Multicast routing through a network with hierarchical topology aggregation

Baruch Awerbuch
Dept. of Computer Science
Johns Hopkins University
baruch@cs.jhu.edu

Tripurari Singh
Dept. of Computer Science
Johns Hopkins University
tsingh@cs.jhu.edu

January 2000

Abstract

The PNNI is an umbrella protocol that allows considerable routing flexibility. While it specifies the network signalling, it leaves route generation undefined. Any non-trivial route generation requires the collection and dessimination of network metrics. Furthermore, hierarchical networks require aggregation and compression of these metrics in order to be scalable. Aggregation and compression methods are also left unspecified by the PNNI, although the PNNI *recommends* using star graphs for aggregation.

In contrast, recent work done by Awerbuch, Du, Khan and Shavitt [ADKS98] suggest that the minimum spanning tree is a better choice. Their work also shows that exponential weight routing is superior to (the ubiquitous) min-hop routing.

In this paper, we extend the work of [ADKS98] to multicast. We define a multicast protocol, model it and incorporate it into their simulator.

Surprisingly, our results are not identical to that of [ADKS98]. However, the recommended schemes of [ADKS98] perform very well, and significantly out perform the standard min-hop routing. This study further strengthens the case for including the minimum spanning tree and exponential weight routing in the PNNI standards.

1 Introduction

Modern networks are typically comprised of several sub-networks or domains. These subnets usually serve independent organizations or distinct geographical location. For reasons of convenience and security, these subnets are independently administered and may be implemented using different technologies. These domains are linked together by a so called *network of networks* - a common protocol running on top of each network. Given a connection request between nodes in distinct domains, this protocol determines a route potentially using intermediate subnets. The protocol then uses these potentially heterogenous subnets to shunt data between the source and destination of the requested connection. The Internet is one such network of networks, and proposed ATM networks are another. One evolving standard for interconnecting ATMs is the so called *Private to Private Network Interface (PNNI)*.

In order to route across networks, the *PNNI* protocol needs to know the properties of the underlying subnets, such as bandwidth availability, latency, routing policies etc. A trivial scheme for doing this would be for each subnet to reveal all its information. This is problematic because the subnetworks may be very diverse in nature making it impossible for the *PNNI* protocol to make sense of all of their technologies. Another important, albeit commercial, reason is the confidential nature of the subnet structure. While networking companies advertise the end to end properties of their networks, the underlying network graph is typically a trade secret. PNNI complies with these constraints by requiring the subnet to *aggregate* its end to end properties and represent them as a set of *border* nodes and the network properties between each pair of border nodes. Specifically, a border node of a subnet is defined as a node with links to other subnets.

Since subnetworks may be potentially huge with a very large number of border nodes, compression of this aggregate information is required. Compression of these aggregates without significant loss of fidelity is the primary focus of this part of my thesis. Briefly, the first problem is of compressing directed additive metrics.

The second problem is to use these metrics to generate efficient routes. In an attempt to maintain flexibility, the PNNI standard does not fully specify the route generation algorithm. It does, however, specify the signalling mechanism for setting up connections along the generated route.

Traditionally *min-hop* routing - that is, routing over paths comprised of lowest *number* of links - is used. While this scheme has some intuitive appeal, it is essentially ad-hoc. Perhaps its greatest shortcoming is that it is a static scheme that completely disregards the existing traffic on the network when attempting to route new connections. We test a promising scheme of [AAP93] that assigns weights to edges depending on their traffic loads, and then picks short routes in this metric.

2 A Brief Description of *PNNI*

PNNI is an "umbrella" protocol that allows participating subnets the freedom to choose their own routing and aggregation protocols for intra subnet connections. Significantly, it also allows a limited degree of freedom in routing *across* networks.

The *PNNI* protocol is hierarchical in nature in that it recursively decomposes the network into subnets. At the bottom of this tree structure are individual participating subnets, which, of course, are free to be hierarchical themselves. *PNNI* supports connection based routing across its hierarchy. The source and destination of each connection request is constrained to be lowest level nodes. *PNNI* supports quality of service and admission control considerations. However, it leaves the criteria and implementation of these decision to the individual subnets.

2.1 The *PNNI* hierarchy

The *PNNI* protocol contracts each subnet into a *logical node* and merges all links connecting it to the same individual subnet into one *logical link*, so that the resulting graph is not a hypergraph. This graph is partitioned into a set of (small) subgraphs known as *Peer Graphs*. *PNNI* does not do this partitioning automatically, but instead relies on the network administrator for this purpose. The logical nodes in a Peer Graph are referred to as its *Peer Nodes*. The *PNNI* hierarchy recursively contracts each Peer Graph into a logical node of the next higher level.

The *PNNI* hierarchy thus formed can be represented as a tree, with participating subnets as its leaves. All leaves are not required to be of the same depth. Each logical node is assigned a name such that a parent's name is a substring of its child's name.

2.2 The *PNNI* database

Information essential for routing within a Peer Graph can be classified into *attributes* and *metrics*. Attributes are considered individually when making routing decisions, while metrics are considered collectively. An example of an attribute is the (low) security status of a node that might preclude some connections from passing through it. Link delay, on the other hand, has a cumulative effect along a path and hence is a metric. Along with the routing algorithm, attributes and metrics are not specified by the *PNNI* standard and are left to each subnet to choose.

Typical link attributes are its available bandwidth and QoS related parameters. An common node metric is the amount of delay it introduces in going from an incoming link to an outgoing link. In a physical node, this delay is primarily due to output queue for each link. In a logical node representing a subnet, it is the delay introduced in traversing the subnet between the given border nodes. Node metrics, such as delay, can be represented as a matrix of pairwise link values. Such a representation is quadratic in the number of border nodes, which can be quite large for large subnetworks. Compression schemes for this matrix will be the focus of our work.

