The TAO algorithm for virtual network management

Andy Collins
acollins@cs.washington.edu

Ratul Mahajan
ratul@cs.washington.edu

Andrew Whitaker
andrew@cs.washington.edu

December 13, 1999

Abstract

Virtual networks are an important tool for deploying new services on top of the Internet. We present the TAO algorithm to automatically manage the topology of virtual networks, allowing nodes to be added and removed automatically, and adjusting the topology to optimize performance. By automating the tasks of maintaining the set of nodes and configuring the tunnels between them, we can reduce the administrative costs and allow scaling to much larger virtual networks than what is currently practical. We focus primarily on the Active Networks testbed, or ABone, though our work has applications in all classes of virtual networks.

1 Introduction

Virtual networks are an important tool used to deploy a variety of new services and algorithms on top of the Internet. Our work is focused on automating the management of virtual networks, both to ease administrative costs and improve network performance as the number of nodes in a virtual network grows.

1.1 Virtual networks

Often referred to as *overlay networks*, or simply *overlays*, virtual networks are a powerful tool for deploying new services on the Internet. This is because they permit applications to implement their own network-level operations, while still using the underlying network (the Internet, in all the cases we will consider, but potentially any other network technology) for connectivity.

A virtual network consists of a relatively small set of nodes from the underlying network, plus a set of connecting "tunnels," which are logical links between the virtual network nodes implemented by encapsulating any data to be sent and using the underlying network to send it to the next node.

Because it encapsulates data in packets destined for the next virtual hop, rather than the final destination, a virtual network is able to implement application-level routing. Packets traverse the network through a series of virtual hops, visiting one node after another and making several separate trips through the underlying network. This is useful for testing new routing algorithms in the wide area, as well as for deploying secure networks that encrypt traffic before allowing it to traverse the network (a so-called *virtual private network*, which to this point has been the prime commercial application). In principle a virtual network is always less efficient than the underlying network (due to the limited control over the exact routes packets take), although the Detour work [6] argues that this difference could often be more than overcome by a sufficiently smart routing algorithm.

The use of actual network paths as logical links is both the power and the challenge of a virtual network. It is a powerful notion because it permits easy deployment (as compared to stringing wires to build a physical network) and potentially improved reliability (since the underlying network can route around failures without affecting the virtual network). In addition, multiple virtual networks can share the same physical resources, so that even systems with small bandwidth requirements can still be cost-effective.

However, unlike a physical network, the performance of a tunnel is not purely a function of the behavior of the overlay, as it is affected (in terms of latency, bandwidth, loss rate and availability) by other users of the underlying network. In addition, logically separate tunnels can share links in the physical network, and therefore can potentially exhibit interference, where traffic sent along one tunnel can affect the performance observed on another.

1.1.1 Some existing virtual networks

As mentioned above, the major commercial application of virtual network technology has been the creation of *virtual private networks*, which allow an organization to connect multiple sites using the Internet for transit, but encrypting all data before it enters the public network. Although commercially successful, these systems are not particularly interesting in the context of our work, because they tend to be small and, by their very nature, centrally-administered.

The largest virtual network deployed so far is the multicast backbone, or MBone. This example is far more pertinent to our work, as it is an example of a virtual network designed to deploy a new network algorithm for multicast routing. The MBone is an example of an incremental deployment system, where the primary reason to build the overlay is to allow the use of algorithms which (it is hoped) will eventually be migrated into the underlying routers, leading to the slow disassembly of the overlay as routers are converted.

1.2 The ABone

Our work has applications to a wide variety of virtual networks, but because of the wide variation in the specific requirements of different virtual network applications, we will focus our implementation on the case of the active network testbed network, the ABone. In section 5, we will revisit some of these application issues, and examine how we might extend this work to support some other classes of applications.

The ABone is an organic network, in the sense that nodes will be contributed with little global coordination and no central administration. Each node is to be operated by the organization contributing it. There is an additional complication introduced by the existence of multiple standard execution environments, so that only some nodes are available to any particular application. We will ignore this issue, because it is especially particular to the ABone implementation, and can be dealt with by running the configuration algorithm multiple times in parallel, establishing a separate virtual network for each execution environment.

From a performance point of view, the important aspect of the ABone is that it is intended to support a wide variety of new network applications, which may have widely varying performance metrics. All of these applications must be served by the tunnels of a single ABone topology, so it is desireable that this topology be general rather than specifically tuned to any particular application.

1.3 Scaling virtual networks

The challenge in our work is to build a virtual network that can scale to a large number of nodes and optimize its performance in the face of the Internet. There are two primary challenges in scaling a virtual network:

- **Node identification:** For a small network it is sufficient for an administrator to configure each node with a list of all other nodes, and to update this list as the membership of the network changes. As the network grows, this list becomes large, and the updates become too frequent for a manual approach to be feasible.
- **Tunnel configuration:** In a small virtual network it is natural to configure a tunnel between each pair of nodes, allowing optimal routing (either directly along the tunnel connecting the communicating nodes or through a true optimizing path in the way of the Detour work). Tunnels do, however, consume resources at the nodes, especially in a performance-optimizing network where it is necessary to keep measurement data for every configured tunnel (and especially for those tunnels *not* actively being used, where we cannot use passive measurements based on TCP data flows, and will have to actually send probe packets). Even if the small overhead of maintaining the tunnel data structures (basically the identity of the remote host and the routing information associated with the link) is reasonable, the measurement cost of an all-pairs configuration will very quickly grow unreasonable.

