Massively Replicating Services in Wide-Area Internetworks

Peter B. Danzig, Dante DeLucia, Katia Obraczka
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
{danzig,dante,kobraczk}@usc.edu
213-740-4780 (Office) 213-740-7285 (Fax)

Abstract

Current and future Internet services will provide a large, rapidly evolving, highly accessed, yet autonomously managed information space. Internet news, perhaps, is the closest existing precursor to such services. It permits autonomous updates, is replicated at thousands of autonomously managed sites, and manages a large database. It gets its performance through massive replication.

This paper proposes a scalable mechanism for replicating wide-area, autonomously managed services. We target replication degrees of tens of thousands of weakly-consistent replicas. For efficiency, our mechanism probes the network and computes a good *logical topology* over which to send updates. For scalability, we organize replicas into hierarchical *replication groups*, analogous to the Internet's autonomous routing domains. We argue that efficient, massive replication does not have to rely on internet multicast.

1 Introduction

Future Internet services will manipulate large, rapidly evolving, highly accessed, yet autonomously managed information spaces. To achieve adequate performance services will have to replicate their data in thousands of autonomous networks. For a current example of such a service, take Internet news. Although it manages a highly dynamic, flat, gigabyte database replicated at thousands of autonomous administrative domains, it responds to queries in seconds. In contrast, archie [5], 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. We believe that archie and other upcoming network services [3] need support to replicate their data efficiently. To appreciate this, we examine existing solutions to replicate data. We will see that, although they are correct and keep replicas consistent, they do not address the scale and autonomy of today's internetworks.

This paper extends an existing replication algorithm to use hierarchical replication groups. It describes a tool that we have built that delivers an update to all of a service's replicas in all of the replication groups. This tool, which we call *flood-d*, floods updates along a logical, update topology. For each replication group, flood-d determines a logical update topology that attempts to minimize the network cost and propagation time needed to transmit updates.

Below, we examine existing replication algorithms. We start by briefly overviewing directory consistency.

1.1 Directory Consistency

Group communication mechanisms for wide-area, massively replicated information services should not trade availability and response time for globally ordered delivery [6]. On one hand, these services still need to guarantee 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, Grapevine [17], the Global Name Service [11], and Network News [9] use weak consistency replication mechanisms. For these reasons we built our replication tool to support asynchronous, weak-consistency replication protocols.

1.2 What Current Algorithms Lack

Existing replication solutions do not scale because they manage single, flat groups of replicas. Distributed systems that scale are organized hierarchically to exploit locality of reference. Grapevine, the Clearinghouse, and the Global Name Service 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. 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 flood updates throughout all physical links of the network. Instead, the replicas flood data to their logical neighbor or peer replicas. Although the word "flooding" sounds inefficient, we claim that the application-level flooding scheme that we propose does use network bandwidth efficiently.

Because layered network protocols hide the network topology from application programs, replicas themselves cannot select their flooding peers to optimize use of the network. Both Grapevine and its commercial successor, the Clearinghouse [13] 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.

Internet news employs flooding to distribute updates among its thousands of replicas. Like the Global Name Service, NNTP site administrators hand-configure their logical flooding topology. 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, specially as the Internet's scale and complexity increase.

1.3 Internet Multicast

Most people think of internet multicast and reliable multicast transport protocols as good foundations on which to build a massive data replication protocol.

IP multicasting [4] delivers best-effort datagrams to a group of hosts sharing a single IP multicast address. Because the network itself supports IP multicast, it has access to the network routing database. IP multicasting builds a minimum delay topology to transmit a datagram packet to the group of recipients. It optimizes delivery delay rather than link bandwidth utilization. Currently, IP multicasting is limited to operate within a single routing domain. It uses virtual point-to-point links, or *tunnels*, to transmit multicast packets between multicast routers in different routing domains. Before transmitting multicast packets through a tunnel, the source multicast router encapsulates them, so that they look like ordinary datagrams to intermediate routers and subnets. Take for example the MBONE [2], 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 consists of *islands*, such as multicast LANs, that can directly support multicast routing linked by tunnels.

Like all lower-level 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 new transport protocol for multiparticipant real-time applications (RTP) [18] provides end-to-end delivery for one or more real-time data flows. It assumes an 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.