The attributes and metrics of all the links and nodes in a given Peer Graph are collectively known as it *Topology State Information.* In a *PNNI* based network, each logical node knows the topology state information of its entire Peer Graph, and also of the Peer Graphs of each of its ancestors. While the former is required for routing within the Peer Group, the latter is required for partially computing routes to nodes outside the Peer Group, in a manner elaborated upon in the next section. As the number of levels in a network is expected to be small, the space required by a node is largely determined by Peer Graph sizes. Since attributes of links incident on a node are linear in the number of links, it is dominated by the node delay metric which is quadratic in the number of incident links.

In order to build its data base, each node determines its own state and those of the links connected to it. It then floods this information throughout its Peer Graph. This exchange of information allows each node to determine the topology state information of its entire Peer Graph. This information is then passed down to its descendent Peer Graphs and flooded in them.

2.3 *PNNI* routing

Two routing schemes commonly used in connection based networks are *source* routing and *hop by hop* routing. Hop by hop routing, as its name suggests, requires the node currently processing a given connection request to determine only the next node along its path. This routing scheme can create loops if the databases of the nodes involved are inconsistent. Source routing, on the other hand, requires the connection source to determine the entire route to the destination. This route is only expected to be a "high level" route and the source is not expected to specify the details of crossing other Peer Graphs. Individual Peer Groups are required to fill in these details while staying consistent with the "high level" route generated by the source.

The source uses the topology state information of its Peer Graph, and those of its ancestors to generate the "high level" route. If topology state information is compressed, the routing algorithm needs to decompress relevant parts of it "on the fly" and it needs to do so without using too much space. Because of the potential inaccuracies introduced by compression and obsolete data this path may not be viable. When this happens a *crankback* is triggered. A crankback tears down the connection to the last node that generated routing information. This could either be the source, or one of the nodes that "filled in" the "details". This node then generates an alternate route and re-attempts a connection. If all attempts fail then the connection request is rejected.

3 Previous Work

3.1 Metric Compression

An obvious way of compressing a graph is to remove some of its edges. The result of such a compression is known as a spanner. A more sophisticated technique would both remove and add edges, and change edge weights. Additional "Steiner" nodes might also be added. The current *PNNI* standard suggests a star graph with a steiner hub vertex connected to each border node by an edge.

Spanners are perhaps the most extensively studied compact representation of graphs. A $t-spanner$ of a graph $G = (V, E)$ is a subgraph $G' = (V, E')$ of G that does not increase the distance between any pair of vertices by more than a factor of t. In other words, given a pair of vertices $u, v \in V$ the distance from u to v in G' is at most t times the distance in G. Our problem of compressing the aggregated matrix of subnets can be readily solved by high quality directed spanners.

The utility of a high quality spanner has been established for a number of networking problems. Specifically [PU87] found that high quality spanners could be used to minimize the time and communication complexity of a network synchronizer. Another study [PU88] found that spanners allow networks to maintain succinct and efficient routing tables.

3.1.1 Undirected Graphs

Spanners were extensively studied by [PS89]. They showed that the problem of constructing the optimal spanner for a graph to be $NP-Complete$. They also developed polynomial time approximation schemes to construct $4r+1$ spanners of *undirected* graphs with $O(n^{1+1/r})$ edges. Setting $r = \log n$, this gives us an $O(\log n) - spanner$ with $O(n)$ edges.

In [AP90] Awerbuch and Peleg added a hierarchical structure on spanners of *undirected* graphs, allowing for a wider range of applications. This construction, however, required $O(n \cdot poly \log n)$ edges. [Bar96] applied yet another form of hierarchical structure - the *hierarchically well separated trees*. The result was a randomized scheme to construct a form of spanners with expected $O(poly \log n)$ stretch with $O(n \log n)$ edges. [CCG+98] deterministically simulated this construction by a set of trees, leading to the de-randomization of a subset of problems solved by [Bar96].

A related paper [CDNS93] addresses the problem of minimizing the *weight* of the spanner. This approach is not as useful for compressing graphs as that of limiting the *number* of edges.

3.1.2 Directed Graphs

Unfortunately, the problem of spanning directed graphs is much harder. [PS89] give a lower bound of $\Omega(n^2)$ as follows: consider a n-node complete bi-partite graph $G = (V, E)$ where V is partitioned into disjoint sets X, Y each of size $\frac{n}{2}$. E consists of weighted directed edges (x, y) directed from every node in $x \in X$ to every node in $y \in Y$. Clearly, the only path in this graph from a node x to node y is the edge (x, y). Hence, the (directed) distances between every pair of vertices $x \in X, y \in Y$ is independent of all other vertex pairs, which in turn implies that no sub-graph of G can be its spanner.

A directed Peer Graph can trivially be "made" undirected by sending dummy traffic so as to equalize the cost along each direction of each link. This dummy traffic may be much larger, or much smaller than the real traffic depending on the link capacities in the two directions of the link, and also on the function used to map link parameters to the edge weights. In a network with equal capacities in both directions of each link, the dummy traffic required is no greater than the real traffic itself, resulting in a modest factor of two loss. Once the directed graph is reduced to an undirected graph, any of the undirected compact representations can be used to compress it.

[AS98] considers an alternate model in which dummy traffic is not sent. Instead, the directed graph is characterized in terms of the *maximum asymmetry*, ρ, of its edge weights. [AS98] present a scheme to represent a directed graph with ρ asymmetry and b border nodes in $O(b)$ space with $O(\sqrt{\rho} \log b)$ distortion.