An additional challenge is posed by an ABone-style virtual network with multiple administrators. This is not a feature of all virtual networks, nor even of all large virtual networks, but since it is an economic issue, where it does occur in generally cannot be easily eliminated. There are a number of issues related to differing priorities of the different administrators, which we will defer to section 5. However, even if all the administrators are cooperative, the coordination alone makes growing the network difficult and argues for an automated system.

1.4 Automating virtual network configuration

The goal of this work is fully automatic virtual network configuration. Ideally each node in the virtual network should be configured independently, given only the minimum amount of information necessary to locate the rest of the virtual network, and plugged in. The system should enable the new node to automatically find its place in the virtual network, inform the rest of the nodes of its presence, and configure the tunnels that will connect it. Furthermore, the system needs to be able to update the global configuration as nodes come and go and as the performance of the underlying Internet changes. The work associated with network maintenance needs to be distributed among the nodes, both to avoid creating a single point of failure and to avoid unduly burdening any of the nodes.

All nodes in the TAO algorithm are created equal. All nodes are expected to provide transit services in accordance with globally optimizing the network. Clearly some nodes will end up providing transit far in excess of their own use of the network, and some nodes will end up with far better average connectivity (in terms of performance), but these inequities will come only from the effects of the actual Internet behavior.

1.4.1 Optimizing virtual network performance

The issue of performance optimization is separate from the goal of automating virtual network maintenance, but there is a huge opportunity in an automatically-configuring virtual network to optimize for performance in a way that is exceedingly difficult for a manually-configured system. Performance of paths in the Internet changes at a very fine timescale. An automated system, if it can be made good enough, has the potential to adjust the configuration to take advantage of these shifts where a human administrator could only base decisions on long-term average behavior.

1.5 Organization

To start, section 2 looks at some related work in the areas of network configuration and discovery.

Our major contribution in this work is an algorithm for managing virtual networks we call the TAO algorithm. This algorithm is detailed in section 3. In order to validate our algorithm, we conducted several simulations, as described in section 4.

Finally, in section 5 we will consider some very different kinds of virtual networks from those described so far, and consider how we might apply our techniques to the management problems they pose.

1.6 Status

The work presented in this paper is not yet fully complete. The design of basic operation of the TAO algorithm is complete, along with an analysis of most of the corner cases. The simulation environment for the algorithm is nearly complete. Section 4 describes our proposed evaluation metrics and comparisons, but does not include any actual results for the TAO algorithm. Once the simulation environment is complete, we should be able to complete the evaluation and begin testing the robustness of the algorithm in the face of abnormal network behavior.

2 Related Work

There has been some research in the area of automatic overlay management (eg. [1, 3]). But most of these works sit in a different design space. The X-Bone [1], is a direct descendant of MBone [7], that provides a graphical overlay management tool. It provides a way to construct multiple overlays over the set of nodes in the MBone. It relies on Multicast capabilities of the underlying network, and also has a central overlay manager for each overlay in operation. Scalibility has not been addressed in the work.

The ongoing work on MANets (Mobile Ad-Hoc Networks) has some relevance to our work. The proposed protocols (eg. [8, 9]), focus on some issues we also address like neighborhood discovery (which is made easier in a wireless world) and clustering (to isolate the effect the local changes and scalability). But the proposed protocols are not directly applicable to this work as their constraints are very different from ours.

A very early piece of work was the Landmark Hierarchy [4], which dealt with organizing the nodes of a network in a special hierarchy that provided efficient routing characteristics. But it imposed a different addressing scheme on the nodes. This brings in another problem of dynamic address translation which we wanted to avoid.

Some recent works have focussed on grouping the network resources based on some property such as locality ([2, 5]). [5] provides a case for a graph-theoretic formulation of the clustering of large internetworks. The solution they provide is *centralized*, while our simple clustering hierarchy is distributed and more suitable to actual deployment in the Internet.

3 The TAO algorithm

In this section, we describe our work on the Two-tiered Adaptive Overlay (TAO) scheme. Although TAO was designed for use with the Active Network virtual network (the ABone), it could be adapted for use with potentially any virtual network.

Designing an adaptive virtual network for the Active Network architecture is a difficult task since different services may have vastly different needs in terms of latency, bandwidth, fault-tolerance, etc. Thus, it is probably impossible to design a single scheme that satisfies everyone. Nevertheless, we feel that the TAO scheme will provide an adequate topology for the vast majority of ABone services.

3.1 TAO overview

The basic idea behind TAO is to organize nodes into a two-level hierarchy. At the bottom level, nodes are organized into clusters based on network locality. These nodes are referred to as cluster members. Each cluster has a designated *cluster head* that serves as the default gateway for the cluster members. The cluster heads collectively comprise the top level of the overlay. Tunnels in the TAO scheme are symmetric: a tunnel from A to B implies a tunnel from B to A.

