 and other receivers
4:    if data is coming from n_0 then
5:       forward them to other receivers
6:    end if
7:    if each chunk completely downloaded then
8:       send feedback message to n_0 containing its average throughput value
9:    end if
10: end while
```

Figure 5.10: AFR algorithm for receivers.

The original file is sent block by block. Each block is partitioned into m chunks depending on the current partition ratio vector. Since the ratios are not integer, the floor operation is used to assign the appropriate amounts of bytes to each chunk, that is, $f_i(n) = \lfloor Bx_i(n) \rfloor$ where B is the block size.[9] Then, the remaining bytes were distributed among chunks with the higher partition ratios in a round-robin manner. In order for receivers to know how much data they need to read, chunk size and block index information are sent prior to the transferred data.

In our implementation, the source need not wait for the arrival of all feedback information from previous blocks prior to sending the next block. To expedite the file transfer, the source keeps pushing blocks, and applies the updated information as it becomes available. In order to handle highly variable Internet traffic, a partition ratio vector is generated by having a weighted average between the previous partition ratio vector $x_i(n-1)$, and new estimate for the partition ratio vector $x_i^{new}(n)$ as shown on Line 7 in Figure 5.9. The impact of varying α and block size will be discussed in Section 5.5.5.

[9] Note that even though block sizes are all equivalent, the last block size may be different.

5.5 Performance Evaluation

We evaluated the proposed scheme on the Internet by conducting experiments with our implementation over the Planetlab testbed [77]. Planetlab is a distributed overlay network providing diverse hosts for experimentation. We explored a number of configurations by varying the source, receiver nodes' locations, and the number of participants. In this section, we present representative results for the 9 hosts in Table 5.1. Table 5.2 shows five source & receiver set configurations that will be discussed below.

We measured two performance metrics: (1) overall replication time, $\max_{i \in N} d_i$, which is our primary objective, and (2) the average replication time, $\frac{1}{m}\sum_{k=1}^{m} d_k$.[10] Ideally, the transfer time at each receiver should be measured from the beginning of the transfer at the source node to the completion of the file download at the receiver node. However, due to clock synchronization problems at different nodes, we measure the file transfer time from the beginning of the transfer at the source node until a feedback message notifying file completion from the receiver. Since we are interested in large file transfers, the addition of a one-way latency of the packet from the receiver to the source will not impact the accuracy of the results. Unless explicitly mentioned, all results are averaged over 10 different runs and the vertical lines at data points represent approximate 90% confidence intervals assuming independent identically distributed samples.

For comparison, we examined the following alternative distribution mechanisms:

Sequential Unicast (SU) This scheme measures the file transfer time from the source to each receiver *independently* via unicast (i.e., in the absence of other receivers), and then takes the worst case transfer time over all receivers assuming such times could be realized

[10] The measured times reported in this chapter do not include any overheads that might be incurred due to the reintegration of chopped segments.

n_0	utexas.edu
n_1	columbia.edu
n_2	gatech.edu
n_3	umich.edu
n_4	ucla.edu
n_5	cam.ac.uk
n_6	caltech.edu
n_7	upenn.edu
n_8	utah.edu

Table 5.1: Participating nodes.

	source	receiver set
CONFIG 1	n_0	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8$
CONFIG 2	n_7	$n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_8$
CONFIG 3	n_0	n_1, n_2, n_3, n_4
CONFIG 4	n_7	n_0, n_5, n_6, n_8
CONFIG 5	n_0	n_1, n_7

Table 5.2: Configurations.

in parallel. Note that Sequential Unicast is a hypothetical "optimistic" construction used for comparison purposes. Our measurements for Sequential Unicast emulate the performance of file distribution using IP multicast.

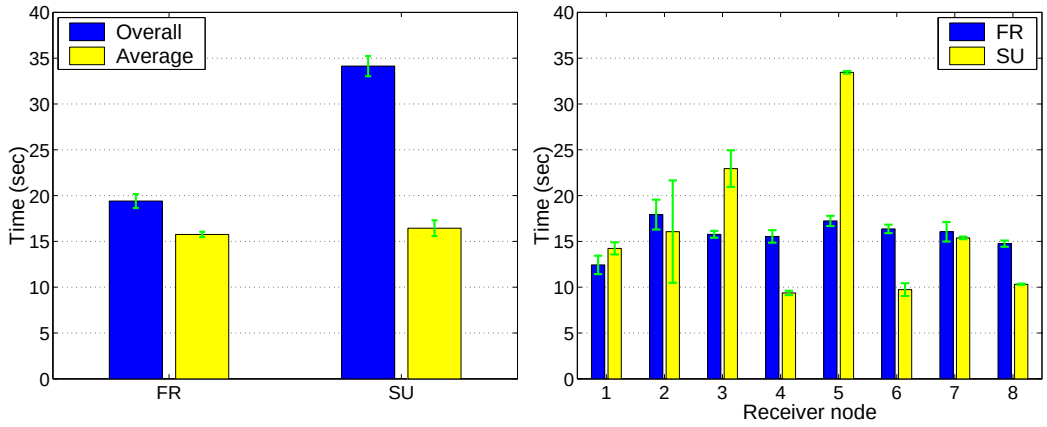
FastReplica (FR) This scheme is a refinement of that proposed in [65]. It is implemented with cut-through functionality at relay nodes.