In a recent experimental work [ADKS98], Awerbuch, Du, Khan and Shavitt study simple and practical techniques for directed graph compression. Their work is based on simulations of multilevel *PNNI* networks on a variety of randomly generated traffic. The objectives of this study was to optimize the throughput of the protocol, as well as it *control load*, i.e. the amount signalling required per connection. All studies were conducted on *unicast* traffic. In the present work we extend their study to the *multicast* case.

3.2 Route Generation

By far the most popular route generation algorithm is "Min-hop Routing". This scheme tries to compute a path from the source to the destination with the smallest number of hops. This is akin to finding a shortest path in a graph with unit edge weights. A minor modification of this scheme is to allow the network administrator to set edge weights. Neither of these schemes adapts to changing traffic. In contrast, the scheme of [AAP93] computes edge weights based on traffic conditions.

4 The Simulator Architecture

4.1 Overview

Our simulator is a "high level" connection oriented simulator that does not simulate individual packets transmitted by the network. Instead it assumes all traffic to be connection oriented, with all packets of a given connection following the same path, and models each source as generating a steady stream of packets.

In reality sources are bursty, and a number of connections are statistically multiplexed into each link. A popular way of characterizing a bursty source is by its *equivalent bandwidth*. This equivalent bandwidth is a function of the average rate of the source, its "burstiness" and also the capacity of the link. For instance, if the link capacity is very large and sources generate traffic independently, only the average rate of each source is relevant. On a low capacity link, on the other hand, the burst size, peak rate etc. of a source become important.

Our simulator is only concerned with equivalent bandwidth of each connection, and its primary function is to pack these into the capacitated links of the *hierarchical* PNNI network. However, provision has been made to "plug-in" a low level simulator that can simulate individual packets. Our simulator is modular in design, with clear interfaces for plug-ins.

The core of the simulator is an event scheduler that processes events in chronological order. When an event is processed, it may generate other events at a future time. The simulation starts with a call to the event scheduler. The event scheduler then calls the *event processor*- the code that implements the event - of the first event. Subsequently, program control shuttles between the event scheduler and the event processor until the scheduler finds the event queue to be empty. At this point the simulation ends.

The network protocol to be simulated is implemented in the *event processor*. This done by implementing the protocol in C++ and compiling it along with the rest of the simulator. The executable file thus produced is hard coded for the particular network protocol. Network and traffic descriptions, however, are inputs the executable.

Our simulator is written in C++ and requires LEDA-3.5.2. It is written on a BSD/OS platform and is readily portable to all flavors of Unix. While the simulator consists of over 13,000 lines of code, only a few modules need to be modified in order to implement aggregation, compression, routing protocols etc.

4.2 Topology input and generation

The simulator proper does not generate random topologies, and requires the topology of the network to be input. However, a separate program is provided to generate random topologies. The multilevel topology is input simply as the set of *logical* nodes followed by a set of *physical* links.

The simulator deduces the hierarchy of a node by its name. Specifically, a node whose name is the prefix of another node is deduced to be its ancestor. This schemes results in a DAG of genealogical relationships, from which the genealogical tree is deduced. Sibling nodes in this genealogical tree are then clustered to form Peer Groups. Physical links are then used to induce logical links. A logical link is labelled an internal link it connects two nodes in the same Peer Group, and an uplink if the nodes belong to different Peer Groups. If more than one logical link is formed between a pair of nodes, they are coalesced into a single logical link.

Random topologies are generated using a standard technique [Wax88]. Specifically, each Peer Graph is generated by first assigning nodes to random locations on a grid. A subset of these nodes are randomly selected to be border nodes. These border nodes are then forced to random locations at the periphery of the domain's grid. Links are then added by randomly generating pairs of nodes and connecting them with probability that decays exponentially with the distance between them. If this raises the degree of either node to more than 4, then the link is not added. Addition of links stops when each node has a degree of at least 2. This process of generating Peer Graphs is recursively applied until a graph of the desired hierarchy-depth is formed.

4.3 The multicast protocol

A *multicast group* consists of a *source* vertex and a set *receiver* vertices spanned by a steiner tree. Receiver vertices arrive one at a time and need to be connected immediately. A receiver node is free to disconnect at any time.

A receiver node requesting connection to a group supplies the name of the group and the name of a node previously connected to this group. The network then forwards the name of the requesting node to the previously connected node specified by it. This connected node then determines the previously connected *receiver* node closest to the requested group and communicates this result to the requesting node. The requesting node then establishes a connection to the closest node. For the sake of simplicity, steiner nodes are not considered when computing the closest node. Also, the name-space of the nodes is used to determine the closest node which, in general, may not be the closest node in the link cost metric.

When a receiver node's request terminates it is disconnected from the tree, unless it is passing on signals to other receiver nodes. In this case the node stays connected until all nodes connected to it terminate their connections.

4.4 Traffic input and generation

Traffic consists of point to point connections and multicasts.

The connection generator outputs a set of connections with randomly chosen *start times, durations* and *bandwidths*. The inter-arrival time, duration and bandwidth can be selected to be either of uniform distribution, or exponential distribution, or a fixed deterministic value.

The multicast generator takes as input the group-id, source node, bandwidth, mean inter-arrival period, the mean connection duration and the number of requests. It then randomly generates nodes requesting connections to the source. The start-time and duration of each request is also randomly generated. As in the case of point to point connections, inter-arrival time, duration and bandwidth can be selected to be either of uniform distribution, or exponential distribution, or a fixed deterministic value.

For authentic results, demand traces from real networks can also be input.

4.5 Statistics collected

The simulator records the success or failure of each connection request, as well as the number of *crankbacks* (see chapter 2) per request. The total throughput is determined by adding up the bandwidths of the successful connections.

The *control load* is estimated from the number of crankbacks per connection. Usually a high quality routing algorithm combined with a high quality aggregation scheme permits the generation of a realistic "high level" route at the source, and thus results in fewer crankbacks down the line.