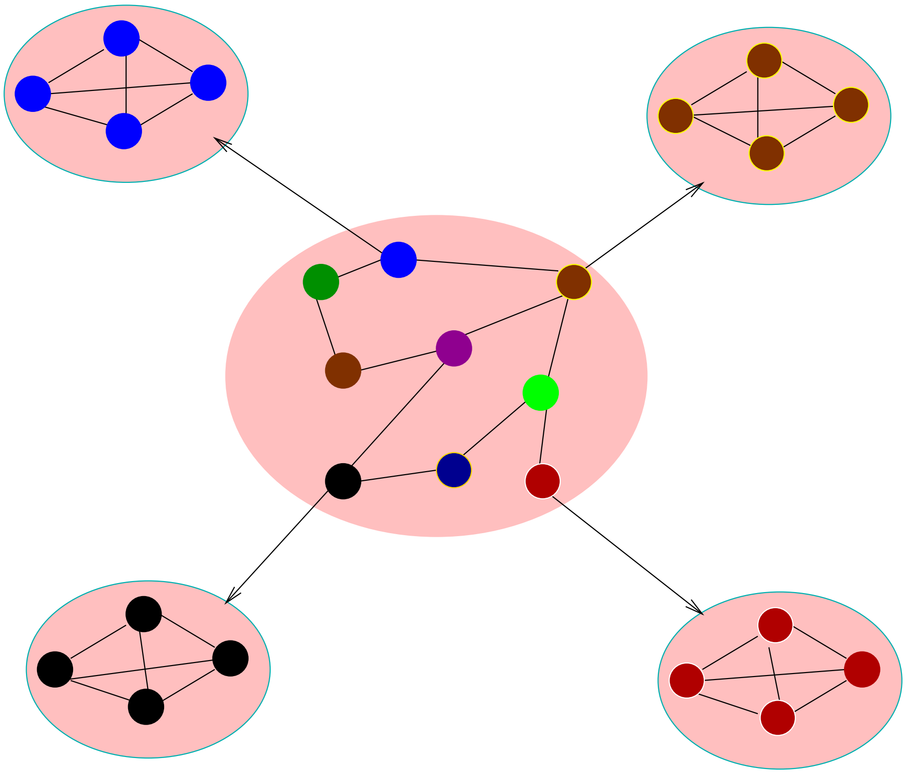

Figure 1: A two-tier virtual network

Within a cluster there exists a tunnel between each pair of nodes, so each node can directly send to every other node if it wants to. The cluster heads maintain tunnels to their associated cluster members and to a set of nearby cluster heads. The network of cluster heads will be running a routing algorithm on top of the overlay. The task of routing has been decoupled from the task of overlay management, so any routing algorithm can be used. TAO provides an API by which a routing algorithm can measure the cost and utilization of a tunnel. Section 3.6 discusses this in more detail.

TAO uses standard IP addresses to identify nodes in the overlay. As a result, it is not generally possible to do route aggregation, since nodes within a cluster are not required to belong to the same network. This was one big hurdle in scaling the scheme. Without any sort of route aggregation support the state required to be maintained at each node to be able to route packets blows up linearly with the number of nodes joining the network. A potential solution could have been assigning addresses to the active nodes based on their position in the topology, but this would have required solving another problem of mapping these addresses to IP addresses.

The remainder of this section describes the algorithms within TAO that make it adaptive to network events. First, we describe the join algorithm by which a new node is incorporated into the virtual network. Second, we describe algorithms for cluster management. Finally, we describe the mechanisms to respond to topology changes and node failure.

3.2 Node initialization

When a node wishes to join the overlay, it must find an appropriate cluster to join. This decision is made on the basis of network locality: the new node joins the closest cluster in terms of latency. For this purpose, it is assumed that the cluster head is indicative of the cluster as a whole, so it is only necessary to measure the distance to the cluster head to decide which cluster to join.

The basic join algorithm is a variant of a hill-climbing search frequently used in AI applications. First, the node obtains the address of a cluster head from the DNS using a well-known address (such as abone.net). The node then measures its distance to the first cluster head using ping or a similar latency measure. To make sure that this latency measurement is not effected by transients it might be necessary to do it multiple times and then consider the minimum latency seen. Then the node measures its distance to the neighbors of the first cluster head. If the first cluster head is closest, then the node joins its cluster. Otherwise, the node expands the neighbors of the closest cluster head seen so far, and the process repeats.

A well-known problem with greedy search techniques is that they can become trapped in sub-optimal local minimums. To combat this problem, TAO uses a variant of the above algorithm called "n-best join". Instead of considering only one candidate cluster head, n-best join maintains a list of the n-best cluster heads seen so far. The search terminates when the n-best list does not change after an entire cycle of pings. Upon completion of the search, the node selects the cluster of the closest member of the n-best list.

Once the new node has chosen an appropriate cluster, it sends a join confirm packet to the cluster head. The cluster head informs the new node of its peer cluster members, so the new node can immediately form its tunnels.

3.3 Cluster dynamics

This section describes algorithms that actively maintain the clusters. Algorithms are provided for cluster splitting, cluster merging, and cluster head election.

```
Vector pingSet = DNS.lookup (``abone.net'');
Vector pingLatencies = new Vector();
Node bestClusterHead = null;

while (true) {
  for (int i=0;i<pingset.size();i++) {
    pingLatencies[i] = ping (pingSet[i]);
  }

  Node closestNode = MIN (pingLatencies);
  if (closestNode.distance < bestClusterHead.distance) {
    bestClusterHead = closestNode;
    pingSet = closestNode.neighbors();
  }
  else return bestClusterHead;
}
```

Figure 2: The simplified join algorithm

3.3.1 Cluster splitting

The TAO scheme does not consider cluster size when choosing a cluster for a new node. As a result, clusters can become arbitrarily large. In the worst case, every new node would choose the same cluster, resulting in a fully-connected virtual network. A big cluster is undesirable because of several reasons:

- Too much overhead on the cluster head.
- Poor network utilization as more packets pass through this head, and
- The cluster is a fully-connected mesh and will therefore be too expensive for a large cluster.

To combat this problem, TAO provides a mechanism for splitting a cluster. After a cluster exceeds a certain size, the cluster head periodically creates a candidate split and tests whether the split is viable. If the test succeeds, then a new cluster is formed from roughly half of the nodes in the original cluster. A cluster head is appointed as the top-level representative for the new cluster. The new cluster head forms a tunnel to the old cluster head and to the old cluster head's neighbors.

