MASSIVELY REPLICATING SERVICES IN WIDE-AREA INTERNETWORKS

by

Katia Obraczka

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Electrical Engineering)

December 1994

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

Ratia Obraczka

under the direction of her..... Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY

Alice C. Parker

Dean of Graduate Studies

Date December 7, 1994

DISSERTATION COMMITTEE

Peter B Dervin
Chairperson

Dedication

To my husband, grandparents,
parents, sister, and brothers.

Acknowledgments

This was the most difficult and at the same time the most enjoyable section to write. As I write it, I realize that it could well be the longest section in the dissertation, since I would like to acknowledge all the wonderful people I have met and interacted throughout my life. Unfortunately, for lack of space, I do not explicitly mention all their names.

First and foremost, I would like to express my sincere thanks to my advisor, Dr. Peter Danzig, for his guidance, encouragement, support, and friendship. I was very fortunate to have had the opportunity to work with him during the past 4 years. He has contributed greatly to this dissertation and my maturity as a researcher. I cannot imagine how a professor could be more dedicated to his students. He will always serve as a model to me.

I wish to thank the members of my qualifying and dissertation committees: Deborah Estrin, Clifford Neumann, Shahram Ghandeharizadeh, and John Silvester. I would especially like to thank Professors Estrin and Silvester for their helpful comments and discussions.

I would also like to acknowledge the Brazilian Education Ministry which provided me with a four-year graduate fellowship as a starting PhD student. This research was supported by the Advanced Research Projects Agency under contract number DABT63-93-C-0052.

Several people have assisted in this research. I would like to acknowledge Dante DeLucia for his implementation of *flood-d* and *mirror-d*. Kitinon Wangpattanamongkol has implemented the logical topology calculation algorithm, and Steve Miller has developed the simulation package I used to build my simulators.

I sincerely thank all the members, old and new, faculty and students, of the Network and Distributed Systems Laboratory at USC. I was fortunate enough to be a member of this friendly, enjoying, and stimulating research community. They

definitely made these years at USC a lot more fun. In particular, I would like to thank Prof. Rafael Saavedra, Gene Tsudik, Abhjit Khale, Lee Breslau, Steve Hotz, Doug Fang, Danny Mitzel, Ron Cocchi, Sugih Jamin, Shih-Hao Li, Jong-Suk Ahn, Daniel Zappala, Brenda Timmerman, John Noll, Kraig Meyer, Louie Ramos and Ari Medvinski. Finally, Gary Frenkel, whose memory provided inspiration to move on and face whatever challenges may appear.

I also wish to thank several friends, and fellow graduate students, whose friendship and support made the cultural chock effects resulting from going to graduate school in a foreign country a learning instead of a painful experience. I would especially like to thank Justine Gilman, Patricia Goldweic, Alfredo Weitzenfeld, Eve Schooler, Bob Felderman, and Steve Schrader.

I was also very fortunate to have a family that has always encouraged and supported me. I will always be grateful to my parents, Bertha and Jayme, who have always given me more than I could ever have asked for. My sister Sandra, and my brothers Marcelo, Ricardo, and Eduardo have always contributed and participated in whatever I set out to accomplish.

Finally, I would also like to thank my husband, Mendel, for the love, support, and encouragement which made it possible for me to overcome the obstacles in the life of a graduate student. He has put up with my fears and frustrations, and my finishing the PhD program is as much his accomplishment as it is mine.

Contents

List Of Tables

List Of Figures

Abstract

Existing and future Internet information services will provide large, rapidly evolving, highly accessed, yet autonomously managed information spaces. Internet news, perhaps, is the closest existing precursor to such services. It permits asynchronous updates, is replicated at thousands of autonomously managed sites, manages a large database, yet presents adequate response time. It gets its performance through massive replication. Other Internet services, such as archie, and WWW/Mosaic must massively replicate their data to deliver reasonable performance.

The problem of replicating information that can be partitioned into autonomously managed subspaces has well-known solutions. Naming services such as the Domain Name Service (DNS) and Grapevine use administrative boundaries to partition their hierarchical name space into *domains*. These domains need only be replicated in a handful of services for adequate performance.

On the other hand, efficient massive replication of wide-area, flat, yet autonomously managed data is yet to be demonstrated, since existing replication algorithms do not address the scale and autonomy of today's internetworks. They either ignore network topology issues entirely, or rely on system administrators to hand-configure replica topologies over which updates travel. Lampson's widely cited Global Name Service (GNS) propagates updates over manually configured topologies. However, the complexity of manually configuring fault-tolerant update topologies that use the underlying physical network efficiently increases with the scale of today's internetworks.

Furthermore, GNS-like replication mechanisms also manage single, flat groups of replicas. While this is appropriate for applications with 20 to 30 replicas that operate within single administrative boundaries, it is unrealistic for wide-area, massively replicated services whose replicas spread throughout the Internet's thousands of administrative domains.

This research investigates scalable replication mechanisms for wide-area, autonomously managed services. We propose a new architecture that extends existing replication mechanisms to explicitly address scalability and autonomy. We target replication degrees of thousands or even tens of thousands of weakly-consistent replicas.

Imitating the Internet's administrative domain routing hierarchy, the proposed architecture organizes replicas into multiple *replication groups*. Organizing replicas into multiple groups limits the size of the consistency state each replica keeps. It also preserves autonomy, since it insulates replication groups from other groups' administrative decisions.

We argue that efficient replication algorithms keep replicas weakly consistent by flooding data between them. Our architecture automatically builds a *logical update flooding topology* over which replicas propagate updates to other replicas in their group. Replicas in a group estimate the underlying physical topology. Using the estimated network topology, we compute a logical update topology that is *k-connected* for resilience, tries to use the physical network efficiently, and yet restricts update propagation delays. Replication groups compute and adopt a new update topology every time they detect changes in group membership and network topology.

We describe our architecture in detail and its implementation as a hierarchical, network cognizant replication tool, which we implemented as part of the Harvest resource discovery system. Through simulations, we investigate the benefits of using hierarchical replication groups and distributing updates over the logical update topologies our architecture computes. We present the results from the wide-area experiments we conducted to evaluate our network topology estimation strategy. We also present our logical topology calculation algorithm in detail, and the results obtained when running our algorithm over randomly generated topologies. Finally, we explore the use of internet multicast as the underlying update propagation mechanism. We argue that since having a single multicast group with all replicas of a service will not scale, each replication group can be mapped to a multicast group. Through *corner replicas*, updates in one group make their way to all groups.

Lampson's Global Name Service has been widely accepted as the solution to the problem of replicating wide-area, distributed, weakly-consistent applications. However, almost 10 years ago, when Lampson wrote "... The system I have in mind

is a large one, large enough to encompass all the computers in the world and all the people that use them...", he did not anticipate that the Internet would grow as fast as it did. The main contribution of this dissertation is to make GNS-like services work in today's exponentially-growing, autonomously-managed internetworks.

Chapter 1

Introduction and Motivation

Existing and future Internet information services will provide large, rapidly evolving, highly accessed, yet autonomously managed information spaces. Achieving adequate performance demands that these services and their data be replicated in hundreds or thousands of autonomous networks.

Take the Internet news distribution system [31] as an example: it manages a highly dynamic, weakly consistent, gigabyte database replicated at thousands of autonomous administrative domains, yet responds to queries in seconds. In contrast, archie [22], a directory service for Internet FTP archives, can take fifteen minutes to answer queries against a much smaller, 150 megabyte database. The difference between archie's and network news' performance is massive replication; there are only about 30 replicas of archie. As WWW/Mosaic [2], archie and other upcoming applications [45] become more popular, their databases must be massively replicated for adequate performance.

Below, we examine existing solutions to replicating data. We argue that, although they are correct and keep replicas consistent, they do not address the scale and autonomy of today's internetworks. We start with an overview of the data consistency problem. We also describe a recently proposed replication mechanism that solves some of the previous replication algorithms' problems, but still does not scale to replication degrees of thousands of replicas. Then, we present a brief overview of multicast communication and relate it to the problem of massive data replication. Finally, we briefly describe the architecture we propose for efficient massive replication of data in wide-area internetworks. We end the chapter by outlining the contents of this dissertation.

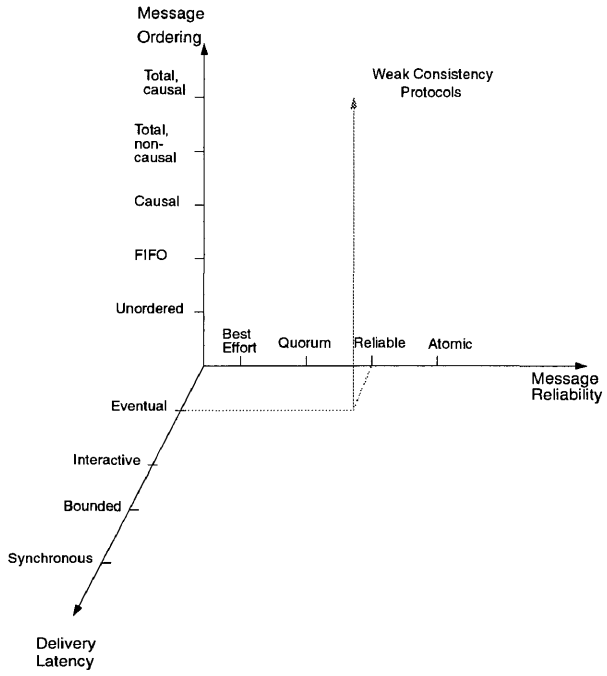

Figure 1.1: Data Consistency Spectrum.

1.1 Data Consistency

Replicas of a service must keep mutually consistent copies of the replicated data. Different applications require different degrees of consistency. Figure 1.1 represents the consistency spectrum as a 3 dimensional space consisting of a *message ordering* axis, and a *message delivery* plane. The axes on this plane are *message reliability* and *delivery latency*. The closer they are to the origin, the weaker their consistency guarantees are. The sample coordinates were extracted from [24].

The *message reliability* axis specifies what subset of the participating replicas are guaranteed to receive a copy of the message. *Atomic* delivery guarantees that a message is either delivered to every group member or to none; delivery is aborted if any replica fails. *Reliable* delivery ensures that a message is delivered to every

functioning replica; if the sender fails, delivery is not guaranteed. In contrast, *quorum* delivery guarantees that some fraction of the replicas are guaranteed to get a message.

The *latency* axis specifies how long replicas may have to wait to receive a message. It depends on when the delivery process begins and ends. *Synchronous* delivery starts immediately and completes within a bounded time. *Bounded* delivery may queue or delay messages, but completes within a bounded time. Both *interactive* and *eventual* delivery may take a finite but unbounded time to complete. Delivery begins immediately in the first case, while messages may be delayed in the second case.

Finally, the *message ordering* axis represents the order messages are delivered to the application. In *total causal* ordering, messages are delivered in the same order to every replica and causal relations between messages are preserved. *Total non-causal* ordering guarantees the same message ordering at every replica, but does not guarantee that causality is preserved. In *Causal* delivery, messages are delivered in an order that preserves potential causal relations. *FIFO* ordering guarantees that only messages from the same source are delivered in order. *Unordered* delivery does not make any ordering guarantees.

Many existing group communication mechanisms populate the consistency spectrum. Their consistency guarantees range from atomic, synchronous, totally ordered delivery to best-effort, eventual delivery. Strong consistency protocols are expensive and do not scale in unreliable, long-haul communication networks. Since they generate considerable latency and overhead traffic and block replicas while propagating updates, they are adequate for services with a small number of replicas or services that need strong consistency guarantees. In contrast, wide-area, massively replicated services should not trade availability and response time for strong consistency guarantees. On the one hand, these services still need to ensure that replicas eventually converge to a consistent, updated state during both normal operation and when recovering from network partition and server or link failures. On the other hand, they need not compromise their availability and response time and incur the extra overhead of strong consistency protocols.

In fact, for availability and robustness, Xerox's Grapevine [59, 60], its commercial successor, the Clearinghouse [46], the Global Name Service [33], and network news

[31] use weak consistency replication mechanisms. When link or node failures result in network partitions, replicas are allowed to diverge and continue to provide service. Once the partition is mended, replicas eventually converge to a consistent state.

Weak consistency replication is also adequate for mobile computing environments. If replicas reside on mobile computers, these replicas will provide service even during disconnection. When reconnected, mobile replicas will exchange updates with the other replicas.

For these reasons, the architecture we propose extends weak consistency protocols to massively replicate data efficiently.

1.2 What Current Algorithms Lack

As existing naming services and distributed file systems have demonstrated, the problem of replicating data that can be partitioned into autonomously managed subspaces has well-known solutions. Naming services such as the Domain Name Service (DNS) [39, 40] and Grapevine organize their name space hierarchically according to well-defined administrative boundaries. They also use these administrative boundaries to partition their name space into several *domains*, which only need to be replicated in a handful of servers to meet adequate performance. In fact, according to the study presented in [17], over 85% of second level domains in the DNS hierarchy are replicated at most three times, while 100% of these domains use at most 7 replicas. The same study also shows that more than 90% of DNS's second-level domains store less than 1,000 entries. Because of the limited domain sizes and small number of replicas, DNS's primary-copy replication scheme performs quite adequately.

Similarly, distributed file systems organize their file space hierarchically, where intermediate nodes are directories and leaf nodes are files. Like LOCUS [51], Andrew-AFS [1] [42, 27], and Coda [58], distributed file systems use locality of reference to partition their file space into directory subtrees. File servers replicate a subset of

[1] Andrew is the name of the research project at Carnegie-Mellon University. AFS is based on Andrew, and has become a product marketed and supported by Transarc Corporation.

files in a directory subtree. Both LOCUS and Andrew provide read-only file replication, while Coda uses distributed updates to keep its read-write file replicas weakly consistent.

On the other hand, efficient massive replication of wide-area, flat, yet autonomously managed data is yet to be demonstrated, since existing replication algorithms do not address the scalability and autonomy of today's internetworks. Like Grapevine, the Clearinghouse, and the Global Name Service, existing replication solutions do not scale because they manage a single, flat group of replicas. While this is appropriate for applications with 20 to 30 replicas that operate within single administrative boundaries, it is unrealistic for wide-area, massively replicated services whose replicas spread throughout the Internet's thousands of administrative domains.

We also argue that efficient replication algorithms flood data between replicas. In fact, Internet news employs flooding to distribute updates among its thousands of replicas. Note that the flooding scheme that we propose differs from network-level flooding as used by routing algorithms: flooding at the network level simply follows the network's physical topology and floods updates throughout all physical links of the network. Instead, replicas flood data to their logical neighbors or peer replicas. Although the word "flooding" sounds inefficient, we claim that the application-level flooding scheme we propose does use network bandwidth efficiently.

Because layered network protocols hide the network topology details from application protocols, in existing replication algorithms replicas cannot select their peers to optimize their use of the underlying physical network. Most existing distributed replication mechanisms ignore network topology: they treat all physical links as having equal delay and bandwidth, and thus do not try to use the network efficiently, deliver timely updates, and adapt to network partition and temporary or permanent site failures. For instance, both Grapevine and the Clearinghouse ignore network and update topology. The Global Name Service assumes the existence of a single administrator who hand-configures the topology over which updates travel. The Global Name Service administrator places replicas in a Hamiltonian cycle, and reconfigures the ring when replicas are added or removed. As the number of replicas grows and replicas spread beyond single administrative boundaries, frequently reconfiguring the ring gets prohibitively expensive. Like the Global Name Service, Internet news site administrators hand-configure the flooding topology over which

their updates travel. Since obtaining current physical topology information is difficult in today's Internet, system administrators frequently confer with one another to plan changes in the logical flooding topology. They try to keep up with the dynamics of the underlying physical topology: a task that becomes increasingly hard as the Internet's scale and complexity increase.

1.3 Timestamped, Anti-Entropy Replication

In [24], Golding proposes a weak consistency group communication mechanism. Inspired by Xerox Clearinghouse's anti-entropy protocol, Golding's message delivery component uses the timestamped anti-entropy (TSAE) protocol which provides reliable, eventual delivery of client data. The basic idea behind anti-entropy protocols is that each replica periodically starts an *anti-entropy* session, in which it selects a peer to exchange updates.

In the TSAE protocol, when a replica has a message to send, it timestamps the message and writes it to a log. During anti-entropy sessions, replicas exchange the contents of their message logs. This message log organizes messages according to their originating replica. To make the anti-entropy update exchange more efficient, each replica keeps a *summary vector* containing the latest message timestamps it has received from each replica in the group. Replicas exchange their summary vectors and then exchange the missing messages. At the end of an anti-entropy session, both replicas have the same continuous sequence of messages sent by each group member. Each replica uses the smallest timestamp in its summary vector as its acknowledgment timestamp and keeps an acknowledgment timestamp for each replica in the group. Replicas also exchange their *acknowledgment vectors* and use them to purge outstanding messages from their message logs.

Through its anti-entropy sessions, the TSAE protocol ensures that replicas eventually converge to a consistent state during normal operation or when recovering from link and node failures, and network partitions. The TSAE protocol combined with a message ordering mechanism provides specific message ordering guarantees. For instance, the *Refdbms* replicated bibliographic database [24] uses the TSAE protocol combined with a total, non-causal ordering mechanism.

Golding's group communication system also includes a group membership component which allows the group of communicating replicas to be dynamic. The proposed weak consistency group membership mechanism guarantees that each participating replica's view of the group eventually converges. It provides the following operations. *Initialization* allows a replica to create a new group. It assumes the existence of a *location service* with which it registers newly created groups. Replicas *join* a group by querying the location service to find at least one current group member. To maintain k-resilience, a joining replica must obtain $k+1$ *sponsors* before becoming a member. It gets its first sponsor, who sends it a copy of the entire group state. It then gets additional sponsors. The new member and sponsors propagate the updated group membership to the rest of the group in anti-entropy sessions. A replica that wants to *leave* the group must first declare its intention to leave and perform anti entropy with at least one member. It must also wait until all current members know it is leaving. A failed replica is *ejected* from the group to avoid that upon recovery, the replica re-injects old messages.

We argue that like other replication mechanisms, which do not try to use the underlying network efficiently and that manage a single, flat replication group, Golding's group communication system will not scale to thousands of replicas spread throughout the Internet's autonomous administrative domains.

1.4 Internet Multicast and Multicast Transport Protocols

Multicast communication is the delivery of data from a source to a group of recipients identified by a single destination address. In particular, Internet multicast [18, 19] delivers best-effort datagrams to a group of hosts sharing a single IP multicast address. Internet multicast architectures build a shortest-path tree to transmit a datagram packet to a group of hosts sharing a single IP multicast address. Multicast sources need only transmit a single copy of a packet, which gets replicated as needed at each branching point of the multicast tree.

Because it minimizes the number of packets sent from a source to its destinations, Internet multicast and reliable multicast transport protocols are often considered to

provide a good foundation on which to build massive replication protocols. Further, since the network itself supports Internet multicast, it has access to the network routing database, and therefore can make routing decisions so that the underlying network is efficiently used.

Like the IP protocol [53], Internet multicast provides best-effort delivery, and leaves reliability up to transport-level protocols. By providing only a minimum functionality delivery service, applications such as real-time voice and video teleconferencing, which can live with data losses, do not have to incur the extra overhead of reliable delivery.

However, contrary to real-time applications, information services, which are the focus of this research, need reliable message delivery. We argue that a reliable multipoint transport protocol does not provide the required reliability. In particular, a reliable multipoint transport protocol does not solve the consistency problem in the case of a replica being temporarily removed from service, losing state used in consistency maintenance. In this case, only the application can re-establish consistency.

The end-to-end argument in layered system design [57] recommends that if a function can only be completely and correctly implemented by the application, that function should be moved upward to the application that uses it. Since replicas of an information service need to perform application-level consistency checks anyway, we argue that replication tools do not have to rely on a reliable multicast transport protocol, although, for efficiency, they can exploit such protocols where available.

1.5 Dissertation Overview and Outline

In the previous sections, we argued that efficient replication algorithms keep replicas weakly consistent by flooding updates between them and don't necessarily have to rely on existing multicast transport protocols. We also argued that although existing replication mechanisms use weak consistency, flooding-based protocols, they will not scale to thousands of replicas spread throughout the Internet's autonomous administrative domains. Existing weak consistency replication algorithms manage single replication groups, and do not try to use the underlying physical network efficiently.

This dissertation proposes an architecture for efficient massive replication of wide-area, autonomously managed services. We target replication degrees of thousands or even tens of thousands of weakly consistent replicas scattered throughout the Internet's thousands of administrative domains. Our architecture extends existing replication mechanisms to address the scale and autonomy of today's internetworks explicitly.

Since distributed systems that scale well are organized hierarchically, the proposed architecture groups replicas into multiple hierarchical *replication groups* analogous to the way the Internet partitions itself into autonomous routing domains. This hierarchical organization limits the size of the consistency state each replica needs to keep to perform periodic end-to-end consistency checks. It also insulates groups against administrative decisions from neighboring, autonomously managed groups, and from most of the network traffic associated with group membership. Replication groups overlap with one another, so that an update that originates in a group makes its way to all groups.

Using client updates as probe messages, group replicas estimate the underlying physical topology. Based on the estimated physical topology, our architecture automatically builds an *update flooding topology* between the group members. The resulting update topology uses the physical network efficiently, is k-connected for resilience, and has limited diameter to restrict update propagation delays. Whenever the probing mechanism detects a significant change in the physical topology connecting members of a replication group, it spawns a topology computation process, which modifies the current topology accordingly.

Over the past 5 years, several information discovery services have been helping users locate and retrieve information available on the Internet. However, the combined growth in data volume, data diversity, and user base created performance bottlenecks early information discovery tools cannot overcome. Through a set of customizable information gathering tools, topic-specific indexes, efficient index searching mechanisms, and massive object replication and caching, Harvest [11] was designed to become the National Information Infrastructure's resource discovery service. Our hierarchical, network-cognizant replication tool supports Harvest's massive object replication.

This dissertation is organized as follows. In the next chapter, we present our hierarchical, network-cognizant replication tool's architecture in detail. We also describe its implementation as *flood-d*, one of the components of the Harvest resource discovery system. Chapter 3 describes the wide-area experiments we conducted to evaluate our network topology estimation strategy, and compares flood-d's communication latency and available bandwidth estimates with measurements obtained from available networking tools. In Chapter 4, we present our topology computation algorithm and the results the algorithm produces using randomly generated topologies as input. Through simulations, we investigate the benefits of using multiple replication groups and distributing updates over the logical topologies our architecture computes, and present the results in Chapter 5. Chapter 6 explores the use of internet multicast as the underlying update propagation mechanism, and proposes a reliable multipoint transport protocol well suited for weak consistent applications. Finally, Chapter 7 summarizes this work, discusses topics for future research, and presents some concluding remarks.

Chapter 2

A Hierarchical, Network-Cognizant Replication Architecture

In this chapter, we present our hierarchical, network cognizant architecture in detail. We describe our network topology estimation strategy and contrast it with topology discovery. We also describe the implementation of our architecture as *flood-d*, a tool that supports *mirror-d*, a weakly-consistent file archiver used to replicate Harvest's [11] indexing databases.

2.1 Overview

The architecture we propose extends weak consistency, flooding-based replication protocols, such as Golding's TSAE protocol, to address scale and autonomy explicitly. It clusters replicas of a service into multiple, autonomously administered replication groups imitating the Internet's administrative domain hierarchy. Organizing replicas into groups limits the size of the consistency state that each replica keeps and reduces the time to reach consistency among replicas of a service. Having multiple replication groups also preserves autonomy, since administrative decisions of one group, such as its connectivity or when it should be split in two, do not affect other groups. Also, it insulates groups from topological rearrangements of their neighboring groups and from most of the network traffic associated with group membership.

Unlike Lampson's Global Name Service, the logical update topologies that interconnect members of a replication group are not restricted to a Hamiltonian cycle. We build k-connected topologies for resilience. Recall that in a k-connected network [5], at least $k-1$ nodes must break before the network is partitioned. The ring topology connecting Global Name Service replicas corresponds to a 2-connected topology where all edges have equal cost. The resulting k-connected topologies also try to use the underlying network efficiently, while limiting update propagation delays.

Our architecture, which automatically builds logical update topologies that have the properties described above, consists of a network topology estimator [1] and a logical topology calculator. Every group replica measures available bandwidth and propagation delay to the other group members. Based on these estimates, the logical topology calculator builds a high bandwidth, limited diameter, k-connected update topology for the group. The topology's diameter corresponds to the maximum number of hops that updates need to travel.

By automating the process of building update topologies among replicas of a service, the proposed architecture offloads logical topology decisions from system administrators.

[1] An alternative strategy for network topology information acquisition is discussed later in this chapter.

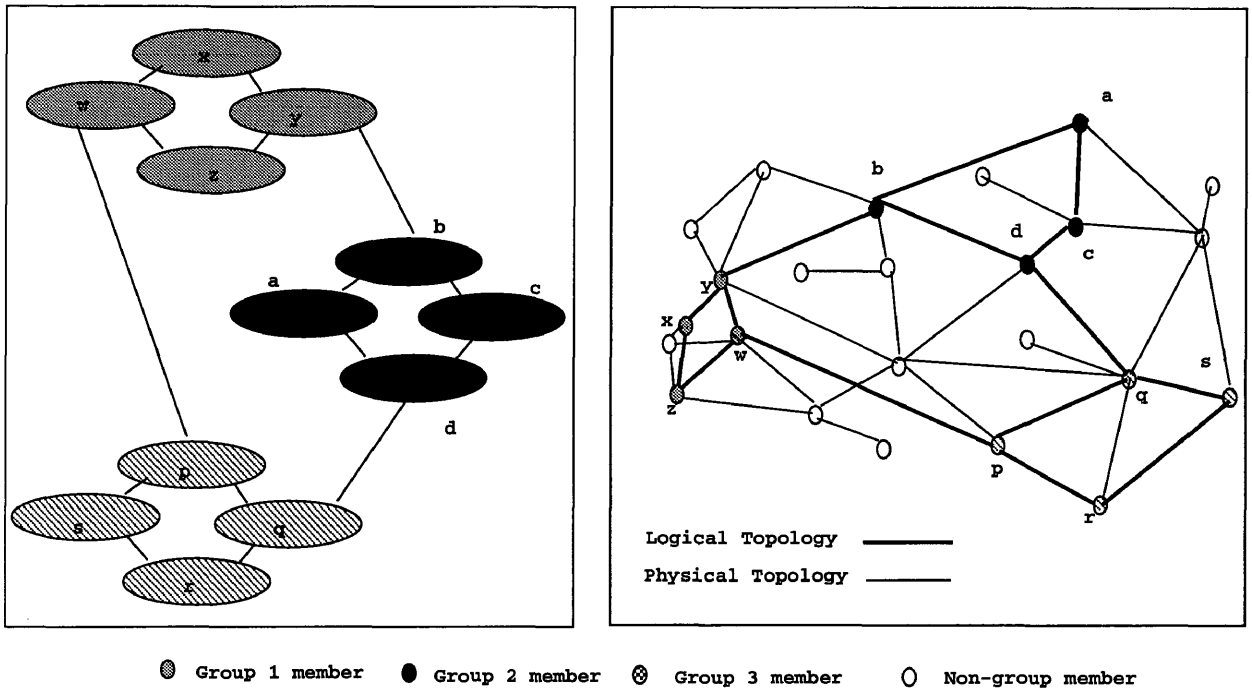

Figure 2.1: Replication groups showing logical versus physical topologies.

2.1.1 Groups and Network Topology

Figure 2.1 illustrates the relationship between logical topologies and the underlying physical network. The left-hand figure shows three replication groups and their logical update topologies. The right-hand figure shows the physical network topology and the logical update topology built on top of it for the three replication groups in the left-hand figure.

We should point out that using logical update topologies does not circumvent Internet routing. On the contrary, network routing can work around occasional bad choices made by the update topology.

2.1.2 Flooding and Peer Selection

Besides organizing replicas into multiple groups, our architecture incorporates the notion of network and logical update topologies into existing weak-consistency, flooding-based replication mechanisms.

For instance, in the original TSAE protocol, every replica randomly selects another replica to exchange missing updates during an *anti-entropy* session. By using the k-connected, high-bandwidth, limited-diameter logical update topologies our architecture builds, TSAE replicas make efficient use of the physical network, while limiting delays when propagating updates. Furthermore, our logical update topologies adapt to group membership and physical topology changes.

2.1.3 Consistency Between Groups

The TSAE protocol maintains consistency between replication groups as easily as it does between members of a group. Between replication groups, it simply communicates with representative individual replicas, or *corner replicas*. Since replicas flood updates to their neighbors in the logical topology, updates in one group make their way to all groups.

Although network node and link failures may result in network partitions, and permanent node failures and group membership changes may introduce temporary inconsistencies, TSAE eventually resolves them as long as our architecture keeps the nodes connected.

2.1.4 Consistency State Size

Organizing replicas into multiple groups limits the amount of consistency state each replica needs to keep. Each replica running the TSAE protocol must store all object updates from other participating replicas in its group. This requires $O(rn)$ space, where r is the the group size and n is the number of UN-purged update log entries. During anti-entropy sessions, a replica exchanges its consistency state with other replicas. When it realizes that all group members have received an update, the replica purges the corresponding update log entry.

By splitting a group into g smaller groups, the size of a replica's consistency state decreases to $O((r/g)n)$. In other words, replicas only keep state for replicas within their own group. Corner replicas also need to keep an aggregate state for each group to which they belong and hence maintain $O((r/g + g)n)$ state.