4.6 Some implementation details

In this section we describe the key classes of our Simulator. Our simulator is quite large, containing over 1500 lines of declarations alone. The description given in this section is extremely brief.

- **PeerGroupNode** defines the state of one node in a Peer Group, and contains functions to process events occurring in it.
- **PeerGroupEdge** defines the state of a physical link.
- **LogicalGroupEdge** defines the state of a (possibly) aggregated edge.
- **PeerGroupGraph** contains functions for routing within the Peer Group.
- **Event** contains the basic information common to all events, such as start time, type and id of event. It also contains a pointer to a message that contains the rest of the information. This information may be different for different types of events.
- **ConnectionRequest** handles the setting up of a point to point connection.
- **BatchMulticastRequest** connects one or more nodes of the *same* multicast group.
- **AggregateGraph** abstracts out the information of a subnet and represents it compactly.

5 The Simulation

We ran a set set of simulations with different values for the following variables:

1. Cost metric
2. Compression Scheme
3. Network Topology
4. Link delay
5. Traffic load

Of these, variables 1 and 2 are engineering parameters whose value we wished to determine, while variables 3,4 and 5 are inputs to the multicast protocol. Next we briefly describe the engineering variables.

5.0.1 Engineering Variables

- **Edge weights**: We compared the traditional min-hop metric with the exponential metric of [AAP93]. The min-hop scheme is implemented by assigning unit weight to each edge, and routing connections on the shortest path in the resulting graph. The scheme of [AAP93] calculates the weight of each edge as an exponential in its fractional utilization. As in the min-hop scheme, the shortest path in the resulting graph is used for each connection.
- **Compression Scheme**: Recall that the subnet can be precisely represented by a matrix of inter-border pair values. We test the following compact, though approximate, representations of it.
  - **Diameter**: A star graph with radius equal to half the network diameter.
  - **Average**: A star graph with radius equal to half the average distance between all pairs of nodes.
  - **MST**: A Minimum Spanning Tree.

- **RST**: A Randomly generated Spanning Tree.
- **Spanner**: A $t-spanner$. We use $t = 2$ in most experiments. That is, we use a sub-graph that does not distort the distance between any pair of nodes by more than a factor of 2.
- **COMPLETE** : No compression is done, and the complete cost matrix is advertised. **COMPLETE** is not a candidate scheme; it merely serves as a control.

5.0.2 Input variables

- **Network Topology** We tested 3 network topologies: 1 randomly generated topology and 2 regular topologies specifically designed to highlight performance differences between aggregation schemes by heavily penalizing poor routing decisions. These topologies are described in greater detail below.
  - **Random:** We generated two level random topologies using the standard technique of [Wax88]. Specifically, each Peer Graph was generated by first assigning nodes to random locations on a grid. A subset of these nodes were randomly selected to be border nodes. These border nodes were then forced to random locations at the periphery of the domain's grid. Links were then added by randomly generating pairs of nodes and connecting them with probability that decays exponentially with the distance between them. If this raised the degree of either node to more than 4, then the link was not added. Addition of links stopped when each node had a degree of at least 2. This process of generating Peer Graphs was recursively applied to generate a two level graph.
  - **Staged Ring:** This two level topology is shown in figure 1. It consists of a 8 node bi-partite graph known as a ladder with two additional nodes - one connected to each node in a bi-partite set. Each node in the ladder is itself a ring subgraph. Each link in the lower level rings has 5 units of capacity, while each link in the higher level ladder has 6 units of capacity.