The first step in the split algorithm is to divide the cluster members into two candidate partitions. The cluster head maintains a matrix of latencies between all pairs of cluster members[1]. Using this information, the cluster head designates two partition leaders, which are the two cluster members separated by the greatest distance. The partition leaders are assigned to different partitions, and every other cluster member is placed in one of the partitions according to which leader it is closer to. If one partition ends up significantly larger than the other, iteratively a member is picked from the bigger partition to go with the smaller partition till the size of the smaller partition exceed the smallest permitted cluster size. Once the partitions are formed, the partition leaders are no longer used.

[1]This information is provided as part of the regular member-to-head updates

The next step is to choose a candidate cluster head for the partition which does not contain the old head. The candidate cluster head is simply the node with smallest average latency to every other node in the partition. This can be quickly computed by adding across the rows of the matrix representing all-pairs latencies within the partition.

The decision to to adopt a proposed candidate split is influenced by two factors. First, the size of the current cluster is taken into account. Each cluster head maintains a constant value MAXNODES representing the maximum number of nodes the cluster head is willing to maintain in its cluster. Of critical importance is the ratio of the current number of nodes to MAXNODES, denoted by the variable p. The number of nodes in a cluster should not (but might if another node joins before the cluster head can effect a split) exceed MAXNODES, so the value for p is usually between 0 and 1 ($p > 1 \rightarrow p = 1$).

The second factor influencing the split test is the quality of the new clusters that would result from the split. Intuitively, it makes sense for a group of nodes to break off if the nodes are tightly grouped around the new cluster head. The split quality is measured by the ratio of the average distance to the new cluster heads versus the average distance to the old cluster head. This value (denoted by the variable q), should lie between 0 and 1 for any reasonable topology [2].

The split test simply compares the values for $(1 - p)$ and q. Specifically, if $(1 - p) \geq q$ then the split is adopted. Otherwise, the current cluster arrangement is maintained. This approach insures that a candidate split will be more likely whenever the cluster size is large, or when the candidate split quality is high.

If the split is effected, then the new cluster head form tunnels to old head and to the neighbors of the old cluster head. Starting from this initial set of tunnels the new head will figure out the right set of tunnels for it by running the dynamic tunneling algorithms outlined in section 3.4.

3.3.2 Cluster merging

Nodes can freely leave the dynamic network, causing clusters to become relatively small. In the extreme case the whole toplogy becomes flat, and the benefits of clustering are lost. This is inefficient because a disproportionately high percentage of messages will traverse the cluster head layer in order to reach their destination. Also, the cluster head for the small cluster is consuming resources (i.e., routing updates) that could be released if the cluster was eliminated.

Everytime a node leaves a cluster, the cluster checks if its size has gone below a threshold. If that is the case then the cluster head sends a merge notification to the remaining member nodes. Upon receiving the merge notification (which also contains the addresses of all neighboring cluster heads), the member nodes join one of the neighboring clusters. Once all the member nodes have vacated, the cluster head also joins a neighboring cluster.

3.3.3 Cluster head election

The current cluster head might not be the best one for the cluster. Since all packets go through the head, it makes sense to have the central node as the cluster head. The split algorithm contains a technique for choosing a cluster head for a candidate partition. It is occasionally desirable to replace the current cluster head without also splitting the cluster for reasons cited above.

The election algorithm operates as follows: periodically, the cluster chooses a candidate cluster head with the lowest average distance to every other cluster member. If the average distance to the candidate cluster head is (significantly)

[2] In the unlikely event that q exceeds 1, then the value for q is simply set to 1

smaller than the average distance to the current cluster head, then the candidate head and the current head swap roles. Since being a cluster head has its own overheads, some nodes in an overlay might not be willing to become heads. This can also be taken into account before chosing a new cluster head for a cluster.

3.4 Tunnels

The whole idea of dynamic tunneling in a virtual network is to have all the "better" options of sending packets from source to destination. This depends on the underlying physical network and the transients. While we can attempt to measure the transients, unfortunately there is no direct way to uncover the underlying network topology. Also since tunnels have their overheads a head cannot have arbitrary number of tunnels. On the other hand having too few a tunnels might be inefficient. So the endeavour is always to have a good set of tunnels. TAO employs a set of simple heuristics for tunnel addition and removal. When a new cluster head arrives (after a successful split), it forms tunnels to the old cluster head, and to the neighbors of the old cluster head. Periodically, a cluster head will attempt to add additional tunnels or remove existing one. First, the cluster head obtains a list of cluster heads that can be reached in two hops; this information is provided as part of the keep-alive message exchanged between cluster heads. If the cluster head can reach a two-hop neighbor significantly faster by sending directly, then a tunnel is formed between the nodes.

Suppose node A is scanning for new tunnels, node B is a one-hop neighbor, and node C is a two-hop neighbor that is reached via B. Let i be the indirect distance to send from A to C via B, and let d be the distance to send directly to C bypassing B. The critical ratio is i over d, denoted by the variable p. Each cluster head has a maximum number of allowed tunnels, denoted by the constant MAXTUNNELS. The threshold for adopting a new tunnel is the ratio of the sum of the current number of tunnels at each node and 2*MAXTUNNELS, denoted by the variable q. An additional factor δ is included to prevent tunnel oscillations. If $p \leq (1 - q - \delta)$, the candidate tunnel is adopted.

A similar procedure is invoked for removing tunnels. The candidate nodes for removing a tunnel to are those which can also be reached by 2-hops. This saves the overlay from totally disconnecting a cluster. Each cluster head can also have a minimum number of necessary tunnels, denoted by the constant MINTUNNELS. A typical value for MINTUNNELS is 2.