Multiple replication groups preserve autonomy by insulating groups against administrative decisions from neighboring, autonomously administered groups. They also limit network traffic associated with group membership.

2.1.5 Updating Logical Topologies

Network nodes and links may fail temporarily, or may be permanently removed from service. Replicas may also join and leave a flooding group. The group membership protocol and physical topology estimation will eventually detect these changes, which will be reflected in the new k-connected graph the group computes.

Our architecture also uses flooding to propagate topology updates to all members of a replication group. Topology update messages carry a sequence number corresponding to the topology identifier, which replicas use to order topology updates, and detect duplicates. Topology update messages also contain the new group membership set. When a replica receives a topology update, it floods the new topology according to the current topology before committing the new topology.

Topology changes may occur while an application-level update is being propagated. In this case, the update propagates according to the current topology, and when it gets to replicas that already committed the new topology, the update propagates following the new topology. Since the new topology is also a k-connected graph, all replicas will eventually get the application-level update. However, more duplicates may be generated since replicas that have already received updates, may receive them again after the topology change.

The topology update process generates additional traffic associated with propagating topology update messages to the participating replicas. The resulting overhead in terms of the total number of messages generated is proportional to the number of participating replicas, and the frequency in which topology updates occur. In a highly replicated service whose copies are spread throughout large internets, the topology update overhead may become prohibitively high as the number of replicas, and the frequency of physical topology and flooding group membership changes can get considerably high. Our hierarchical approach helps limit this overhead. Grouping replicas located physically close to one another restricts the scope of the changes

that trigger topology updates. It also limits the scope of the resulting topology updates to the local group, and therefore restricts topology update traffic on the more expensive, long-haul physical links.

2.2 Physical Topology

We claim that, analogously to the fact that database applications need to have access to raw disks for efficiency while the underlying operating system exports file system abstractions, existing and future network applications will benefit from exposed physical network topology information. In particular, our replication architecture builds logical update topologies that try to make efficient use of the underlying network, while limiting update propagation delays. It builds these logical topologies based on information about the current state of the network.

Because of its size, complexity, and autonomy, the Internet's physical topology is neither well known nor stable. While our current strategy is to probe the underlying physical network to estimate its characteristics, in the next section, we present some topology discovery approaches. In Section 2.2.2, we describe our topology estimation strategy.

2.2.1 Topology Discovery

The topology discovery approach relies on getting topological information indirectly through some existing lower-level mechanism, such as routing protocols. We briefly examine some existing internetwork routing protocols and topology discovery tools.

Today's Internet connects thousands of autonomous Administrative Domains (ADs). An AD is a set of hosts, networks, and gateways administered by a single authority [7]. Roughly speaking, the current inter-AD topology can be described as a hierarchy consisting of long-haul backbone, regional, metropolitan and local-area networks. This hierarchy is augmented with bypass links connecting non-adjacent levels.

Interior gateway protocols (IGPs) are routing protocols used within an AD. Several IGPs are currently in use on the Internet. The Routing Information Protocol(RIP) [25], the HELLO protocol [38] are examples of IGPs that use distance-vector algorithms [66], whereas the OSPF protocol [43] exemplifies link-state based [66] IGPs. Because IGPs are limited to operate within single ADs, each one of them may use a different cost metric.

Exterior Gateway Protocols have been used in inter-AD routing. They support heterogeneity by interconnecting ADs that run different IGPs. EGP [55] is a distance vector exterior gateway protocol. Because there is no universally accepted measure of distance between two networks in two different autonomous domains, EGP gateways exchange reachability information using the number of intermediate EGP gateways along the path from source to destination as the hop count metric. The Border Gateway Protocol (BGP) [35] has been proposed as a successor to EGP. BGP is also a distance vector protocol that uses hop count as the routing metric. As an enhancement to EGP, BGP avoids loops by including full path information in its routing updates.

Due to the complexity and heterogeneity of today's large internets, topology discovery based on inter-domain routing does not provide complete topology information. By using Exterior Gateway Protocol routing data, we can find out about the participating ADs, and the corresponding reachability information. However, it does not provide real link costs.

A number of automated topology discovery tools have been developed as part of network management systems. These tools employ one or more of the following methods to discover topology. The first approach relies on sending ICMP [52] echo requests to a range of addresses, and then adding the addresses that respond to a list of nodes. This approach finds all nodes on the network that respond to an ICMP, but can be very time consuming. The second method uses the Simple Network Management Protocol (SNMP) [8] to collect topology and routing information from the Management Information Base (MIB) [37] kept in SNMP-reachable nodes. Using this method, more complete information can be collected, but only SNMP-reachable nodes will be discovered. The third approach discovers nodes by monitoring local Ethernet segments. Based on addresses in the headers of the observed packets, it creates a list of nodes. This method does not discover nodes on remote segments.

In [64], Schwartz et. al. present an architecture for discovering the characteristics of large internets, including topology, congestion, routing, and protocol usage. The proposed tool uses a wide range of existing protocols such as SNMP, ICMP, ARP [50], and available information sources such as DNS [41, 39], and routing protocols. It employs passive monitoring, and active probing of various internet segments, and caches relevant information at various points around the network. Although this tool can generate more complete and detailed internet maps than traditional network management systems, it still does not discover link costs.

Rouvellou et. al. [56] propose a method for discovering the topology of a network based on partial or corrupted neighbor information provided by participating nodes. The proposed method accomplishes topology identification through a combinatorial optimization technique that minimizes cost. Costs are assigned to measure how well a graph fits the data provided by the participating nodes given that a set of corruptions might have occurred. Because it has been designed to locate nodes of mobile networks, which have dynamically changing topology, this technique only helps in assessing node reachability. It does not estimate real link costs.

2.2.2 Estimating Physical Topology

Our current strategy is to estimate the characteristics of the underlying physical topology. Physical topology estimation relies on probing the physical network to estimate relevant cost parameters and computing a cost matrix based of a pre-defined cost function.

Making communication cost a function of hop count accounts for the utilization of network resources when propagating data; the higher the hop count, the more expensive it is to propagate data. However, using hop count as the cost metric makes all communication links look the same.

Specifying the cost function only in terms of the available bandwidth can be misleading. It would make higher bandwidth paths look better than lower bandwidth ones, even if the higher bandwidth paths traverse longer geographical distances. For analogous reasons, defining cost only in terms of latency would result in not getting any benefit from higher bandwidth paths.

We are currently using a cost function that combines latency and available bandwidth between a given source-destination pair. Latency accounts for propagation and queuing delays incurred at the intermediate nodes and links. We should point out that for shorter distances, which is the case of sites in a local-area network, bandwidth is the dominating factor in communication cost. On the other hand, for longer distances, like in long-haul networks, propagation delays are no longer negligible.

2.3 Implementation

As a result of the Internet Resource Discovery Task Force (IRTF) efforts [63, 6], the Harvest resource discovery system [11] has been designed and implemented to solve the scalability and efficiency problems of early resource discovery services. For availability and response time, Harvest relies on massive replication of its indexing databases, or *brokers*. such as its directory service. In particular, Harvest's directory, which stores information about all available brokers, is expected to be highly accessed and must be massively replicated for adequate performance.

Our hierarchical, network-cognizant replication architecture, which we call *flood-d*, supports Harvest broker replication. Flood-d currently provides a flooding-based, group communication layer, which information systems can use to propagate data among their replicas. These applications use flood-d as their underlying data propagation mechanism and perform periodic end-to-end consistency checks to detect missing updates.

Flood-d provides the data distribution mechanism which *mirror-d*, a weakly-consistent, replicated file archiver, uses to replicate files. Harvest brokers use *mirror-d* to replicate their data.

2.3.1 Flood-d

When a replica receives data to propagate, it floods an update message to replicas that are its logical neighbors, according to the current logical topology. Flood-d estimates available peer-to-peer bandwidth when a replica sends an update to another replica. To measure available bandwidth to non-neighbors, a replica occasionally

exchanges updates with one or more non-neighbor replicas within its replication group.

Since flood-d uses TCP for point-to-point update transmission, it estimates the effective, *flow-controlled* bandwidth between two replicas. If updates are small, then propagation delay and TCP slow start dominate link speed in this estimation, yielding higher bandwidths to closer peers.

Flood-d also measures communication delays between pairs of replicas. Each replica periodically sends short UDP messages to other group members. We use UDP to avoid that the connection setup overhead of TCP interfere with our measurements. Probe messages are short so that each probe fits in a single packet.

Any flood-d replica can perform topology calculations. Currently, a replica in a group is designated as the *group master* and is responsible for computing logical update topologies for the group. From time to time, the current group master collects cost estimates from replicas in its group and then computes the all-pairs shortest-paths between group members, generating a fully-connected cost matrix. Using this cost matrix and the group's desired connectivity (typically 2 or 3), it computes a new logical update topology and, if the new topology is significantly better than the old, distributes it to all the group members.

Topology update messages carry a sequence number corresponding to the topology identifier, which replicas use to order topology updates and detect duplicates. Topology update messages also contain the new group membership set. When a replica receives a topology update, it floods the update according to the current topology before committing the new topology.

Replica update messages also carry a topology sequence number. If a replica learns of an update from one of its peers, but the update now carries a higher topology sequence number, the replica must re-flood the update as if it hadn't received it before.

Replicas may join or leave a replication group at any time. To join a group, a new replica copies a neighbor's database, and floods its existence to the rest of the group. When a new member joins the group, a topology calculation is spawned. The resulting topology includes the new replica and is distributed to the group.

```
;
; Configuration for site w.
;

; Define site 'w'
(:site-define    ; What we are defining.
 (:site-name   w)        ; A convenient name for the sites.
 (:hostname    w)        ; The hostname where this site is located.
 (:client-port 2000)   ; Where a client can talk to me.
 (:data-port   2001)     ; Where other flood daemons talk to me
 (:longitude -118.0)     ; The physical coordinates of
 (:lattitude 34.0)   ;   a site.
 (:topology-period 86400)) ; Generate a new topology every day.
```

```
;
; Configuration for the group of which site w is a member.
;
;
(:group-define
 (:group-name gray)      ; Identify the group by name.
 (:site w)       ; This site is in the group.
 (:site x)               ; and so on...
 (:site y)               ; *GATEWAY* to black group.
 (:site z)
 (:site p)       ; *GATEWAY* to slash group.
 (:bandwidth-period 3600.0)   ; Estimate bandwidth to another site every hour.
 (:estimates-period 900.0)    ; Send estimates out every 15 minutes.
 (:master-site w))            ; Who is responsible for generating topology.
```

Figure 2.2: Example flood-d site and group configuration.

Sites leave a replication group silently; if members of a group have not heard from a site in a specified period of time, they assume the silent site has left the group, and is excluded from the group membership set and topology computations.

Several configuration parameters can be set to modify flood-d's operation. The left-hand side of Figure 2.2 illustrates a sample configuration for a flood-d replica. In particular, this is the configuration for Figure 2.1's replica **w**. In the right-hand side of Figure 2.2, we show the configuration for the replication group of which replica **w** is a member.

Figure 2.3 illustrates flood-d's group monitoring tool. Using this graphical interface to flood-d, group managers can view the current membership list, as well as information about individual sites. This tool also displays the current logical topology of a group from a specific site's perspective.

2.3.2 Mirror-d

Mirror-d is a weakly-consistent, replicated file archiver that uses flood-d as its underlying update propagation mechanism. As shown in Figure 2.4, mirror-d is designed

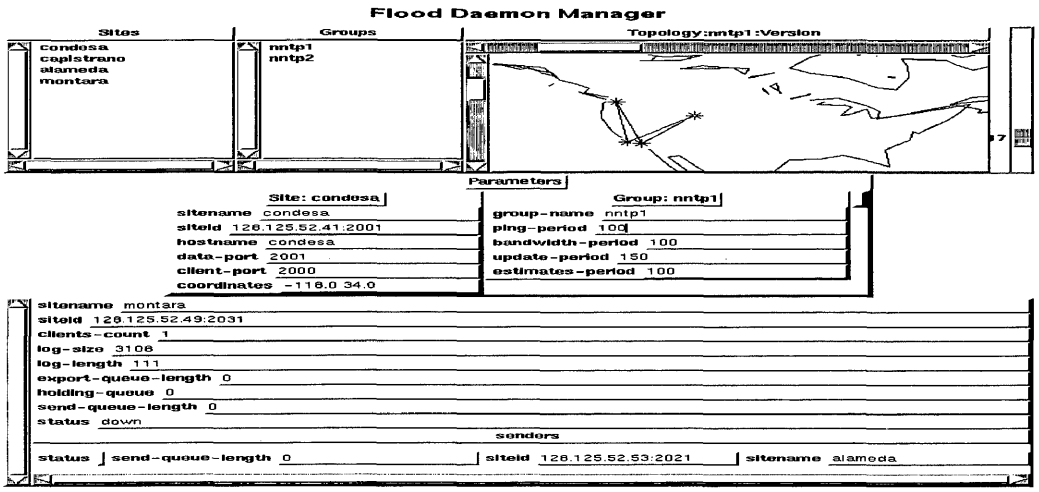

Figure 2.3: Flood-d's group monitoring tool.

as a collection of master copies that update a common archive. Every replica in the common archive has a copy of everything that is being replicated. In the context of Harvest for example, master copies represent brokers' databases; this means that the same pool of replicas may replicate different brokers.

Whenever a file is added or deleted, the corresponding master copy sends out an update to the common archive. The master copy sends the update to one of the replicas which propagates it to the other replicas using flood-d. Notice that in Figure 2.4, the archive replicas use a 2-connected update logical topology.

Periodically, master copies generate a state message with their current file lists. State messages propagate to all replicas, who check whether their copies are up-to-date. If a replica detects it is missing a file, it requests that file from one of its neighbors. In case a replica detects it has a file which is no longer present in the master copy's list, the replica deletes that file. Therefore, any file additions or deletions in a master copy eventually propagate to all replicas. We should point out that only updates to the master copies propagate to the archive. Therefore, if a replica adds a file to its copy, it will not get propagated to the other replicas. Furthermore, eventually, that file will get deleted when the replica receives a state message from the master copy.

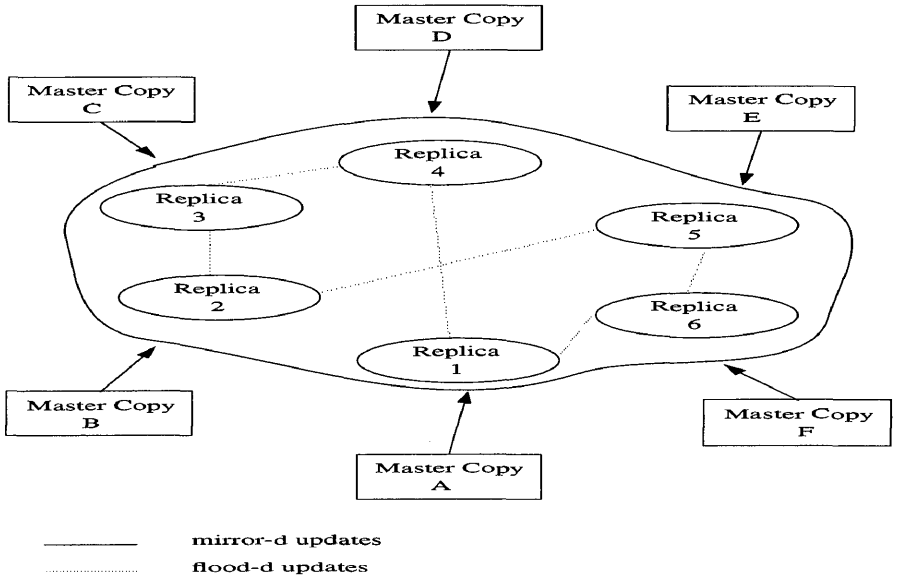

Figure 2.4: Mirror-d master and slave copies.

Chapter 3

Evaluating Network Topology Estimation

In the previous chapter, we presented our hierarchical, network-cognizant replication architecture. We also described its implementation as flood-d, a multi-point communication tool, which service replicas use as their data propagation mechanism. Flood-d propagates data according to a pre-computed logical topology, which tries to use the underlying physical network efficiently, while limiting update propagation time. To compute these logical update topologies, replicas running flood-d probe the network, and estimate latency and available bandwidth.

This chapter describes the wide-area experiments we conducted to evaluate our network estimation strategy. We present flood-d's latency and bandwidth estimates, and compare them with measurements obtained from other tools, such as *traceroute* [28].

3.1 Wide-Area Experiments

For our wide-area experiments, we used a single replication group, whose size fluctuated around 10 replicas. Participating replicas were located at different internet sites: some replicas were run locally at the USC Networking and Distributed Systems Laboratory; at least one replica was located outside of the US, more specifically in Sidney, Australia; the remaining replicas were located throughout the continental US.

The experiments consisted of collecting latency and bandwidth estimates from participating replicas during a period of time. They were conducted at different times of the day and on different days of the week to capture the variation in network load.

Currently, replicas running flood-d periodically send probe messages to other group members to measure communication latency. When it is time for a replica to estimate latency, the replica sends a probe message to a randomly selected replica in its group. Probe messages carry only header information, so that they fit in a single UDP packet. Before sending a probe, a replica timestamps it. The probed replica sends back a response, which carries the probe's original timestamp. Latency is computed as

$$latency = timestamp_2 - timestamp_1$$

where $timestamp_1$ is the probe's original timestamp and $timestamp_2$ is the time when the replica that originated the probe received its reply. Notice that the value of $timestamp_2$ accounts for the load at the replica being probed.

The frequency at which replicas probe one another is one of flood-d's group configuration parameters. Replication group administrators adjust latency estimation frequency depending on how fast they want their group to adapt to membership or network topology changes.

Replicas estimate available bandwidth during update propagation: when a replica sends an update to its neighbors, it also measures available bandwidth to them. The replica that is being updated records the time it receives the update's first and last bytes. Available bandwidth is estimated as

$$bandwidth = update\_size/(time_{last\_byte} - time_{first\_byte})$$

where $update\_size$ is the size of the update message, and $time_{first\_byte}$ and $time_{last\_byte}$ are the times the replica being updated received the update's first and last bytes, respectively.

To find out whether there are currently better alternate logical links, replicas also measure available bandwidth to non-neighbor replicas. Periodically, a replica randomly selects a non-neighbor replica in its group, sends it a fixed-size, dummy update, and estimates the corresponding available bandwidth. The frequency at which replicas perform bandwidth estimation to non-neighbors is another one of flood-d's group configuration parameters, which group administrators can adjust according to how adaptive to network topology and group membership changes they want their group to be. However, besides generating additional overhead traffic, high latency and bandwidth estimation frequencies may also cause groups to adapt to transients.

3.2 Latency Estimates

Below, we present the latency estimates we collected during our wide-area experiments. While collecting flood-d latency estimates, we also collected round-trip time values from *traceroute* probes to serve as a basis for comparison.

Traceroute [28] is a network management tool used to track routes IP packets take between a given source-destination pair. It uses UDP probe packets, and increments the packet's IP protocol's time-to-live value by 1 each time, so that the packet goes 1 hop further until it reaches the destination or the maximum time-to-live (30 hops by default). For each time-to-live value, traceroute issues 3 probes by default, and besides the route, it also reports the round-trip time of each probe. In our experiments, we periodically invoke traceroute from the replica that is collecting flood-d estimates to the other group members, and record the average round-trip time among the values reported by the 3 traceroute probes. The time between

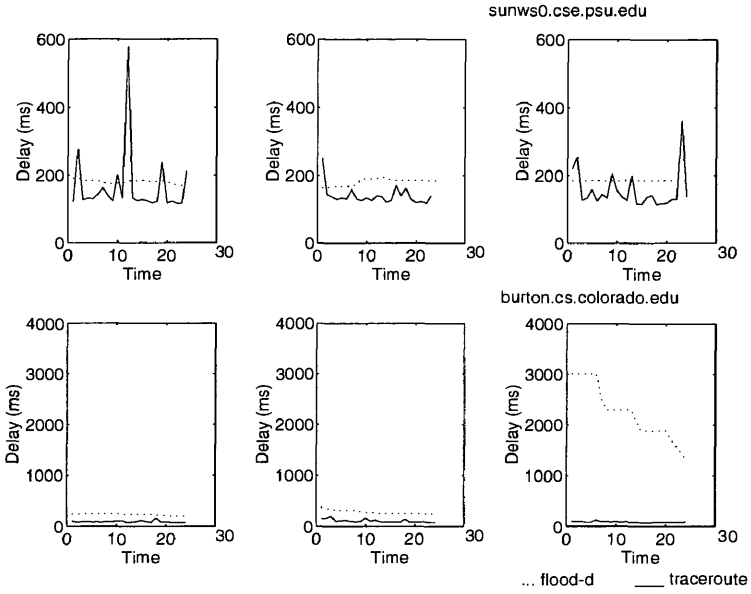

Figure 3.1: Flood-d's latency estimates and traceroute's round-trip time measurements.

consecutive traceroute invocations is configured to be the same as flood-d's latency estimation interval.

The graphs in Figure 3.1 show 3 sets of latency estimates and traceroute round-trip times collected at different times of day on different days of the week from a USC site, *warthog.usc.edu*, to 2 remote sites: *sunws0.cse.psu.edu* located at Pennsylvania State University, and *burton.cs.colorado.edu* located at the University of Colorado in Boulder. The graphs in the first, second, and third columns plot values collected Saturday, June 4 1994, from 3:30 pm to 3:30 pm, Monday, June 6 from 2:30 pm to 2:30 am, and Tuesday, June 7 from 5:30 pm and 5:30 am, respectively. Data was collected every 30 minutes.

The top and bottom graphs show measurements from *warthog.usc.edu* to *sunws0.cse.psu.edu* and *burton.cs.colorado.edu*, respectively. The values plotted in the first column graphs were collected Saturday, June 4 1994, from 3:30 pm to 3:30 am. The second and third column graphs contain values collected between 2:30 pm and 2:30 am on Monday, June 6, and 5:30 pm and 5:30 am on Tuesday, June 7, respectively.

We observe that flood-d estimates do not capture as much oscillation in communication latency as traceroute round-trip times, which means that flood-d estimates tend not to capture short-lived changes in network topology. This is due to the way flood-d estimates are computed: to dampen the effects of transients, flood-d takes into account previous history, by combining the current estimate with the previous one. In fact, as the values in the first-row graphs show, flood-d latency estimates average out the transients traceroute detects. We should also point out that because flood-d's latency estimates are user-level measurements, they are typically higher than traceroute round-trip times. Therefore, latency estimates to local sites are dominated by the protocol stack overhead.

The results reported in the second-row graphs show how machine load can affect flood-d's latency estimates. Flood-d's estimates taken on June 7 (right-most graph) present a big discrepancy in relation to the values collected on June 4, and June 6. Furthermore, the June 7 values also report a large discrepancy between flood-d estimates and traceroute round-trip times. This is due to the fact that the target machine, *burton.cs.colorado.edu*, was probably overloaded at the time. Notice that towards the end of the measurement period, flood-d latency measurements became considerably lower.

Application-level estimates of the underlying physical topology capture both network and machine load. This means that distributed applications that are network cognizant are able not only to use the underlying network efficiently, but also to improve performance. For instance, suppose that the user interface of a resource discovery service is trying to decide to which directory service replica it should submit the user query, so that it gets a reasonable response time without incurring unnecessary network load. Using traceroute round-trip times will not help in this case, since the client also wants to avoid going to highly loaded servers.

The graphs in Figure 3.2 compare latency data collected at replicas running on 2 different internet sites: *aloe.cs.arizona.edu* at University of Arizona in Tucson, and *sunws0.cse.psu.edu* at PennState University. These replicas measure latency between themselves and to other group members, namely *burton.cs.colorado.edu*, *casino.cchs.su.edu.au*, *condesa.usc.edu*, and *warthog.usc.edu*, respectively. Data was collected every 30 minutes between 3 pm and 3 am, Friday, June 10 1994.

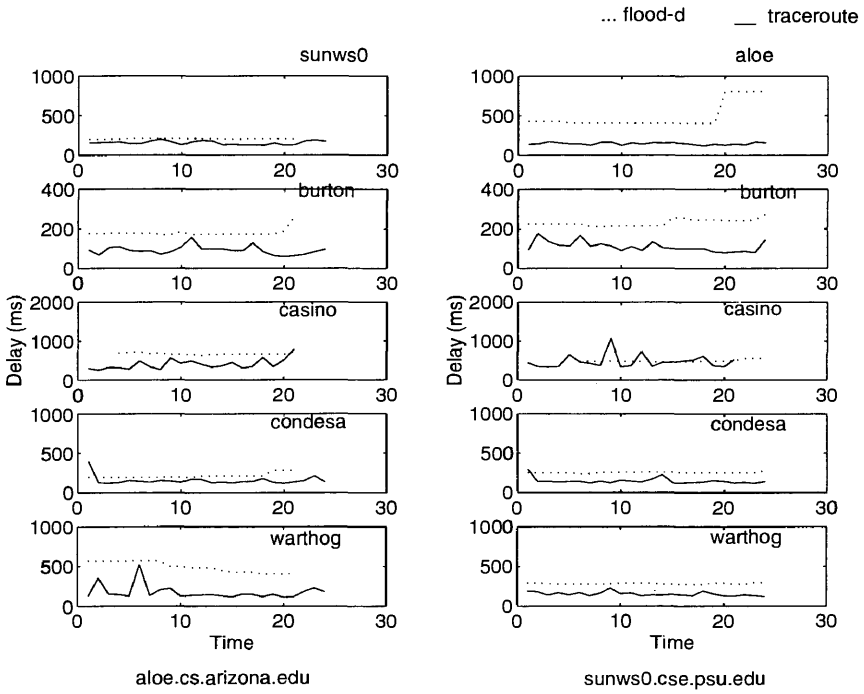

Figure 3.2: Flood-d's latency estimates and traceroute's round-trip time measurements.

The graphs in the first row show latency estimates from *aloe.cs.arizona.edu* to *sunws0.cse.psu.edu*, and vice-versa. Notice that although traceroute round-trip time values in both graphs are similar, the actual physical paths are asymmetric. The difference between flood-d's estimates in the 2 graphs shows logical link asymmetry: yet another phenomenon that could only be captured by end-to-end measurements.

The graphs in the 2 bottom-most rows show measurements to the 2 USC replicas. Since they are on the same local-area network, the traceroute round-trip times from both *aloe.cs.arizona.edu* and *sunws0.cse.psu.edu* to these 2 machines are similar. Notice that flood-d estimates to *warthog.usc.edu* were higher than to *condesa.usc.edu* probably because *warthog* had a bigger load at the time of the measurements.

Figure 3.3 shows latency measurements taken from *alameda.usc.edu* and *hidalgo.usc.edu*, both at USC, to the other members in the group, namely *burton.cs.colorado.edu*, *casino.cchs.su.edu.au*, and *sunws0.cse.psu.edu*. Data was collected every 30 minutes between 4:30 pm to 4:30 am, Thursday, June 16 1994.

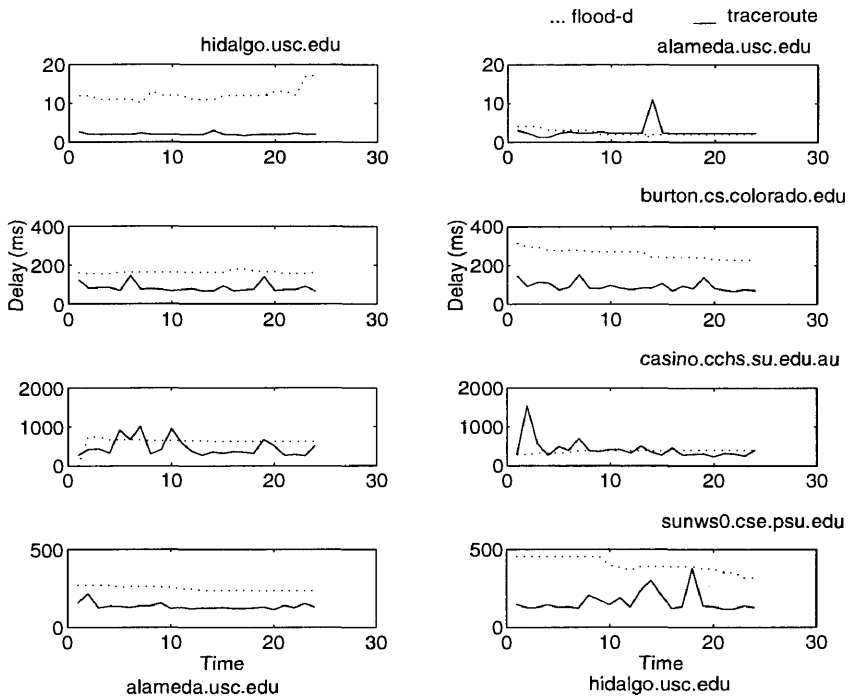

Figure 3.3: Flood-d's latency estimates and traceroute's round-trip time measurements.

The graphs in the first row plot flood-d estimates and traceroute round-trip times from *alameda* to *hidalgo*, and vice-versa. Notice that although the average traceroute round-trip times in both graphs are very similar, the latency estimates from flood-d are considerably different. This is because *hidalgo*, which is one of the machines that has the biggest memory in the USC Networking and Distributed Systems Laboratory, is usually running big simulations, and had the additional load of being the group master during this experiment.