We define the *speedup* for the overall replication time of scheme Y over scheme Z in percentage as $100(r^Z/r^Y - 1)\%$. Thus, for example, 30% speedup of scheme Y over scheme Z means that scheme Y is 1.3 times faster than scheme Z.

5.5.1 FR versus SU - Does partitioning and path diversity help?

Figure 5.11(a) shows the performance results for the SU and FR schemes when an 8MB file is transferred in CONFIG 1 shown in Table 5.2. We observe that there is a significant

speedup for the overall replication time (75%) and a marginal speedup for the average replication time (4%) of FR over SU. Figure 5.11(b) depicts the download times for individual receivers. We observe that the limited bandwidth from n_0 to n_5 significantly impacts the performance of SU. However, FR can overcome this limitation by realizing additional concurrent connections with path diversity. Also note that FR achieves much smaller variance in download times among receivers - a desirable characteristic if fairness among receivers is an issue.



(a) Overall & average time. (b) Download times.

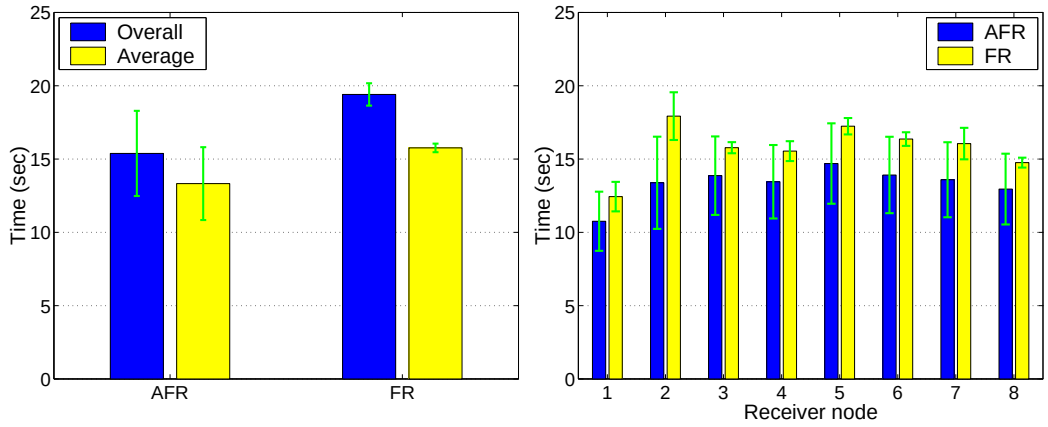
Figure 5.11: FR vs. SU in CONFIG 1.

5.5.2 AFR versus FR - Does adaptivity help?

Figure 5.12(a) shows the performance gain of AFR over FR for an 8MB file transfer in CONFIG 1. In the experiments discussed in this subsection, AFR had a block size of 512KB and α = 0.1.(We will discuss the impact of varying block size and α later.) As can be seen, there is a 26% and 18% speedup for AFR over FR for overall and average replication time respectively, corresponding to a 121% overall and 23% average time speedup

of AFR over SU. Figure 5.12(b) depicts download times at receiver nodes. Observe that AFR achieves its speedup gains over FR by reducing download times for all receiver nodes. As expected, AFR's performance gain come from exploiting heterogeneity in network resources and dynamic traffic conditions, i.e., putting more on "good" paths and less on "bad" paths.

Figure 5.13 shows a particular realization of the partition ratio vector for AFR as a function of the block index. The partition ratio vector stays fixed at 0.125 at the beginning of file transfer until feedback messages containing average throughput information return. Subsequently, partition ratios keep evolving to better exploit the heterogeneity of overlay paths by load balancing. Note that the partition ratios for FR would be fixed at 0.125.



(a) Overall & average time. (b) Download times.

Figure 5.12: AFR (B = 512KB, α = 0.1) vs. FR in CONFIG 1.

5.5.3 Diverse configuration results

After verifying that both AFR and FR do better than SU, we consider another scheme, *m-Sequential Unicast(mSU)* for comparison. As with SU, this scheme measures file transfer

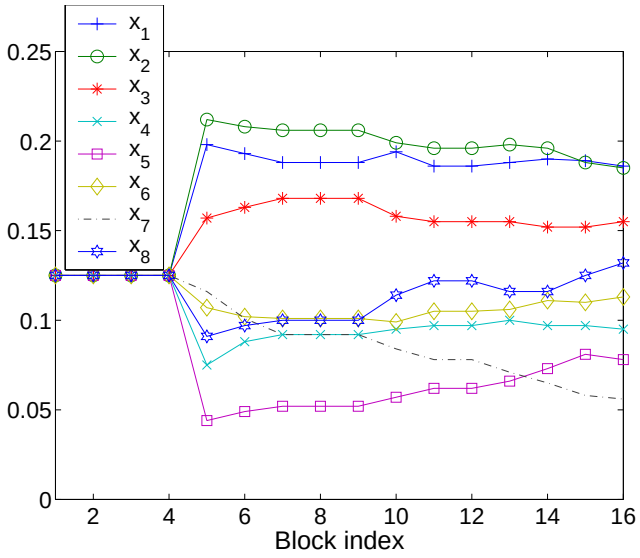


Figure 5.13: Partition ratio vector for AFR (B = 512KB, $\alpha = 0.1$) in CONFIG 1.