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 multipoint transport protocol based on IP multicasting can not meet these stringent requirements. In particular, it can not 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 [16]; 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 algorithm does not rely on a reliable, multicast transport protocol, although it can exploit it where available.

Recently, a new multicast transport protocol, *Muse*

[12], 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.

1.4 Timestamped, Anti-Entropy Replication

Golding modified Grapevine's consistency maintenance protocol to eliminate its garbage collection problems and use it as a replication algorithm. He named the modified algorithm *Timestamped Anti-Entropy (TSAE) Protocol* and used it to build a replicated, distributed bibliographic database system [6]. Like other good replication algorithms, TSAE floods updates.

Periodically, a replica starts an *anti-entropy session*, in which it selects a peer to exchange updates. Through these anti-entropy sessions, the TSAE protocol ensures that replicas eventually converge to a consistent state during normal operation or when recovering from link failures, replica failure, and network partitions. However, like the other replication mechanisms, TSAE was not designed to scale to thousands of autonomous replicas.

1.5 Outline

We propose a multi-point, hierarchical replication group tool that extends the TSAE protocol to address scalability and autonomy. The next section describes our replication tool, *flood-d* [1], and reviews the state it keeps. It also describes how flood-d probes the physical network during normal data exchange for later use. Section 3 describes how flood-d computes logical update topologies and shows some preliminary results. Section 4 presents simulation results that show how flood-d's state and time to propagate items grows with replication group size, replica failure rate, and parameters of the TSAE algorithm.

[1] We copied the way UNIX daemons are named and called our replication tool *flood-d* because it is implemented as an update flooding daemon

2 Flood-d

Flood-d is designed with autonomy and scalability in mind. 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 minimizes the time to reach a consistent state.

Flood-d's multi-point communication layer guarantees reliable delivery, although we recommend that information systems built on top of it should check for inconsistency themselves. Grapevine did this final, end-to-end check for a simple reason. It's easier to recover from inconsistency than it is to guarantee that every autonomous system administrator properly configures his replicas. Flood-d's lower layer propagates updates from peer to peer along end-to-end paths that it perceives are good. We call the graph of these paths a logical update topology.

Unlike Lampson's Global Name Service, flood-d's logical topology is not restricted to a Hamiltonian cycle. Flood-d builds k-connected topologies for resiliency. Recall that in a k-connected network [1], that 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. Flood-d measures available peer-to-peer bandwidth and flood-d's logical topology calculator tries to build a high bandwidth, update topology. It also tries to limit the topology's diameter, the maximum number of hops that updates need to travel. This means that flood-d's 2-connected topologies are not cycles, but more star-like. Hence, flood-d offloads logical topology decisions from system administrators.

2.1 Implementation

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. Occasionally, a replica exchanges updates with one or more non-neighbor replicas within its replication group.

Flood-d estimates available peer-to-peer bandwidth when a replica sends (via TCP) an update to another replica. Hence flood-d 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.

Any flood-d replica can initiate topology calculation. It spawns a process that collects cost estimates from

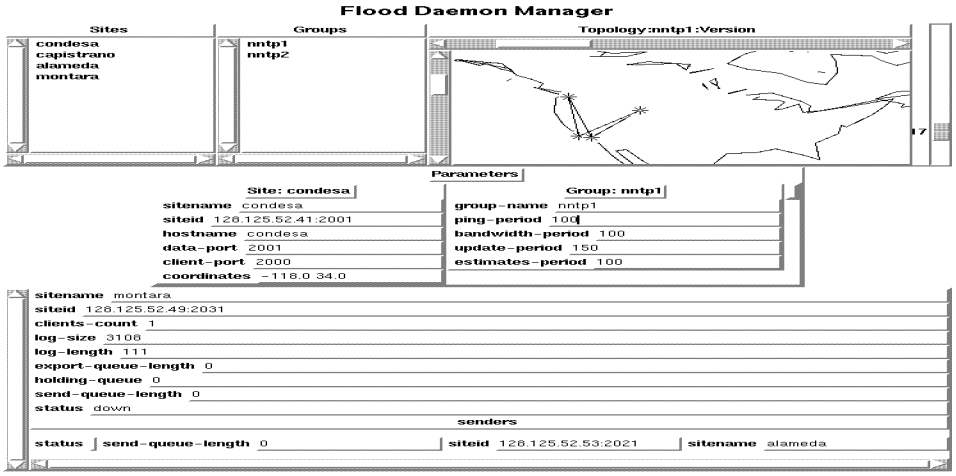

Figure 1: Flood-d's group monotoring tool.

the replicas in its group and then computes the all-pairs shortest-paths between group members. 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, it knows it must re-flood the update as if it hadn't received it before.