Most of flood-d's estimates in the remaining graphs reflect *hidalgo*'s higher load; even though *hidalgo* and *alameda* are on the same local-area network, *hidalgo*'s latency estimates to a given machine are higher than *alameda*'s. This is due to the way latency measurements are being computed. Since the replica that originated the probing message timestamps it upon receipt, this timestamp accounts for the probe originator's load. An alternative is to have the probed replica issue the receipt timestamp, and then the originator computes the round-trip latency as 2 times the

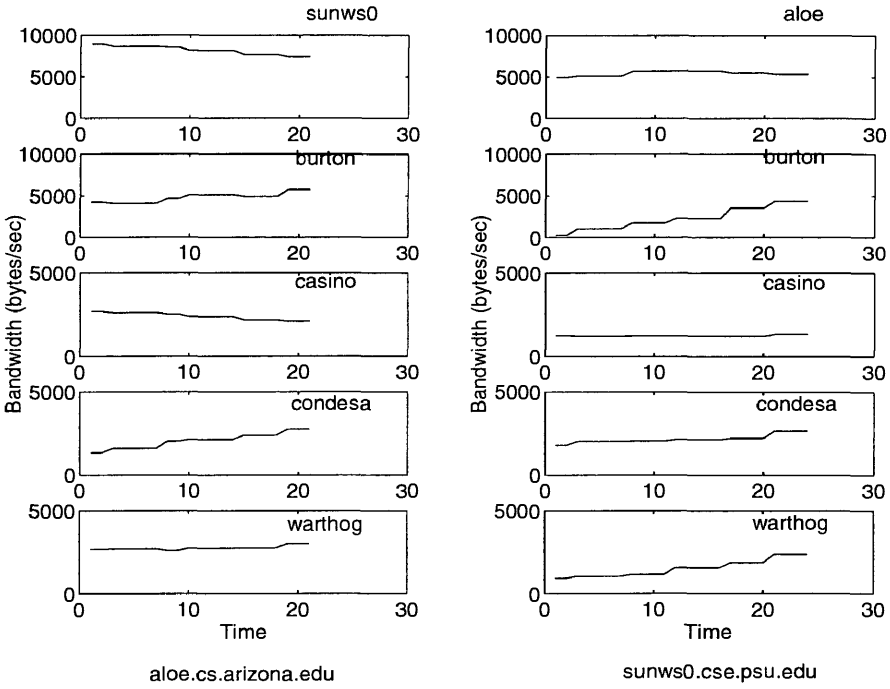

Figure 3.4: Flood-d's bandwidth estimates.

difference between the originator's and receiver's timestamps. The disadvantage of this approach is that we do not account for logical link asymmetry.

3.3 Bandwidth Estimates

Recall that besides measuring available bandwidth when updating neighbors, flood-d replicas periodically estimate bandwidth to members of their group. This way replicas also estimate bandwidth to non-neighbors and keep fresh bandwidth estimates even when there are no updates to propagate.

The first set of bandwidth estimates were collected by periodically sending fixed-size, dummy update messages among group replicas. Figure 3.4 shows measurements collected at the University of Arizona and PennState University replicas, namely *aloe.cs.arizona.edu*, and *sunws0.cse.psu.edu* to the other group members, *burton.cs.colorado.edu*, *casino.cchs.su.edu.au*, *condesa.usc.edu*, and *warthog.usc.edu*. Data was collected every 30 minutes between 3 pm and 3 am, Thursday, June 9 1994.

In all graphs, we observe the effects of slow-start, TCP's flow control mechanism, which gradually opens up the sender's transmission window as it receives acknowledgments from the receiver. However, because the periodic bandwidth probing messages are currently 32 KBytes long, in some cases they are too small to stretch the transmission window to its maximum. Take for example the bandwidth measured to *condesa.usc.edu* and *warthog.usc.edu* from both *aloe.cs.arizona.edu* and *sunws0.cse.psu.edu* (graphs in the fourth row): each probing message detects a higher bandwidth. This is not the case for the bandwidth measurements to *casino.cchs.su.edu.au*, the replica in Australia (graphs in the third row): messages from *aloe.cs.arizona.edu* detect a gradual decrease in 'available bandwidth, while the bandwidth on the logical link from *sunws0.cse.psu.edu* stays constant around 1 KBytes/sec through most part of the experiment.

The graphs in the first row show the asymmetry between the logical links connecting PennState to University of Arizona and vice versa. While the link from Arizona to PennState started with almost 10 KBytes/sec and gradually saturated, the bandwidth from PennState to Arizona stayed around 5 KBytes/sec.

The set of graphs in Figure 3.5 plot bandwidth estimates between 2 USC replicas, *alameda.usc.edu* and *hidalgo.usc.edu*, and from them to the other members of their group: *burton.cs.colorado.edu*, *casino.cchs.su.edu.au*, and *sunws0.cse.psu.edu*. Data was collected every 5 minutes between 12:30 pm and 10:30 pm, Wednesday, August 10 1994.

Notice that both *alameda* and *hidalgo* detect high bandwidths between themselves since they are connected to each other through a local-area network. The slightly lower bandwidth that *alameda* measures to *hidalgo* is due to *hidalgo*'s higher load during the experiments. Because *alameda* and *hidalgo* are located on the same local-area network, we observe that their bandwidth estimates to other group replicas are very similar.

The graphs in Figure 3.6 plot bandwidth estimates collected from replicas while files were being replicated. More specifically, we replicate 331 files in the gcc-2.5.8 software distribution with a total size of 17.5 MBytes (largest file is 430,000 bytes and the smallest 16 bytes). After the first hour of data collection, we started the file replicator on *hidalgo.usc.edu*. We collected measurements between *hidalgo.usc.edu* and *madero.usc.edu*, and from them to the other members of the group, namely,

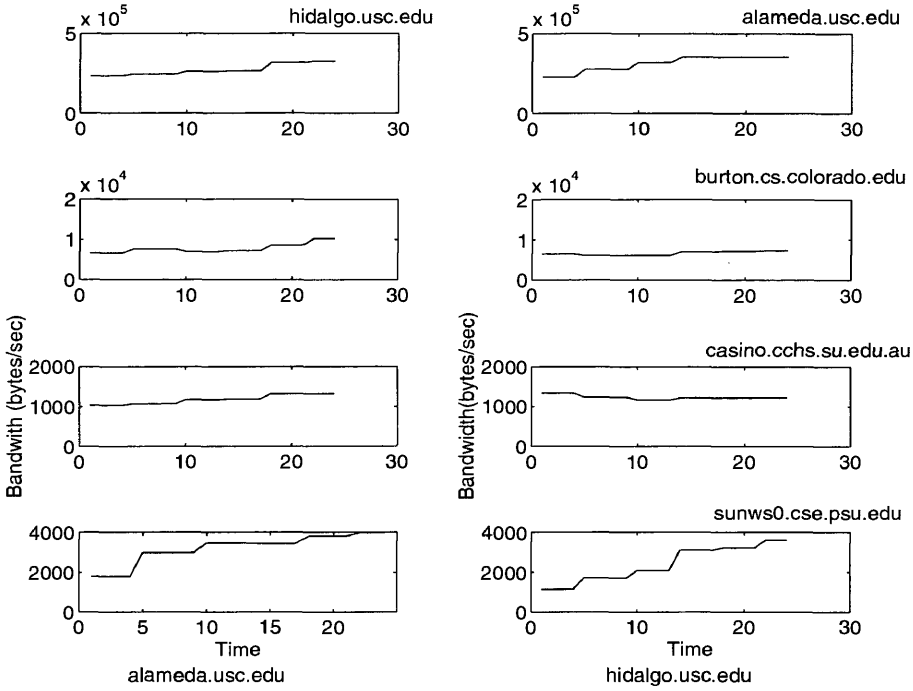

Figure 3.5: Flood-d's bandwidth estimates.

burton.cs.colorado.edu, *casino.cchs.su.edu.au*, and *sunws0.cse.psu.edu*. Data was collected every 30 minutes between 3:30 pm and 3:30 am, Thursday, June 16 1994.

When real data is being flooded, sites measure bandwidth when updating their neighbors according to the current logical topology. To measure bandwidth estimates to the other group members, replicas periodically send 32-KByte probing messages. Since we wanted to collect bandwidth estimates generated when propagating real data propagation, we forced the topology calculator to compute a fully-connected logical topology during this experiment. This way every replica tries to update every other replica in its group.

We observe that *hidalgo*'s bandwidth estimates to both *burton.cs.colorado.edu* and *sunws0.cse.psu.edu* are higher than *madero*'s. Since *hidalgo* is the one running the file replicator, it receives the data first, and updates the other group members. Flood-d's duplicate detection mechanism prevents *madero* from sending the same data to the other replicas. Therefore, *madero* estimates bandwidth by sending periodic 32-Kbyte messages. In fact, if we compare *madero*'s bandwidth estimates to

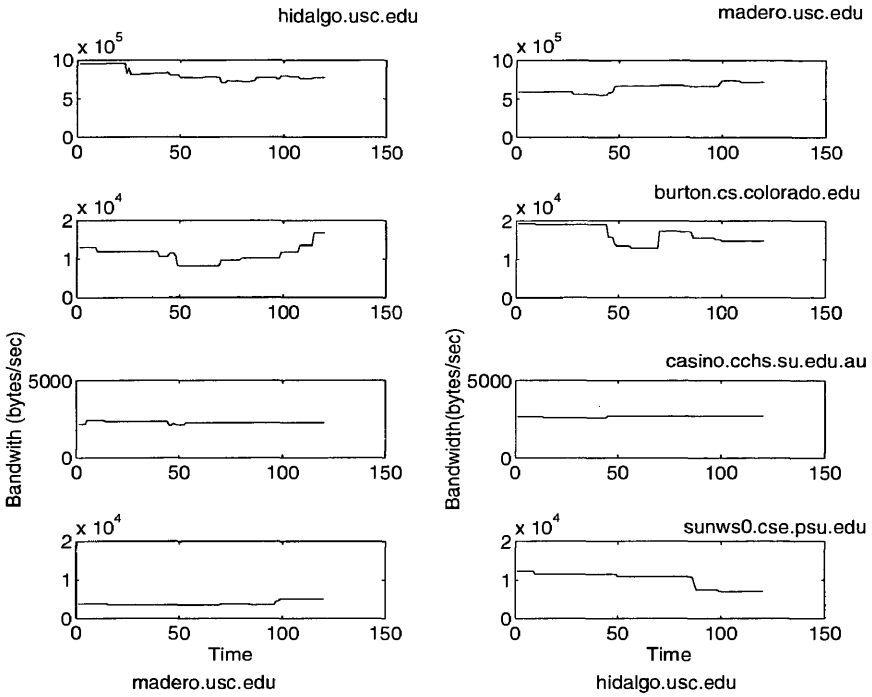

Figure 3.6: Flood-d's bandwidth estimates.

burton and *sunws0* in Figure 3.6 to the values in Figure 3.5, we notice that they are similar.

As in the previous bandwidth estimation experiments, we observe that *madero.usc.edu* and *hidalgo.usc.edu* detect high bandwidth between themselves since they are in the same local-area network. Notice that *madero*'s bandwidth estimates to *hidalgo* are higher than *hidalgo*'s estimates to *madero*. We would expect the opposite, since *hidalgo* is the one updating *madero*. However, because they are on a local-area network, and *madero* estimates bandwidth to *hidalgo* by sending periodic 32-KByte messages, these messages get to *hidalgo* almost instantaneously. Furthermore, since *hidalgo* is the group master and is updating everybody else, it has a high load, and probably saw the first and last bytes of the bandwidth probing message back-to-back. This means that the time interval between when it sees the first and the last bytes is very small, which results in the high bandwidth value *madero* detects.

Alternatively, we could have senders measure bandwidth. The sender timestamps the first and last bytes sent, and the receiver acknowledges when it receives them.

This way senders can tell whether high bandwidths resulted from receivers being overloaded.

Notice that *hidalgo*'s and *madero*'s estimates to *casino.cchs.su.edu.au* are very similar. This is due to physical bandwidth limitations on the connection between the continental US and Australia.

3.4 Summary

To compute logical topologies that use the underlying network efficiently, flood-d replicas probe the physical network and estimate communication latency and available bandwidth. In this chapter, we presented the results of the wide-area experiments we conducted to evaluate flood-d's network topology estimation strategy.

We showed that when compared to network-level monitoring tools such as traceroute, flood-d's estimates tend not to capture short-lived changes in network topology. We also showed that because they are performed at the application level, flood-d's estimates account for both network and machine load.

Like flood-d, distributed applications that are network cognizant are able not only to use the network efficiently, but also provide better response time, availability, and data convergence time. Based on the end-to-end information they have about the underlying network topology, network-cognizant applications can decide what is the best way to service a client's request, or the most efficient way of making their replicas converge to a consistent state. Because they periodically collect these end-to-end estimates, network-cognizant applications can adapt to changes in network and machine load.

Chapter 4

Logical Topologies

Using the estimated network topology, our architecture builds logical update topologies that are k-connected for resilience, use the physical network efficiently, and restrict update propagation delays.

We start this chapter by stating our topology computation problem as a graph theory problem. Then, we review existing solutions to similar problems, and describe in detail our topology calculation algorithm. We present the results obtained from running our algorithm using randomly generated topologies as input. At the end of the chapter, we present alternate approaches to computing our logical update topologies.

4.1 Definitions

Below, we provide the definitions [5] of the graph theory terms we use throughout the chapter.

- **k-Connected**: A graph G is said to be k-connected if no removal of any $k-1$ vertices together with all their incident edges disconnects G.
- **k-Connected Regular Graph**: A graph G is said to be a k-connected regular graph if all its vertices are exactly k-connected.
- **Diameter**: The diameter of a graph G is defined as the maximum shortest path between any two of G's vertices.
- **Degree**: The degree of a vertex v in G is the number of edges of G incident with v.

4.2 Statement of the Problem

We represent the underlying physical topology by a graph $G(V, E)$, where V is a set of vertices of G representing network nodes and E is a set of edges of G representing network links.

Our problem can be stated as follows. Given a graph $G(V, E)$ and a cost matrix with cost values for all the edges, construct a graph $G'(V, E')$ with the following properties.

- G' is k-connected.
- G' has minimum diameter.
- G' has minimum total edge cost.

This type of optimization problem is NP-complete, but the literature records several approximations for similar problems. Below, we review several of them, and describe our approximation algorithm.

4.3 Related Work

Plesnik [49] proves that any algorithm that generates a minimum spanning subgraph of G, say $G'(V, E')$, by selecting E' as subset of E with a given budget constraint and minimum diameter is NP-complete.

Johnson [30] states that constructing a subgraph which connects all vertices, and minimizes shortest path cost between all vertex pairs subject to a budget constraint on the sum of its edge costs is also NP-complete.

Schumacher [62] provides an algorithm for generating topologies which have minimum number of edges, are k-connected and have minimum diameter. However, his method assumes that all the edges have equal weights. We cannot make that assumption since our problem is to build logical topologies on top of real networks.

4.3.1 Steiglitz's Algorithm

Steiglitz et al. [65] propose a heuristic solution to a problem similar to ours. The problem consists of finding an undirected graph with the following properties.

- Feasibility : The redundancy between any two nodes i and j is at least $R_{i,j}$.
- Optimality: No network which satisfies the first property has lower cost.

When we map the above onto our problem, the redundancy matrix $R_{i,j}$, the number of disjoint paths between i and j, represents our connectivity requirement. Therefore, Steiglitz's algorithm [65] not only fulfills our connectivity requirements but also provides the option of having different connectivity for each pair of nodes. As stated above, trying to satisfy the redundancy and the minimum cost requirements results in NP-complete problems. Steiglitz's algorithm provides a heuristic method which leads to satisfactory low costs solutions.

Steiglitz's algorithm has two parts: the starting and the optimizing routines. The starting routine generates a random feasible solution. The optimizing routine iteratively applies heuristics to generate lower cost topologies. It uses a local transformation called *x-change*, which randomly selects four nodes connected pairwise and swaps the edges connecting them (see Figure 4.3). It then records the lowest cost feasible topology generated by these local transformations.

The algorithm uses hill climbing heuristics to generate local optimal solutions from different starting configurations. It terminates with a set of feasible solutions, from which it chooses the one with the lowest cost.

We extended Steiglitz's algorithm to fulfill all our topology requirements. Below, we describe our topology computation algorithm in detail.

4.4 Topology Calculation Algorithm

Steiglitz's redundancy matrix, $R_{i,j}$, can represent our topology's connectivity requirements. Notice that specifying the node connectivity imposes a lower bound on the degree of each node.

Instead of optimizing with hill climbing, we employed simulated annealing which has been successfully used to approximate solutions to the traveling-salesman problem. Simulated annealing is an analogy with thermodynamics annealing, in which metals arrive at a low energy state by slowly decreasing the temperature [32].

The Metropolis algorithm written in 1953 was one of the first to incorporate this concept into numerical calculations. It simulates a thermodynamic system which changes its configuration from energy state E_1 to energy state E_2 with probability given by

$$P(\triangle E) = exp(-\triangle E/kT) \tag{4.1}$$

where $\triangle E = E_2 - E_1$, T is the temperature, and k is a constant.

Notice that if $E_2 < E_1$, $P(\triangle E)$ is actually greater than unity, and is assigned the value of 1. This means that the algorithm always accepts a downhill configuration, and sometimes accepts an uphill one. Thus, one of the distinguished features of simulated annealing is that it can escape from local minima by accepting intermediate solutions that not always decrease the objective function.

In [54], Rose uses simulated annealing to find network topologies with small mean distances between nodes, and shows that annealing helps in equalizing the initially uneven distribution of mean distances.

When applying simulated annealing to our problem, we need to specify an objective function. We present the objective functions we investigated in Section 4.4.3.1.

4.4.1 Generating a Starting Topology

We feed our logical topology calculator with the participating nodes' redundancy requirements and the estimated communication cost matrix. Notice that this cost matrix contains estimates of the underlying physical topology, and is not necessarily fully connected. To obtain a fully-connected cost matrix, we compute Dijkstra's all-pairs shortest-paths [14] over the estimated cost matrix, and generate the costs for the logical links connecting every node pair.

Using the fully-connected, logical-link cost matrix, the algorithm starts by generating random topologies until it finds one that is *feasible*. Recall that a topology is *feasible* if it satisfies the specified connectivity requirement. Figure 4.1 presents the algorithm for generating a feasible starting topology.

Because the nodes' redundancy requirements are typically uniform, most of the randomness in generating a starting solution comes from the initial labeling of the nodes. Thus, whenever it needs another initial topology, the algorithm re-labels the nodes. For a topology with n nodes, there are $n * (n-1) * (n-2) * ... * 1 = n!$ possible starting configurations.

Topologies whose connectivity requirement is k are considered feasible if there are at least k disjoint paths between every pair of nodes. However, one of the theorems stated in [65] demonstrates that to check topology feasibility, it is not necessary to calculate the redundancy between all possible node pairs. In particular, when all nodes have the same redundancy requirement k, then we only need to perform the following redundancy checks: between any node m_1 and all remaining nodes; then, between any node m_2, different than m_1, and the remaining nodes, except m_1; and, so on up to node m_k. Therefore, instead of having to perform $n(n-1)/2$ feasibility checks, whose time complexity is $O(n^2)$, we only need

$$(n-1) + (n-2) + ... + (n-k) = kn - k(k+1)/2 \tag{4.2}$$

where n is the number of nodes, and k the redundancy requirement. Notice that since $n >> k$, and k is a constant [1], we need to perform $O(n)$ feasibility checks.

[1] In practice, topologies typically are either 2- or 3-connected.

```
feasible_start(redundancy matrix, cost matrix)
{
  do {
     randomly label all nodes.
     do {
        using the labeling order,
          pick first node with highest redundancy
                                        requirement (node A).
          decrement its redundancy requirement by 1.
        pick another node with highest redundancy
                                        requirement (node B).
          if there is more than 1 node with highest redundancy
          then pick the one that has the cheapest link to A.
            if there are 2 or more nodes with
                                          same link cost to A
            then pick the one with the lowest label.
          decrement the selected node's redundancy
                                         requirement by 1.
      } until all nodes have redundancy requirement = 0.
      feasible = check_feasibility().
    } until feasible.
}
```

Figure 4.1: Generating a feasible starting topology.

The feasibility check algorithm, which we present in Figure 4.2, employs the Ford-Fulkerson method for solving the maximum flow problem in flow networks [14], which produces the number of disjoint paths between the flow network's source and sink when all edges have flow capacity of 1.

After generating a feasible starting topology, our algorithm computes the starting topology's cost by calculating its total edge cost and diameter. The total edge cost computation takes time $O(e)$, where e is the number of edges. To calculate the topology's diameter, we use Dijkstra's all-pairs shortest-path algorithm [14]. Our current implementation of Dijkstra's all-pairs shortest-path algorithm uses a binary heap to implement the priority queue the algorithm uses to keep all vertices whose

```
int check_feasibility(topology)
{
  for nodes A_i, 1 <= i <= k {
    for every other node B {
      if ( disjoint_paths(A_i, B) < k )
      then return(FALSE).
    }
  }
  return(TRUE).
}

disjoint_paths(G, s, t)
{
  initialize residual network R to graph G.

  while there exists a flow f from s to t in R {
    extract f from R.
    paths++.
  }
  return paths.
}
```

Figure 4.2: Checking topology feasibility.

shortest paths have not yet been computed. The time complexity of the resulting algorithm is $O(ne \log n)$.

4.4.2 Applying Local Transformations

The next step is to apply transformations to the initial configuration. These transformations consist of some combination of the basic add- and delete-edge operations. For instance, one of the transformations we use is Steiglitz's *x-change* operation. It randomly selects a pair of connected nodes **a** and **b**. Then it selects another pair of connected nodes **c** and **d**, such that **c** is not connected to **a**, and **d** is not connected to **b**, or vice-versa. *X-change* then deletes edges **a-b** and **c-d**, and adds edges **a-c** and **b-d** (see Figure 4.3). The *x-change* operation has the property of not changing the degree of any node.

We also use the *split*, *delete*, and *add* transformations. *Split* randomly selects a pair of connected nodes, breaks the edge connecting them and connects each of them to another node. *Delete* randomly chooses a pair of connected nodes with degree greater than the required connectivity and deletes the edge connecting them. *Add* selects a pair of nodes that are not connected and adds an edge connecting them. Figure 4.3 illustrates these transformations.

Each transformation is assigned some probability of being selected as the next operation the topology calculation algorithm will apply to the current topology. One of our experiments consisted of using different combinations of these probabilities. The results of these experiments are presented in Section 4.5.

After each transformation, the resulting topology is checked for feasibility. If the feasibility check fails, the algorithm rejects the resulting topology, and restores the previous one.

Steiglitz [65] proves that in the case that the connectivity requirement is the same for all nodes, after a *x-change* operation, the resulting topology's feasibility can be checked just by verifying whether the connectivity between the 2 pair of nodes that got disconnected is still greater than or equal to the connectivity requirement. Therefore, the feasibility check after a *x-change* just needs to compute the number of disjoint paths between the 2 node pairs involved in the *x-change* operation.

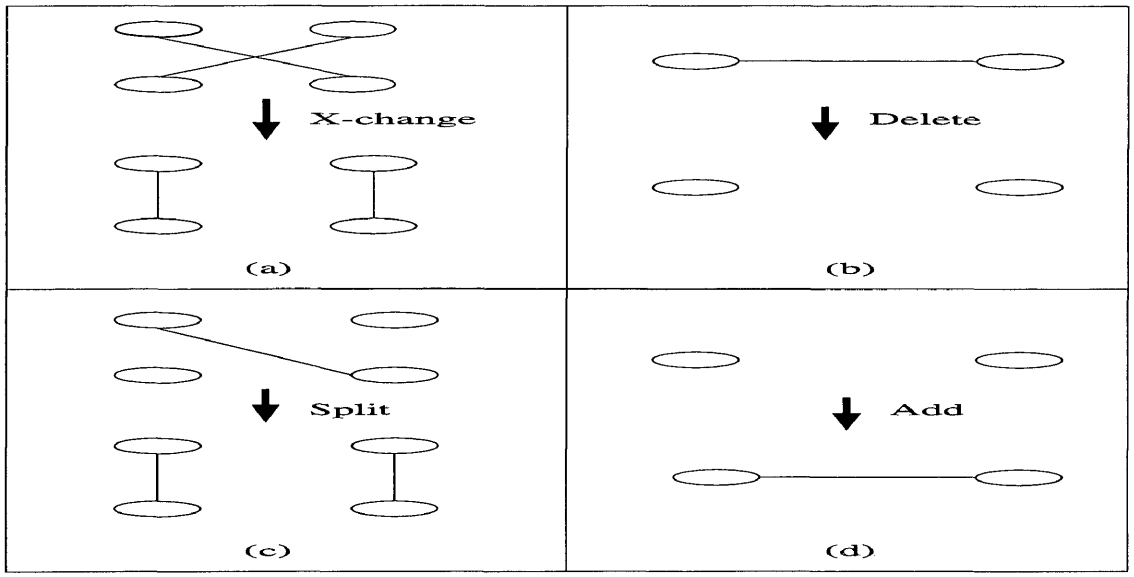

Figure 4.3: Possible Transformations.

The *split* operation deletes 1 edge, and then adds 2 new ones. Steiglitz's theorem reduces the feasibility test after a *split* to the pair of nodes that had their connecting edge removed. No feasibility check is needed after *add* since it does not remove any edges, while a full check (the same that is performed when generating a starting configuration) needs to be executed after a *delete*.

4.4.3 Annealing

The annealing schedule decides whether a feasible topology that is generated after each transformation should be accepted or not. It checks whether the new configuration improves cost. In this case, it accepts the new configuration, and goes back to the transformation step. If cost increases, the new configuration is accepted or rejected according to the Boltzmann probability distribution given by expression 4.1 in Section 4.4, where $\triangle E$ is the difference in cost between the new and old topologies.

Figure 4.4 shows our annealing algorithm. Analogously to thermodynamics annealing, we decrease the temperature T at a specified rate D after a certain number

of transformations have been applied. We discuss the tuning of these annealing parameters in Section 4.4.3.2.

4.4.3.1 Objective Functions

Recall that our goal is to generate logical topologies that try to use the underlying physical network efficiently and, at the same time, reduce the time to propagate updates. In other words, we want logical topologies with low total edge cost and small diameter. Thus, one of our objective functions combines both edge cost and diameter. In this case, we compute the difference $\triangle E$ as

$$\triangle E = \frac{edge\_cost_{new} - edge\_cost_{old}}{edge\_cost_{old}} + \frac{diameter_{new} - diameter_{old}}{diameter_{old}}$$

where $edge\_cost_{old}$, $edge\_cost_{new}$, $diameter_{old}$ and $diameter_{new}$ are, respectively, the total edge cost and diameter of the topologies before and after a transformation is applied. We use $edge\_cost_{old}$ and $diameter_{old}$ as normalization factors for the edge cost and diameter, respectively.

Clearly, the choice of the objective function depends on the requirements of the problem being solved. If we just want to minimize the maximum number of links an update needs to travel when being propagated to all group members, then in the objective function, we compute diameter as the maximum number of hops between any pair of nodes, instead of the maximum edge cost sum.

Now suppose our problem is to find a k-connected topology that minimizes the amount of wiring needed to connect nodes in a network. In this case, the corresponding objective function just needs to take into account edge cost. This means that after a transformation, if the new topology's total edge cost is less than the old topology's total edge cost, the new topology is accepted. Thus, $\triangle E$ is computed as

$$\triangle E = edge\_cost_{new} - edge\_cost_{old}$$

On the other hand, if we just want to minimize the delay in propagating updates, then the objective function should only depend on diameter. In this case $\triangle E$ is computed as

```
anneal()
{
  for i=1 to ITERATIONS {
    old_cost = current_cost.
    select local_transformation.
    new_topology = local_transformation(current_topology).
    feasible = check_feasibility(new_topology)
    if ( !feasible ) continue.
    current_cost = cost(new_topology).
    delta_cost = current_cost - old_cost.
    p = exp(-obj_function(delta_cost)/temperature).
    r = random(0,1).
    if (r < p) {
      accept new_topology.
      success++.
    }
    else
      undo transformation.
    operation++.
    if ( (success == SUCCESS_OPS) or (operations == TOTAL_OPS) ) {
      temperature = temperature * (1-decrease_rate).
      success = 0.
      operations = 0.
    }
  }
}
```

Figure 4.4: The annealing algorithm.

$$\triangle E = diameter_{new} - diameter_{old}$$

We should point out that diameter reduction is usually achieved with the addition of edges, which usually increases edge cost. Conversely, reducing edge cost by deleting extra edges [2] tends to increase diameter.

We have experimented with all previously described objective functions and we present the results in Section 4.5.

4.4.3.2 Annealing Parameters

According to expression 4.1, the probability of accepting a new topology is a function of $\triangle E$, the difference in cost between the new and old topologies, and the current temperature T. Notice that the higher the temperature, the higher the probability of accepting a new topology in case its cost is greater than the old topology's cost. As we lower T, the probability of accepting a higher cost topology decreases.

The decrease rate D determines how fast T decreases. The lower D, the slower T decreases, and the more higher-cost topologies get accepted. Lower decrease rates also mean that the annealing process takes longer to get to a low-energy state.

We studied how T's initial value and the decrease rate D affect the annealing algorithm. Section 4.5 below starts by presenting the results of running our algorithm with different initial values of T and D.

4.5 Results

In this section, we present the results of the several experiments we conducted with our topology calculation algorithm. Recall that besides the connectivity requirement, we need to feed our algorithm with the cost matrix resulting from estimating the underlying physical topology. For our simulations, we use NTG [26] to generate random physical topologies. Using number of nodes, average node degree, and link speed as parameters, NTG places nodes in a plane and randomly connects pairs

[2] Extra edges are the ones that can be deleted without making the resulting topology infeasible.

of vertices. The communication cost between each directly-connected pair of nodes is a function of the link bandwidth and the physical distance between the nodes. Section 4.6 discusses the impact of the randomly-generated link cost distributions in our logical topology calculation algorithm.