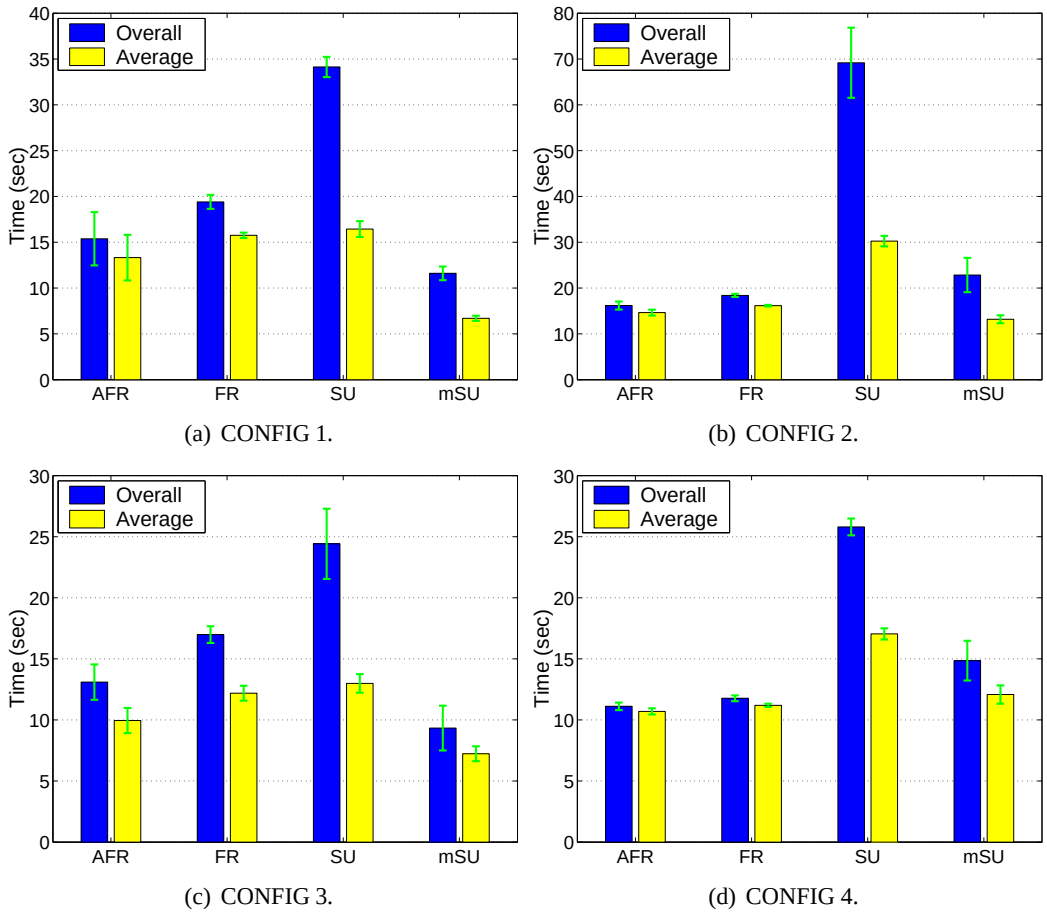
	FR (overall)	SU (overall)	FR (average)	SU (average)
CONFIG 1	26 %	121 %	18 %	23 %
CONFIG 2	13 %	327 %	10 %	106 %
CONFIG 3	29 %	86 %	22 %	30 %
CONFIG 4	5 %	132 %	4 %	59 %

Table 5.3: Speedup gains for AFR over FR and SU.

time from the source to each receiver *independently*. However, *m* (the number of receivers) concurrent unicast connections are used to realize the file transfer. That is, the file is equally partitioned into *m* chunks and those chunks are delivered via *m* parallel connections from the source to the receiver. This scheme will benefit from an acceleration in transfers due to parallel connections, but not the diversity in paths provided by AFR/FR.

Figure 5.14(a)-(d) shows performance results for AFR, FR, SU and mSU with CONFIG 1, 2, 3 and 4 respectively under an 8MB file transfer.

First, let us consider comparison results among AFR, FR and SU. Table 5.3 summarize the speedup gains for AFR over FR and SU. With different configurations we obtain



(a) CONFIG 1. (b) CONFIG 2.

(c) CONFIG 3. (d) CONFIG 4.

Figure 5.14: AFR (B = 512KB, $\alpha = 0.1$) vs. FR vs. SU.

different amounts of performance gains, but the results consistently show that (1) AFR outperforms FR and SU in both overall and average replication time, and (2) as expected AFR achieves more speedup in overall replication time than average replication time. We also verified this result with a large number of configurations by varying source or receiver nodes' locations with different number of participants.