To join a replication 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.

Figure 1 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.

2.2 Groups and Network Topology

Figure 2 illustrates the relationship between logical topologies and the underlying physical topologies. The left-hand side of Figure 2 shows three replication groups and their logical update topologies. The right-hand side of Figure 2 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. The logical topology, hopefully, does not send the same data too many times over the same physical links. We should point out that using a logical update topology does not circumvent Internet routing. On the contrary, network routing can work around occasional bad choices made by the update topology.

The left-hand side of Figure 3 illustrates a sample configuration for a flood-d replica. In particular, this is the configuration for Figure 2's replica **w**. In the right-hand side of Figure 3, we show the configuration for the replication group of which replica **w** is a member.

2.3 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

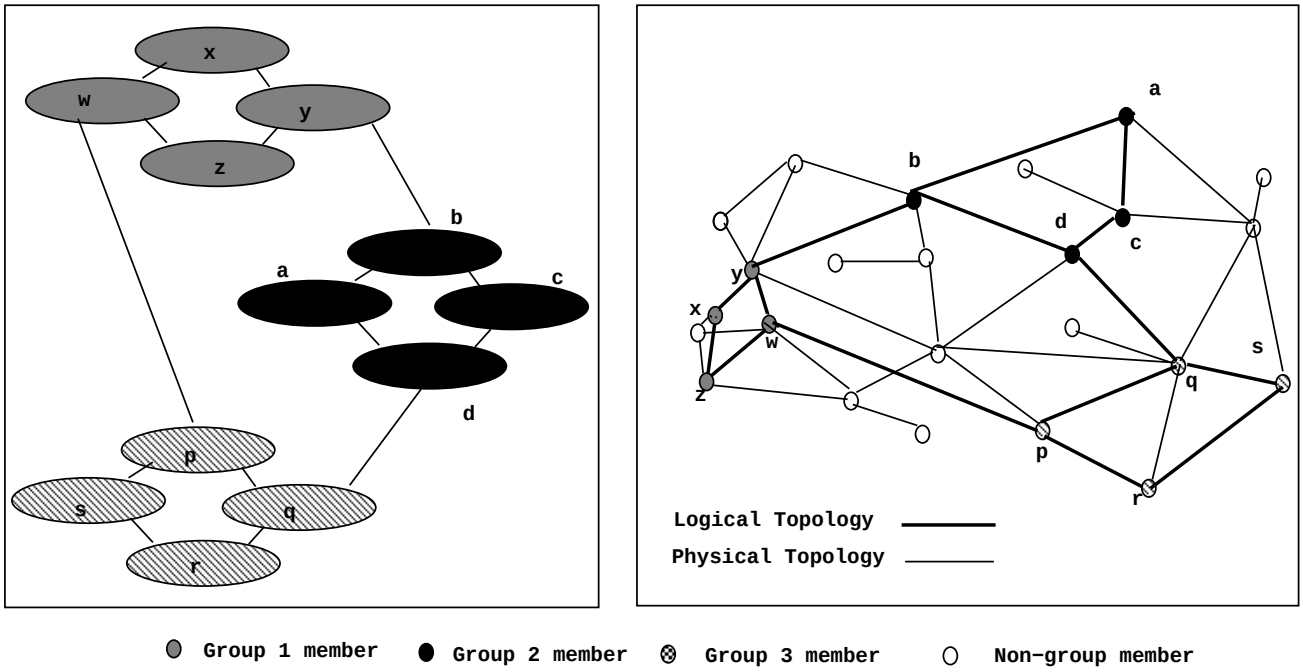

Figure 2: Replication groups showing logical versus physical topologies.

inconsistencies, TSAE eventually resolves them.

2.4 Consistency State Size

A hierarchical organization 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 group size and n is the number of unpurged 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.

3 Computing Logical Topologies

Below, we state flood-d's logical topology computation as a graph theory problem, summarize solutions to similar problems, and describe our algorithm for solving it.