4.5.1 Setting the Annealing Parameters

We conducted some preliminary experiments to select the initial value of T. For each objective function, we ran the algorithm to determine the range of $\triangle E$ values resulting after each transformation. Using the highest $\triangle E$ found, we set T's initial value, T_0, so that the probability of accepting higher cost topologies starts at 50%.

To study the effects of T's initial value on the annealing process, we ran our algorithm for T_0, $T_0/10$ and $T_0 * 10$. Figure 4.5 plots edge cost and diameter values for the different values of T_0. These runs used edge cost and diameter as objective function to generate 2-connected topologies.

We observe that the higher the temperature, the higher the bump in the edge cost curve, since at high temperatures, the probability of accepting higher cost topologies increases. We should also point out that because our initial random topologies already have low edge cost, it is hard to eliminate edges without making the resulting topology infeasible. Therefore, in the beginning of the annealing process, most operations add edges, which reduces diameter [3], but increases edge cost. Also notice that higher temperatures allow more oscillation in diameter.

According to Figure 4.6, generating 3-connected graphs presents similar behavior: higher initial temperatures allow higher total edge cost topologies to be accepted and consequently, more oscillation in diameter in edge cost. We also observe that since we are generating 3-connected graphs, the resulting topologies have more edges, which explains why the total edge cost values are higher, and both diameter in edge cost and hop count are lower than in the 2-connected graphs.

Figures 4.7 and 4.8 show the results of running the same set of experiments using only edge cost as the objective function. Although it is controlled by total edge cost only, the topology calculator initially accepts topologies with higher total edge cost.

[3]The path that corresponds to the diameter in edge cost may not be the same as the one corresponding to diameter in hop count.

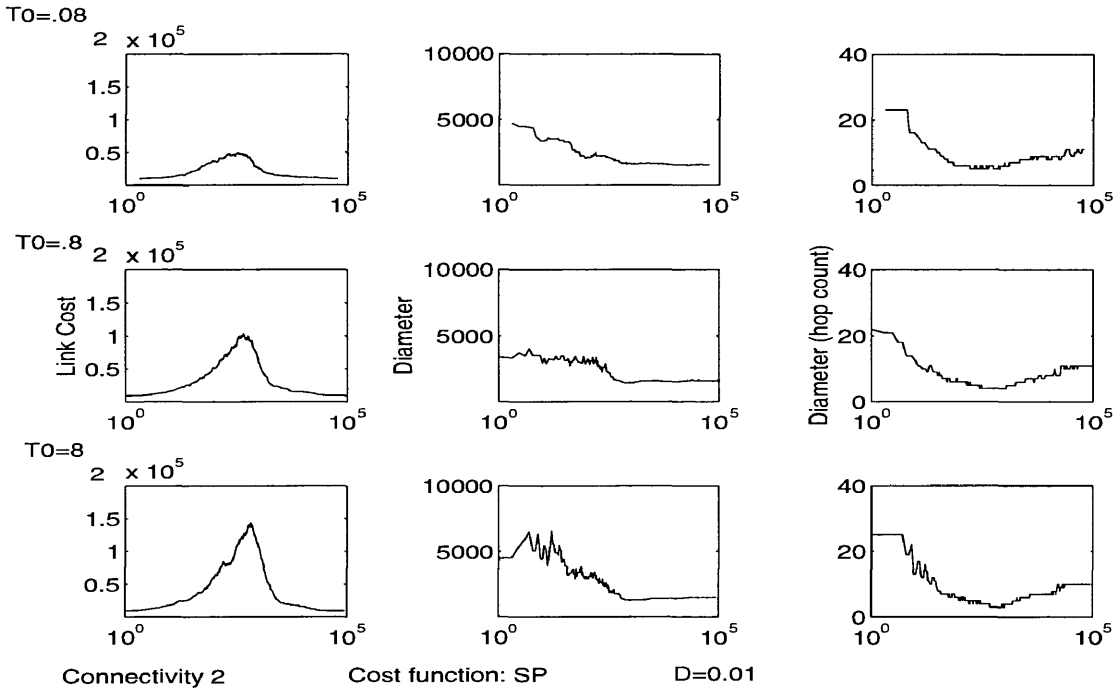

Figure 4.5: The effects of the initial temperature T_0 in the annealing process. For $T_0 = .08$, $T_0 = .8$, and $T_0 = 8$, we plot total edge cost, diameter, and diameter in hop count when generating 2-connected graphs using the combination of edge cost and diameter as objective function.

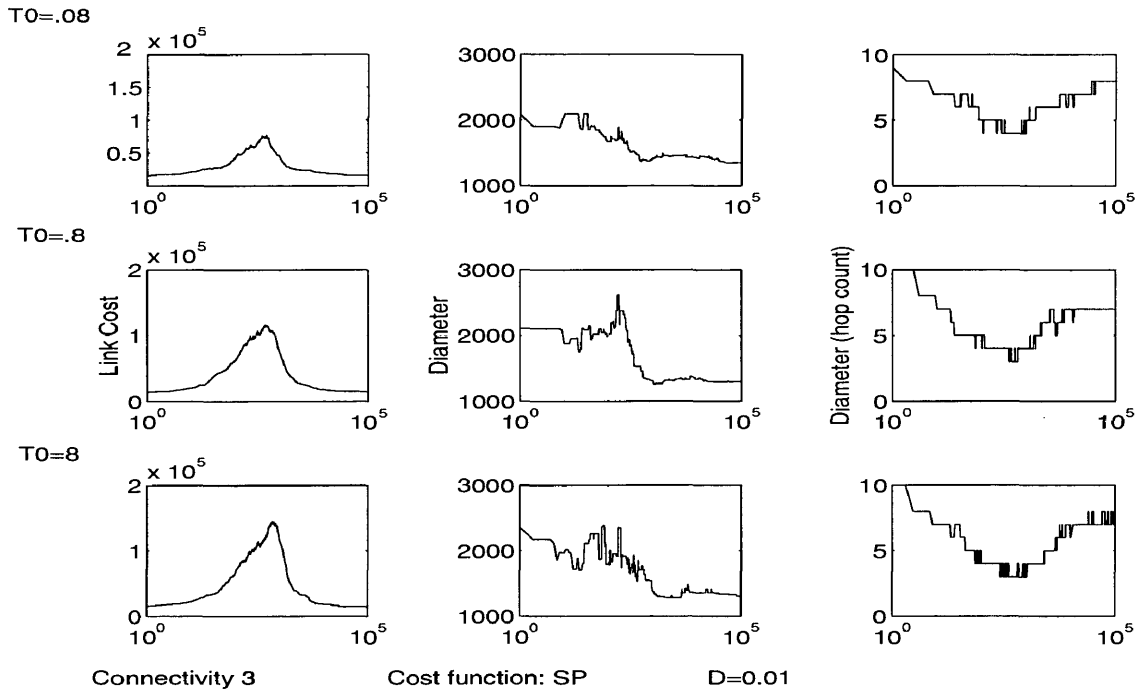

Figure 4.6: The effects of the initial temperature T_0 in the annealing process. For $T_0 = .08$, $T_0 = .8$, and $T_0 = 8$, we plot total edge cost, diameter, and diameter in hop count when generating 3-connected graphs using the combination of edge cost and diameter as objective function.

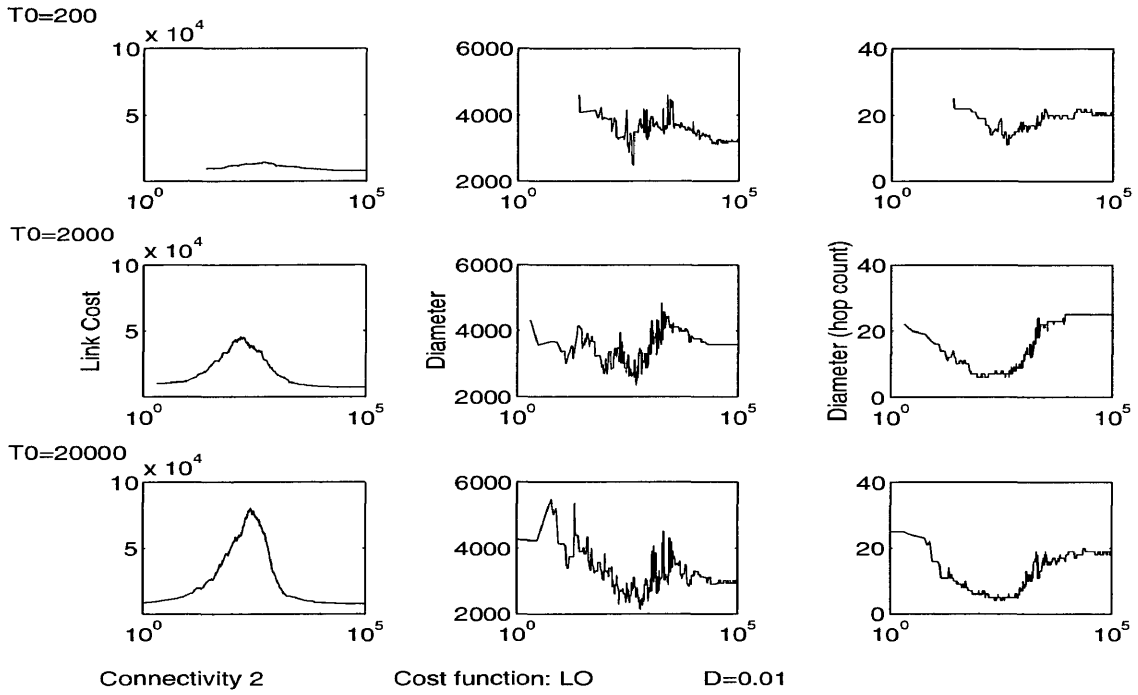

Figure 4.7: The effects of the initial temperature T_0 in the annealing process. For $T_0 = .08$, $T_0 = .8$, and $T_0 = 8$, we plot total edge cost, diameter, and diameter in hop count when generating 2-connected graphs using only edge cost as objective function.

Again, this is because the random topologies the annealing process uses as input already have low edge cost. We notice however that edge cost values do not go as high as those in Figures 4.5 and 4.6. Since the cost function does not control diameter, we notice higher fluctuations in diameter if compared to Figures 4.5 and 4.6.

Besides the fact that the bump in the edge cost curve increases as T_0 increases, we also observe that the lower the initial temperature, the lower the probability of accepting higher-cost topologies. This explains the gaps in the first and second-row graphs, and the fact that the gaps in the first-row graphs are bigger than in the second-row graphs.

Again, we notice that in the 3-connected topologies the edge costs are higher than in the 2-connected topologies, but diameter values are lower.

We also studied how the temperature decrease rate D affects the annealing process. Figure 4.9 shows edge cost and diameter curves for different decrease rates

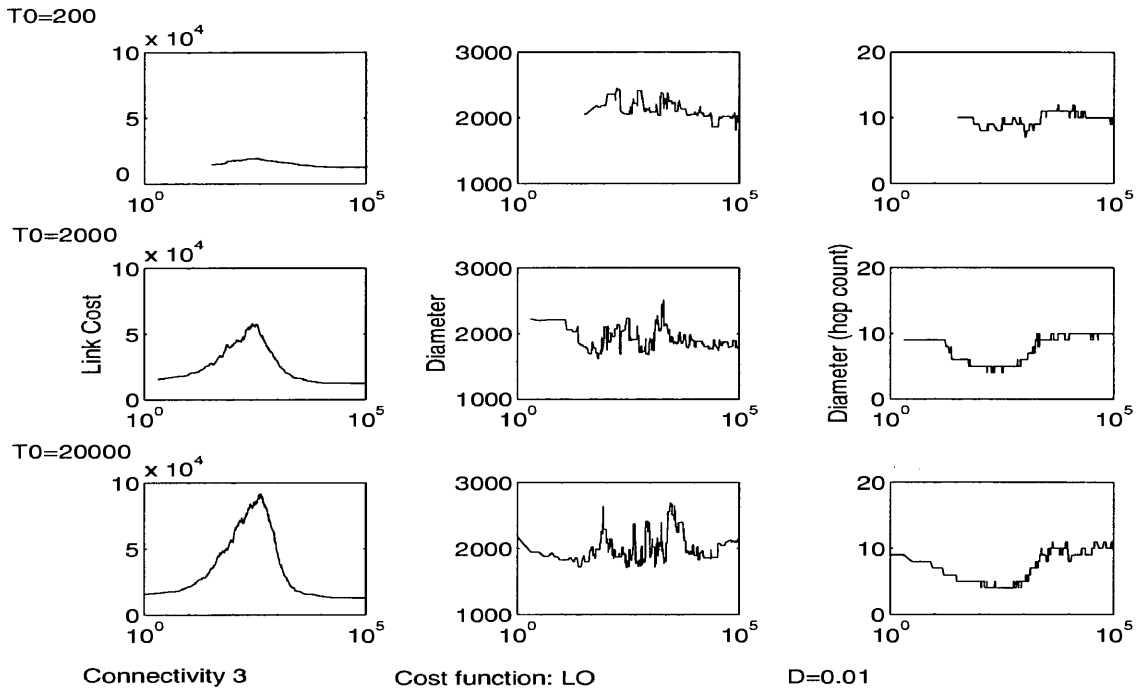

Figure 4.8: The effects of initial temperature T in the annealing process. For $T = .08$, $T = .8$, and $T = 8$, we plot total edge cost, diameter, and diameter in hop count when generating 3-connected graphs using only edge cost as objective function.

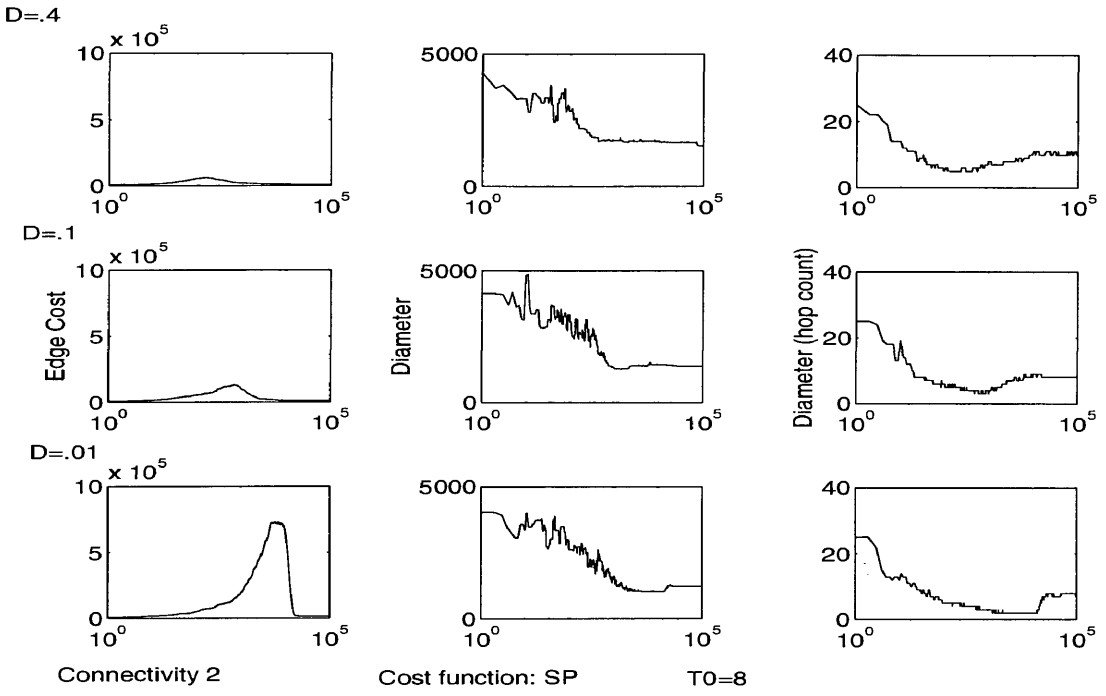

Figure 4.9: The effects of temperature decrease rate D in the annealing process. For $D = .4$, $D = .1$, and $D = .01$, we plot total edge cost, diameter, and diameter in hop count when generating 2-connected graphs using the combination of edge cost and diameter as objective function.

using edge cost and diameter as objective function. Lower decrease rates make the temperature decrease slower, which allows higher-cost topologies to get accepted. This explains why the edge cost curves have higher bumps that get shifted to the right as D decreases. Notice that lower decrease rates also cause more diameter fluctuation. However, we observe that on average [4], lower decrease rates result in topologies with lower diameter and consequently, higher edge cost.

We also conducted the temperature decrease rate experiment using edge cost as the objective function, and show the results in Figure 4.10. The edge cost curves are similar to those in Figure 4.9, but the edge cost objective function does not allow edge costs to go as high. Consequently, diameter values are higher than the ones in Figure 4.10. In fact, as expected, when using only edge cost in the objective

[4]We conducted 3 runs for each value of D.

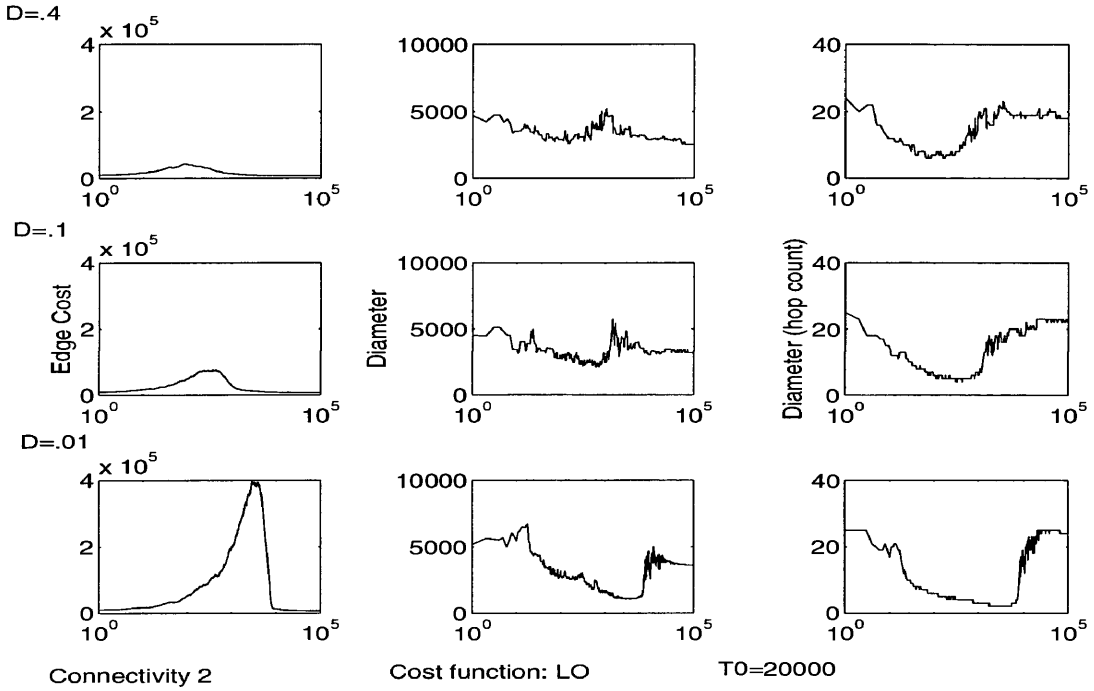

Figure 4.10: The effects of temperature decrease rate D in the annealing process. For $D = .4$, $D = .1$, and $D = .01$, we plot total edge cost, diameter, and diameter in hop count when generating 2-connected graphs using only edge cost as objective function.

---

function, we get topologies with lower total edge cost and higher diameter than the ones obtained when using both edge cost and diameter in the objective function.

4.5.2 Objective Functions and Transformation Probabilities

Besides experimenting with different objective functions, we also varied the probability with which the annealing process selects among the *add*, *delete*, *x-change*, and *split* transformations. For these experiments, we fixed the initial topology the annealing process uses, so that we can compare the resulting topologies' costs.

Figure 4.11 and Table 4.1 show the results of running the topology computation algorithm using edge cost and diameter in the objective function to generate 2-connected graphs. Besides different transformation probabilities, we also used different decrease rates.

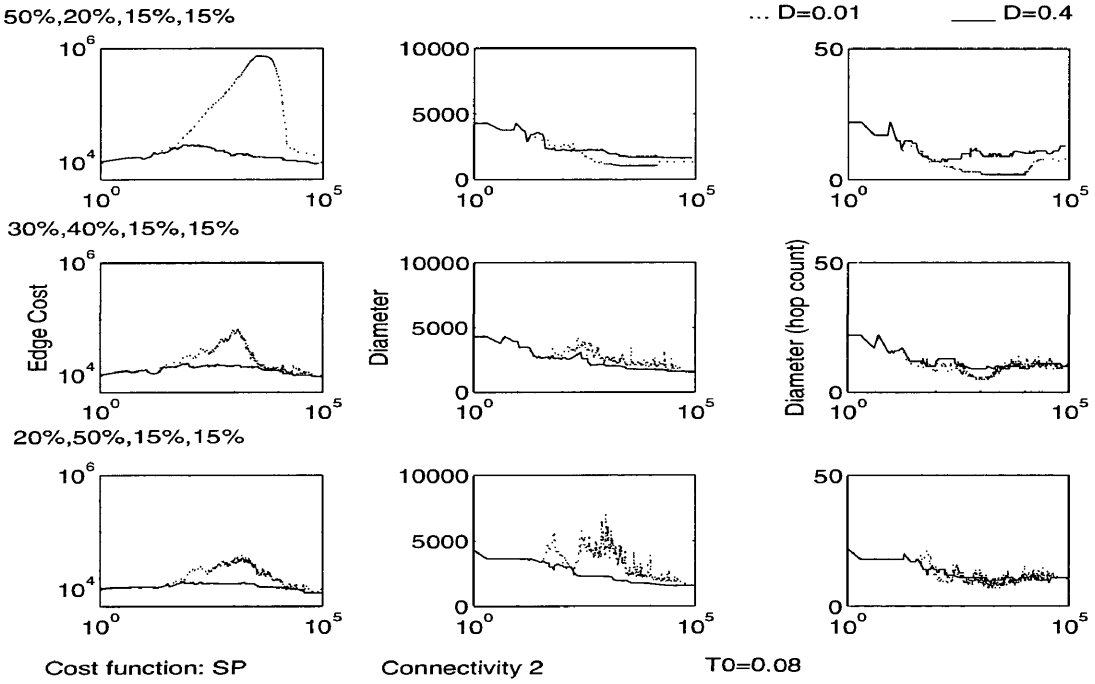

Figure 4.11: The effects of different transformation probabilities in the annealing process. We plot total edge cost (log scale), diameter, and diameter in hop count using different transformation probabilities for $D = 0.01$ and $D = 0.4$. T_0 is set at 0.08, and as cost function, we use total edge cost and diameter in edge cost.

Probabilities (%)	D	Total Edge Cost		Diameter in Edge Cost		Diameter in Hop Count	
Add, Del, Swp, Spt		BA	AA	BA	AA	BA	AA
50, 20, 15, 15	0.01	9517	13158	4755	1336	25	7
	0.4	9517	9495	4755	1664	25	13
30, 40, 15, 15	0.01	9517	9322	4755	1578	25	11
	0.4	9517	9398	4755	1572	25	10
20, 50, 15, 15	0.01	9517	8872	4755	1594	25	11
	0.4	9517	8660	4755	1580	25	11

Table 4.1: Logical topology costs before (BA) and after (AA) annealing. The initial temperature is set at 0.08, and as cost function, we use total edge cost and diameter in edge cost.

Because the initial topology already has low total edge cost, we observe that with total edge cost and diameter in the objective function, the annealing process cannot improve the total edge cost considerably, but produces topologies with diameter up to 3.5 times lower. As expected, the values in Table 4.1 show that as the probability of *delete* increases and the probability of *add* decreases, we get lower total edge cost topologies with higher diameter [5].

For the highest *delete* probability, $D = 0.4$ generates a lower total edge cost topology with lower diameter than the one generated for $D = 0.01$. This is because the higher decrease rate causes the probability of accepting higher cost topologies to decrease faster. In fact, the graphs in Figure 4.11 show that the lower decrease rate, $D = 0.01$, generates more cost oscillation during the annealing process.

We should point out that even though we report the total edge cost for a given logical topology, it does not mean that this is the cost flood-d incurs when propagating updates over that logical topology. Flood-d's duplicate detection mechanism avoids sending updates to replicas that already received them. Similarly to internet multicast, flood-d ends up propagating updates according to a spanning tree that tries to minimize cost. The same way as some adaptive routing mechanisms keep alternate routes, flood-d keeps alternate logical paths for resilience.

[5]Notice that since all starting topologies are rings, they have 25 hops as their hop-count diameter.

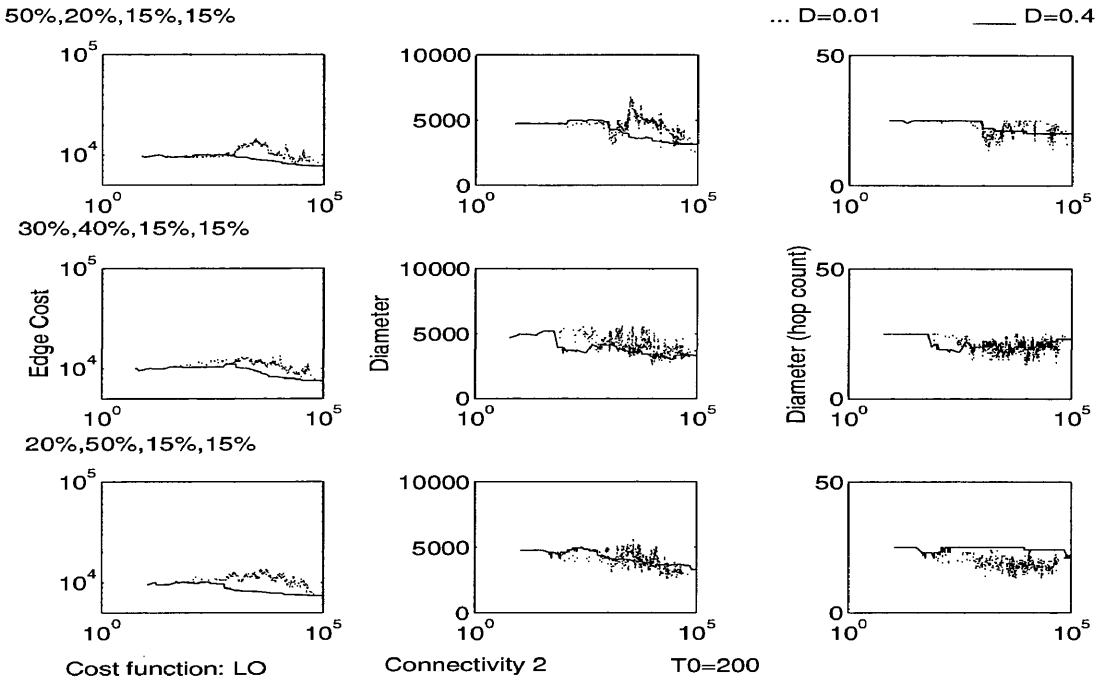

Figure 4.12: The effects of different transformation probabilities in the annealing process. We plot total edge cost (log scale), diameter, and diameter in hop count using different transformation probabilities for $D = 0.01$ and $D = 0.4$. T_0 is set at 200, and we use total edge cost as objective function.

In the next set of simulations, we use only total edge cost in the objective function. Figure 4.12 plots the cost curves corresponding to the intermediate topologies our algorithm generates, and Table 4.2 shows the starting and final topologies' costs.

Since the annealing process tries to optimize the total edge cost only, it favors the runs with higher *delete* probability. In fact, according to Table 4.2, when the *delete* probability is 50%, the resulting topology's total edge cost is approximately 25% lower than the initial topology's total edge cost. Notice that we also get a reduction in diameter. The highest diameter reduction is approximately 50% and as expected, happens when the *add* probability is 50% and the delete probability is only 20%. However, we should point out that except for the cases where the *add* probability is 50%, the diameter in hop count does not go down significantly. This means that annealing generated topologies with almost the same number of edges than the initial topology, but the selected edges have lower cost.

Probabilities (%)	D	Total Edge Cost		Diameter in Edge Cost		Diameter in Hop Count	
Add, Del, Swp, Spt		BA	AA	BA	AA	BA	AA
50, 20, 15, 15	0.01	9517	7980	4755	2540	25	17
	0.4	9517	7736	4755	3144	25	20
30, 40, 15, 15	0.01	9517	7528	4755	3683	25	25
	0.4	9517	7527	4755	3323	25	23
20, 50, 15, 15	0.01	9517	7464	4755	2944	25	21
	0.4	9517	7418	4755	3289	25	21

Table 4.2: Logical topology costs before (BA) and after (AA) annealing. The initial temperature is set at 200, and we use total edge cost as objective function.

Probabilities (%)	D	Total Edge Cost		Diameter in Edge Cost		Diameter in Hop Count	
Add, Del, Swp, Spt		BA	AA	BA	AA	BA	AA
50, 20, 15, 15	0.01	9517	111364	4755	1364	25	2
	0.4	9517	13378	4755	1953	25	7
30, 40, 15, 15	0.01	9517	12303	4755	2156	25	8
	0.4	9517	11783	4755	2307	25	8
20, 50, 15, 15	0.01	9517	11254	4755	2088	25	8
	0.4	9517	11155	4755	1973	25	8

Table 4.3: Logical topology costs before (BA) and after (AA) annealing. The initial temperature is set at 0.08, and we use total edge cost and diameter in hop count as objective function.