Next, note that mSU always beats SU, which verifies that multiple concurrent file transfers will achieve better throughput than that of a single one. For CONFIG 1 and 3, mSU

is better than AFR and FR, but a bit surprisingly, for CONFIG 2 and 4, the performance of mSU is worse. Furthermore note that for cases where the source is at n_7, e.g., CONFIG 2 and 4, we observed that AFR and FR performs significantly better than SU, and there is not much gain for AFR over FR. The main reason behind these results is that node n_7's transmitting rate is quite limited, implying there is a bottleneck at the source. This can also be verified in Figure 5.13, where x_7 is the smallest over time. From the perspective of the source, the amount of data to be transferred at its outgoing link is the same for all schemes. This indicates that not only having parallel concurrent connections but also having path diversity like FR or AFR schemes can greatly help in cases where the bandwidth at the source is limited. This result is consistent with those presented in [65] where the source is located at hp.com, which is bandwidth-limited compared to other receivers connected to Internet2. Since the bottleneck link is at the source, it is likely that the same bottleneck link is equally shared by all chunks. This will result in a roughly equal partition ratio for all chunks, so, the performance of AFR becomes similar to that of FR.

5.5.4 AFR - A case for robustness in a dynamic environment

For these experiments, we observed fairly tight confidence intervals on our measurements indicating the network loads were fairly stable. This is probably because most hosts on the Planetlab testbed are connected on Internet2 [78]. To study how more dynamic traffic patterns impact the performance of AFR, we devised the following experiment based on CONFIG 5 in Table 5.2. The source n_0 transfers an 8MB file to receivers, n_1 and n_7. After 5 seconds from initiating the transfer, we generate an interfering traffic by having host n_7 transmit a 32MB file to all the hosts in Table 5.1 except n_0 and n_1. Figure 5.15 shows the performance results for AFR (B=128KB, $\alpha = 0.1$) and FR with/without traffic interference.

Without traffic interference, there was a 5% speedup for AFR over FR in overall replication time. However, the speedup gain becomes significant (28%) with traffic interference. We observe that AFR puts a smaller portion of its traffic to n_7 due to its limited sending ability after 5 seconds. This result clearly indicates that AFR adaptation can enhance performance in a truly dynamic network environment, as might be expected on the Internet versus our testbed Internet2.

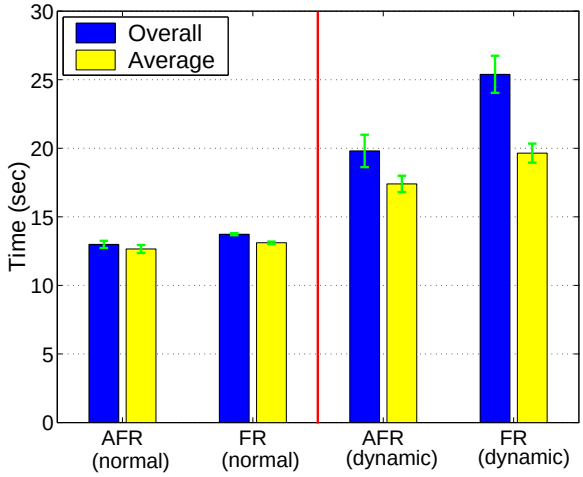


Figure 5.15: AFR (B = 128KB, $\alpha = 0.1$) vs. FR with a dynamic network environment in CONFIG 5.

5.5.5 Tuning AFR's parameters

There are two parameters to be selected in AFR: block size and α. Both parameters play a role in determining how quickly partition is adapted. With large blocks, we lose opportunities to respond to dynamic changes in network bandwidth. (At the extreme end where making block size equal to the entire file size, no adaptation is performed.) On the other hand, there will be overheads incurred in processing feedback messages and reintegration when blocks are small. Likewise, with small α we are too conservative to quickly follow

changes in network state, and vice versa for large α value. To see how different α values impact the performance, we conducted the following experiment: in CONFIG 5, the source starts a transfer of 32MB file using AFR (B=128KB), and 10 seconds later n_7 transfers 1MB to $\{n_2, n_3, n_4, n_5, n_6, n_8\}$. Figure 5.16 shows the partition ratio vector values with α = 0.1 and 0.5. As expected, larger α values (0.5) quickly track changes in network state but experience higher variability due to measurement noise.

Our experiments varying block size and alpha under the configurations in Table 5.2 shows that the performance difference is not noticeable except when block size is too small (e.g., making each chunk lower than 32KB) and $\alpha = 0$, i.e., no adaptation. This again indicates that the network is fairly stable.