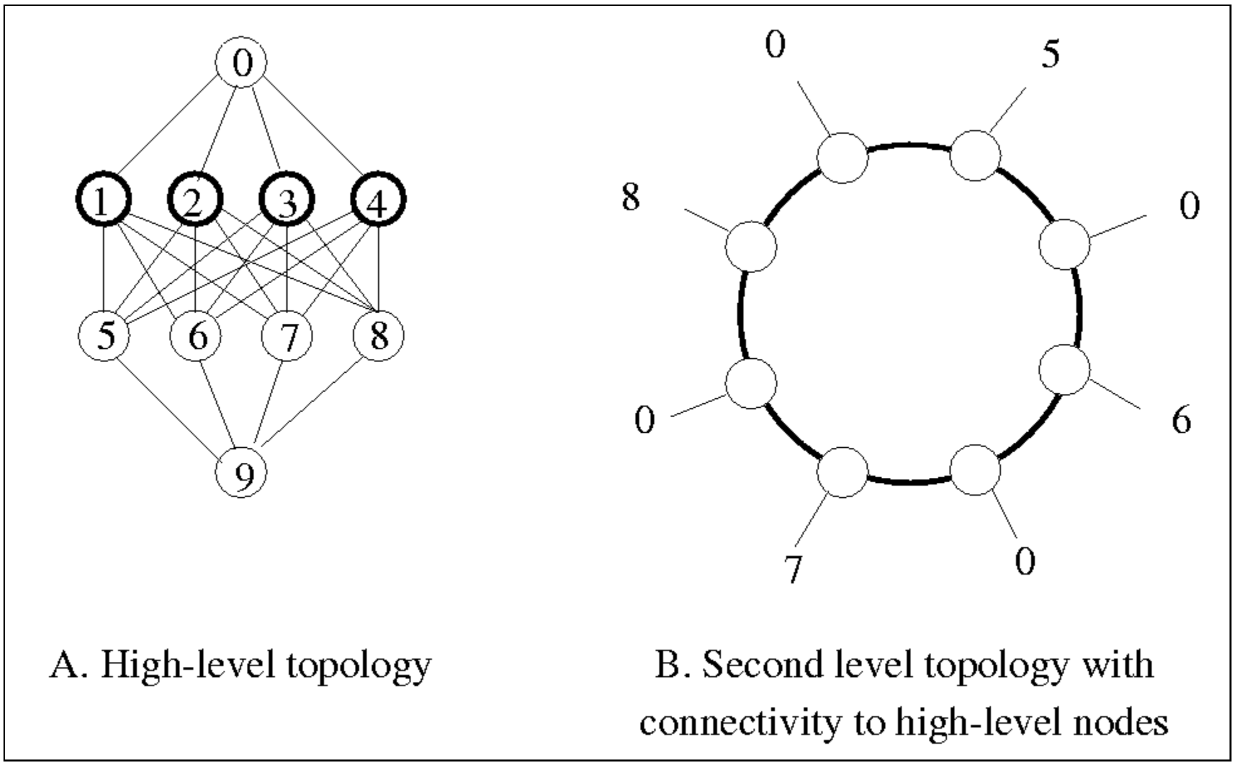

Figure 1: Staged Ring Topology

  - **Self-similar topology:** This topology is shown in figure 2. This topology has three layers. The first/top layer is a line graph shown on the right. The second and third layers are have similar topologies. This topology is shown on the left. The highlighted node in the first layer is composed of the second layer subnetwork. The remaining two first layer nodes are singleton nodes. Each node in the second layer is recursively composed of graphs of similar topologies. Link capacities are 5 for plain internal edges, 10 for bold internal edges, 10 for links between border nodes and 15 for bold links between border nodes.

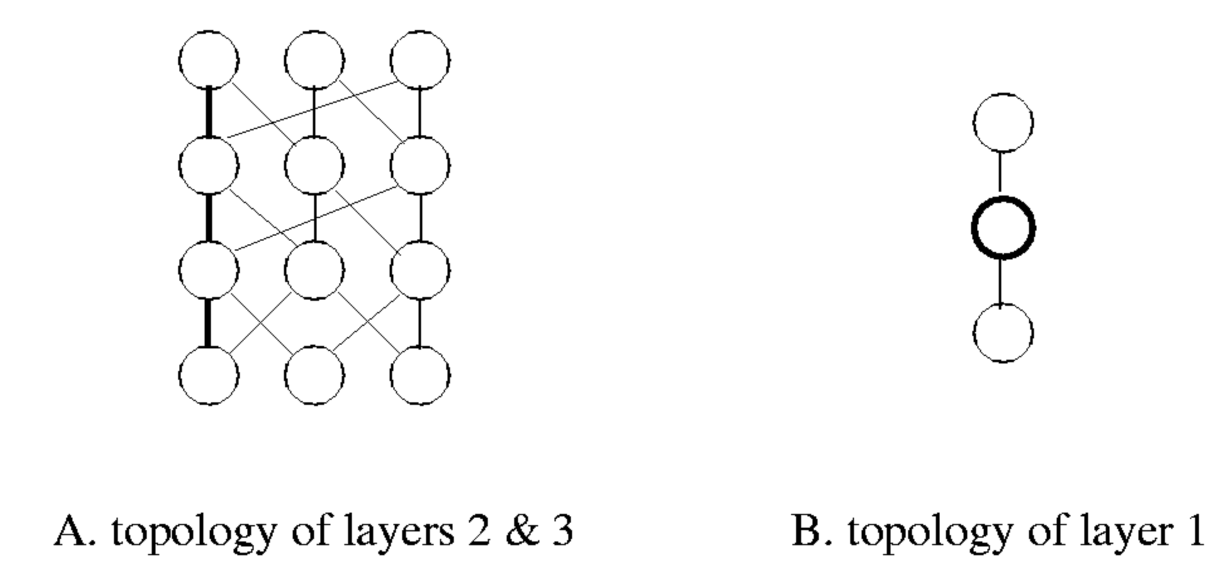

Figure 2: Self-Similar Topology

- **Link Delay** Link delays contribute to both the time needed to set up a connection as well as the latency of data transmission on an established connection. Here we concentrate on the first aspect of the problem. Large set up times result in a large number of connection requests being active at any given moment. This places a greater load on the signalling and control mechanisms of the network, and magnifies the difference in performance of different protocols.

  We ran simulations using two values of link delay: 1 time unit and 10 time units.

- **Traffic Load** Like link delay, heavy traffic also increases the control load on the network. In general, heavier traffic increases the performance difference between protocols.

  We modeled requests to each multicast group as a Poisson process, with exponentially distributed arrival times and connection durations. Note that once a connection is accepted, it has to be maintained until its duration, and those of nodes (subsequently) connected to it, expire. We assume that connection requests are not persistent. That is, once a node is denied a request, it does not keep re-trying for the same connection. If the rejected nodes did, in fact, re-try indefinitely, they would saturate the network's control/signalling mechanism.

  We tested each protocol at 4 distinct traffic loads.

The above mentioned variables yield a total of 228 combinations of values. For each of these combinations we run a simulation of about 5000 connection requests, and measure the total throughput and the number of crankbacks per request. We only generate multicast requests, and not a combination of point to point and multicast requests as would typically occur in real life. We do this in order not to dilute our results, or equivalently get by with a smaller simulation.

5.1 Results

The results of our simulations are summarized in the tables below. We first ran our simulations with the min-hop edge weight scheme and then repeated these experiments with the exponential weight scheme.

We first present the throughput values of each set of experiments and then their crankback values.