From the graphs in Figure 4.12, we observe that the lower decrease rate, $D = 0.01$, generates more cost oscillation than $D = 0.4$. Again, this is due to the fact that the lower D causes the temperature to decrease slower, and consequently allows more higher-cost topologies to be accepted by the annealing process.

The last cost function with which we experimented combines total edge cost and diameter in hop count. For networks where physical link costs are roughly uniform, it makes more sense to improve diameter in hop count than diameter in edge cost. Figure 4.13 plots the intermediate topologies' cost curves, while Table 4.3 shows the initial and final topologies' costs.

From the results in Table 4.3, we observe that all of the resulting topologies have higher total edge costs, but show considerable reductions in diameter. However, with 50% as the probability of adding edges, we ended up with a very high edge

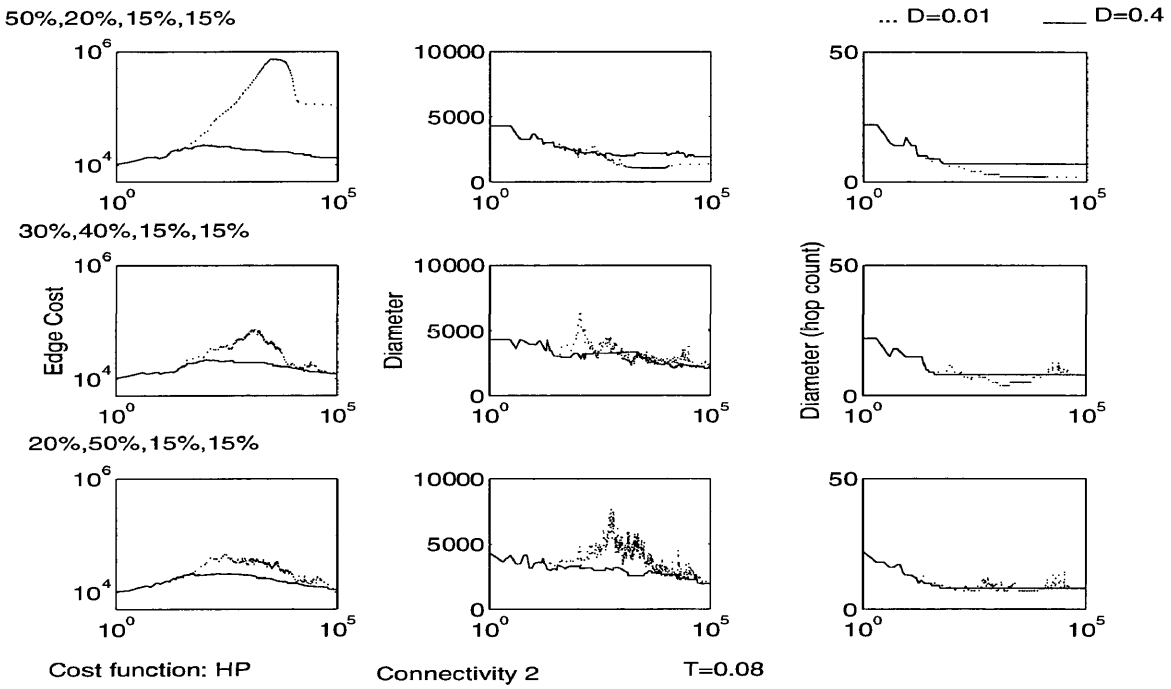

Figure 4.13: The effects of different transformation probabilities in the annealing process. We plot total edge cost (log scale), diameter, and diameter in hop count using different transformation probabilities for $D = 0.01$ and $D = 0.4$. T_0 is set at 0.08, and we use total edge cost and diameter in hop count as objective function.

cost topology which has very low diameter [6]. While this is not a practical solution, it gives us a sense of edge cost upper bounds [7]. Notice that the resulting topologies present lower diameter in hop count than the topologies obtained when diameter in edge cost is used in the objective function (Table 4.1).

For the same transformation probability combination, the 0.4 decrease rate generates a more reasonable topology. As the curves in Figure 4.13 show, the lower decrease rate does not allow total edge costs to go very high, since the probability of accepting higher-cost topologies decrease faster. We also observe that the discrepancy between the different decrease rates decreases as the *add* probability decreases.

4.5.3 Other Approaches

In this section, we examine other, more practical, approaches to solving the logical topology graph problem. The first alternative consists of applying our annealing schedule to a fully connected topology. The other approaches consist of using heuristics to select edges to be added to or deleted from a given topology. Below, we explain these approaches further and report their results.

4.5.3.1 Fully-Connected Initial Topology

Since fully connected topologies have minimum diameter and maximum total edge cost, in this experiment we use only total edge cost in the objective function. Table 4.4 shows the costs before and after annealing. Notice that since we start with a fully-connected topology, the total edge cost is maximum and the diameter is minimum.

If we compare the results of these experiments with the values in Table 4.2, which used the same objective function but started with a randomly generated, feasible topology, we notice that for the higher probability of adding edges, the

[6] The minimum diameter is 1 hop, which happens when the topology is fully connected. The minimum diameter in edge cost is 1048.

[7] In fact, the fully-connected topology corresponding to this link cost distribution has total edge cost of 740,762.

Probabilities (%)	D	Total Edge Cost		Diameter in Edge Cost		Diameter in Hop Count	
Add, Del, Swp, Spt		BA	AA	BA	AA	BA	AA
50, 20, 15, 15	0.01	740,762	7343	1048	3489	1	23
	0.4	740,762	7561	1048	3270	1	21
30, 40, 15, 15	0.01	740,762	7736	1048	3423	1	23
	0.4	740,762	7514	1048	2983	1	20
20, 50, 15, 15	0.01	740,762	7331	1048	3411	1	24
	0.4	740,762	7936	1048	3696	1	23

Table 4.4: Logical topology costs before (BA) and after (AA) annealing. The initial topology is fully-connected, and we use total edge cost as objective function. The initial temperature is set at 200

fully-connected approach generates lower total edge cost, higher diameter topologies. Since the initial topology is fully-connected, even with higher probabilities of selecting the *add* transformation, the only possible transformation is *delete*. As the *delete* probability increases, we observe the opposite, that is, the fully-connected approach generates higher total edge cost, lower diameter topologies.

Because it allows more *delete* operations, the fully-connected approach has the disadvantage of taking longer to finish. The *delete* operation is more expensive than the others because it requires a full-connectivity check.

4.5.3.2 Adding Selected Edges

In these experiments, we started with a random feasible initial topology generated by the algorithm described in Section 4.4.1. Recall that these initial topologies tend to have a low total edge cost since they do not include many extra edges. Therefore, the next step is to add a number of extra edges so that we can reduce the diameter.

We used different edge selection heuristics. In the first experiment, we add direct edges connecting node pairs whose distance in edge cost is maximum. In other words, we inspect the current topology's adjacency matrix, and add an edge connecting the first pair of nodes, whose distance is equal to the diameter in edge cost. The first entry in Table 4.5 shows the starting solution's costs. The remaining entries show the costs after a new link is added to the previous topology.

Total Edge Cost	Diameter in Edge Cost	Diameter in Hop Count
9376	4687	25
10294	4684	25
11035	3216	17
11411	3099	17
12042	3038	17
12882	2761	14
13632	2675	12
14364	2530	11
15159	2358	11
15706	2328	10
16094	1985	10

Table 4.5: Logical topology costs after adding 10 extra edges to the initial topology. Links are added according to diameter in edge cost.

As expected, total edge cost goes up as we add new links, and diameter goes down. Notice that after adding the second link, both diameter in edge cost and hop count decrease more than 30% with an increase of less than 15% in total edge cost. We also observe that as diameter in edge cost decreases, so does diameter in hop count. However, every time a new link is added diameter in edge cost decreases, but diameter in hop count may stay the same.

In the next experiment, instead of using diameter in edge cost, we based our selection criteria in diameter in hop count; when adding an edge, we inspect the current adjacency matrix, and add a link connecting the first pair of nodes, whose distance is equal to diameter in hop count. Each entry in Table 4.6 below shows total edge cost, diameter in edge cost, and diameter in hop count every time a new link is added.

Again, as we add new links both diameter in edge cost and hop count go down. Notice that after adding the second link, diameter in hop count decreases almost 50%, while total edge cost increases less than 20%.

Finally, in the last experiment, we tried to minimize the increase in total edge cost each time a new edge is added. So, we use the same edge selection criteria as before, except that we choose the cheapest edge connecting nodes whose distance in hop count is equal to the diameter in hop count.

Total Edge Cost	Diameter in Edge Cost	Diameter in Hop Count
9376	4687	25
10290	4635	25
11184	3774	14
12141	3774	14
12786	3774	13
13676	3328	13
14344	3068	11
14734	2936	10
15352	2936	10
15870	2936	9
16543	2936	9

Table 4.6: Logical topology costs after adding 10 extra edges to the initial topology. Links are added according to diameter in hop count.

When we compare these results with the values in Table 4.6, we notice that for the same diameter decrease, the total edge cost increase is lower. For example, after we add the second link we get the same reduction in diameter in hop count, but only a 10% increase in total edge cost. We also observe that because we select the cheapest link every time, the reduction in diameter in edge cost is higher than in the previous experiment.

Instead of using the regular *add* transformation in the simulated annealing approach, we can add selected edges according to one of the criteria presented above. The disadvantage of using the *selected add* operation is that it is more expensive than the regular *add*.

4.5.3.3 Deleting Selected Edges

The same way as we use heuristics to choose edges to add, we can select good edges to delete. When choosing an edge to delete, the goal is to lower the topology's total edge cost, yet keeping the diameter constant. Recall that diameter is the maximum shortest path between any pair of nodes. If we want to keep diameter in edge cost constant, then we should find a directly connected pair of nodes whose connecting edge cost is equal or higher than the diameter. Another possibility is to find all pair

Total Edge Cost	Diameter in Edge Cost	Diameter in Hop Count
9376	4687	25
9751	4626	25
10508	3254	14
11103	3132	14
11895	2908	13
12527	2642	13
13222	2555	11
14046	2553	11
14921	2553	10
15559	2553	9
15874	2390	9

Table 4.7: Logical topology costs after adding 10 extra edges to the initial topology. Links are added according to diameter in hop count. The cheapest link is selected every time.

of nodes that satisfy the above requirements, and choose the most expensive link to delete.

Similarly to *selected add*, we can use the *selected delete* operation instead of the normal *delete* operation in the simulated annealing approach. Like *selected add*, *selected delete* is also more expensive than the regular *delete*.

4.6 Summary and Discussion

We presented our topology calculation algorithm and the results obtained when feeding our algorithm with randomly generated topologies for different objective functions controlling the annealing process. Choosing the objective function depends on the type of optimization problem being solved. For instance, in building logical topologies that use the network efficiently, yet limiting update propagation delays, we use total edge cost and diameter in the objective function. Because the initial topologies our algorithm generates already have low total edge cost, the resulting topologies show total edge cost reductions of up to 10%. However, we achieve diameter reductions of more than 30% in edge cost and more than 50% in hop count.

If we want to optimize propagation delays in terms of number of hops, we use an objective function that combines total edge cost and diameter in hop count. The

corresponding simulation results showed reductions of 70% in diameter in hop count at the expense of total edge cost increase of approximately 15%. This is because reductions in diameter in hop count can only be achieved by adding more edges, while swapping more expensive edges with cheaper ones can result in reductions in diameter in edge cost.

We also used total edge cost as objective function, and the resulting topologies showed a 25% reduction in total edge cost. All topologies also showed diameter reductions.

We argued that even though we report logical topologies' total edge costs, it does not mean that these are the costs flood-d incurs when propagating updates. Flood-d's duplicate detection mechanism avoids sending updates to replicas that already received them. Similarly to internet multicast, flood-d ends up propagating updates according to a spanning tree that tries to minimize cost. The same way as some adaptive routing mechanisms keep alternate routes, flood-d keeps alternate logical paths for resilience.

Finally, we presented some alternate approaches to solving our logical update topology problem. In particular, the selected add and selected delete schemes consist of using heuristics to select edges to be added to or deleted from a given topology. For example, if we want to reduce a topology's diameter, we can add some extra edges. Our simulation results show that depending on the edge selection criteria, we can get diameter reductions of almost 50% with a total edge cost increase of less than 20%.

The selected add and selected delete approaches can also be combined with our simulated annealing algorithm. One way of doing this is to replace the regular *add* and *delete* edge operations with the *selected add* and *selected delete*. Alternatively, we can run simulated annealing on a feasible starting topology, and then use the selected add or selected delete schemes. For instance, suppose we use total edge cost as objective function to generate a low edge cost topology. Then, we can throw in a couple of cheap edges that will reduce diameter, but will not increase total edge cost considerably.

Our simulations showed that heuristic approaches like *selected add* are a practical way of solving our logical topology calculation problem. However, this only became visible after we used our optimization algorithm and inspected its solution space.

By doing so, we could verify that adding selected edges to the initial topologies our algorithm generates did indeed generate acceptable topologies.

While the simulations we ran helped us understand the behavior of our algorithm for different parameters, they used artificially generated physical topology maps. In particular, the topology maps used presented roughly uniform link cost distributions [8]. Having uniform link cost distributions have several implications. For instance, when selecting an edge to add or delete, choosing the cheapest or the most expensive ones does not have as big of an impact as when link cost distributions have higher variation. In our future work, we will use the physical topology estimates collected during larger, live experiments to further tune our topology computation algorithm.

[8] As previously mentioned, we used NTG [26], a random topology generator, to generate the maps we feed to our topology calculation algorithm.

Chapter 5

Replication Groups and Logical Update Topologies

Chapter 2 presented flood-d, our replication tool that keeps replicas weakly consistent by propagating updates according to a logical topology. Flood-d organizes replicas into hierarchical replication groups, and for each group determines a logical update topology that attempts to use the physical network efficiently and minimize the time to propagate updates.

In the first part of this chapter, we investigate via simulation how replication group size, the percentage of time replicas are down, and the parameters of the consistency protocol affect the size of flood-d's consistency state, the time to propagate updates to all members of a group, and the time to learn that all group members have received an update. This study models pure end-to-end TSAE algorithm, not update propagation via flood-d's logical update topology.

The simulations in the second part of the chapter incorporate the notion of a logical topology for propagating updates. Recall that a TSAE replica periodically chooses another replica to compare consistency state, and exchange the missing updates. We assign communication costs to the logical links between each pair of replicas, and contrast random and cost-based partner selection policies. The results show how these partner selection policies impact the cost and the time to propagate updates for different replication group sizes.

5.1 Consistency State Size and Propagation Time

Figure 5.1 demonstrates the benefits of aggregating replicas into multiple replication groups. The top graph shows how the average consistency state size for different sized replication groups varied during a simulation run. The anti-entropy rate is the frequency that replicas poll each other, looking for inconsistency. Let's assume that an information service's database is replicated in 200 sites. If replicas are configured in a single, flat replication group, they need to keep state for each individual replica. Decomposing the original flat group into two groups of 100 replicas each results in smaller consistency state sizes. And, as groups get smaller, so does the replica consistency state size. Clearly, as replication degrees reach thousands or tens of thousands of replicas, having a single, flat replication group becomes prohibitively expensive. We should point out that the consistency state grows as replicas generate new updates, and as these updates propagate to the group. The consistency state shrinks as replicas purge updates that have been received by all other replicas in the group. This explains the oscillation in the average consistency state size over time.

The middle and bottom graphs in Figure 5.1 show the cumulative distribution of the time required to propagate an update consistently to all replicas and the time required for all replicas to purge their consistency state. As expected, as the group gets bigger, it takes longer for an update to get to all replicas in the group. This explains the large difference in the consistency state size. When replicas are organized in multiple, smaller groups, once the update reaches a group, it propagates faster to the rest of the group members, and the nodes in this group can purge their consistency state. Of course, this does not change the total time to propagate the message to every node in every group. Furthermore, for bigger groups, the slopes of these distributions also get smoother. This means that smaller groups achieve a consistent state sooner.

5.1.1 Anti-Entropy Rate

This next set of results show how the consistency state size and the time to propagate updates and acknowledgments scale with the anti-entropy rate. In this first set of graphs, we made the group update generation and failure rates constant, and varied

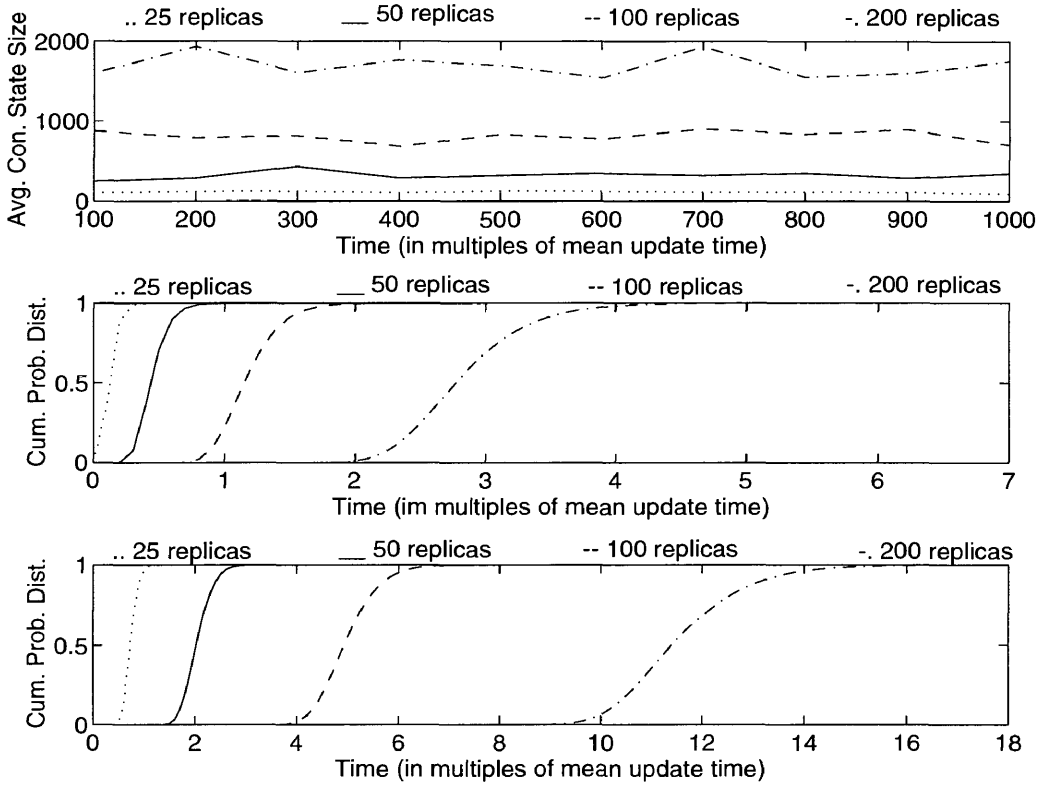

Figure 5.1: Average consistency state size (in number of update messages), time to propagate updates and acknowledgments for different group sizes. In all 3 graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The global update, anti-entropy, and failure rates were kept constant.

the group anti-entropy rate. Notice that the anti-entropy rate was kept constant for different group sizes [1].

In Figures 5.2, 5.3, and 5.4, anti-entropy rates are twice and equal to the update rate in the top and bottom graphs, respectively. In all graphs, replicas stay up 99.9% of the time. We use such high availability because at the same time that we want to minimize the influence of failures in this set of simulations, we also want to introduce some degree of unavailability. We study the effects of lower availability in Section 5.1.2 below. Figure 5.2 shows the average consistency state size for different anti-entropy rates and group sizes. We notice that as state exchanges get less frequent, which saves network resources, consistency state sizes get bigger.

The anti-entropy rate also impacts the time it takes for updates and acknowledgments to propagate to all replicas. Figures 5.3 and 5.4 show that less frequent state exchanges cause updates and acknowledgments to propagate slower. Because we did not model message exchange using flood-d's logical topology, these are worst case assessments. Note how it takes 2 to 3 times more time to purge state than it does to reach consistency.

Should each replica's anti-entropy rate decrease as the group size increases so that the group's global anti-entropy rate remains a constant? If not, for approximately the same number of updates, bigger groups will generate more anti-entropy sessions. Figures 5.5, 5.6, 5.7 plot consistency state size, time to reach global consistency, and time to purge all logs for three replica anti-entropy rates. The higher the anti-entropy rate, the smaller the consistency state size, the faster consistency is obtained and state purged.

5.1.2 Replica Availability

Lower replica availability increases the consistency state size and the time to propagate updates and acknowledgments. In Figures 5.8, 5.9, and 5.10, replicas stay up 99% and 90% of the time in the top and bottom graphs, respectively. The lower the

[1] For instance, to generate the same group anti-entropy rate, the replica anti-entropy rate needs to be 2 times higher in a group of 100 replicas than in a group of 200 replicas.

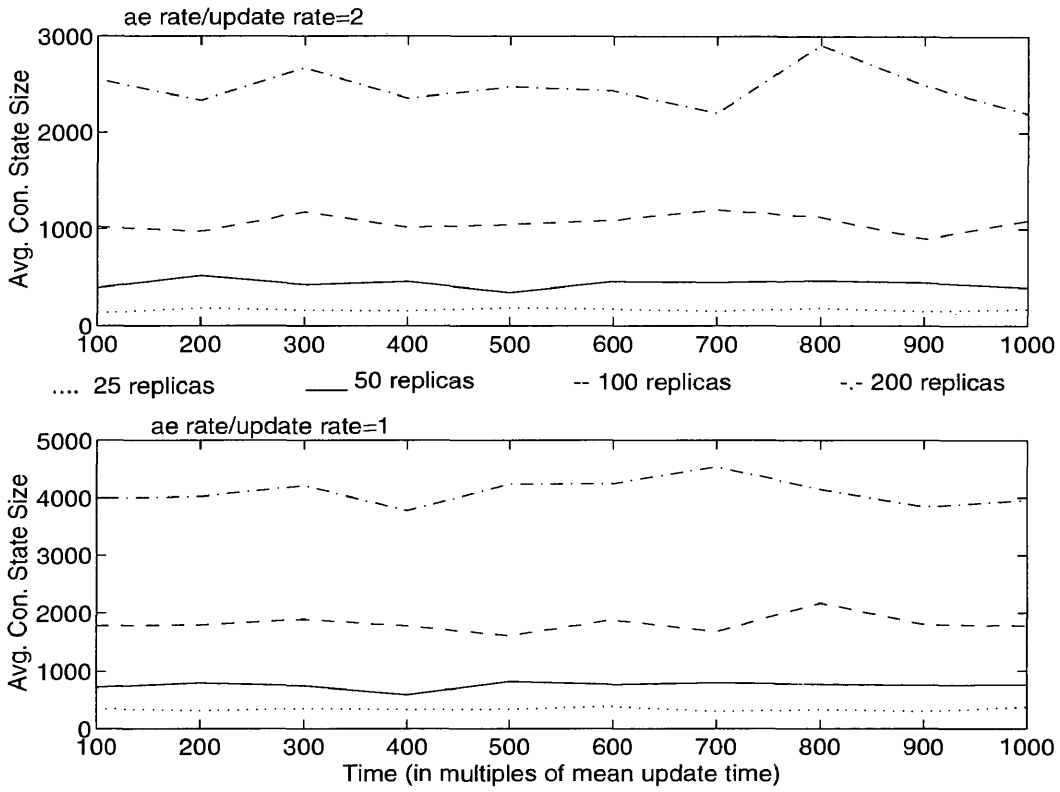

Figure 5.2: Average consistency state size sample paths (in number of update messages) for different group sizes. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

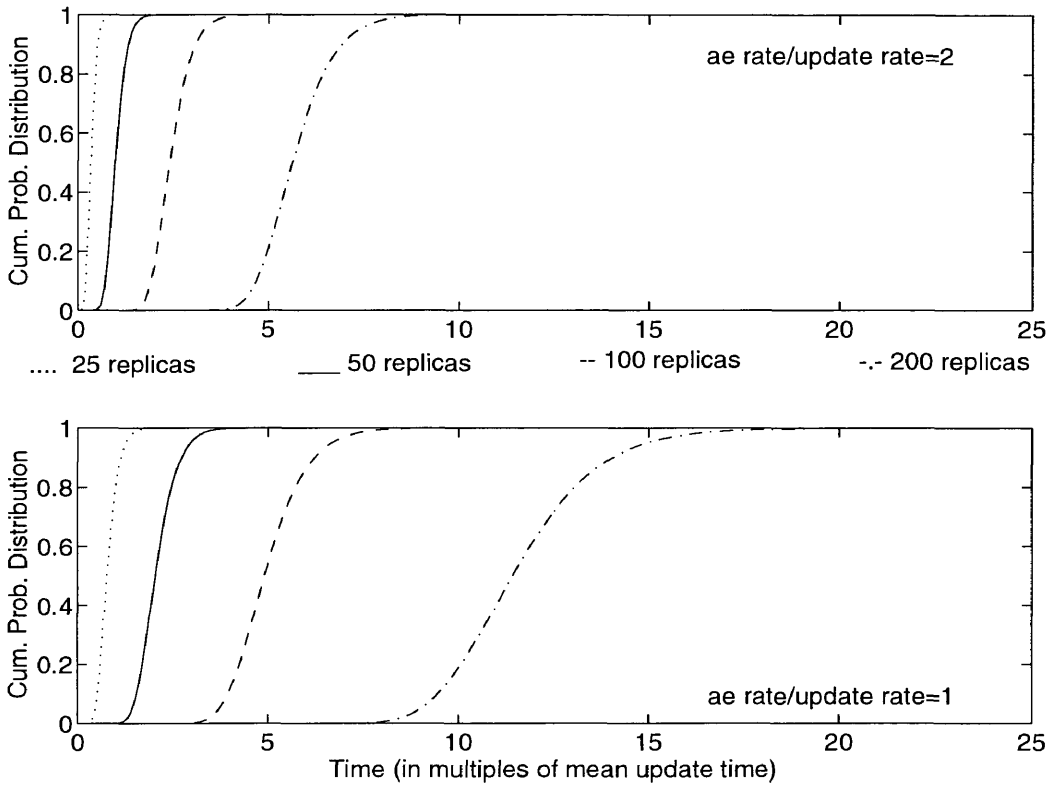

Figure 5.3: Cumulative probability distribution for propagating updates to all replicas consistently. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

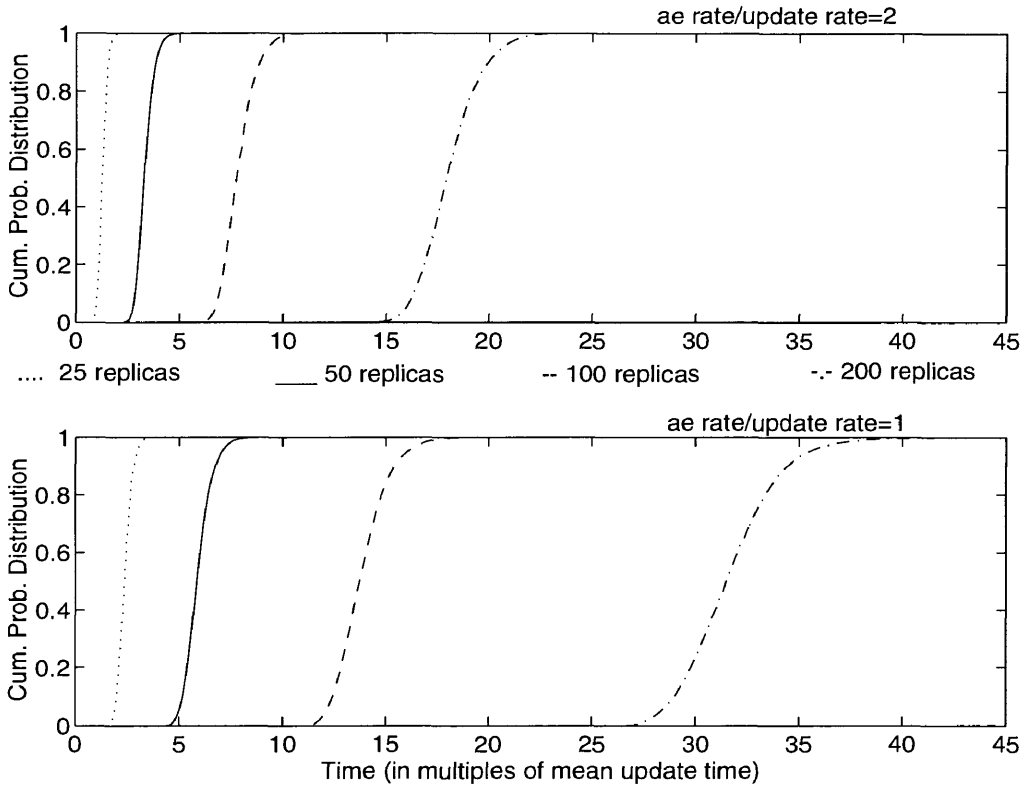

Figure 5.4: Cumulative probability distribution for receiving acknowledgments from all replicas and purging consistency state. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