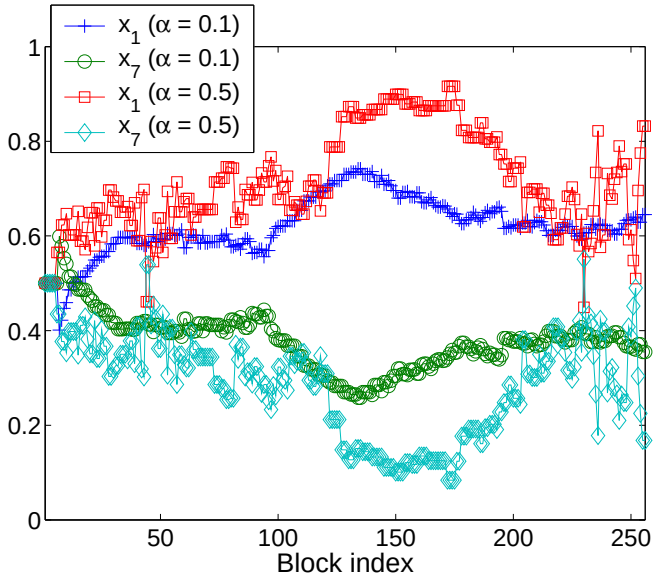


Figure 5.16: Partition ratio vectors for AFR (B = 128KB) with a dynamic network environment in CONFIG 5.

5.6 Additional Related Work

Expediting file transfers using concurrent connections is not a new idea. One area of active research in this direction has been exploiting the benefits of path diversity between clients and servers with replicated content in CDNs. That is, since there are multiple servers storing the same content deployed at geographically distributed locations in CDNs, a client can access multiple servers in parallel to reduce the download time, or achieve fault tolerance. The work in [79] reduces the download times by having a client receive a Tornado encoded file from multiple mirror servers. The dynamic parallel access scheme in [80] also demonstrates the improved download time observed by clients by requesting different parts of a file from different servers. Note that this research work is of the many-to-one communication type, and it is assumed that the original content is already replicated across the edge (mirror) servers. By contrast, the problem in our chapter is about content replication across edge servers within content delivery networks.

As mentioned earlier, many existing end-system approaches try to optimize overlay tree structure based on target application requirements. For example, Yoid [60], Narda [2] and Scattercast [61] use delay as the routing metric. Overcast [3] constructs an overlay tree optimized for available bandwidth from the source to the receivers. End system multicast [62] uses a combination of delay and available bandwidth when selecting a routing path. Available bandwidth from the source to the receivers is the key quantity for our approach since our objective is to minimize the overall replication time. However, unlike [3], [62], we do not explicitly measure the available bandwidth on an end-to-end path, but rely on feedback messages containing throughput information from receivers. Furthermore, unlike all the above end-system approaches, our approach uses a mesh-structure versus a tree structure.

Recent work in [81] proposes the use of additional cross-connections between peers to exchange complementary content that nodes have already received. This approach provides peer-to-peer type content delivery service supporting asynchrony, i.e., receivers can leave or join at arbitrary times. Note that our approach is related to the infrastructure based content distribution network (CDN) (e.g., Akamai [82]). In a CDN setting, we assume that the set of receivers is known priori to file transfer.

5.7 Conclusion

In this chapter, we propose the Adaptive FastReplica mechanism for a fast delivery of large files to distributed nodes. Three key ideas were added to the original FastReplica concept [65]. First, chunk sizes are no longer necessarily equal. The source may assign a larger portion of the original file to paths that are efficient and smaller ones to paths that are not. Second, in order to efficiently obtain path quality information, we propose an iterative throughput-based approach where each receiver sends a feedback message containing the perceived average throughput information for each chunk. This average throughput information is used to generate new partition ratios for subsequent transfers. Third, a large file is partitioned into multiple equal-size blocks, and each block is again split into m chunks according to our partitioning algorithm. This allows the source to dynamically generate partition ratios based on feedback messages from receivers, and effectively adapt to changing Internet traffic loads during a single large file transfer.

Unlike most existing end-system approaches, ours does not try to construct optimized overlay structure. Instead, it uses a full fixed set of heterogeneous overlay paths in an 'optimized' manner, i.e., application-level load balancing. It exploits 'good' paths by putting more data on them. This can significantly reduce overhead otherwise incurred from

active probing, maintaining and reconfiguring overlay structure.

Although the performance gains for our approach over FastReplica and Sequential Unicast schemes depend on a case by case basis, the experiment results suggest that our approach consistently outperforms other schemes in minimizing both the overall and average replication time. Furthermore, this speedup gain becomes significant when the network is highly dynamic, as might be the case on more congested networks than those on which we conducted our experiments.

When there is a large number of receivers, e.g., hundreds/thousands of nodes, the basic FastReplica framework can be iteratively applied using a hierarchical *k*-ary tree structure where *k* is small enough to support efficient file replication. One interesting and important issue is "how one might build such a structure", i.e., efficient clustering techniques. Recent research [83] shows that large-scale Internet applications could benefit from incorporating IP-level topological information in the construction of the overlay to significantly improve overlay performance. In [83], a new distributed technique is introduced where the nodes partition themselves into bins in such a way that nodes within a given bin are relatively close to one another in terms of network latency. This may be an interesting way for clustering "close" nodes into replication groups in FastReplica framework.

Chapter 6

Conclusion

In this dissertation we address four problems: (1) IP multicast topology discovery, (2) DDoS attack prevention, (3) efficient subgroup communication, and (4) fast content delivery. Each problem has its own value and significance in today's Internet. Our contribution is that for each problem, we have presented practical and simple solutions based on a suite of novel ideas. The following is the summary of key ideas and mechanisms used to solve these four problems.