5.1.1 Min Hop: Fraction of connection requests accepted

5.1.2 Ring-Ladder

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.351	0.353	0.358	0.345	0.365	0.365
10	0.318	0.317	0.342	0.315	0.335	0.335
15	0.331	0.349	0.315	0.309	0.343	0.343
20	0.302	0.305	0.307	0.31	0.311	0.311

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.341	0.335	0.346	0.322	0.336	0.336
10	0.327	0.342	0.342	0.326	0.312	0.312
15	0.334	0.334	0.33	0.306	0.322	0.322
20	0.319	0.322	0.308	0.301	0.297	0.297

5.1.3 Self-similar Hierarchy

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.411	0.395	0.423	0.416	0.415	0.415
10	0.435	0.443	0.441	0.435	0.44	0.44
15	0.532	0.529	0.519	0.517	0.527	0.527
20	0.582	0.581	0.577	0.578	0.573	0.573

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.423	0.391	0.397	0.406	0.408	0.408
10	0.465	0.441	0.456	0.463	0.445	0.445
15	0.53	0.532	0.531	0.525	0.531	0.531
20	0.59	0.584	0.571	0.578	0.59	0.59

5.1.4 Random Graph

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.389	0.382	0.392	0.406	0.401	0.401
10	0.431	0.426	0.428	0.435	0.424	0.424
15	0.535	0.535	0.544	0.536	0.539	0.539
20	0.532	0.531	0.524	0.522	0.52	0.52

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.41	0.409	0.402	0.421	0.42	0.42
10	0.438	0.438	0.449	0.452	0.441	0.441
15	0.543	0.526	0.53	0.523	0.548	0.548
20	0.551	0.558	0.53	0.531	0.545	0.545

5.1.5 Min Hop: Average number of crankbacks per request

5.1.6 Ring-Ladder

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.396	0.392	0.444	0.673	0.555	0.555
10	0.397	0.365	0.494	0.849	0.456	0.456
15	0.376	0.476	0.456	0.607	0.581	0.581
20	0.554	0.576	0.497	0.835	0.695	0.695

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.438	0.434	0.499	0.691	0.465	0.465
10	0.556	0.516	0.605	1.018	0.536	0.536
15	0.478	0.478	0.5	0.587	0.487	0.487
20	0.551	0.573	0.516	0.779	0.497	0.497

5.1.7 Self-similar Hierarchy

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	1.574	1.206	1.337	1.469	1.697	1.697
10	1.74	1.783	1.89	1.633	1.56	1.56
15	1.387	1.368	1.566	1.584	1.443	1.443
20	1.978	2.011	1.942	1.739	1.497	1.497

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	1.696	1.423	1.547	1.869	2.004	2.004
10	1.452	1.404	1.388	1.365	1.455	1.455
15	1.36	1.185	1.146	1.325	1.257	1.257
20	1.378	1.375	1.371	1.36	1.347	1.347

5.1.8 Random Graph

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	1.286	1.199	1.555	1.467	1.322	1.322
10	1.474	1.577	1.662	1.805	1.85	1.85
15	1.657	1.442	1.455	1.574	1.431	1.431
20	1.119	1.115	1.056	1.118	1.152	1.152

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.742	0.636	0.557	0.756	0.642	0.642
10	0.847	0.902	0.759	0.8	0.852	0.852
15	1.076	1.027	0.791	0.961	1.005	1.005
20	0.799	0.794	0.687	0.682	0.743	0.743

5.1.9 Exponential: Fraction of connection requests accepted

5.1.10 Ring-Ladder

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.354	0.376	0.357	0.339	0.357	0.357
10	0.321	0.344	0.337	0.319	0.337	0.337
15	0.333	0.322	0.314	0.334	0.314	0.314
20	0.317	0.32	0.315	0.312	0.315	0.315

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.335	0.333	0.336	0.321	0.335	0.33
10	0.317	0.342	0.336	0.317	0.332	0.336
15	0.318	0.323	0.328	0.302	0.364	0.341
20	0.301	0.32	0.303	0.307	0.308	0.303

5.1.11 Self-similar Hierarchy

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.409	0.417	0.397	0.398	0.408	0.41
10	0.436	0.45	0.452	0.436	0.453	0.449
15	0.537	0.537	0.517	0.523	0.519	0.514
20	0.567	0.576	0.56	0.572	0.576	0.576

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.403	0.399	0.402	0.395	0.412	0.403
10	0.459	0.464	0.446	0.441	0.453	0.44
15	0.527	0.532	0.524	0.532	0.518	0.515
20	0.592	0.587	0.59	0.592	0.59	0.583

5.1.12 Random Graph

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.391	0.388	0.387	0.37	0.39	0.398
10	0.428	0.426	0.419	0.425	0.424	0.415
15	0.523	0.523	0.522	0.524	0.525	0.532
20	0.539	0.534	0.536	0.531	0.534	0.528

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.393	0.412	0.392	0.367	0.392	0.414
10	0.448	0.423	0.444	0.377	0.435	0.429
15	0.537	0.546	0.517	0.506	0.54	0.541
20	0.551	0.544	0.54	0.52	0.547	0.556

5.1.13 Exponential: Average number of crankbacks per request

5.1.14 Ring-Ladder

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.344	0.309	0.214	0.551	0.214	0.214
10	0.435	0.336	0.243	0.673	0.243	0.243
15	0.386	0.432	0.371	0.465	0.371	0.371
20	0.387	0.425	0.266	0.822	0.266	0.266

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.337	0.432	0.349	0.541	0.354	0.358
10	0.41	0.429	0.347	0.554	0.429	0.347
15	0.456	0.479	0.414	0.642	0.448	0.359
20	0.444	0.443	0.415	0.789	0.347	0.415

5.1.15 Self-similar Hierarchy

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	1.516	1.407	1.553	1.428	1.383	1.316
10	1.368	1.472	1.468	1.499	1.394	1.584
15	1.06	1.06	1.199	1.22	1.094	1.33
20	1.255	1.415	1.516	1.51	1.19	1.19

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	1.639	1.467	1.705	1.443	1.504	1.441
10	1.164	1.173	1.19	1.129	1.251	1.32
15	1.014	0.922	1.062	1.127	0.988	1.1
20	1.167	1.135	1.06	0.969	1.247	1.01