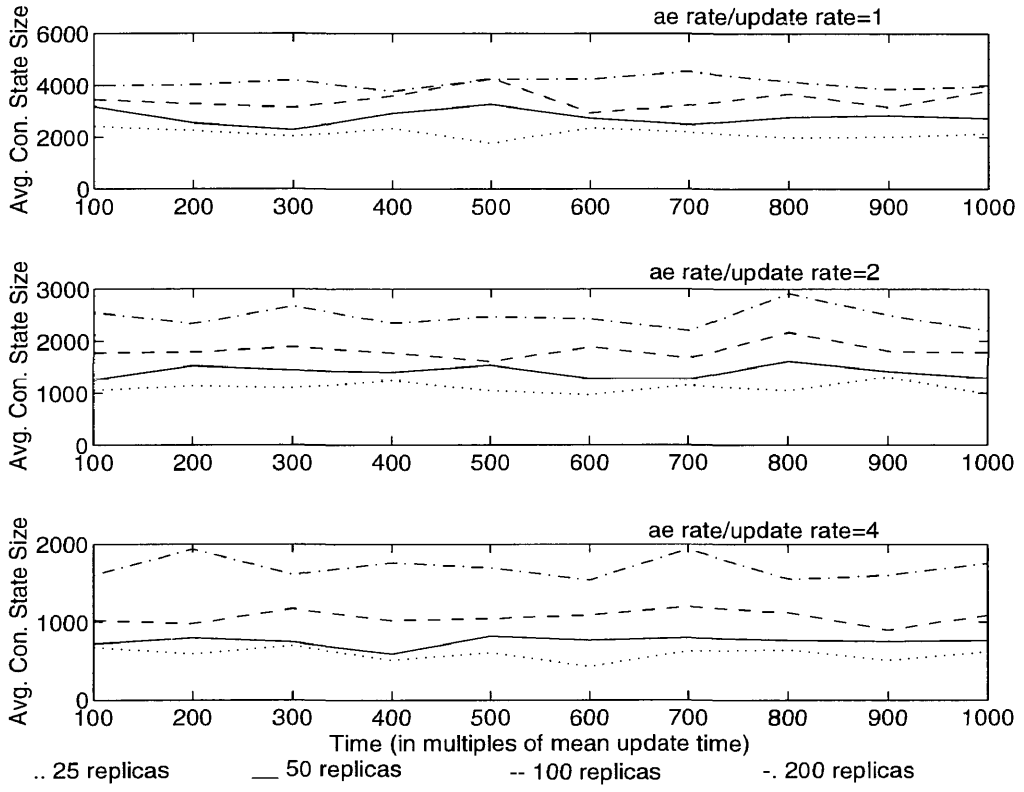

Figure 5.5: Average consistency state size (in number of update messages). In all three graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

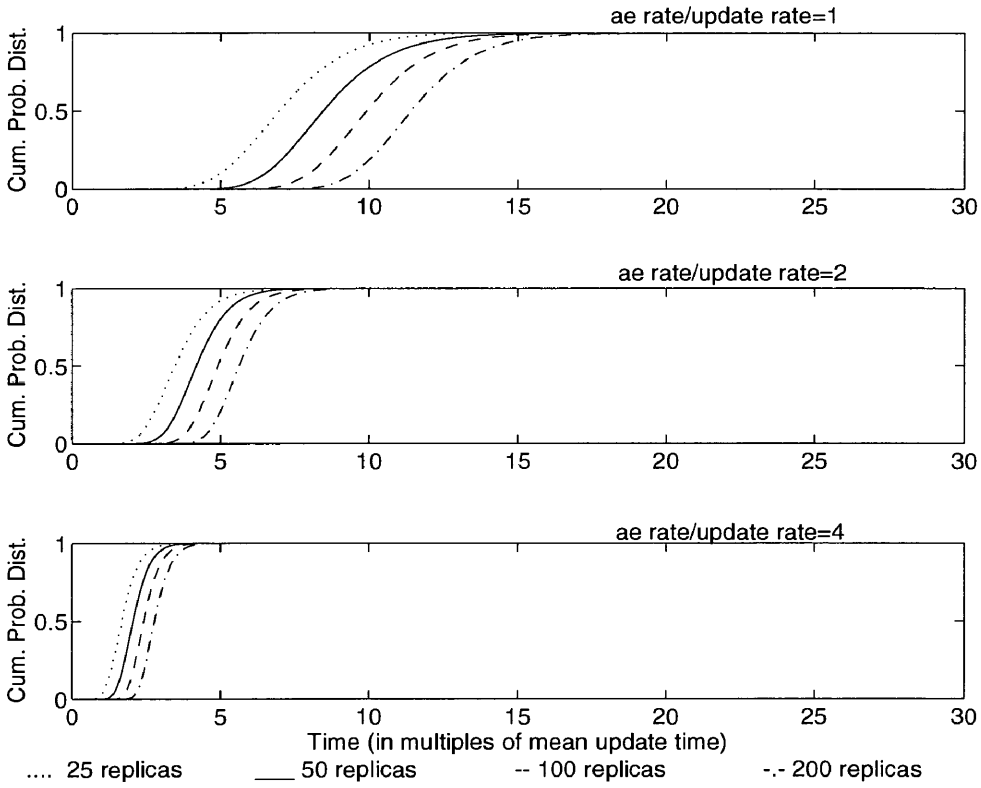

Figure 5.6: Cumulative probability distribution for propagating updates to all replicas. In all three graphs, the x-axis is the simulation time scale in multiples of when updates are generated.In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

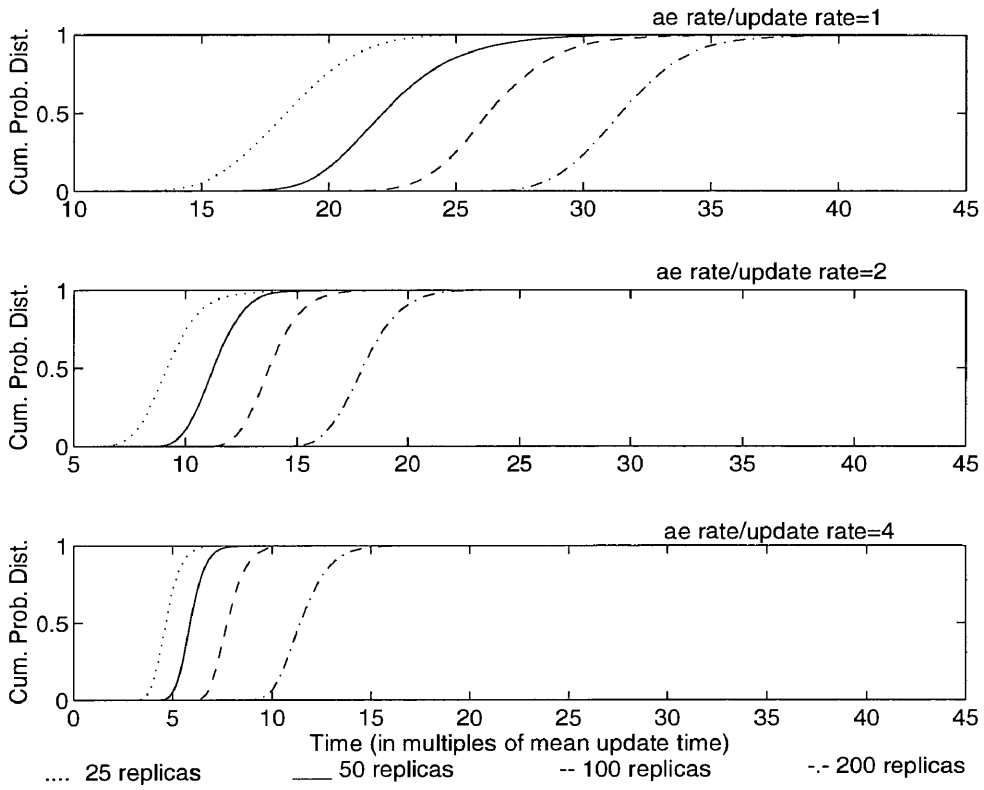

Figure 5.7: Cumulative probability distribution for receiving acknowledgments from all replicas. In all three graphs, the x-axis is the simulation time scale in multiples of when updates are generated.In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

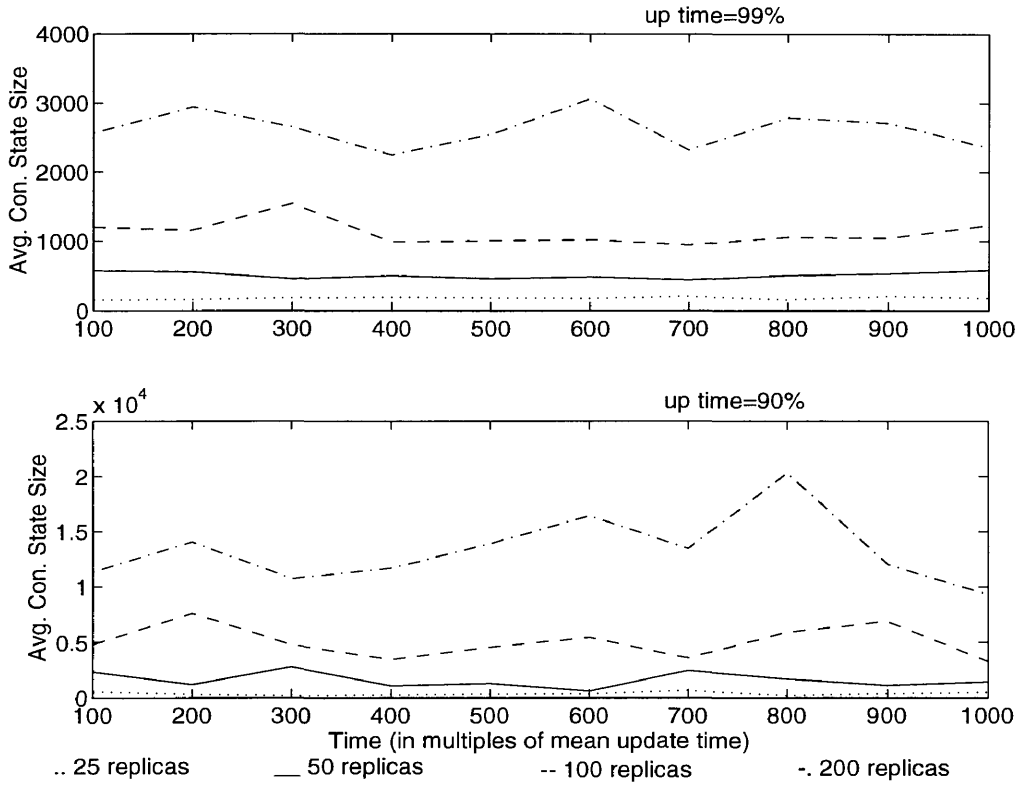

Figure 5.8: Average consistency state size (in number of update messages). In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

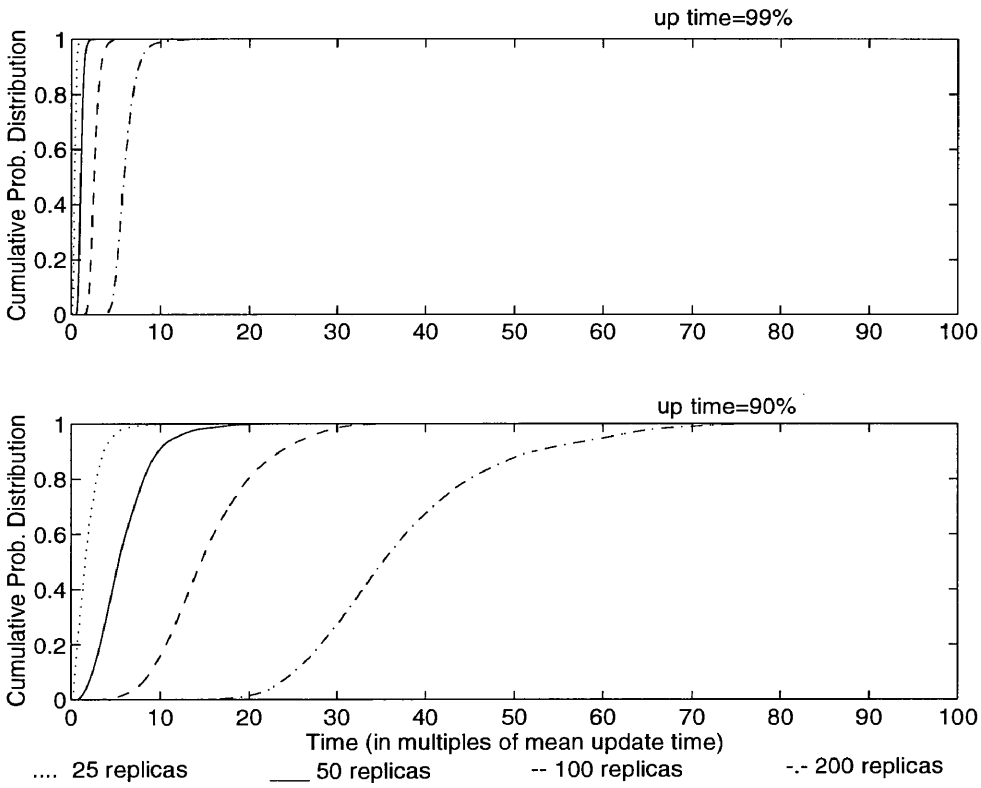

Figure 5.9: Cumulative probability distribution for propagating updates to all replicas. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

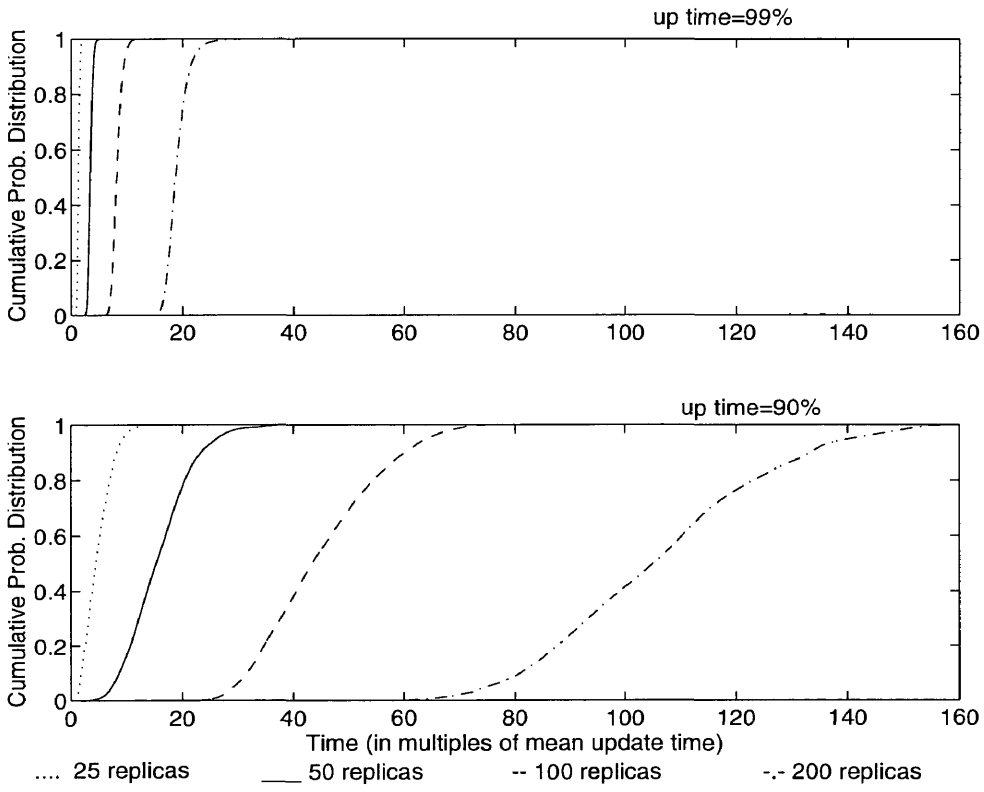

Figure 5.10: Cumulative probability distribution for receiving acknowledgments from all replicas. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

replica availability, the larger the consistency state gets. Consequently, the argument for having smaller replication groups becomes even stronger in an environment like the Internet, where site and link failures are reasonably frequent.

Figures 5.9 and 5.10 show the times to propagate updates and purge state for lower replica availability. As expected, lower availability cause longer delays in propagating items to all members of a group. However, a difference of less than 10% in availability has a huge impact on the time to propagate an update to all replicas in a group and the group convergence time. This is specially true for bigger groups, and it means that not only replicas will take longer to converge to a consistent state, but also that they will take longer to purge their message logs. Clustering replicas into smaller groups reduces the effect of higher site or link failure rates.

5.2 Cost

Besides organizing replicas into multiple replication groups, flood-d also suggests good logical topologies that replicas can use to propagate updates. These logical update topologies attempt to use the underlying network efficiently and at the same time, reduce update propagation time. In this section, we show the effects of using topological information on the overall cost of propagating updates in replication groups of different sizes. We compare two different partner selection policies: random, in which a replica randomly chooses another replica to exchange consistency state, and cost-based, in which the peer with the minimum cost link is selected.

To assign communication costs to links, we use NTG [26], a random topology generator, to generate logical topologies for replication groups of different sizes. We feed the topology generator with the group size, average node degree, and link bandwidths. The topology generator randomly places nodes on a plane, and connects them with links whose costs are a combination of link bandwidth and physical distance between nodes. The resulting logical topology which is represented by a fully-connected, symmetric cost matrix [2], serves as input to our simulator. For these

[2]We should point out that in the real world the cost matrix will not be symmetric since logical links can use asymmetric paths.

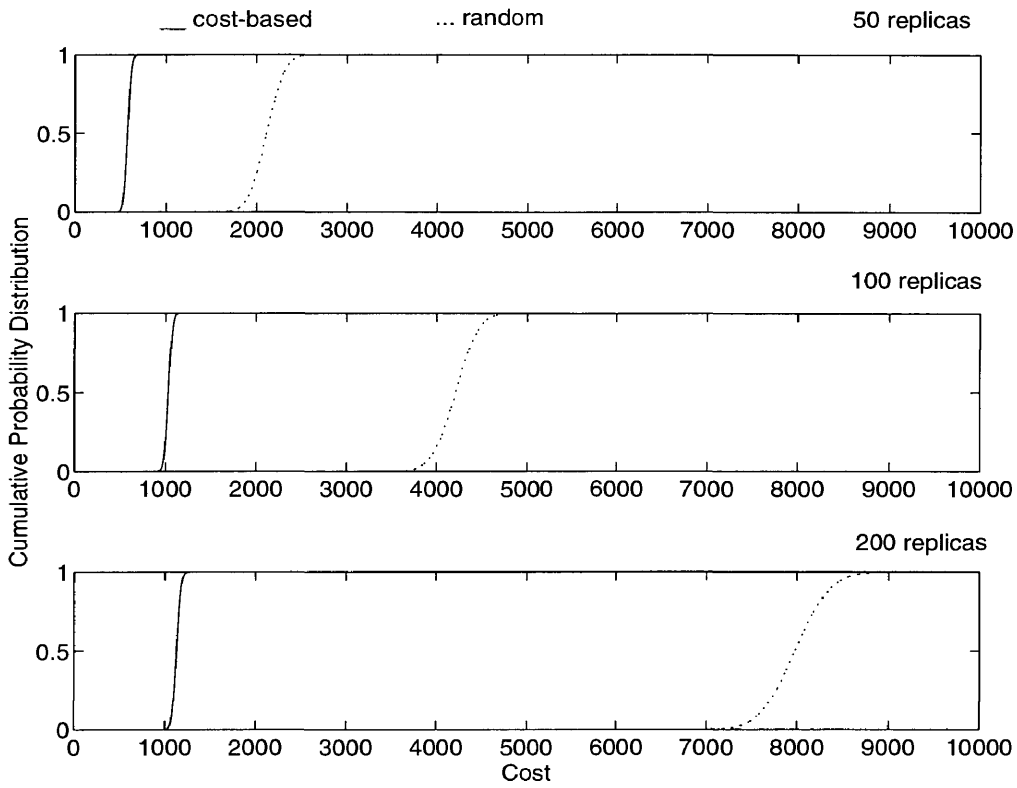

Figure 5.11: Cumulative probability distributions for the total cost of propagating updates to all replicas using cost-based and random partner selection policies for different group sizes.

cost simulations, we set the anti-entropy rate equal to the update rate, and replica availability at 99%.

The top graph in Figure 5.11 plots the cumulative probability distribution for the total communication cost for propagating updates to all members in a group of 50 replicas using random and cost-based partner selection policies. These distributions show that it costs at least 3 times less to propagate updates when replicas choose their peers based on the communication cost to get to them instead of randomly selecting them. For these simulations, each replica chooses its 3 lowest-cost peers, and each time the replica performs an anti-entropy session, it randomly chooses among its 3 previously selected peers. In other words, we generate a logical update topology in which replicas have connectivity degree of 3.

As replication groups get bigger, the discrepancy between cost-based and random peer selection policies increases. The middle and bottom graphs of Figure 5.11 show

the cost distributions for 100- and 200-replica groups. Notice that for a group with 200 replicas, the cost-based approach is approximately 7 times cheaper than choosing peers at random.

To evaluate how partner selection policies affect the time to propagate items to all replicas, Figure 5.12 presents the cumulative probability distributions for propagating updates to all replicas for different group sizes. We notice that items take longer to propagate to all replicas when the cost-based partner selection policy is used. This can be explained by the following arguments. First, since link costs remain constant during the whole simulation, the set of partners that a replica chooses using cost-based selection is always the same. In the random selection approach, however, each replica can select any other replica in its group with whom to exchange consistency state. Thus, each replica has connectivity degree of $n - 1$, where n is the group size. The other argument is that the logical topology generated in the cost-based approach does not take diameter into account.

In the next set of simulations, we increase the replica's logical connectivity and observe what happens to the total cost and the time to propagate updates. The connectivity is set to 10% of the group size, which means that every time a replica performs an anti-entropy session, it randomly chooses one among its $0.1n$ lowest-cost peers, where n is the replication group size. Figures 5.13 and 5.14 show the total cost and the time to propagate updates to all replicas in groups of different sizes. There is a slight increase in cost for groups of 50 replicas when compared to the 3-connected case (Figure 5.11). For bigger groups, the difference in cost between the $0.1n$- and the 3-connected cases increases, since more expensive logical links are being used. On the other hand, because of the higher connectivity, the difference in time to propagate updates to all replicas using cost-based and random selection policies decreases when compared to the 3-connected case (Figure 5.12).

5.3 Summary

Our simulation results demonstrate the benefits of splitting a single, flat replication group into multiple smaller groups. We observe that the smaller the replication group, the smaller the size of the consistency state each replica in the group needs

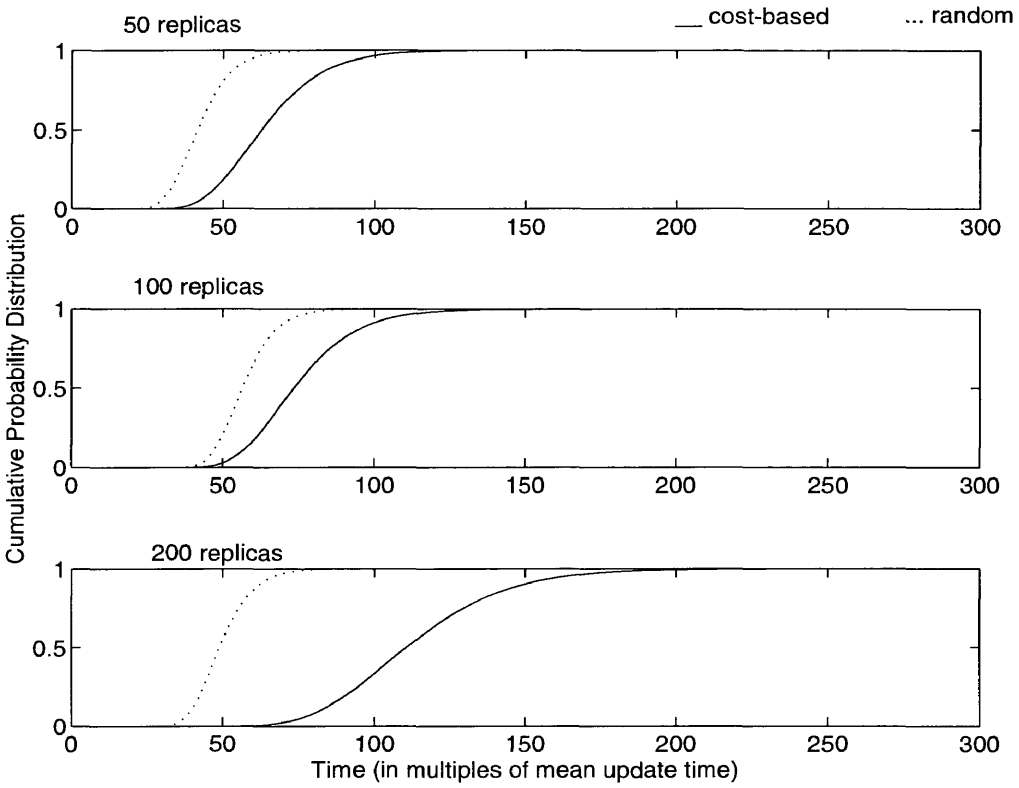

Figure 5.12: Cumulative probability distributions for the time to propagate updates to all replicas using cost-based and random partner selection policies for different group sizes.

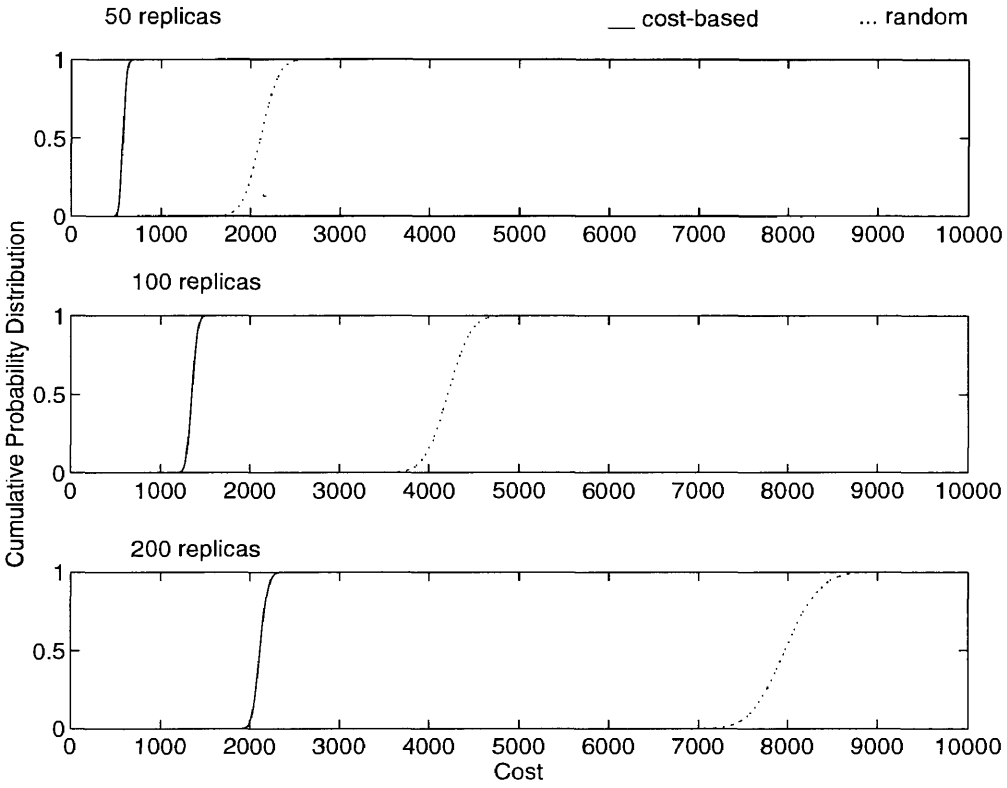

Figure 5.13: Cumulative probability distributions for the total cost of propagating updates to all replicas using cost-based and random partner selection policies for different group sizes.

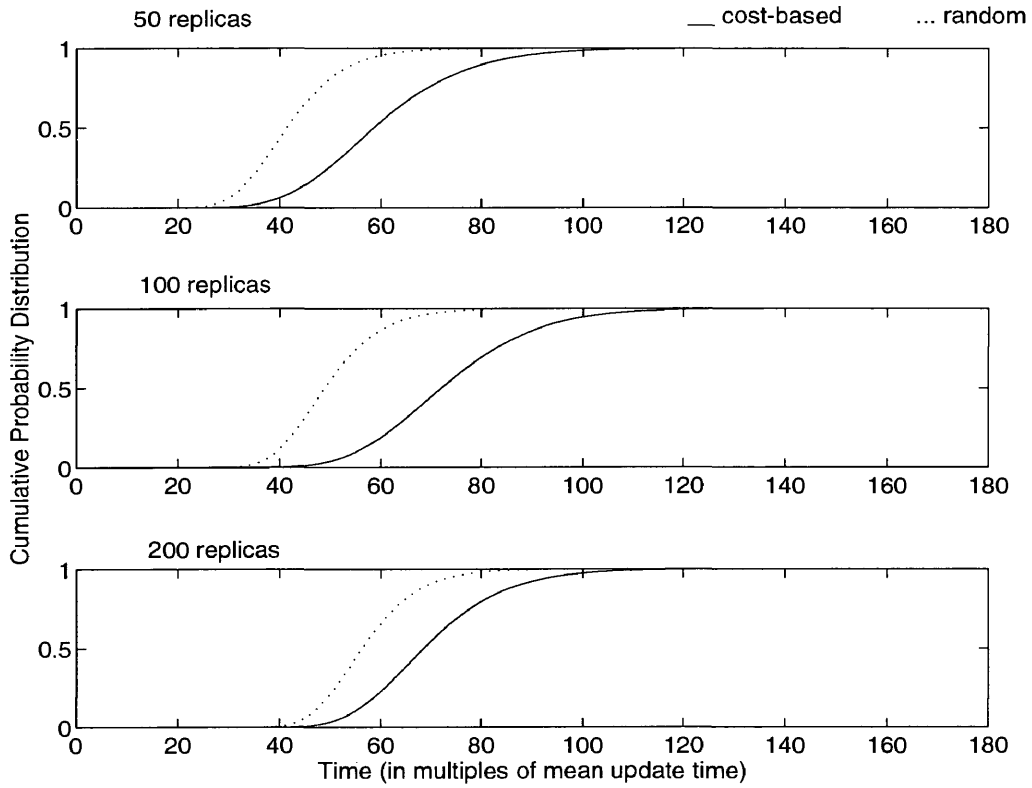

Figure 5.14: Cumulative probability distributions for the time to propagate updates to all replicas using cost-based and random partner selection policies for different group sizes.

to keep. This difference in consistency state size is amplified because in smaller groups, replicas can purge their consistency state sooner.