- IP multicast topology discovery
  - Fan-out decrement mechanism
  - Topology discovery based on path/fan-out distance information
- DDoS attack prevention
  - Filtering and tracing mechanisms based on IP multicast
- Efficient subgroup communication
  - Group communication based on combining unicast and scoped multicast

  - Resource and topology discovery

- Fast content delivery

  - Use of a fixed set of overlay paths to exploit resource diversity

  - Application-level load adaptation

Although at first glance the proposed solutions might seem quite different, we were able to make the following observations.

First, a common property of the problems we studied and of today's Internet is that they are dynamic and unpredictable in the sense that (1) members (or attack nodes) can join and leave at any time, and (2) traffic is highly variable. To cope with this dynamicity, which is seemingly an immutable characteristic in computer networks, we believe that *adaptivity* and *robustness* are key elements, which are incorporated in all of the solutions we proposed.

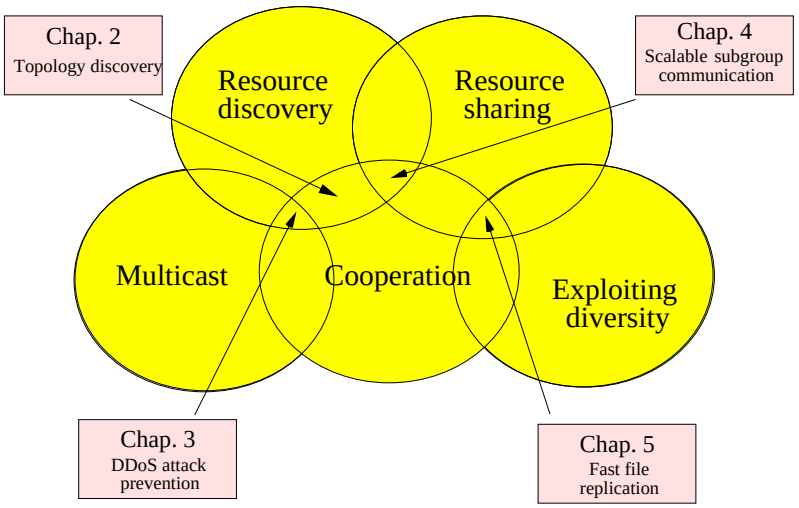


Figure 6.1: Relationships between problems and characteristics used to solve them in the dissertation.

Second, a common thread in our work is the exploitation of inherent characteristics of group communications. Figure 6.1 summarizes the relationships between the problems

we addressed in this dissertation. The items in squares are the problems, while those in circles are characteristics used to solve them. In all the solutions, *cooperation* among members is the key element. Topology or resource information within neighborhoods becomes a foundation for sharing resources or further cooperation. Thus, our results on topology discovery in Chapter 2 can be used to solve the problems in DDoS attack prevention in Chapter 3 and efficient subgroup communication in Chapter 4, wherein *resource discovery and sharing* are key components. The approach used for DDoS prevention in Chapter 3 is a good example illustrating the powerful use of group communications. We note that, to our knowledge, this is the first attempt to apply IP multicast to defeat DDoS attacks, moving away from its basic use as a data distribution medium. The key idea used for fast content delivery in Chapter 5 is to simplify the exploitation of path diversity. Recall that this diversity can exist only when there are multiple nodes.

In this dissertation, we explore how *diversity* in group communications can be turned into an advantage in achieving various objectives. Throughout our studies, we learned that cooperation, discovery and sharing network resources can be used to attack various problems associated with group communications.

Bibliography

[1] S. E. Deering, *Multicast Routing in a Datagram Internetwork*, Ph.D. thesis, Stanford University, 1991.

[2] Y-H Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *ACM SIGMETRICS*, 2000.

[3] J. Jannotti, D. Gifford, K. Johnson, F. Kasshoek, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *USENIX OSDI*, 2000.

[4] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: an application level multicast infrastructure," in *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[5] B. N. Levine, S. Paul, and J. J. Garcia-Luna-Aceves, "Organizing multicast receivers deterministically according to packet-loss correlation," in *Proceedings of ACM Multimedia'98*, 1998.

[6] D. Li and D. R. Cheriton, "OTERS(On-Tree Efficient Recovery using Subcasting): A reliable multicast protocol," in *Proceedings of IEEE ICNP'98*, 1998.

[7] J.C. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," in *Proceedings IEEE Infocom*, 1996.

[8] I. Kouvelas, V. Hardman, and J. Crowcroft, "Network adaptive continuous-media applications through self-organised transcoding," in *Proceedings of the Network and Operating Systems Support for Digital Audio and Video*, 1998.

[9] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast applications," in *Proceedings IEEE Infocom*, 1998.

[10] S. Ratnasamy and S. McCanne, "Scaling end-to-end multicast transports with a topologically-sensitive group formation protocol," in *Proceedings IEEE International Conference of Network Protocols, ICNP'99*, 1999.

[11] W. Fenner and S. Casner, "A traceroute facility for IP Multicast," IETF draft, draft-ietf-idmr-traceroute-ipm-06.txt, March, 2000.

[12] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *Proceedings IEEE Infocom*, 1999.