5.1.16 Random Graph

Link delay = 1 time unit

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.784	0.653	0.686	0.552	0.694	0.82
10	1.231	0.888	1.049	1.099	1.327	0.879
15	0.727	0.638	0.587	0.86	0.571	0.542
20	0.508	0.49	0.542	0.582	0.511	0.523

Link delay = 10 time units

ArrivalMean	Diameter	Average	MST	RST	TSpanner	COMPLETE
5	0.463	0.692	0.497	0.536	0.625	0.507
10	0.601	0.553	0.523	0.656	0.533	0.513
15	0.7	0.594	0.448	0.587	0.629	0.543
20	0.333	0.338	0.331	0.352	0.327	0.322

5.2 Analysis

Of the two measures of performance, throughput is probably the more important one. We first analyze the throughput data of both min-hop and exponential edge weight routing schemes.

5.2.1 Throughput

A brief glance at the throughput data shows little dependence on engineering variables. RST was the only compression scheme that frequently under-performed.

When min-hop routing was used, all compression schemes performed within 5% of each other in 21 of the 24 input instances, and well within 10% in the remaining 3. Within this narrow band of throughput, Average performed the best followed by Diameter. Average performed the best in 7 of the 24 inputs followed by Diameter also best in 7 inputs. However, Average performed better under conditions of higher loads and link delays.

When exponential weighting was used, the spread in performance was slightly larger, though it was still quite small. On 12 out of 24 inputs, all compression schemes performed within 3% of each other, on 11 other inputs all performed within 7% of each other and on the remaining input t-Spanner was about 17% better than Diameter. As in the min-hop case, Average performed the best, winning in 12/24 inputs followed by Diameter winning in 8 inputs. RST again performed the worst.

What's more interesting is that both min-hop and exponential weighting schemes performed similarly. The best compression schemes under both were within 3% of each other in 21 of the 24 inputs, within 5% in 2 of the inputs and 10% apart in only one input.

An apparent anomaly in the data is that COMPLETE, our control scheme, is frequently outperformed by other schemes. This is due to the online nature of the problem. Complete information of the present does not necessarily result in good decisions for future input, which may be completely unrelated to the past.

We summarize our results by averaging throughput data over all inputs and plotting it in figure 3.

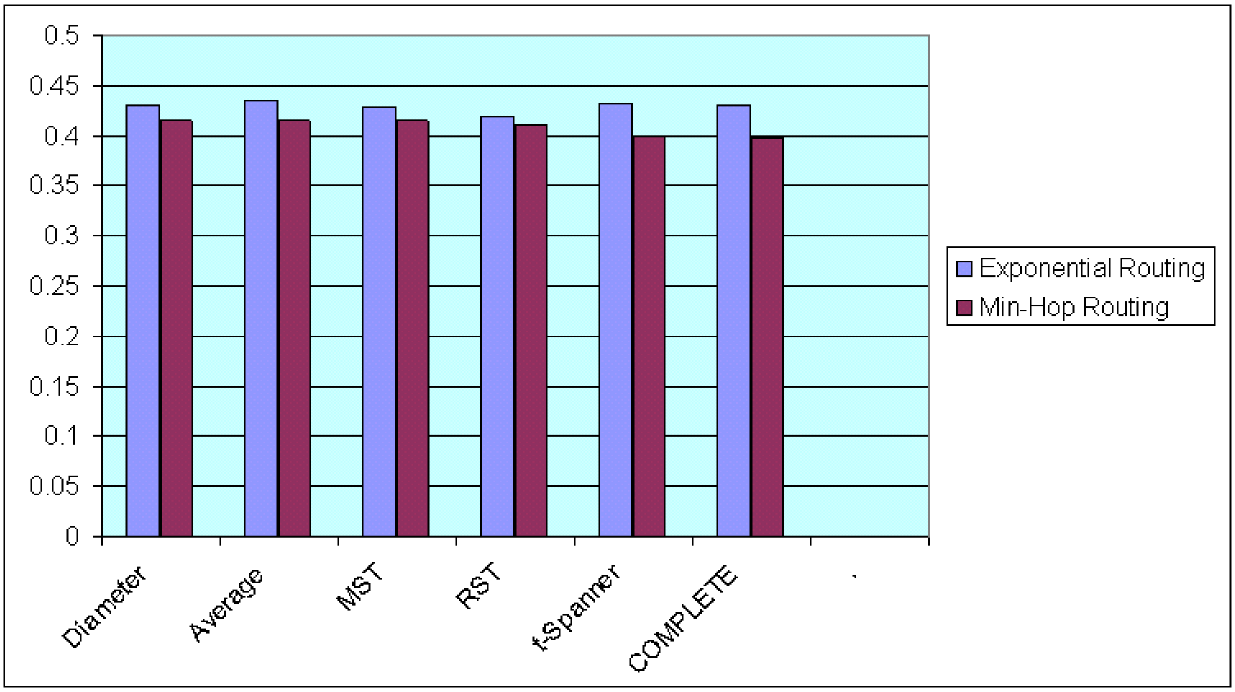

Figure 3: **Throughput**, as computed by averaging the the fraction of realized connection requests over all inputs. There is little difference in the performance of the different schemes. This leads us to conclude that throughput is largely independent of our engineering variables.

5.2.2 Crankbacks

There was greater spread in crankback performance than throughput performance. When min-hop routing was used, Average performed the best, winning on 9 out of 24 inputs. MST was second with 6, while RST was the worst with 2.

When exponential weighting was used, t-Spanner performed the best, winning on 10 out of 24 inputs, followed closely by MST at 8 wins.