3.5 Reacting to network events

A dynamic virtual network must respond gracefully to a variety of network events including node and link failures. Though we believe that the set of overlay nodes will be more stable than what is seen in adhoc networks, there will be some dynamism (nodes joining and leaving the overlay) in the overlay. Also cluster heads become the single point of failure within a cluster. This section talks about how we deal with such events in our scheme.

3.5.1 Node failures

In order to detect node failure, a cluster head periodically exchanges keep-alive packets with its cluster members and with neighboring cluster heads. These packets can include other overlay management information. Packets from the head to the members include the full list of cluster members and the identidy of the backup cluster head (described below). Packets from the members to the head include a vector of costs (latencies) to all the other members. Failure to receive a keep-alive packet for three successive periods is interpreted as a node failure.

If the failing node is a cluster member, the cluster head simply removes the child from its list of children. The rest of the sibling nodes will become aware of this in the next update period. If the failing node is a cluster head, then the situation is potentially more grave. Every cluster maintains a backup cluster who automatically assumes command. The identidy of the backup cluster head is known both by the member nodes and by the neighboring cluster heads. The backup cluster head will have all the information a cluster head has which enables him to take over the responsibility as soon as the death of the cluster head is discovered. Upon gaining control, the backup cluster head forms tunnels to all the cluster members and to the neighboring cluster heads, and broadcasts its routing information to the cluster heads. While the updates are being processed, the neighboring cluster heads will automatically forward packets bound for the deceased cluster head to the backup cluster head as soon as they discover the death of this cluster head.

3.5.2 Handling network partition

Although network partitions are hopefully rare, it is important that the TAO algorithm be able to tolerate them. The ideal behavior, we believe, is for the part of the overlay on each side of the partition to continue operating, including accepting new members and adjusting the clusters to fit the visible behavior. When the partition is repaired, the cluster heads should rediscover each other, and combine the two hierarchies back into a single overlay.

We can use the cluster heads referenced by the well-known DNS address for the purpose of recognizing partitions. Each cluster head should periodically verify that it can reach (either directly or through the other cluster heads) each of the cluster heads in the DNS. If the node cannot be reached through the network of cluster heads, then it is either down or partitioned. If it can be reached directly, then it must have recently recovered. The first node to notice a recovery establishes a tunnel, thereby healing the network. The resulting hierarchy may be suboptimal, but the normal process of adjusting the clusters and tunnels between cluster heads should repair this over time.

3.6 Routing in TAO

A potential use of virtual networks is to try out new routing algorithms among other things. But every application possibly written will not be packaged with a built-in routing algorithm. And some are also not concerned about the efficiency of routing mechanism. While we claim to have decoupled the problem of routing with the overlay management, we cannot get away with providing no routing whatsoever.

The cluster heads are running a standard routing protocol among themselves, so every cluster head knows how to reach another cluster head in the network. Now the main obstacle here is that no one knows about a member of another cluster head. So to route a packet destined at some other cluster head's child, it becomes necessary to discover the cluster head associated with the desintation. There are several ways of dealing with the problem. This trivial one is that each node maintains a comprehensive list of all members of all nodes. This scheme is impractical for two reasons. The first one being that there maybe too much state to be stored in the absence of route aggregation. The second problem is that in this case what happens within a cluster is not restricted to it. News of birth and death of every node will have to be propagated to the whole network.

Another possible solution problem is a reactive routing scheme like the ones used in adhoc networks. These tend to have large startup latency for flows and to require flooding. The default routing mechanism we plan to support does the following. When a cluster head receives a packet from its child destined for a node which is not a cluster head (and not present in the cache), he sends out a first packet directly. This packet reaches the destination using the underlying physical network. The response to this packet will contain the cluster head of the destination. This will make the source cluster head cache this little piece of information and use it for routing all the packets to that destination received after that. This ensures no major route setting up cost in terms of time (like in the reactive solution). By caching routes, we can expect that the popular destinations will likely exist in the cache, making th/e scheme appealing.

4 Evaluation

Note: We have not yet completed the simulation study part of the work. This section describes our evaluation plan, rather than any actual results. The implementation of the TAO algorithm inside the simulator is mostly complete, as is the trace-based dataset. The synthetic datasets have not yet been implemented

To evaluate the TAO algorithm, we have simulated its behavior on both a trace-based and a synthetic network, and compared it to two simple algorithms, one based on a centralized database and full connectivity, and the other on expanding-ring search with IP multicast. Both of our current simulations involve static networks, where the latency between each pair of nodes is constant. This permits us to test the behavior as nodes are added and removed in the overlay, but does not permit us to consider the dynamic behavior in response to Internet load. In the future we hope to experiment with more sophisticated trace-based studies and/or live Internet experiments to better understand these dynamics.

There are two metrics that we hope to optimize with the TAO algorithm:

- The network cost of building the overlay, measured in link-packets (i.e. one packet traversing one link is one link-packet; packet size is not considered), and
- The average delay between any two overlay nodes in the finished network.

4.1 Simulation setup

Our simulation environment is a simple work-queue algorithm focusing on the messages being exchanged by the protocol. We do not account for processing overhead, which we believe will be insignificant. We do, however, account for the delay of messages in the network. We start with an empty overlay, and add nodes one at a time in a random order.

Our network model consists of a set of nodes and latency measurements between those nodes. As the algorithm runs, it is permitted to query the latency data and charged appropriately. It is assumed that the algorithm can obtain both the current latency along a path (i.e. using ping) or the estimated physical propagation delay (which would be estimated in practice as the minimum observed latency over some history). The simulation model is able to provide either sort of latency value.