3.1 Definitions

We provide several useful definitions [1] below:

- **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 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.

3.2 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 that repre-

```
;
; 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 3: Example flood-d site and group configuration.

sent network nodes and E is a set of edges of G that represent 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' has node connectivity k.
- Let D_i be the degree of node i of G'. Then,

$$D_i \leq \delta$$

  where δ is an upper bound on the node degree of G'.

- The weighted sum of G's diameter and edge cost function is minimal.

This optimization problem is NP-complete, but the literature records approximations for similar problems.

Plesnik [14] 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-hard.

Johnson [8] 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-hard.

Schumacher [19] 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.

Steiglitz [20] proposes 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 [20] not only fulfills our connectivity requirements but 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-hard problems.

We extended Steiglitz's algorithm to fulfill all our topology requirements. The original algorithm and our modifications to it are described in detail below.

3.3 Steiglitz's Algorithm

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 local transformations called *X-change*, which randomly selects four nodes connected pairwise and swaps the edges connecting them (see Figure 4). 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. Steiglitz justified his algorithm as follows: as the size of the optimization problem increases, local optima get closer to the global optimum, and worst-case time and sub-optimality becomes less important than average-case performance.

3.4 Our Modifications

Steiglitz's redundancy matrix, $R_{i,j}$, can represent our topology's connectivity requirements. Furthermore, specifying the node connectivity imposes a lower bound on the degree of each node. To distribute work fairly among nodes and to limit the amount of update duplicates, we extended Steiglitz's algorithm to include upper bounds on node degrees.

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 [10].

In [15], 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. To generate logical topologies with low edge costs and small diameters, we use

$$total\_cost = A * edge\_cost + B * diameter$$

where A and B are weighting constants assigned to *edge_cost* and *diameter*, respectively.

We feed our logical topology calculator with participating nodes' redundancy requirements and the estimated communication cost matrix. An initial feasible configuration is generated randomly.