[13] X. Yang and L. W. Lehman, "Inferring characteristics of multicast trees," Dec 1998. MIT 6.896 Topics in Computer Networks term project paper.

[14] R. Caceres, N.G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley, "Loss-based inference of multicast network topology," in *Proceedings 1999 IEEE Conference on Decision and Control*, 1999.

[15] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proceedings IEEE Infocom*, 2000.

[16] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of Internet instrumentation," in *Proceedings IEEE Infocom*, 2000.

[17] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering Internet topology," `www.cs.cornell.edu/skeshav/papers/discovery.pdf`.

[18] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology discovery in heterogeneous IP networks," in *Proceedings IEEE Infocom*, 2000.

[19] M. Stemm, R. Katz, and S. Seshan, "A network measurement architecture for adaptive applications," in *Proceedings IEEE Infocom*, 2000.

[20] T. Ballardie, P. Francis, and J. Crowcroft, "Core based trees(CBT): An architecture for scalable inter-domain multicast routing," in *Proc. ACM SIGCOMM*, 1993.

[21] A. Ballardie, "Core Based Trees(CBT) Multicast Routing Architecture," RFC 2201, Sep., 1997.

[22] B. Cain, S. Deering, and A. Thyagarajan, "Internet group management protocol, version 3," IETF draft, draft-ietf-idmr-igmp-v3-04.txt, June, 1997.

[23] S. Keshav, *An Engineering approach to computer networking:ATM Networks, the Internet, and the Telephone Network*, Addison-Wesley, Inc., 1997.

[24] D. Thaler, D. Estrin, and D. Meyer, "Border gateway multicast protocol (BGMP):Protocol specification," IETF draft, draft-ietf-bgmp-spec-01.txt, Mar. 2000.

[25] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification," RFC 2362, Jun., 1998.

[26] R. Caceres, N.G. Duffield, J. Horowitz, D. Towsley, and Tian Bu, "Multicast-based inference of network-internal characteristics: Accuracy of packet loss estimation," in *Proceedings IEEE Infocom*, 1999.

[27] M. Pullen, M. Myjak, and C. Bouwens, "Limitations of Internet protocol suite for distributed simulation in the large multicast environment," Tech. Rep., RFC 2502, 1999.

[28] Computer Emergency Response Team, "CERT advisory ca-2000-01 denial-of-service developments," `http://www.cert.org/advisories/CA-2000-01.html`.

[29] P. Ferguson and D. Senie, "Network ingress filtering: Defeating Denial of Service attacks which employ IP source address spoofing," RFC 2267, Jan. 1998.

[30] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets," in *Proc. ACM SIGCOMM*, 2001.

[31] CISCO Corporation, "Characterizing and tracing packet floods using cisco routers," `www.cisco.com/warp/public/707/22.pdf`.

[32] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in *Proc. ACM SIGCOMM*, 2000.

[33] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for IP traceback," in *Proc. IEEE Infocom*, 2001.

[34] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash-based IP traceback," in *Proc. ACM SIGCOMM*, 2001.

[35] "Skitter," http://www.caida.org/tools/measurement/skitter/.

[36] "Internet mapping," http://cm.bell-labs.com/who/ches/map/dbs/index.html/, 1999.

[37] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," in *IEEE/ACM Transactions on Networking*, Oct. 2000, vol. 8.

[38] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an Internet," in *Proc. IEEE Infocom*, 1996.

[39] B. Li, M. J. Golin, G. F. Ialiano, and X. Deng, "On the optimal placement of web proxies in the Internet," in *Proc. IEEE Infocom*, 1999.

[40] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," in *http://www.aciri.org/pushback/pushback-toCCR.ps, submitted to CCR*, July 2001.

[41] G. Sager, "Security fun with OCxmon and cflowd," in *Internet 2 Working Group Meeting*, Nov. 1998, http://www.caida.org/projects/NGI/content/security/1198.

[42] William Stallings, *SNMP, SNMP v2, SNMP v3 and RMON 1 and 2(Third Edition)*, Addison-Wesley, Inc., 1999.

[43] "IETF secure multicast group," http://www.securemulticast.org.

[44] Xianjun Geng and Andrew B. Whinston, "Defeating distributed denaial of service attacks," in *IT Pro*, July 2000.

[45] G. Banga, P. Druschel, and J. Mogul, "Resource containers:A new facility for resource management in server systems," in *Proc. of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, Feb. 1999.

[46] O. Spatscheck and L. Peterson, "Defending against denial of service attacks in Scout," in *Proc. of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, Feb. 1999.

[47] H. Wang, D. Zhang, and K. G. Shin, "Detecting SYN flooding attacks," in *Proc. IEEE Infocom*, 2002.

[48] S. M. Bellovin, "ICMP traceback messages.," IETF draft, draft-bellovin-itrace-05.txt, Mar. 2000.

[49] S. F. Wu, L. Zhang, D. Massey, and A. Mankin, "Intention-driven ICMP trace-back," IETF draft, draft-wu-itrace-00.txt, Feb. 2001.

[50] K. Park and H. Lee, "On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack," in *Proc. IEEE Infocom*, 2001.