On the whole, exponential weighting spectacularly outperformed min-hop. On 5 of the inputs, the best compression scheme using exponential weights was over 100% better than the best using min-hop and on a further 5 inputs exponential was over 50% better. Min hop was better only on 2 inputs, outperforming exponential by 2% and 9%.

We summarize our results by averaging crankback data over all inputs and plotting it in figure 4.

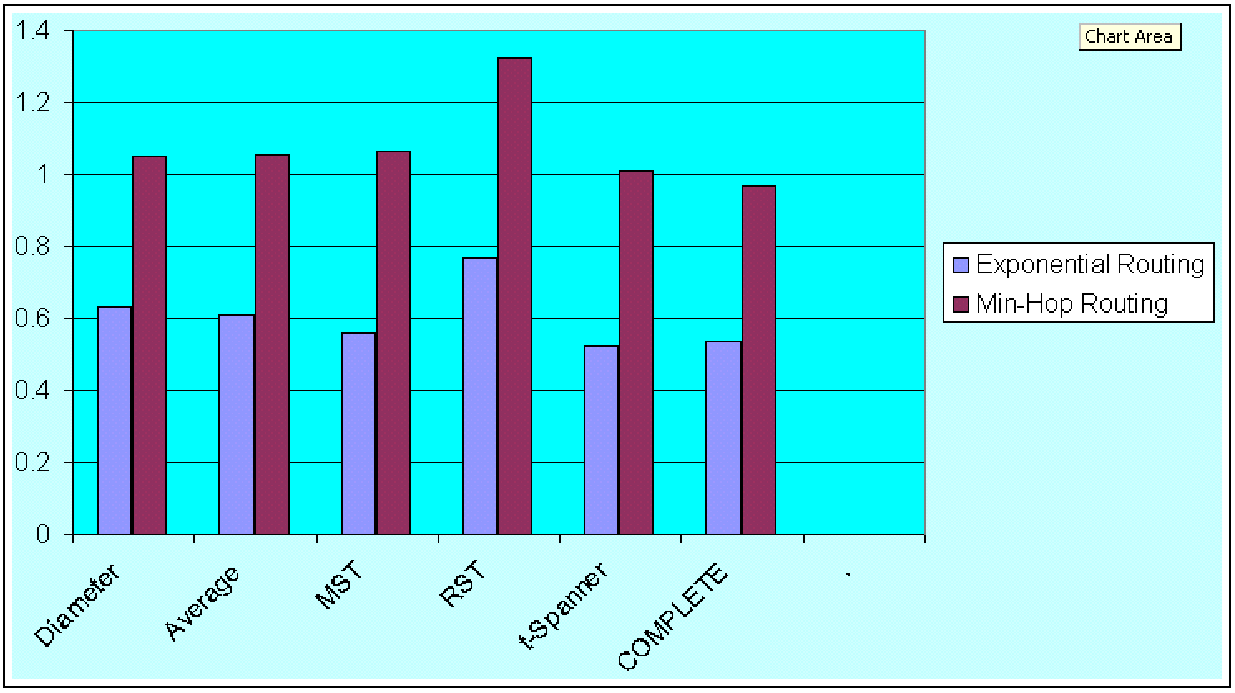

Figure 4: **Crankbacks**, averaged over all inputs. Note the clear superiority of the exponential weight routing scheme.

5.3 Recommendation

We present three sets of recommendations, depending on the operating environment:

- If maximization of throughput is the only criteria, and all applications running on the network are multicasts, then min-hop routing with Average star compression should be used.
- If all applications are multicasts, and both high throughput and low setup times/low control load are desired then exponential weighting with t-spanner compression should be used. This scheme is clearly the best in terms of crankbacks, and is within a few percent of the best in terms of throughput.
- If both point-to-point connections and multicast applications run on the network, as typically is the case, then exponential weighting and MST compression should be used. This is clearly the best combination for point-to-point connections [ADKS98]. In the multicast situation, it is within 10% of the best in terms of throughput and 20% of the best in terms of crankbacks.

References

[AAP93] Baruch Awerbuch, Yossi Azar, and Serge Plotkin. Throughput competitive on-line routing. In 34^{th} *Annual Symposium on Foundations of Computer Science,* Palo Alto, California, pages 32–40. IEEE, November 1993.

[ADKS98] Baruch Awerbuch, Yi Du, Bilal Khan, and Yuval Shavitt. Routing through networks with hierarchical topology aggregation. *Journal of High Speed Networks*, 7(1):57–73, 1998.

[AP90] Baruch Awerbuch and David Peleg. Sparse partitions. In 31^{st} *Annual Symposium on Foundations of Computer Science*, pages 503–513, 1990.

[AS98] Baruch Awerbuch and Yuval Shavitt. Topology aggregation for directed graphs. In *Proceedings of IEEE ISCC*, 1998.

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In 37^{th} *Annual Symposium on Foundations of Computer Science,* Burlington, Vermont, 1996.

[CCG+98] M. Charikar, C. Chekuri, A. Goel, S. Guha, and S. Plotkin. Approximating arbitrary metrics by o(n log n) trees. In 39^{th} *Annual Symposium on Foundations of Computer Science,* Palo Alto, California, November 1998.

[CDNS93] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Proc. 8'th ACM Symposium on Computational Geometry*, 1993.

[PS89] D. Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

[PU87] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the* 8^{th} *Annual ACM Symposium on Principles of Distributed Computing*, 1987.

[PU88] David Peleg and Eli Upfal. A tradeoff between size and efficiency for routing tables. In *Proceedings of the* 20^{th} *Annual ACM Symposium on Theory of Computing.* ACM, May 1988.

[Wax88] Bernard M. Waxman. Routing of multipoint connections. *Journal on Selected Areas in Communications*, pages 1617–1622, august 1988.