Number of Nodes	2-connected		3-connected	
	Initial	Final	Initial	Final
100	15000	11900	23200	19900
50	8900	6900	13000	10500
30	7400	5600	11800	9800
25	6200	4700	9800	7500

Table 1: Logical topology costs before and after annealing.

The next step is to apply transformations to the initial configuration. One type of transformation is the Steiglitz's X-change operation. We also use *Delete*, *Split*, and *Move* transformations. *Delete* randomly choses a pair of connected nodes with degree greater than the required connectivity and deletes the edge connecting them. *Create* randomly selects a pair of connected nodes, breaks the edge connecting them and connects each of them to another node. *Move* randomly selects a node and one of its edges, disconnects it and connects the node with another one. Figure 4 illustrates these four transformations.

The annealing schedule decides whether the configuration resulting from a transformation should be accepted or not. First it tests feasibility. If the new configuration is not feasible, annealing restores the previous configuration, and goes back to the transformation step. Otherwise, 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 according to the Boltzman probability distribution.

Table 1 records results of running the annealing algorithm to create various 2-connected and 3-connected groups. The initial cost column lists our initial feasible topology's cost. The final costs column lists the final logical topology's cost. While cost reductions are not dramatic, this is true because our initial feasible solutions are pretty good. In our simulations, we generated physical topologies by placing nodes in a square plane and randomly onnecting pairs of vertices. We plan to tune the topology computation algorithm once we conduct larger, live experiments.

4 Replication Groups and Logical Topologies

We investigated via simulation how replication group size, the percentage of time replicas are down, and the parameters of the TSAE 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

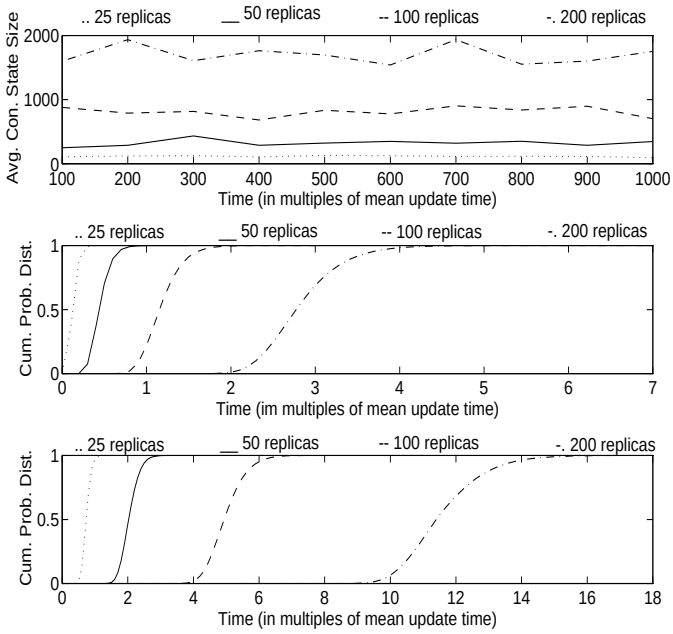

Figure 5: Average consistency state size (in number of update messages), time to propagate updates and acknowledgments for different group sizes. In all 3 graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The global update, anti-entropy, and failure rates were kept constant.

smoother. This means that smaller groups achieve a consistent state sooner.

4.2 Anti-Entropy Rate

The 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 the group anti-entropy rate. Notice that the anti-entropy rate was kept constant for different group sizes [2].