We also show how anti-entropy and failure rates impact the size of the replica consistency state, and that smaller replication groups attenuate the effect of less frequent state exchanges and lower replica availability.

Finally, we assigned communication costs to logical links, and compared 2 different anti-entropy partner selection policies: random and cost-based. We observed that while for a 50-replica group, propagating updates with a cost-based partner selection policy is about 3 times less expensive than using a random selection policy, this cost ratio jumps to 7 for 200-replica groups.

We also noticed that due to the lower logical connectivity and the fact that the logical topology generated in the cost-based approach does not take diameter into account, items take longer to propagate to all replicas when the cost-based partner selection policy is used. As proof of concept, we increased the replica's logical connectivity and observed that the difference in time to propagate items to all replicas using cost-based and random selection policies decreases when compared to the less connected case. As expected, the higher connected topology resulted in slightly higher propagation costs.

However, as described in previous chapters, flood-d uses its delay and bandwidth estimates to compute logical update topologies. Flood-d's logical topologies optimize not only network utilization costs by restricting total edge cost, but also propagation delays by limiting diameter. Therefore, when updates propagate using flood-d's network-cognizant logical topologies, they use the network efficiently, yet reducing their propagation delays.

Chapter 6

Exploiting Internet Multicast

Internet multicast and reliable multicast transport protocols are often thought as good foundations on which to build a massive data replication protocol. This chapter presents the state-of-the-art of internet multicast architectures and multipoint transport protocols. We discuss the applicability of existing multipoint transport protocols for weakly consistent applications, and propose a reliable multipoint transport service specially tailored for this kind of applications.

Through simulations, we evaluate how the use of IP multicast can influence the performance of flood-d, our weakly-consistent replication tool. We discuss the assumptions we use in these simulations, and speculate on how to improve our simulation model.

6.1 Internet Multicast

IP multicasting delivers best-effort datagrams to a group of hosts sharing a single IP multicast address. Current Internet multicast routing protocols, such as DVMRP [20] and MOSPF [44] are extensions to traditional unicast routing mechanisms.

Because MOSPF is a link-state protocol, MOSPF routers compute source-rooted shortest-path delivery trees based on their global view of group membership and network topology. Broadcasting group membership information about all existing groups to all MOSPF routers is expensive, and severely limits the scalability of MOSPF and other link-state multicast protocols. The shortest-path delivery tree computation is another factor that limits the scalability of link-state multicast protocols.

DVMRP is a distance-vector multicast routing protocol that builds source-rooted shortest-path distribution trees as follows. A DVMRP router forwards data packets destined to unknown groups out all interfaces except the one through which it received the packet. When a leaf router receives a packet for a group that has no members on its attached subnetworks, the router sends a prune message upstream towards the source of the packet. Since prune messages cause the tree branches with no group members to be removed from the multicast tree, the resulting tree is a source-specific shortest-path tree where all leaves are attached to group members. Pruned branches will timeout, grow back, and be pruned again if they still don't have any attached group member. Distance-vector multicast protocols incur the overhead of having sources periodically broadcast data packets which trigger routers that are not on the delivery tree to process these messages and generate the corresponding prunes.

DVMRP has been in use on hundreds of routing domains that form the MBONE [9], a semi-permanent, hand-configured virtual network that was originally engineered to carry audio and video transmissions from IETF meetings to destinations around the world. The MBONE uses virtual point-to-point links, or *tunnels*, to transmit multicast packets between DVMRP routers in different routing domains. Before transmitting multicast packets through a tunnel, the source DVMRP router encapsulates them, so that they look like ordinary datagrams to intermediate routers and subnets.

With the argument that traditional IP multicasting presents scalability and efficiency limitations, alternative multicast routing architectures have been proposed. Instead of building multiple source-based trees, the Core Based Tree (CBT) architecture [4] employs a single delivery tree shared by all the members of a multicast group. The shared tree has a *core* router (or a set of cores for robustness). *Non-core* routers are the intermediate nodes in the shortest paths between the core and routers directly attached to group members. Data packets are forwarded using unicast until they reach a CBT router. From then on they are multicast using the shared tree [1]. CBT trades source-specific shortest-path delivery trees for scalability: rather than having to keep an entry per [*source, group*] pair, CBT routers need only keep forwarding information for each multicast group. However, shared trees may impose unacceptable bandwidth limitations for high bandwidth, concurrent-sender applications, such as audio and video multiparty conferencing.

The Protocol Independent Multicast (PIM) architecture [21] supports both shared and shortest-path delivery trees. While high bandwidth applications may choose to use PIM's *dense mode* source-specific shortest-path trees, shared trees may be adequate for low data data rate applications or sparse multicast groups. By using shared delivery trees, or PIM's *sparse mode*, sparsely populated groups avoid the overhead incurred by protocols such as DVMRP, where sources occasionally have to broadcast data packets, and routers with no attached group members have to respond with prune messages. Furthermore, sparse-mode routers need only keep state for each multicast group.

6.2 Multipoint Transport Protocols

IP multicast relies on transport-level protocols for reliability and sequencing. For instance, real-time applications like voice and video teleconferencing, which are delay-sensitive but can live with data losses, are layered on top of UDP and Internet multicast. A transport protocol for multiparticipant real-time applications (RTP) [61] provides end-to-end delivery for one or more real-time data flows. It assumes an

[1] Actually, according to [4], each non-core router uses unicast to forward data packets out all branches of the shared tree.

unreliable datagram service and does not provide reliable, ordered delivery. RTP can transfer data to multiple destinations if the underlying network provides a multicast service.

Traditional reliable unicast transport protocols, such as TCP, use positive acknowledgments to recover from packet loss. This approach to achieving reliability is often referred to as *sender-initiated*, since it is the responsibility of the sender to detect packet losses. In a multicast environment, as group sizes increase, the sender-initiated scheme may cause *acknowledgment implosion* [16] since each delivered packet triggers an acknowledgment from every receiver in the group.

Alternatively, in the receiver-initiated approach to reliability, receivers detect packet losses and request its retransmission by generating a negative acknowledgment. Placing the responsibility of recovering from packet losses on the receiver alleviates the acknowledgment implosion problem. The performance comparison study presented in [48] confirms that receiver-initiated multicast transport protocols deliver better performance than their sender-initiated counterparts.

Most reliable multipoint transport protocols are either pure receiver-initiated or use a hybrid approach by combining receiver- and sender-initiated reliability. Below, we overview some of these protocols.

6.2.1 Reliable Broadcast Protocol

The Reliable Broadcast Protocol (RBP) [10] provides multipoint communication between sites connected by a local-area broadcast network. RBP combines negative and positive acknowledgments to achieve reliability, ordering, and resilience to failures. Messages are multicast to the group through the *token site*. In other words, the token site is responsible for multicasting an acknowledgment for each message it receives. These positive acknowledgments inform the sender that the token site received a message. Since acknowledgments carry a global timestamp, receivers use them to order and detect lost messages.

When a receiver detects a lost message, it unicasts a negative acknowledgment to the current token site, which replies with the missing message. For resilience and to limit the amount of state each site needs to keep, the role of the token site rotates among all group members. A member accepts to be the next token site only if it

has all the messages earlier than the timestamp of the token passing message. Thus, after a message is generated and the token site is rotated K times, K sites can fail and the message will still be delivered to the sites that are active. Furthermore, the current token site can purge all messages whose timestamps are earlier than the last time this site had the token.

6.2.2 Multicast Transport Protocol

Influenced by RBP, the Multicast Transport Protocol (MTP) [3] provides reliable and globally ordered delivery of client data among a group of communicating processes. MTP is built on top of IP multicasting, and provides strong consistency by multicasting messages atomically under centralized control. One of the participating processes is assigned to be the *master* of the group. The group master ensures that data is delivered reliably and in order by giving out *transmit tokens* and controlling the status of tokens and data messages. It implements an implicit congestion control mechanism by not granting tokens if a certain number of messages are still in the *pending* state. A message is pending until the master sees all of its packets.

The master is also responsible for supervising group membership. If a group member that is holding a token becomes disconnected, the master removes it from the group and rejects that member's outstanding messages. A process that wants to join the group sends a request to the master, who may accept or reject it. If the request is accepted, the master must be in possession of all the transmit tokens before sending an acceptance message. This way, the master ensures that the new member sees only complete messages. The master may notify its client application about the new member, and it is up to the application to send client state to the new member.

MTP is a negative acknowledgment protocol. When a group member detects a lost message, it sends a negative acknowledgment to the message producer, who multicasts the requested message to the entire group. However, if the producer has discarded the requested message, it informs the source of the negative acknowledgment. It is up to the application to recover or decide to withdraw from the group. MTP's flow control mechanism is based on a fixed-size transmission window. Every

group member agrees on the current window size upon joining a group. This window specifies the maximum number of data packets that a member can send to the group within a specified period of time, and is shared between new and retransmitted packets. Thus, the more retransmissions, the less new packets are injected into the group.

6.2.3 Reliable Multicast Protocol

The Reliable Multicast Protocol (RMP) [67] provides a totally ordered, reliable, atomic multipoint transport protocol on top of an unreliable multicast service. Like the Multicast Transport Protocol (MTP), RMP evolved from the Reliable Broadcast Protocol (RBP).

Besides using multicast where available, RMP extended RBP with a name service that advertises the existence of multicast groups, and a flow and congestion control mechanism. RMP's name service uses the RMP protocol to manage its distributed database. Each existing group has at least one *global name server* that is registered with the name service group. The global name server's database consists of entries mapping a group's address to the group name. Name servers in each group keep the mappings between group address and the membership set and group name. RMP's group membership protocol allows sites to join or leave a group. This protocol updates the members' group membership view accordingly, and guarantees the resilience requirements. When messages updating the group membership reach a site that is running a local name server, the local mappings are updated. Updating the global mappings once a membership update message reaches a global name server is a strong consistency, atomic transaction across the global name service group.

RMP's flow and congestion control mechanism uses the algorithms Van Jacobson developed for TCP. Each sender in a group keeps a sliding window regulated by retransmission timers, negative acknowledgments (NACKs) from receivers, and positive acknowledgments (ACKs) from the token site. A sender uses slow-start to increase its window size by 1 packet every time an ACK is received. When a retransmission timer expires signaling congestion, the sender exponentially reduces its window size. To implement flow control, a RMP receiver multicasts a NACK to the group (instead of unicasting it to the token site) when it detects a lost packet.

When a sender receives a NACK, it treats it just like an expired timer and backs off exponentially.

6.2.4 Uniform Reliable Group Communication Protocol

The Uniform Reliable Group Communication (URGC) protocol [1] also provides atomic and ordered communication among the members of a group. To achieve ordering and decide when to purge old messages, URGC uses centralized control, which rotates among all members of the group. All sites keep a *history* of all processed messages which allows other sites to recover missing messages. Besides broadcasting its message ordering decisions, the current *coordinator* also informs the group which is the most up-to-date member. Through unicast communication, other group members can recover missing messages from the most up-to-date site.

6.2.5 Propagation Graph Algorithm

Garcia-Molina's Propagation Graph algorithm [23] guarantees ordered delivery in a multicast environment. The Propagation Graph algorithm was inspired by RBP and also attempts to reduce the overhead of fully distributed solutions. However, instead of ordering all messages at a single central site, it uses a collection of sites structured into a message propagation graph. These intermediary nodes are in the intersections of the existing multicast groups. Instead of sending messages to the destinations and then ordering them, messages follow the propagation graph and are ordered along the way. The Propagation Graph method is basically an ordering protocol and needs additional mechanisms to recover from site failure, network partition, and group membership changes.

6.2.6 Muse

Muse [34] has been developed to multicast news articles on the MBONE. Muse sends news articles as UDP packets to the multicast group consisting of participating news servers. Because Muse is not a reliable news propagation protocol, it must be used in conjunction with another mechanism that performs reliability checks, such as NNTP.

According to its authors, making Muse a reliable news transport protocol would result in greatly limiting its scalability because of retransmissions requests from clients that lost or received corrupted articles.

6.2.7 Imm

Satellite receivers use the *imm* protocol [15] to multicast earth observing data files to users on the MBONE. Contrary to some replicated database applications, there is no need to provide an ordered, atomic multicast service for distributing satellite data files.

The imm protocol uses UDP on top of IP multicasting, but unlike Muse, it allows its clients to request selective retransmission of loss packets from imm servers. Servers may also request explicit acknowledgments from members of a group. New members join a group by multicasting a join message to that group. However, imm does not provide any explicit mechanisms for a member to leave a group or for a group to eject a member that has permanently failed.

6.2.8 A Transport Service for a Distributed Whiteboard

The distributed whiteboard tool presented in [36] uses a multipoint transport protocol to reliably distribute data among all participating sites. These sites use negative acknowledgments to request retransmission of lost data, which can be answered by any member that has the information. To avoid generating multiple copies of NACKs and retransmitted data, they are multicast to the group. To further reduce the multiple copy problem, a site waits a random period of time before sending a NACK or retransmitting data.

A new site joins an ongoing whiteboard conference by announcing its presence with a join message. Current members update their view with the new site. The new site's view of the group is gradually updated through periodic status messages generated by current members when the group stays quiet for a certain period of time.

6.2.9 Adaptive File Distribution Protocol

The Adaptive File Distribution Protocol (AFDP) [13] provides a reliable file distribution service on top of UDP. To achieve reliability, AFDP uses a selective negative acknowledgment scheme, in which receivers explicitly request senders to retransmit lost packets. AFDP's rate-based flow mechanism uses negative acknowledgments to slow down senders, who decrease their transmitting rate every time they receive a negative acknowledgment. Senders gradually increase their transmitting rate after successful transmissions. Using multicast as its preferred transmission mode, AFDP packets can also be transmitted using broadcast, or unicast, whenever multicast is not available.

AFDP allows sites to join and leave a distribution group, but does not provide a recovery mode to deal with site failures or network partitions. AFDP has been implemented and tested on a cluster of workstations connected by an ethernet, and has proven to deliver better performance than existing file distribution mechanisms.

6.3 A Multicast Transport Protocol for Weakly Consistent Applications

In contrast to sending real-time audio and video, updating the database of an information service requires reliable message delivery, crash recovery, and eventual database consistency. A reliable multipoint transport protocol based on IP multicasting cannot meet these requirements. In particular, it cannot solve the consistency problem raised when replicas temporarily crash or when IP routers crash and lose state crucial to reliable, multipoint delivery. In such cases, the application itself must re-establish consistency.

Recall the end-to-end argument in layered design [57]: functions that can only be completely and correctly implemented by the application should be moved into the application. In our case, since each replica must keep its database consistent, we let the replica manage reliable multipoint delivery. For these reasons, our hierarchical replication group consistency tool, *flood-d*, does not rely on a reliable, multicast transport protocol, although, for efficiency, it can exploit it where available.

In this section, we examine the transport service requirements of weakly consistent applications, and contrast them to the needs of other types of applications, such as real-time and strong consistency services. Based on these requirements, we propose a multipoint transport protocol specially tailored for weakly consistent services. Below, we describe the mechanisms needed to implement the proposed transport protocol's functionality.

6.3.1 Transport-Level Reliability

At one extreme of the consistency spectrum lie strong consistency multicast transport protocols such as MTP and RMP. Because of their inherent latency and overhead, strong consistency protocols are not adequate for massively replicating weakly consistent applications. These protocols offer a reliable and ordered delivery service to the application. Both protocols achieve reliability and ordering through centralized control. MTP, for example, relies on a group master to oversee group communication and membership. This centralized control compromises MTP's scalability and fault tolerance. That is probably why MTP has not yet been implemented.

Although RMP solves many of MTP's problems by rotating the master's role among all group members, its latency and overhead are still very high for large groups. In fact, the implementation reported in [67] uses a local-area Ethernet, and the performance results refer to groups of less than 10 sites.

Since the ultimate consistency check must be done at the application layer, a multipoint transport protocol for propagating updates in a replicated information service could be totally unreliable. The resulting multicast transport service could use UDP on top of IP multicasting, and would be analogous to the unicast service provided by the UDP/IP protocol suite. In this case, reliability and group membership management are completely implemented at the application using a weak-consistency, hierarchical group communication tool such as *flood-d*.

For efficiency, weakly consistent applications could benefit from early detection and retransmission of lost packets. While these applications would still need to recover from inconsistencies due to site failures, they could use a reliable multipoint transport service such as the one provided by the imm protocol and the one used by the distributed whiteboard tool.

In the proposed reliable multipoint transport protocol, whenever a replica wants to propagate an update to the rest of the group, it passes the update to the transport layer, which assigns monotonically increasing sequence numbers to each packet, and multicasts them to the group. The transport protocol at the receiver's side detects lost packets from holes in the sequence number space. These holes are detected when a source sends subsequent data. Notice that if the last packet in a burst is dropped, receivers will only detect the loss when they receive the next burst. An alternative is to have sites periodically announce their current state, so that the group can detect losses earlier. This can get expensive, and may not be needed since the application will perform periodic consistency checks.

When a receiver detects a missing packet, it sends a negative acknowledgment requesting the retransmission of that packet. When receiving the NACK, the sender replies by retransmitting the requested packet. If the NACK or the retransmitted packet gets lost, the receiver times out and retransmits the NACK. Since it is possible that other group members are missing the same packet, having the receiver multicast the NACK and the sender reply by multicasting the missing packet is a way to avoid having the group generate multiple NACKs and copies of the same packet. Furthermore, since NACKs are multicast to the group, any member that has the requested packet could retransmit it. A site that is close to the requester is likely to be the one that retransmits the packet. However, if several receivers detect a packet loss simultaneously, they will generate multiple NACKs for the same packet. Similarly, multiple copies of the missing packet may be generated since any member may answer requests. Using randomization to schedule the transmission of NACKs and retransmission of missing packets alleviates this multiple copy problem.

One problem with a pure negative acknowledgment protocol is that the state senders need to keep can grow indefinetly, since there is no explicit mechanism to inform senders that all receivers have received a given message. To solve this problem, besides issuing NACKs to recover from missing packets, protocols like RBP and RMP have a central site generate positive acknowledgments for every message they receive, and have the role of the central site rotate among group members.

The imm protocol uses a more optimistic approach: servers assume that the transmission of a packet was successful if they don't receive any negative acknowledgments within a pre-defined time interval. Since they currently store a week's

worth of data, imm servers appear as virtually infinite data repositories, and clients can later retrieve missing data.

A transport protocol for weakly consistent applications should use imm's low-overhead approach to reliability. Senders retransmit on demand and keep previously transmitted messages for some pre-defined period of time. Since the application performs periodic consistency checks, it will eventually fix inconsistencies left unresolved by the transport service.

Besides update messages, the multipoint transport protocol also propagates group membership messages. For instance, when a replica running flood-d joins a group, it copies another replica's database and current group membership view. The new replica then sends a join message to all replicas in that replication group. The underlying multipoint transport protocol reliably multicasts the join message to the corresponding multicast group. The joining of a new member spawns a topology calculation, whose result is multicast to the group along with the new membership set. Notice that once the new replica is added to the underlying multicast group (see Section 6.3.4), it starts receiving messages multicast to the group. However, the new replica only starts getting application-level state exchanges from other replicas, when it gets added to the other replicas' group membership view.

When a replica gets disconnected temporarily, its transport entity will request missing updates once the replica gets reconnected to the group. However, if the replica fails and looses its state, only the application can re-establish consistency. Flood-d's periodic probe messages will detect that a replica has failed. In case the failure persists for some time, the failed replica may be deleted from the group membership set and the next topology map. Once the replica comes back up, by re-joining the group, it will re-establish consistency.

6.3.2 Flow and Congestion Control

Like RMP, the proposed transport protocol employs a window-based flow-control mechanism regulated by slow-start and negative acknowledgments. Senders start transmitting with a pre-defined window size and linearly increase it. Upon receiving a negative acknowledgment, senders backoff by exponentially reducing their transmission window size.

Because there are no positive acknowledgments, negative acknowledgments also act as a sign of congestion, and cause senders to exponentially backoff. Recall that in the original TCP slow-start approach, the initial window size is set to 1 packet and gradually stretches as transmitted packets arrive at their destinations and senders have more data to transmit. Since distributed information services source traffic patterns are likely to be bursty, instead of using an initial window size of 1 packet, senders could be more aggressive and use a bigger initial window based on the expected update message size and frequency. When joining a group, a new member agrees to use the current transmission window size.

Besides reducing the number of copies of NACKs and retransmitted packets, broadcasting negative acknowledgments to the group instead of only unicasting them to the sender can also improve the effectiveness of the congestion control mechanism. When the group receives a negative acknowledgment, all its members will adapt by reducing their transmission windows.

6.3.3 Resource Reservation

The Internet's original best-effort service model is inadequate to support the diverse set of existing and future wide-area applications that range from traditional data communication services like electronic mail and remote file transfer to real-time multimedia applications such as audio and video conferencing, and shared whiteboard tools. Since several of these tools are very sensitive to the quality of service the network offers to their packets, the network should allow these applications to request enough resources for their data flows.

There has been considerable research focused on extending the current Internet architecture to support other service models. Besides traditional best-effort, new models include guaranteed, and predictive services. Guaranteed service provides an absolute upper bound on the delay of every packet. However, some real-time applications do not need absolute delay bounds, and only require that some (high) fraction of their packets obey a predictive bound. This type of service called predictive service offers a fairly reliable but not absolute delay bound.

To support different service models, new components need to be added to the current Internet architecture. Data sources need a pre-defined flow specification

language, that enables them to specify to the network the traffic characteristics of their data flows. An example of a flow specification language is described in [47]. For the network to decide whether to grant or deny a reservation request, it needs some kind of admission control mechanism. One of the proposed admission control algorithms is presented in [29]. Once the network accepts a data flow, it needs to set aside the required resources, such as a portion of the available bandwidth or buffer space in the routers. RSVP [68] is one of the resource reservation protocols being proposed. It allows the network to create and maintain resource reservations on each link along multipoint transport connections. The network also needs an accounting and billing infrastructure to produce monetary incentives [12] so that end users can select the service model that best suits their needs at a price they are willing to pay.

Keeping the database of a distributed information service weakly consistent implies that updates must eventually propagate to all replicas, and that every replica is a potential source of updates. However, unlike high data rate applications, source traffic patterns are likely to be bursty, with senders going through shorter active periods followed by longer silent intervals. Furthermore, eventual consistency applications are very sensitive to packet losses but not as sensitive to delay as real-time applications. On one hand, they can live with the network's best-effort service model, and don't need to explicitly reserve resources. In this case, flood-d's estimates of the underlying physical network and the resulting logical topology help in propagating updates efficiently.

On the other hand, some weakly consistent applications may want to have an upper bound on the time updates take to propagate. If they are willing to pay for a more expensive service, they could request predictive bandwidth. In this case, flood-d should probably based its logical topology on propagation delay, rather than available bandwidth.

6.3.4 Application-Level Functionality

From time to time, the application running at each replica needs to perform consistency checks to catch inconsistencies from which the underlying transport protocol could not recover. In particular, every replica running flood-d periodically chooses another replica, with whom it compares its consistency state. Replicas recover from

inconsistencies by getting missing updates from their peers through reliable point-to-point communication. When a replica finds out that all the other replicas have received a specific message, it can then purge that message from its consistency state.

Since the application needs to perform periodic consistency checks, it needs to know the set of members in the group. However, weakly consistent replicas only need to keep weakly consistent views of the group, which will eventually converge to a single consistent view once group membership reaches steady state. In flood-d for example, to join a replication group, a new replica copies another replica's database, which is more efficient than relying on the reliability mechanisms of the underlying multipoint transport service. The new replica also floods its existence to the rest of the group. The joining of a new member spawns a topology calculation, which propagates the updated group membership set along with the new topology.

Internet multicast architectures use the Internet Group Management Protocol (IGMP) [18] to manage multicast groups. Therefore, local application-level joins are mapped to IGMP joins, which inform multicast routers about new group members. Notice that only the application and the underlying internet multicasting protocol need to keep group membership information. The multipoint transport protocol only translates the application's group name to the corresponding multicast group address.

Because different applications have different message ordering requirements, the ordering mechanism should be implemented at the application. This way applications that have looser ordering constraints do not have to pay the performance penalties of more strict ordering requirements.

6.4 Multicast and Replication Groups

Similarly to having a single replication group, having a single multicast group with all service replicas as members will not scale. Since flood-d organizes replicas into multiple groups, replicas within a group can run flood-d on top of a multipoint transport service. In this scenario, each replication group corresponds to a multicast group. Through representative individual replicas in each replication group, or *corner replicas*, flood-d maintains consistency between groups, as well as recovers

from inconsistencies within a group. This scheme fits well with the way IP multicasting is currently deployed on the Internet, where there are regions that can directly support multicast routing connected by point-to-point tunnels.

Alternatively, multicasting can also be used as the primary inter-group propagation mechanism. Each replication group will still be mapped to a multicast group, and each corner replica will be a member of as many multicast groups as replication groups the corner replica interconnects. Another possibility is to set up a multicast group with all the corner replicas, and also have each corner replica be a member of the multicast group corresponding to one of the replication groups the corner replica interconnects. Flood-d will still be responsible for recovering from inconsistencies which the underlying transport service cannot detect.

In the context of the underlying multicast architecture, we can speculate that since updating the replicated database of an information service has low bandwidth requirements when compared to audio and video applications, each replication group could use a shared delivery tree to distribute updates. However, one can argue that since replication groups may densely populate Internet routing domains, intra-group source-specific delivery trees may be more adequate. On the other hand, inter-group communication can be done over a shared delivery tree, since it will likely use backbone links, which are shared anyway.

6.5 Simulations

To evaluate the use of IP multicast as the underlying update propagation mechanism for a weak-consistency replication tool such as flood-d, we modify the simulator described in Chapter 5 so that it multicasts updates to all replicas in a group.

The goal of these simulations is to answer the following question: assuming that IP multicasting is the primary update propagation mechanism, what are the benefits of performing the end-to-end consistency state exchanges over a logical topology that tries to use the underlying physical network efficiently, and, at the same time, reduce update propagation time? Application-level state exchanges with randomly selected peers serve as a basis for comparison.

We use a very simple simulation model, which we discuss in more detail in Section 6.5.2 below. When a replica generates an update, this update will get propagated

to the other replicas via multicast subject to some drop rate. In other words, the multicast drop rate which we use as one of the simulation parameters, determines whether a given update will be dropped or received by all members of a replication group. Periodic application-level state exchanges will fix inconsistencies which the underlying propagation mechanism could not resolve.

6.5.1 Results

These set of simulations record the total communication cost incurred to propagate dropped messages through application-level consistency state exchanges (anti-entropy). These state exchanges used 2 different policies to select the partner with whom to exchange state: cost-based and random. To assign communication costs to links, we use NTG to generate random logical topologies for different group sizes. While the random selection policy, picks anti-entropy partners at random, the cost-based policy chooses from a subset containing replicas with minimum link cost to the replica originating the anti-entropy. The size of the selection set is currently 10% of the replication group size. Note that the bigger this selection set, the higher the link costs it contains. Consequently, the bigger the selection set, the higher the overall cost for propagating messages.

The graphs in Figure 6.1 show how the overall cost to propagate updates to all replicas in a group scales with the multicast drop rate for different group sizes. As expected, the higher the multicast drop rate, the more evident the benefits of propagating updates using a cost-based partner selection policy as opposed to random partner selection.

Furthermore, the bigger the replication group, the bigger the discrepancy in cost between cost-based and random partner selection policies. This is due to the fact that in bigger groups, updates need to travel more logical hops to get to all replicas.

The set of graphs in Figure 6.2 show how the the overall communication cost to propagate updates to all replicas using cost-based and random partner selection policies scale with different update rates. In the middle and bottom graphs, the update rate is twice and four times higher than the top graph's update rate. In all graphs, we plot communication cost as a function of the multicast drop rate and use a replication group size of 100 replicas.