4.2 Trace-based simulation

Our trace-based simulation is built using the UW3 trace from the Detour project [6]. This trace consists of periodic traceroute measurements between 38 hosts on the Internet, taken over a period of a week. For each pair of hosts, we compute the minimum and average round-trip time to serve as the estimates of propagation delay and instantaneous latency for the TAO algorithm. It is known that the UW3 dataset, in contrast to some of the other Detour traces, does not suffer from excessive route flapping that might affect the minimum value.

Unfortunately, despite the fact that they consist of a series of measurements, the Detour traces cannot easily be used to simulate the behavior of the TAO algorithm in response to changes in network performance. This is because the measurements are independent along each path, and therefore we cannot in general compare the instantaneous delay along two paths, because we do not have simultaneous measurements. There is another Detour trace, UW4, which does have appropriate simultaneous measurements, but it has only 15 hosts, which is too small to show interesting behavior in the TAO algorithm.

4.3 Synthetic simulation

In order to run simulations with a larger number of nodes, we hope to construct several synthetic topologies using the Tiers [11] topology generator. The idea is to compute routes over the generated topology using a simple shortest-path routing algorithm (because of the strongly hierarchical nature of the Tiers topology we feel it is unnecessarily complicated to simulate the behavior of separate interdomain/intradomain routing as in the actual Internet).

Tiers supplies a propagation delay on each link in the topology, based on its length, as well as a link bandwidth, based on the level in the hierarchy of networks. In order to properly simulate actual network delays, we will most likely need to add an additional queuing delay to each link. How to do this properly remains an open problem.

4.4 Comparisons

We are unaware of any other performance results for the virtual network configuration problem, so we have implemented two very simple algorithms to use by way of comparison. Both treat the overlay as a flat structure without any sort of hierarchy, and both attempt to build a topology similar to a minimum spanning tree, but incrementally as new nodes are added. Neither does any sort of reconfiguration to optimize older parts of the overlay in response to the arrival of new nodes.

4.4.1 The CDFM algorithm

The CDFM (centralized database, full measurement) algorithm separates the tasks of finding the existing nodes in the overlay and determining which tunnels to configure to connect to them. A simple, centralized database is used to track the nodes in the network. This database runs at a single, well-known server. When a new node arrives, it contacts this server, informing it of the new node's existence and obtaining a list of all other nodes. The server serializes all requests, for the sake of easy analysis, although a real implementation could probably get by with a weaker consistency model.

Once the new node has the list of other nodes, it probes each of them for latency and establishes a tunnel to the closest node. The CDFM algorithm measures propagation delay (as described in section 4.1), rather than instantaneous latency, because it is a more stable metric. As noted above, once a tunnel is established in the CDFM algorithm it is not torn down until the node leaves the system.

The database cost in the CDFM algorithm is roughly comparable to the DNS cost of the TAO algorithm (ignoring for the moment the single point of failure issue). In both cases, each node must query the database when it starts up. The DNS requires more probes to locate the correct server (though as long as the DNS server for the overlay is a top-level domain only two probes should be required), while the CDFM database is more expensive to implement (since it must track all the nodes) and more prone to failure. The tradeoffs are too subtle to attempt to capture given our simple simulations, so we will ignore the database costs for both algorithms.

Note that, as specified, the CDFM algorithm is incomplete, because node departures could lead to network partition. Given the centralized database it should not be too difficult to recover from this situation (because the overlay structure is a spanning tree we know that removing any edge partitions the network, and we can have the remaining node take responsibility for replacing the tunnel). Dealing with node departure, however, is not required for our comparisons, so we will leave that aspect of the algorithm unspecified.

4.4.2 The ERST algorithm

The ERST (expanding-ring spanning tree) algorithm avoids the use of a centralized database by relying on IP multicast instead. When nodes start up, they perform an expanding-ring search to locate their nearest neighbor in the overlay, and establish a tunnel to that node. Unlike both the TAO and CDFM algorithms, ERST requires no central database at all.

Expanding-ring multicast (originally described by Boggs in his thesis on Internet broadcasting [12],[14]) is a common technique for locating nearby services using IP multicasting. The idea is to send a request packet *all* servers (using multicast), but with an extremely small TTL (generally starting with a TTL of one hop). If there is no response, then the TTL is increased and the request is sent again, until some server responds. Hopefully the first server to respond will be the closest (at least in terms of hops, and ignoring some of the odd ways that multicast TTLs are used for access control). This style of search is aesthetically pleasing because it requires no central coordination at all (disregarding the work done by the IP multicast implementation, of course).

The problem with expanding-ring multicast is that it is expensive, particularly for searches that ultimately find distant nodes. This has the potential to be a serious problem for ERST, especially for the first few searches. One can, however, hope that the small number of receivers on the multicast tree during the early part of the algorithm will help to compensate for the costs of the wide-area searches.

Our simulation framework makes it easy to determine which node will be found by an expanding-ring multicast (since we know the hopcount on all possible paths between nodes). In order to compute the cost of the search, which we hope to measure in total links traversed, we apply the scaling law of Chuang and Sirbu [13],[15], which states that the total cost of a multicast tree is $\hat{a} \cdot m^{0.8}$ where $\hat{a}$ is the average number of hops between the sender and a receiver, and m is the total number of receivers (compare this to $\hat{a} \cdot m$, which is the cost of repeated unicast, to see the efficiency benefit of multicast). This implies that the cost of a truncated multicast is $t \cdot m^{0.8}$, where t is the TTL, and therefore that the cost of a full expanding-ring search is $(\Sigma_{t=1}^{h} t) m^{0.8}$ or $\frac{h(h+1)}{2} m^{0.8}$, where h is the number of hops to the closest receiver (where the search terminates), and m is the total number of receivers.