In Figures 6, 7, and 8, anti-entropy rates are twice and equal to the update rate in the top and bottom graphs, respectively. In all graphs, the simulation parameter corresponding to the percentage of time replicas stay up is set to 99.9%. 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 4.3 below. Figure 6 shows the average consistency state size for different anti-entropy rates and group sizes. We notice that as state exchanges get less frequent, saving 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 7 and 8 show that less frequent state exchanges cause updates and acknowledgments to propagate slower. Because we did not model flood-d's normal message exchange, 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 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 9, 10, 11 plot consistency state size, time to reach global consistency, and time to purge all logs for three global anti-entropy rates. The higher the anti-entropy rate, the smaller the consistency state size, the faster consistency is obtained and state purged.

4.3 Replica Availability

Lower replica availability increases the consistency state size and the time to propagate updates and acknowledgments. In Figures 12, 13, and 14, replicas stay up 99% and 90% of the time in the top and bottom graphs, respectively. The lower the 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 13 and 14 show the times to propagate updates and purge state for lower availability. As expected, lower availability causes 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 will replicas 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 lower site or link availability.

5 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

[2] 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.

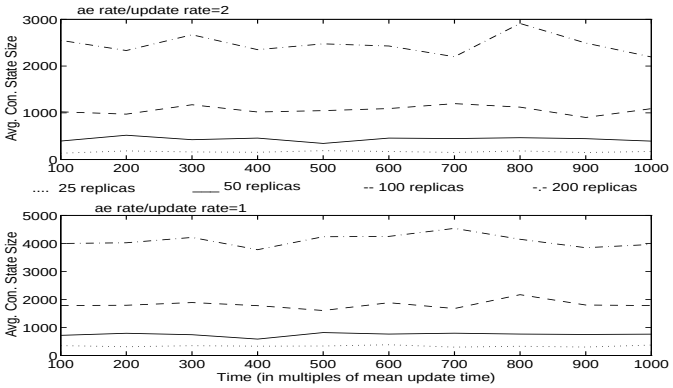

Figure 6: Average consistency state size sample paths (in number of update messages) for different group sizes. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

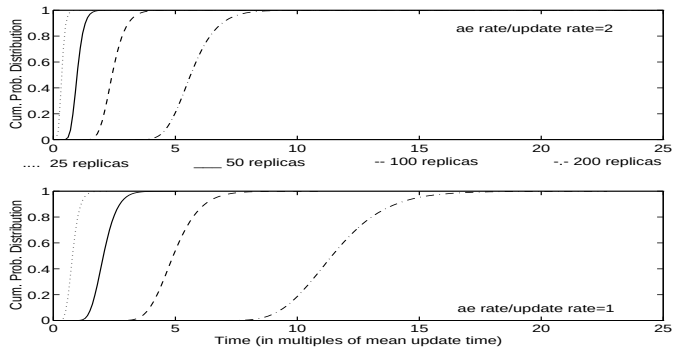

Figure 7: Cumulative probability distribution for propagating updates to all replicas consistently. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

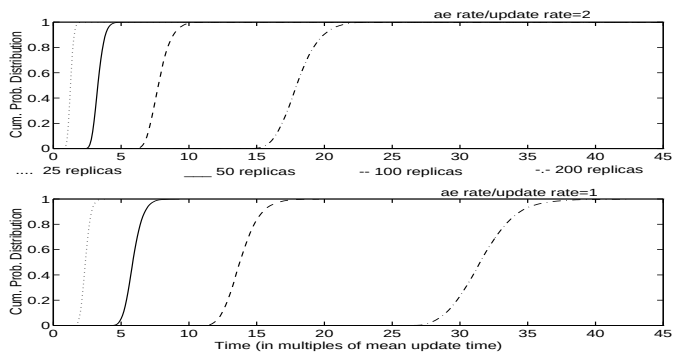

Figure 8: Cumulative probability distribution for receiving acknowledgments from all replicas and purging consistency state. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. The upper graph employs a faster anti-entropy rate (ae rate).

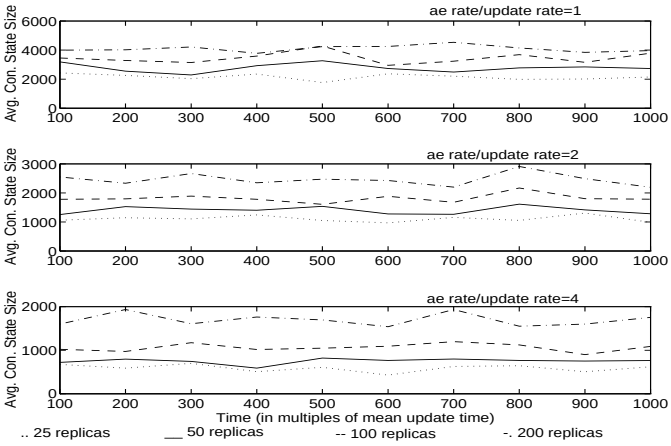

Figure 9: Average consistency state size (in number of update messages). In all three graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

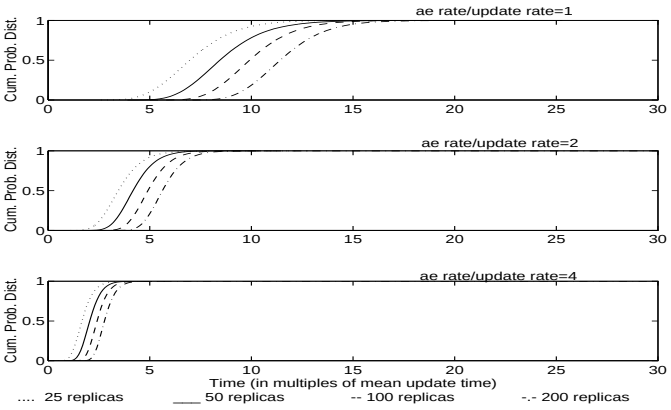

Figure 10: Cumulative probability distribution for propagating updates to all replicas. In all three graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

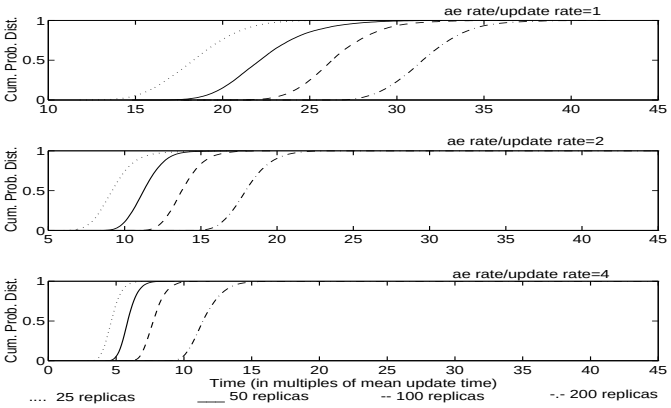