[51] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for intrusion detection and response," in *Proc. First DARPA Information Survivability Conference and Exposition*, 2000.

[52] T. Wong, R. Katz, and S. McCanne, "An evaluation of preference clustering in large-scale multicast applications," in *Proc. IEEE Infocom*, 2000.

[53] D. Thaler and M. Handley, "On the aggregatability of multicast forwarding state," in *Proc. IEEE Infocom*, 2000.

[54] T. Wong and R. Katz, "An analysis of multicast forwarding state scalability," in *International Conference on Network Protocols(ICNP)*, 2000.

[55] E. Lety and T. Turletti, "Issues in designing a communication architecture for large-scale virtual environments.," in *Proc. of Networked Group Communication Workshop*, 1999.

[56] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu, "Discrete mobile centers," in *17th Symposium on Computational Geometry(SoCG)*, 2001.

[57] R. C. Chalmers and K. C. Almeroth, "Modeling the branching characteristics and efficiency gains in global multicast trees," in *Proc. IEEE Infocom*, 2001.

[58] J. Tian and G. Neufeld, "Forwarding state reduction for sparse mode multicast communication," in *Proc. IEEE Infocom*, 1998.

[59] I. Stoica, T. E. Ng, and H. Zhang, "REUNITE: A recursive unicast approach to multicast," in *Proc. IEEE Infocom*, 2000.

[60] P. Francis, "Yoid: Extending the Internet multicast architecture," in *Tech. reports, ACIRI, http://www.aciri.org/yoid*, 2000.

[61] Y. Chawathe, *Scattercast: An architecture for Internet broadcast distribution as an infrastructure service*, Ph.D. thesis, University of California, Berkeley, 2000.

[62] Y. Chu, S. Rao, S. Seshan, and H. Zhang, "Enabling conferencing applications on the Internet using an overlay multicast architecture," in *Proc. ACM SIGCOMM*, 2001.

[63] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001.

[64] B. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault resilient wide-area location and routing," in *Tech. Report. UCB//CSD-01-1141, U. C. Berkeley*, 2001.

[65] L. Cherkasova and J. Lee, "FastReplica: Efficient large file distribution within content delivery network," in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[66] M. Sharma and J. W. Byers, "How well does file size predict wide-area transfer time?," in *Global Internet*, 2002.

[67] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, 1992.

[68] F. P. Kelly, A. K. Maullo, and D. K. H. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of Operational Research Society*, 1998.

[69] J. Roberts and L. Massoulié, "Bandwidth sharing and admission control for elastic traffic," in *ITC Specialist Seminar*, 1998.

[70] T. Lee, *Traffic management & design of multiservice networks: the Internet & ATM networks*, Ph.D. thesis, The University of Texas at Austin, 1999.

[71] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput," in *Proc. ACM SIGCOMM*, 2002.

[72] E. M. Gafni and D. Bertsekas, "Dynamic control of session input rates in communication networks," in *IEEE Trans. Automatic Control*, 1984.

[73] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation:Numerical methods*, Prentice Hall, 1989.

[74] G. Urvoy-Keller and E. W. Biersack, "A multicast congestion control model for overlay networks and its performance," in *Proc. of Networked Group Communication (NGC)*, 2002.

[75] D. E. Comer, *Internetworking with TCP/IP*, Prentice Hall.

[76] R. Stevens, *UNIX Network Programming, Volume 1. Second Edition: Networking APIs: Sockets and XTI*, Prentice Hall, 1998.

[77] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of the First ACM Workshop on Hot Topics in Networks*, 2002.

[78] "Internet2," `http://www.internet2.edu/`.

[79] J. W. Byers, M. Luby, and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: Using Tornado codes to speedup downloads," in *Proc. IEEE Infocom*, 1999.

[80] P. Rodriguez, A. Kirpal, and E. W. Biersack, "Parallel-access for mirror sites in the Internet," in *Proc. IEEE Infocom*, 2000.

[81] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay," in *Proc. ACM SIGCOMM*, 2002.

[82] "Akamai," `http://www.akamai.com/`.

[83] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Topologically-aware construction and server selection," in *Proceedings IEEE Infocom*, 2002.

Vita

Jangwon Lee was born in Seoul, Korea on May 21, 1972, the son of Shinhyo Lee and Soonsim Kim. He graduated from Myongji High School, Seoul, Korea in February 1991 and received his B.S. degree in Electrical Engineering at Seoul National University, Seoul, Korea in February 1995. He began his graduate studies and completed his M.S. degree in Electrical Engineering at Seoul National University in February 1997 and served military duty as a public agent in Seodaemun-Gu, Seoul, Korea from February 1997 to August 1998. He entered the Department of Electrical and Computer Engineering at the University of Texas at Austin in August 1998 and joined Wireless Networking and Communications Group in May 1999. In 2000, he married Jimin Lee and together they now have one son. He finished his Ph.D. in May 2003.

Permanent Address: 3367 Lake Austin Blvd. APT. C

Austin, TX, 78703

This dissertation was typeset with LaTeX 2ε[1] by the author.

[1] LaTeX 2ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the

American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.