The analysis ignores the cost of the very first node determining that there are in fact no other nodes to connect to. This will take a large number of probes, until the TTL is made sufficiently large for the probe packets to potentially visit the entire Internet, but no packets should actually leave the local subnet, because there are no listeners on the multicast channel.

As with the CDFM algorithm, we will leave unspecified the node departure behavior, because it is not required for the simulations we will present.

4.5 Simulation results

To evaluate the TAO algorithm in comparison with the CDFM and ERST algorithms, we will consider separately the cost of running the algorithm with the quality of the generated topology.

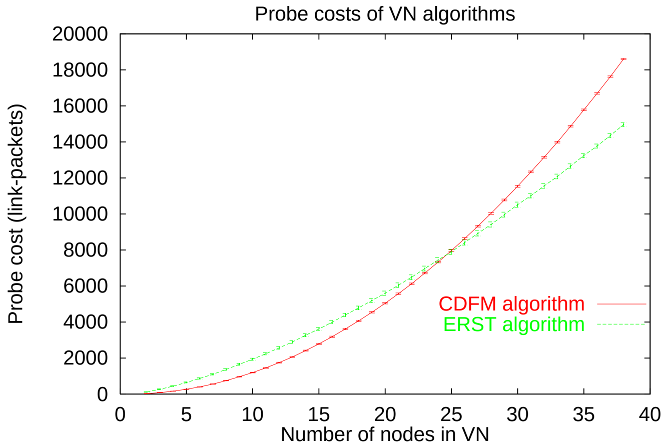

Figure 3: Cost to build a virtual network topology, in total link-packets transmitted, averaged across 100 runs of each virtual network management algorithm. Errorbars show 95% confidence intervals on the mean costs.

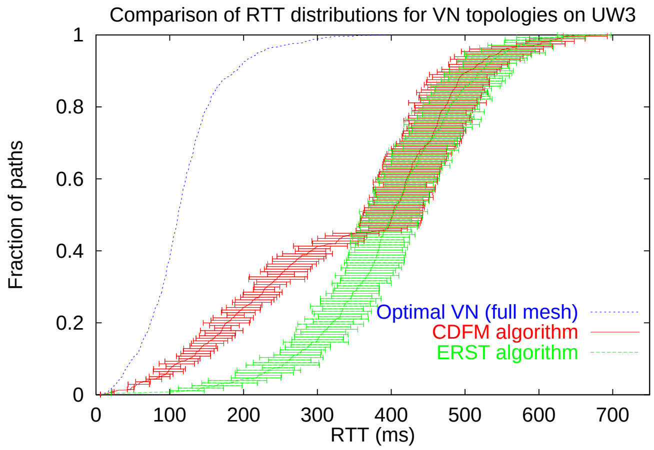

Figure 4: CDF of the virtual network latencies in the final generated topologies, averaged across 100 runs of each virtual network management algorithm. Errorbards show 95% confidence intervals on the mean latency for each path.

Figure 3 shows the cost of running the CDFM and ERST algorithms. The x axis counts the number of nodes in the overlay while the y axis shows the total cost to construct an overlay of that size. The costs are measured in total link traversals, with the cost of the expanding-ring searches in the ERST algorithm estimated as described above. The figure was generated using the UW3 trace data. Each algorithm was run 100 times, with the order in which the nodes were added chosen at random; the costs shown are the mean over these 100 runs. The 95% confidence intervals are shown as well.

Both CDFM and ERST exhibit quadratic growth in the number of measurements required to build the overlay. Our hope is that the TAO algorithm will require substantially fewer and shorter measurement probes. It is interesting to note the crossover between the CDFM and ERST algoriths. For large networks, expanding-ring multicast is superior because it does not require probing all existing nodes when a new node is added. However, for smaller networks, the wasted work in the expanding-ring algorithm (i.e. all those probes which terminate without reaching the nearest target) dominate the cost and make the simple CDFM algorithm superior.

Figure 4 shows the quality of the overlays generated by the algorithms. Again each algorithm was run 100 times. After each run, we ran a latency-optimizing routing algorithm over the generated overlay topology, and computed the resulting latency between each pair of nodes. The figure shows a CDF of these resulting mean latencies, as well as 95% confidence intervals. The variation is much larger here than it was in figure 3, showing that for CDFM and ERST, although the cost to build the overlay varies little with the order in which the nodes are added, the quality of the final topology varies substantially. This is not surprising, considering that neither attepts any sort of reconfiguration as the topology grows.

Because both ERST and CDFM generate very sparse topologies, most routes are long and therefore most latencies large. TAO should have somewhat smaller latencies, both because of its superior intra-cluster routing and its generally shorter inter-cluster routes. For comparison, the figure shows the best-possible latencies that can be acheived using a full-mesh overlay topology. All of the topologies we consider are far to restrictive to even approach this lower bound.

5 Extending to other classes of virtual networks

The problem examined up to this point is a fairly conventional example in the space of virtual network configuration. In this section, we will briefly consider some more exotic examples of problems and solution structures, and consider how we might apply our ideas and/or extend the TAO algorithm to support them.

5.1 Client-server virtual networks

One of the basic assumptions in the TAO algorithm is that all nodes are essentially equal, and that any of them is willing to provide transport to any other, with the goal of globally optimizing performance. This seems reasonable for an academic system such as the ABone, but seems less feasible in a commercial system, where clients might expect to be paying for a service directly, and have that service provided by a set of servers distinct from the client population.