Figure 11: Cumulative probability distribution for receiving acknowledgments from all replicas. In all the x-axis is the simulation time scale in multiples of when updates are generated. In the middle and bottom plots, the global anti-entropy rate (ae rate) is 2 and 4 times the rate of the top plot.

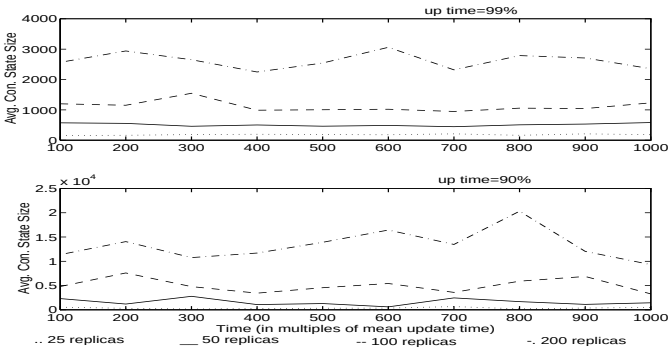

Figure 12: Average consistency state size (in number of update messages). In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

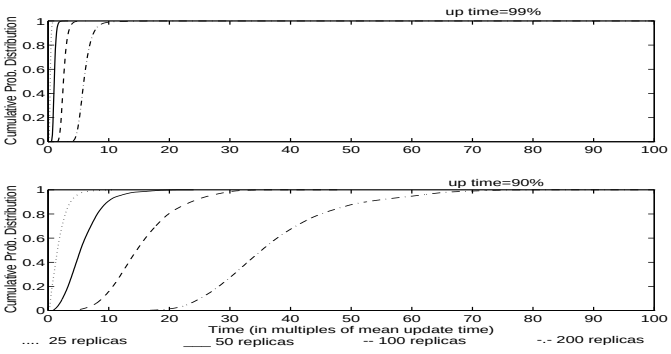

Figure 13: Cumulative probability distribution for propagating updates to all replicas. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

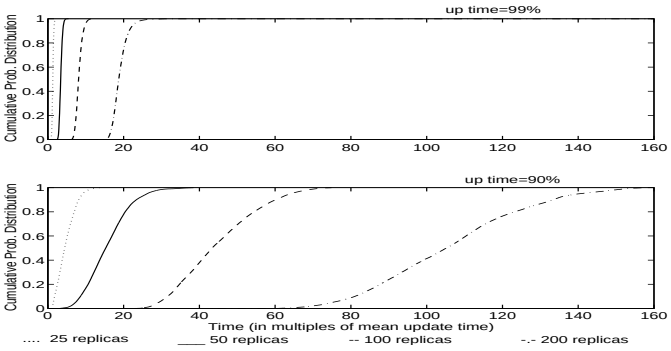

Figure 14: Cumulative probability distribution for receiving acknowledgments from all replicas. In both graphs, the x-axis is the simulation time scale in multiples of when updates are generated. In the bottom graph, replicas stay down longer than in the upper graph.

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 [7], 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 is represented by a fully-connected, symmetric cost matrix [3].

The top graph in Figure 15 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

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

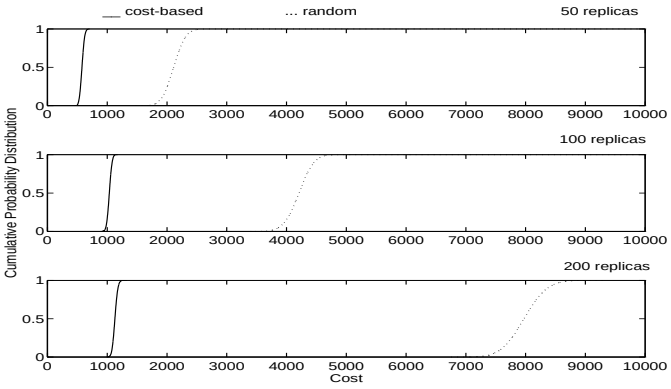

Figure 15: Cumulative probability distributions for the total cost of propagating updates to all replicas using cost-based and random partner selection policies for different group sizes. In all graphs, the x-axis is the simulation time scale in multiples of when updates are generated.

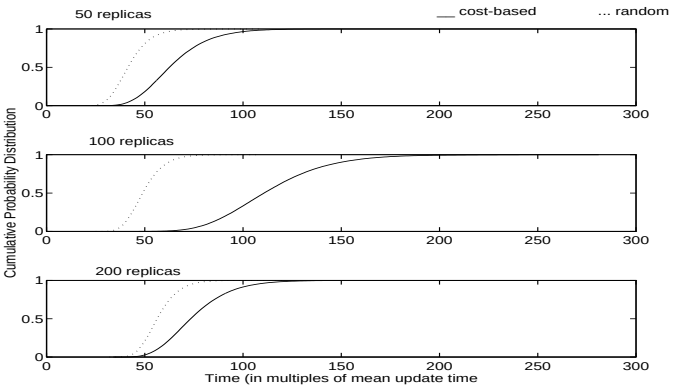

Figure 16: Cumulative probability distributions for the time to propagate updates to all replicas using cost-based and random partner selection policies for different group sizes. In all graphs, the x-axis is the simulation time scale in multiples of when updates are generated.

increases. The middle and bottom graphs of Figure 15 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 16 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 17 and 18 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 15). 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 16).

5.1 Summary

Our simulation results demonstrated the benefits of splitting a single, flat replication group into multiple smaller groups. We observed that the smaller the replication group, the smaller the size of the consistency state each replica in the group needs to keep. This difference in consistency state size is amplified because in smaller groups, replicas can purge their consistency state sooner.

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

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

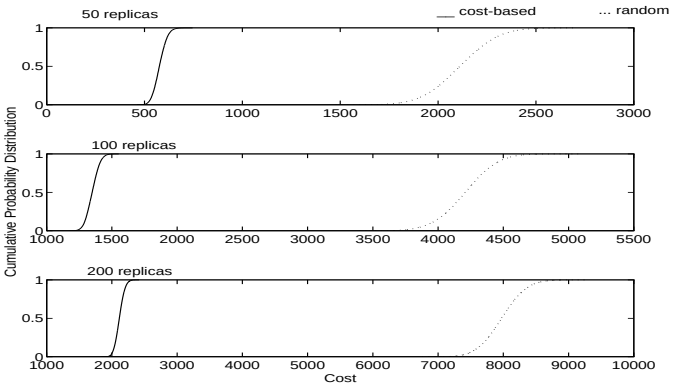