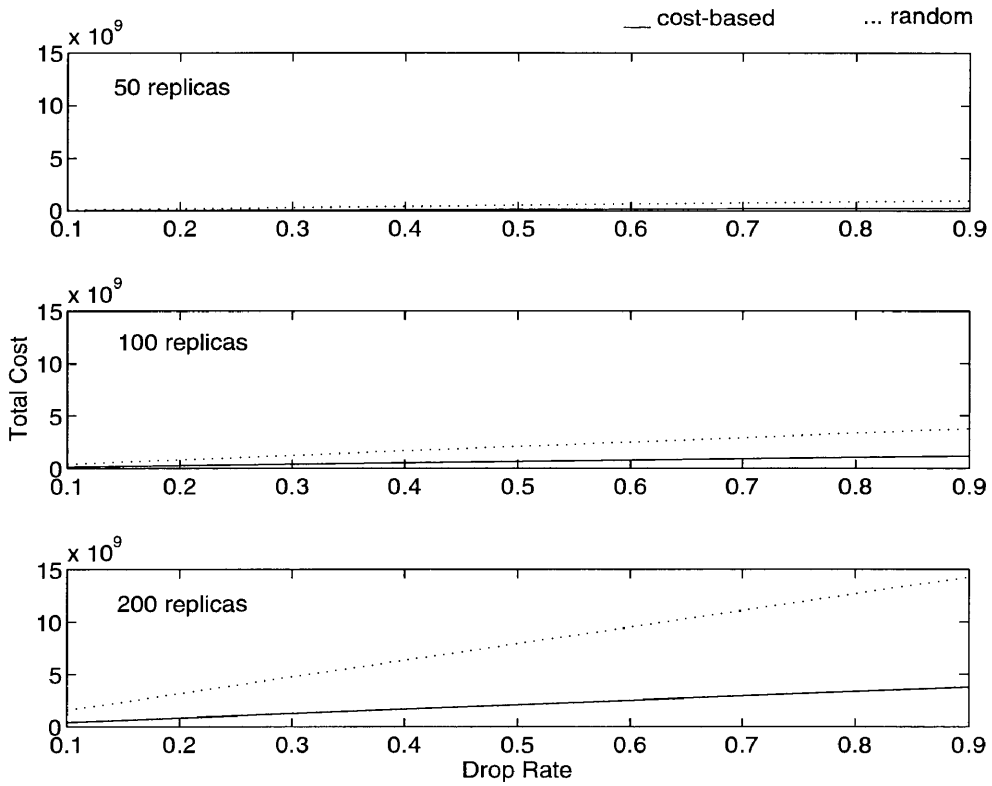

Figure 6.1: Total communication cost for propagating messages to all replicas in a replication group as a function of the multicast drop rate using cost-based and random partner selection policies. The top, middle, and bottom graphs use 50-, 100-, and 200-replica groups, respectively.

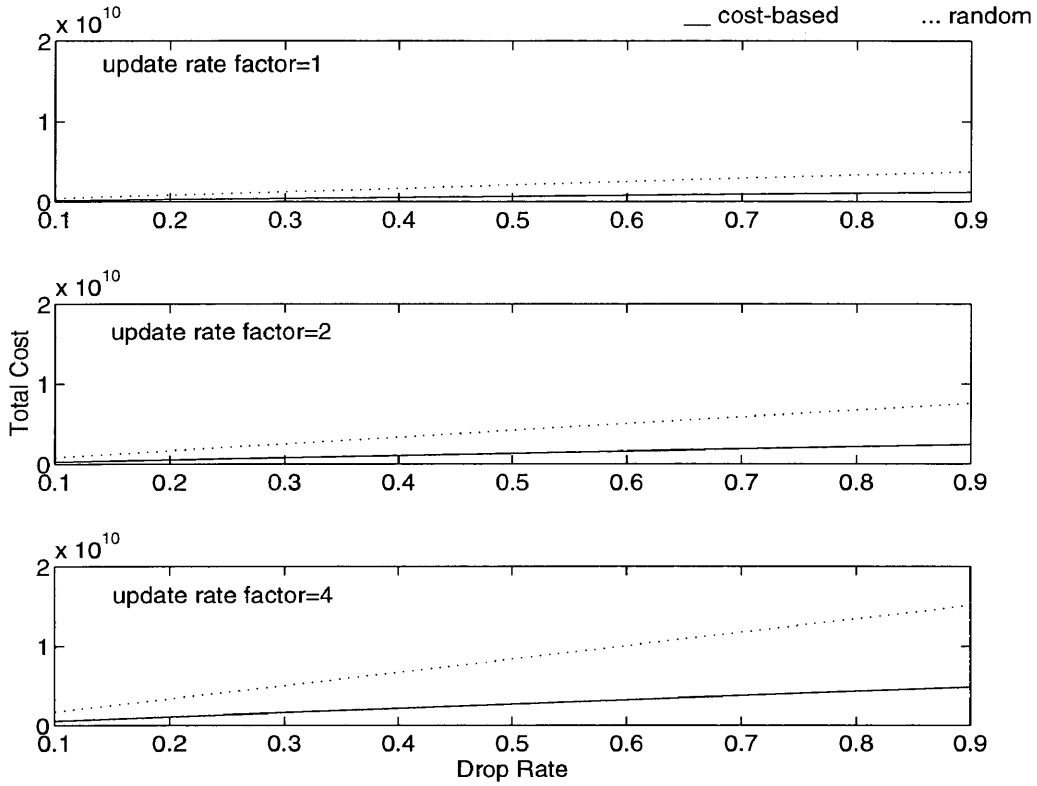

Figure 6.2: Total communication cost for propagating messages to all replicas in a replication group as a function of the multicast drop rate using cost-based and random partner selection policies. The middle and bottom graphs use update rates 2 and 4 times higher than the top graph, respectively.

We observe that for higher update rates the discrepancy in cost between cost-based and random update propagation increases. This means that the higher the update rates, the higher the communication cost reduction if, instead of using a random propagation policy, updates are propagated over a cost-based logical topology.

6.5.2 Simulation Modeling Considerations

The application-level simulations described in the previous section are based on some unrealistic assumptions. First, they use pure IP multicasting without any transport-level reliability. Without a reliable transport service on top of unreliable multicast, reliability detection and recovery is fully performed at the application level. This results in a best-case scenario for using flood-d's logical update topologies since the

amount of application-level exchanges that needs to be done to fix inconsistencies left by the lower-level protocols is maximum.

Modeling multicast unreliability by using arbitrary drop rates is an over-simplifying assumption. We use Bernouilli distributions to determine whether a given update is to be dropped or successfully multicast. Besides using arbitrary Bernouilli probabilities, we also assume that either all replicas in a group receive an update or none of them receive it.

We also assume that every update message can be propagated as a single IP multicast packet. In other words, we simulate update message multicasting and not packet multicasting. While this is coherent with the fact that we are not using a transport level protocol, it does not provide a realistic model for the target applications. For instance, update messages in a replicated file archive service may generate hundreds of packets.

In the sections below, we speculate on how to build a simulation model that could more realistically represent weakly consistent applications that use a replication tool like floodd-d running on top of a reliable multipoint transport service such as the one described in Section 6.3.

6.5.2.1 Source Traffic

Every participating replica is a potential traffic source, which generates a burst of packets corresponding to every client update. Burst frequency and duration correspond to the frequency to which client updates are generated and their size in number of packets, respectively, and are application dependent.

We expect weakly consistent service' update rates to be relatively low at steady state. For instance, in the Domain Name Service (DNS) [39, 40], changing host names and addresses, or adding and deleting new database entries are not likely to happen more than once a month. Rates of one update per hour seem high even for applications such as the Internet's global directory service [6]. Clearly, the ratio between the application-level state exchange rate and the update rate is an important parameter: it determines how fast inconsistencies are detected and patched. The tradeoff is that using higher state exchange rates implies achieving

eventual consistency sooner at the expense of generating more network traffic and consuming more network resources.

The duration of the packet burst corresponding to the transmission of an update depends on the update message size. Replicated directory services or indexing databases will likely generate updates corresponding to bursts of tens of packets. On the other hand, replicated file archives may transmit whole files as updates. The contents of the replicated files, which may range from entire software distributions, textual documents, or even images and sound, determine the size of the updates. One could then argue that update message sizes can be modeled by a bimodal distribution, where fixed-size smaller and bigger updates occur with probabilities p and $1-p$. Since directory service records and replicated files can span a whole spectrum of update sizes, a discrete uniform distribution could be used instead.

6.5.2.2 Multicast Drop Rate

A multicast propagation model should contain information such as the probability that a message is successfully multicast to a given number of replicas. Furthermore, instead of being assigned arbitrarily, these probabilities should depend on parameters such as the underlying physical topology. For instance, if a multicast delivery tree (either source-specific or shared) uses a given physical link to send packets to a subset of replicas in a replication group, the probability that these replicas successfully receive a message multicast over this delivery tree depends on the packet drop rate of the shared physical link.

6.5.2.3 Transport-Level Mechanisms

Using a reliable, flow-controlled, multipoint transport protocol on top of IP multicasting has several implications. Because the transport protocol provides reliable delivery, replicas can recover from losses earlier instead of having to wait for application-level state exchanges. Furthermore, the transport service achieves reliable delivery through selective negative acknowledgments, which means that individual lost packets may be recovered, as opposed to transmitting the whole update, which is what happens at the application level. This is particularly beneficial when client updates generate a large number of packets.

The flow and congestion control mechanisms used by the proposed multipoint transport protocol employ negative acknowledgments and slow start to adjust the transmission window size. A sender transmits enough packets to fill up the current window size or until it does not have any packets to send, and sets the retransmission timer. Because there are no positive acknowledgments, if the sender does not receive any NACK until the timer expires, it assumes that the transmission was successful, and increases its window size. This means that different senders may have different window sizes at a given time. However, if NACKs are multicast to the whole group, all members exponentially backoff when a NACK is received, which causes bigger reductions for senders with bigger windows.

6.5.2.4 Bandwidth

Claiming that as link speed increases, network bandwidth will become abundant is analogous to the argument stating that as computer memory gets cheap, memory will become plentiful for computer applications. However, as the complexity of applications increases, so does their demand for memory and other computer resources. Similarly, current and future network applications will tend to take advantage of the available bandwidth.

Suppose that the network supports different classes of service. Some weakly-consistent application that wants to have an upper bound on the time its updates take to propagate may request the network to reserve the necessary bandwidth for its packets. However, even if there is enough bandwidth to handle the packet bursts produced by every update, packets may still be dropped at the receivers if they cannot process them fast enough, or if they do not have enough buffer space. Therefore, slow receivers may slow the whole group down since they will generate NACKs as packets get dropped, causing the transmission window size to oscillate around a value compatible with their service time and buffer size.

Let's assume that the current Internet is roughly hierarchically structured and uses links of similar speeds at each hierarchical level: slower links connect nodes at the lower hierarchical levels, while nodes at the higher levels are connected through faster links. If Internet services set up their replication groups according to the

Internet's hierarchical structure, then we can consider the physical links connecting replicas within a group to be roughly homogeneous.

However, because link utilization may vary drastically from link to link as well as over time, the actual available bandwidth at each link can be considerably different. For instance, suppose that in a multicast delivery tree, there is a subset of replicas that share the same upstream physical link. If for some reason this link is frequently congested (it may also be used by other applications' replication groups), then the downstream replicas often lose packets and generate negative acknowledgments requesting their retransmission. Therefore, in this case, this shared link becomes a bottleneck, and dictates the group's maximum transmission window size. Link utilization can get even higher when instead of source-specific trees, replication groups use shared delivery trees.

6.6 Summary

In this chapter, we overviewed existing and new internet multicasting architectures and multipoint transport protocols. We argue that most existing multipoint transport protocols are not well-suited for weakly consistent applications. Like the Real-Time Transport Protocol (RTP), they are either unreliable, or like the Reliable Broadcast Protocol (RBP), the Multicast Transport Protocol (MTP), and the Reliable Multicast Protocol (RMP), they provide strong consistency at the expense of latency and overhead.

The features and mechanisms of the reliable multipoint transport protocol we proposed are summarized in Table 6.1. The main idea is that weakly-consistent applications can use a low overhead, reliable multipoint transport service to reduce the amount of application-level state exchanges without compromising the application's availability and response time.

Finally, we evaluated the use of IP multicasting as flood-d's primary update propagation mechanism. As expected, the higher the multicast drop rate, the more evident the benefits of using flood-d's logical topology to perform application-level consistency state exchanges. We also observed that as replication groups get bigger, so does the discrepancy in cost when propagating updates according to cost-based logical topologies as opposed to random topologies.

Features	Mechanisms
Transport-Level Reliability	• Sequence Numbers • Selective NACKs • Selective Retransmissions • NACKs/retransmitted packets can be unicast to sender/requesting receiver or multicast to group. If multicast, scheduling transmission of NACKs and retransmitted packets may be randomized.
Flow and Congestion Control	• Window-Based • Regulated by NACKs and Slow-Start
Service Model	• Best Effort
Application-Level Functionality	• End-to-End Reliability • Ordering • Group Membership

Table 6.1: Reliable multipoint transport protocol for weakly-consistent applications: features and mechanisms.

Chapter 7

Summary and Conclusions

Existing and future Internet information services will provide large, rapidly evolving, highly accessed, yet autonomously managed information spaces. Achieving adequate performance demands that these services and their data be replicated in hundreds or thousands of autonomous networks.

Although the problem of replicating data that can be partitioned into autonomously managed subspaces has well-known solutions, efficient massive replication of wide-area, flat, yet autonomously managed data has yet to be demonstrated. This dissertation investigated scalable replication mechanisms for wide-area, autonomously managed services.

We claimed that efficient replication mechanisms keep replicas weakly consistent by flooding data between them. While existing replication algorithms use flooding to propagate updates among their replicas, they do not address the scalability and autonomy of today's internetworks. For instance, Lampson's widely cited Global Name Service manages a single, flat group of replicas. While this works for applications with 20 to 30 replicas operating within single administrative boundaries, it is unrealistic for wide-area, massively replicated services whose replicas spread throughout the Internet's thousands of administrative domains. Furthermore, like other existing replication mechanisms, Lampson's Global Name Service ignores network topology issues. It assumes the existence of a single administrator who hand-configures the topology over which updates travel. The Global Name Service administrator places replicas in a logical ring, and reconfigures the ring every time a replica is added or removed. As the number of replicas grows and replicas spread beyond single administrative boundaries, reconfiguring the ring gets prohibitively expensive.

The architecture we proposed extends existing replication algorithms to address scalability and autonomy explicitly. We target replication degrees of thousands or even tens of thousands of weakly consistent replicas scattered throughout the Internet's thousands of administrative domains.

In the following sections, we summarize our contributions, present our plans for continuing work, and directions for future research.

7.1 Contributions

Lampson's Global Name Service (GNS) has been widely-accepted as the solution to the problem of replicating wide-area, distributed, weakly-consistent applications. However, almost 10 years ago, when Lampson claimed GNS could "encompass all the computers and all the people that use them", he did not anticipate that the Internet would grow as fast as it did. The main contribution of this dissertation is to make GNS-like services work in today's exponentially-growing, autonomously-managed internetworks.

Our hierarchical, network-cognizant architecture which has been implemented as a replication tool we called *flood-d*, provides a flooding-based, group communication layer information services can use to efficiently propagate data among their replicas. It organizes replicas into multiple replication groups. This multiple group organization limits the size of the consistency state each replica needs to keep. It also preserves autonomy since it insulates groups from other groups' administrative decisions, and from most of the network traffic associated with group membership. Replication groups overlap with each other, so that an update that originates in a group makes its way to all groups.

Using client updates and probe messages, group replicas estimate the underlying physical topology. Based on the estimated physical topology, our architecture builds a *logical update topology* connecting the members of a replication group. The resulting update topology is *k-connected* for resilience, uses the physical topology efficiently, and has limited diameter to restrict update propagation delays. Whenever the probing mechanism detects a significant change in the underlying physical topology or any group membership change, it spawns a topology computation process, which updates the group's logical topology accordingly.

Through simulations, we investigated several points in the design space of replication algorithms. First, we simulated Golding's Timestamped Anti-Entropy (TSAE) algorithm, and studied how group size, replica availability, and the frequency with which replicas exchange consistency state affect the algorithm's performance. Our simulation results demonstrate the benefits of splitting a single, flat replication group into multiple, smaller groups. The smaller the replication group, the smaller the size of the consistency state each replica needs to keep. Replicas' consistency state grows and shrinks as new updates are generated and old ones are purged. The difference in consistency state size is amplified because in smaller groups, replicas can purge their consistency state sooner. Our simulations also showed that smaller replication groups attenuate the effect of less frequent state exchanges and lower replica availability.

The other set of points in the design space we investigated consisted of incorporating the notion of a logical update topology to an existing replication algorithm, such as Golding's TSAE, and then using TSAE with topology and multiple replication groups. We assigned communication costs to the logical links connecting pairs of replicas, and contrasted random and cost-based update propagation. Our simulations showed that while for a 50-replica group, propagating updates over a low-cost logical topology is about 3 times less expensive than when a random topology is used, this cost ratio jumps to 7 for 200-replica groups.

Finally, we studied the use of internet multicast as our replication tool's primary update propagation mechanism. We claimed that a reliable multipoint transport protocol on top of IP multicast cannot provide the reliability information services need. In particular, a reliable multipoint transport protocol does not solve consistency problems such as the one associated to temporarily removing a replica from service, and consequently losing its consistency state. In these cases, only the application can re-establish consistency. Since service replicas need to perform periodic application-level consistency checks, our replication tool does not rely on a reliable multipoint transport protocol, but for efficiency, can exploit it where available.

Since having a single multicast group with all service replicas will not scale, we argued that each replication group should be mapped to a multicast group. *Corner replicas* can use flood-d to propagate updates between groups, or can be members of a separate multicast group. Using this multicast group, which contains only corner

replicas, updates from one group are multicast to the other groups through the corresponding corner replicas. Like group replicas, corner replicas use flood-d to periodically perform point-to-point consistency checks between themselves.

Using a simple simulation model, we evaluated the use of IP multicasting as flood-d's primary update propagation mechanism. We used multicast drop rate and group size as simulation parameters. As expected, the higher the multicast drop rate, the more evident the benefits of using flood-d's logical topology to perform application-level consistency checks. We also observed that as replication groups get bigger, so does the discrepancy in cost when propagating updates according to cost-based logical topologies as opposed to random topologies.

The other contribution of this work is having proposed and solved the graph theory problem associated with building our logical update topologies. Our algorithm uses as input the required connectivity and the communication cost matrix containing flood-d's estimates of the underlying physical topology. The algorithm computes a random starting topology, and uses simulated annealing as the optimization technique in finding a low edge cost, low diameter topology. Our simulations showed that our algorithm's initial topologies already have low total edge cost, and therefore the resulting topologies did not show considerable total edge cost reductions. However, we achieved diameter reductions of more than 30% in edge cost and more than 50% in hop count.

We also showed that since the initial topologies our algorithm generates already have low total edge cost, an alternate approach to producing a low edge cost, low diameter topology is to select some extra edges and add them to the initial topology. We investigated different edge selection criteria, and showed that by adding a couple of edges, we can get significant diameter reductions without incurring a considerable total edge cost increase. We argued that although this heuristic approach constitutes a practical way of solving our logical update topology problem, only after having used our optimization algorithm could we verify that adding selected edges to our initial topologies did indeed generate acceptable topologies. Furthermore, we showed that we can use our algorithm with different cost functions to solve other topology optimization problems.

7.2 Continuing Work and Future Directions

We have conducted wide-area experiments in which a dozen sites in a single group replicate a directory tree. In our continuing work, we plan to experiment with multiple, bigger groups, replicating bigger objects such as the contents of Harvest brokers. We will then measure flood-d's performance including the overhead it generates.

These wide-area, big group experiments will also allow us to evaluate how well our network topology estimates reflect the state of the underlying physical network. It will also allow us to tune our logical topology computation algorithm. So far we have been feeding it with randomly-generated physical topologies. Once we have Harvest deployed on the Internet, we will have real data on reasonable sized topologies to drive our algorithm. We also plan to experiment with different objective functions to control the annealing schedule. For instance, we plan to use average edge cost, so that sites that are connected through slow links do not become transit nodes.

Another direction for future work is to incorporate in our replication tool the option of using internet multicast to propagate updates. This includes the design and implementation of a streaming, reliable multipoint transport protocol that suits the needs of weakly consistent applications. Flood-d will use multicast as its primary update propagation mechanism. Periodically, replicas will perform end-to-end consistency checks using flood-d's logical topology to select neighbors with whom to exchange consistency state.

Other directions for future research include investigating the dynamics of groups. For example, when a new replica is added to a service, it needs to know to which group it should join. This is a resource discovery problem in itself. A service can advertise its existence by registering itself with a directory of services. This service's entry in the directory of services' database could keep a list of the existing replication groups along with some information about them, such as the group's geographic location, and group size. When a new replica wants to join the service, the replica will query the directory of services, and then based on the information about that service's existing groups, can decide where to join.

Other group dynamics issues include how big a group can get before it is split in two, or when should 2 or more groups be coalesced into one group. By having

multiple replication groups, the resulting topological rearrangements will be limited to the groups involved.

Reference List

[1] R. Aiello, E. Pagani, and G.P. Rossi. Design of a reliable multicast protocol. *Proc. of the IEEE Infocomm'93, San Francisco, CA*, pages 75–81, March 1993.

[2] M. Andreessen. NCSA Mosaic technical summary. Techcical Report, available via Mosaic from ftp://zaphod.ncsa.uiuc.edu/Web/mosaic-papers/mosaic.ps.Z, May 1993.

[3] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol. Internet Request for Comments RFC 1301, February 1992.

[4] A.J. Ballardi, P.F. Francis, and J. Crowcroft. Core based trees (cbt). *Proc. of the ACM SIGCOMM'93, San Francisco, CA*, August 1993.

[5] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* North Holland, 1976.

[6] M. Bowman, P. B. Danzig, U. Manber, and M. Schwartz. Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8), August 1994.

[7] L. Breslau and D. Estrin. Design of inter-administrative domain routing protocols. *Proceedings of the ACM SIGCOMM '90*, pages 231–241, September 1990.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol. Internet Request for Comments RFC 1067, August 1988.

[9] S. Casner. Frequently asked questions (FAQ) on the multicast backbone (MBONE). On-line documentation; available from venera.isi.edu:mbone/faq.txt, January 1993.

[10] J.M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[11] D.R. Hardy U. Manber C.M. Bowman, P.B. Danzig and M.F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS 732-94, Department of Computer Science, University of Colorado-Boulder, July 1994.

[12] R. Cocchi, D. Estrin, S. Shenker, and L. Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transcations on Networking*, pages 614–627, December 1993.

[13] J.R. Cooperstock and S. Kotsopoulos. Adaptive flow control for reliable high volume group communications. Technical Report, Department of Electrical and Computer Engineering, University of Toronto, July 1994.

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press and McGraw-Hill.

[15] W. Dang and T.N. Nielsen. The imm protocol. On-line source code, anonymous ftp from ftp.Hawaii.Edu:/paccom/imm-3.3, May 1994.

[16] P. B. Danzig. *Optimally Selecting the Parameters of Adaptive Backoff Algorithms for Computer Networks and Multiprocessors.* PhD thesis, University of California, Berkeley, December 1989.

[17] P.B. Danzig, K. Obraczka, and A. Kumar. An analysis of wide-area name server traffic: A study of the domain name system. *Proc. of the ACM SIGCOMM '92, Baltimore, Maryland*, August 1992.

[18] S. Deering. Host extensions for IP multicasting. Internet Request for Comments RFC 1112, August 1989.

[19] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[20] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–111, May 1990.

[21] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. An architecture for wide-area multicast routing. Submitted to ACM Sigcomm 1994, February 1994.

[22] A. Emtage and P. Deutsch. archie: An electronic directory service for the Internet. *Proceedings of the Winter 1992 Usenix Conference*, pages 93–110, January 1992.

[23] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM Transactions on Computer Systems*, 9(3):242–271, August 1991.

[24] R. A. Golding. *Weak-Consistency Group Communication and Membership.* PhD thesis, University of California, Santa Cruz, December 1992. Computer and Information Sciences Technical Report UCSC-CRL-92-52.

[25] C.L. Hedrick. Routing information protocol. Internet Request for Comments RFC 1058, June 1988.

[26] S. Hotz and R. Nagamati. Network topology generator (NTG): A tool for generating network topology and policy for protocol simulation purposes. Technical Report, Computer Science Department, University of Southern California, Spring 1992.

[27] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[28] V. Jacobson. traceroute(8) manual page. SunOS Release 4.1 Reference Manual, Sun Microsystems, February 1989.

[29] S. Jamin, S. Shenker, L. Zhang, and D. Clark. Admission control algorithm for predictive real-time service. *Proceedings on the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1992.

[30] D.S. Johnson. The complexity of network design problem. *Networks*, 8:279–285, 1978.

[31] B. Kantor and P. Lapsley. Network news transfer protocol - a proposed standard for the stream-based transmission of news. Internet Request for Comments RFC 977, February 1986.

[32] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, May 1983.

[33] B. Lampson. Designing a global name service. *Proceedings of the 5th. ACM Symposium on the Principles of Distributed Computing*, pages 1–10, August 1986.

[34] K. Lidl, J. Osborne, and J. Malcolm. Drinking from the firehose: Multicast USENET news. *Proceedings of the 1994 Winter USENIX Conference*, 1994.

[35] K. Lougheed and Y. Rekhter. Border gateway protocol. Internet Request for Comments RFC 1163, June 1990.

[36] S. McCanne. A distributed whiteboard for network conferencing. UC Berkeley report, May 1992.

[37] K. McCloghrie and M. Rose. Management information base for network management of TCP/IP-based internets. Internet Request for Comments RFC 1066, August 1988.

[38] D.L. Mills. DCN local-network protocols. Internet Request for Comments RFC 891, December 1983.

[39] P. Mockapetris. Domain names - concepts and facilities. Internet Request for Comments RFC 1034, November 1987.

[40] P. Mockapetris. Domain names - implementation and specification. Internet Request for Comments RFC 1035, November 1987.

[41] P. Mockapetris and K. Dunlap. Development of the Domain Name System. *Proc. of the ACM SIGCOMM '88*, pages 11–21, August 1988.

[42] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Smith. Andrew: A distributed personal computing environment. *Communications of the ACM*, 29(3):184–201, March 1986.

[43] J. Moy. The open shortest path first (OSPF) specification. Internet Request for Comments RFC 1131, August 1989.

[44] J. Moy. Multicast extension to OSPF. Internet Draft, September 1992.

[45] K. Obraczka, P. Danzig, and S.H. Li. Internet resource discovery services. *IEEE Computer*, 26(9):8–22, September 1993.

[46] D. Oppen and Y. Dalal. The Clearinghouse: A decentralized agent for locating named objects in a distributed environment. *ACM Transactions on Office Information Systems*, 1(3):230–253, July 1983.

[47] C. Partridge. A proposed flow specification. Internet Request for Comments RFC 1363, March 1992.

[48] S. Pingali, D. Towsley, and J. F. Kurose. A comparison of sender-initiated and receiver initiated reliable multicast protocols. *Proc. of the IEEE Infocomm'94, Toronto, Canada*, June 1994.

[49] J. Plesnik. The complexity of designing a network with minimum diameter. *Networks*, 11:77–85, 1981.

[50] D.C. Plummer. An Ethernet address resolution protocol. Internet Request for Comments RFC 826, November 1982.

[51] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, and G. Rudisin ang G. Thiel. LOCUS: A network transparent, high reliability distributed system. *Proc. of the 8th. Symposium on Operating Systems Principles*, pages 169–177, December 1981.

[52] J. Postel. Internet control message protocol. Internet Request for Comments RFC 792, September 1981.

[53] J. Postel. Internet protocol. Internet Request for Comments RFC 791, September 1981.

[54] C. Rose. Low mean internodal distance network topologies and simulated annealing. *IEEE Trans. Commun.*, pages 1319–1326, 1992.

[55] E. C. Rosen. Exterior gateway protocol (EGP). Internet Request for Comments RFC 827, October 1982.

[56] I. Rouvellou and G.W. Hart. Topology identification for traffic and configuration management in dynamic networks. *IEEE INFOCOM*, pages 2197–2204, 1992.

[57] J.H. Saltzer, D.P. Reed, and D.D. Clark. End-To-End arguments in system design. *Proceedings of the 2nd International Conference on Distributed Systems*, pages 509–512, April 1981.

[58] M. Satyanarayanan. Scalable, secure, and highly available distributed file access. *Computer Magazine*, 23(5):9–21, May 1990.

[59] M. Schroeder, A. Birrell, and R. Needham. Grapevine: An exercise in distributed computing. *Communications of the ACM*, 25(4):260–274, April 1982.

[60] M. Schroeder, A. Birrell, and R. Needham. Experience with Grapevine: The growth of a distributed system. *ACM Trans. on Computer Systems*, 2(1):3–23, February 1984.

[61] H. Schulzrinne. A transport protocol for real-time applications. Internet Draft, Internet Engineering Task Force, Audio-Video Transport WG, March 1993.

[62] Ulrich Schumacher. An algorithm for construction of a k-connected graph with minimum number of edges and quasiminimal diameter. *Networks*, 14:63–74, 1984.

[63] M. Schwartz, M. Bowman, P. Danzig, and U. Manber. IRTF/ARPA resource discovery project design specification. Version 1, January 1994.

[64] M.F. Schwartz, D.H. Goldstein, R.K. Neves, and D.M. Wood. An architecture for discovering and visualizing characteristics of large internets. Technical Report CU-CS-520-91, Department of Computer Science, University of Colorado, Boulder, Colorado, February 1991.

[65] K. Steiglitz, P. Weiner, and D.J. Kleitman. The design of minimum-cost servivable networks. *IEEE Trans. Circuit Theory*, CT-16(4):455–460, November 1969.

[66] A.W. Tanenbaum. *Computer Networks.* Prentice-Hall, Inc. Second Edition.

[67] B. Whetten and S. Kaplan. A high performance totally ordered multicast protocol. Submitted to ACM Sigcomm 1994, February 1993.

[68] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource ReSerVation protocol. Internet Draft.