What this leads to is a different kind of overlay structure with two classes of nodes. *Core* nodes provide the service (be it routing or distributed computation or caching or what not). They are willing to provide transit and computation services as necessary for the application. It is safe to assume that they will be well-configured and most likely dedicated to their task, and that there will be a relatively small number of them in the system (say comparable to the number of cluster heads in the TAO algorithm). This actually makes the topology configuration problem between the core nodes much simpler, and in most cases a simple full connectivity pattern should not be unreasonable in this part of the network.

The clients in this core-centric network, on the other hand, are characterized by their unwillingness to provide transit service. They will connect to the core through a small set of tunnels, and will have no tunnels to other client nodes. All communication between clients must either travel directly between them (bypassing the virtual network entirely) or from the source, into the core nodes, which route optimally among themselves, and then to the destination. Unless there is some reason to take a longer route, this can always be done in three virtual hops, assuming full connectivity between the core nodes.

5.1.1 Adapting the TAO algorithm

One can think of the core of a core-centric network as corresponding to the cluster heads in the TAO algorithm. One can either use the incremental tunnel configuration algorithm that the TAO algorithm uses between cluster heads to connect the core, or use full-connectivity and suffer somewhat in the maximum size core that can be managed.

The challenge in configuring the overlay lies in determining which core nodes to connect a client to, and the hill-climbing algorithm used in TAO should work reasonably well here. Clients will end up connecting to the core node that would have been their cluster head, but will have no special relationship with the other clients which happen to connect to the same point.

5.2 Per-request virtual networks

Another basic assumption of the TAO algorithm is that we are constructing a single virtual topology for all data on the overlay. This topology will inherently be a compromise for many kinds of applications (consider an overlay aimed at intelligently trading off latency versus bandwidth for different data transfers). In a *per-request* virtual network we separate the task of maintaining measurements between the nodes from the task of configuring tunnels, and delay the tunnel configuration until we actually have data to send.

When a flow starts up in such a system, the nodes consult their measurement databases, figure out which route makes sense for the flow, and then configure only those tunnels that are actually needed. Some time after the flow ends the tunnels are torn down. In general, it will make sense to keep up the measurements between nodes constantly, in order to reduce the latency at startup. Although the TAO algorithm cannot be directly applied in this space, many of the ideas, including the notion of a hierarchy where nodes communicate only inside their clusters, can be adapted to maintaining the measurements needed to quickly locate the appropriate tunnels to satisfy a flow's requirements.

6 Conclusions

We have presented the TAO algorithm for automatically configuring and managing virtual networks. This algorithm obviates the need for administrators to manually configure all the nodes of an overlay and determine the best set of tunnels to use to connect them. By moving to an automated system we can build overlays which are more reactive to changing network conditions, better able to deal with multiple administraive domains, and more scalable.

Our algorithm is designed with the ABone environment in mind, but is easily extensible to a wide variety of similar virtual networks, and the ideas can be applied to wider variety of overlay structures.

References

[1] J. Touch and S. Hotz, "The X-BONE", *Third Global Internet Mini-Conference in conjunction with Globecom '98* Sydney, Australia Nov. 8-12, 1998.

[2] Wroclawski J., "The Metanet", *NGI Workshop white paper*, April 1997.

[3] Delgrossi, L., Ferrari, D., "A virtual Network Service for Integrated-Services Internetworks", *7th International Workshop on Network and Operating System Support for Digital Audio and Video*, St. Louis, May 1997.

[4] Paul F. Tsuchiya, "The Landmark Hierarchy: A New Hierarchy for Routing in Very Large Networks", *ACM SIGCOMM 1988*.

[5] Rajesh Krishnan, Ram Ramanathan, Martha Streenstrup, "Optimization Algorithms for Large Self-Structuring Networks", *IEEE INFOCOMM 99*.

[6] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The End-to-End Effects of Internet Path Selection. In *Proceedings of the ACM SIGCOMM '99*, pages 289–299, Boston, MA, September 1999.

[7] Eriksson, H., "MBone: The Multicast Backbone", *Communications of the ACM*, Aug. 1994, Vol. 37, pp. 54-60.

[8] Zygmunt J. Haas, Marc R. Pearlman, "The Zone Routing Protocol (ZRP) for Ad Hoc Networks", *Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-zone-zrp-02.txt*.

[9] Mingliang Jiang, Jinyang Li, Y.C. Tay, "Cluster Based Routing Protocol(CBRP)", *Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-cbrp-spec-01.txt*.

[10] Abone Web Pages, *http://www.isi.edu/abone/*.

[11] K. Calvert, M. Doar, E.W. Zegura. Modeling Internet Topology. *IEEE Communications*, IEEE, June 1997.

[12] David R. Boggs. Internet broadcasting, January 1982. Ph.D. Thesis. Technical Report CSL-83-3, Xerox Palo Alto Research Center, October 1983.

[13] J. Chuang and M. Sirbu. Pricing multicast communications: A cost-based approach. In *Proceedings of the INET '98*, 1998.

[14] James D. Guyton and Michael F. Schwartz. Locating Nearby Copies of Replicated Internet Servers. Technical Report CU-CS-762-95, University of Colorado at Boulder, Department of Computer Science, February 1995.

[15] Graham Phillips, Scott Shenker, and Hongsuda Tangmunarunkit. Scaling of Multicast Trees: Comments on the Chuang-Sirbu scaling law. In *Proceedings of the ACM SIGCOMM '99*, pages 41–51, Boston, MA, September 1999.