Figure 17: Cumulative probability distributions for the total cost of propagating updates to all replicas using cost-based and random partner selection policies for different group sizes. In all graphs, the x-axis is the simulation time scale in multiples of when updates are generated.

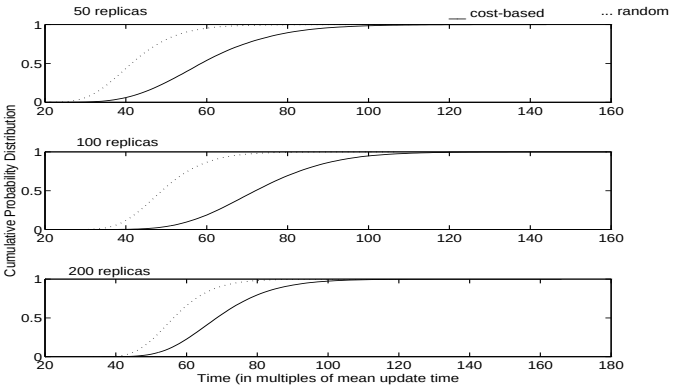

Figure 18: Cumulative probability distributions for the time to propagate updates to all replicas using cost-based and random partner selection policies for different group sizes.

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.

6 Conclusions

To achieve adequate performance, many future Internet services will need to replicate their data in thousands of autonomous domains. This paper described flood-d, our tool to support high degrees of replication through application-layer flooding of data.

Flood-d automatically builds logical update topologies that attempt to use the network efficiently and attempt to propagate updates quickly and robustly. It organizes replicas into *replication groups*, analogous to the Internet's autonomous routing domains.

Organizing replicas into multiple, smaller groups is vital for wide-area services with replication degrees in the thousands. Hierarchical organization limits the size of the consistency state that replicas needs to keep and permit autonomous management of group topology, group membership, and group membership traffic.

References

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

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

[3] Peter Danzig, Katia Obraczka, and Shih-Hao Li. Internet resource discovery services. *IEEE Computer*, pages 8–22, September 1993.

[4] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–111, May 1990.

[5] Alan Emtage and Peter Deutsch. archie: An electronic directory service for the Internet. *Proceedings of the Winter 1992 Usenix Conference*, January 1992.

[6] 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.

[7] Steve Hotz and Romklau Nagamati. Network topology generator (ntg): A tool for generating network topology and policy for protocol simulation purposes. Spring 1992.

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

[9] 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.

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

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

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

[13] 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.

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

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

[16] 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.

[17] 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.

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

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

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