A Scalable Virtual Circuit Routing Scheme for ATM Networks*

Cengiz Alaettinoğlu — Information Sciences Institute, University of Southern California, Marina del Rey, CA 90292

Ibrahim Matta, A. Udaya Shankar — Department of Computer Science, University of Maryland, College Park, MD 20742

Abstract

We present a scalable VC routing protocol based on the recently proposed viewserver hierarchy. Each viewserver maintains a partial view of the network. By querying these viewservers, a source obtains a merged view that contains a path to the destination. The source then sends a setup request packet over this path to reserve resources. We use simulation to compare our protocol to a simple approach that maintains a full view of the network. In addition to the savings in storage, our protocol performs as well or better in terms of VC carried load and blocking probability over a wide range of real-time workload.

1 Introduction

Integrated services packet-switched networks, such as Asynchronous Transfer Mode (ATM) networks [22], are expected to carry a wide variety of applications with heterogeneous quality of service (QoS) requirements. For this purpose, new resource allocation algorithms and protocols have been proposed, including link scheduling, admission control, and routing. Link scheduling defines how the link bandwidth is allocated among the different services. Admission control defines the criteria the network uses to decide whether to accept or reject a new incoming application. Routing concerns the selection of routes to be taken by application packets (or cells) to reach their destination. In this paper, we are mainly concerned with routing for real-time applications (e.g., voice, video) requiring QoS guarantees (e.g., bandwidth and delay guarantees).

To provide real-time QoS support, a number of *virtual-circuit* (VC) routing approaches have been proposed. A *simple* (or straightforward) approach to VC routing is the link-state full-view approach. Here, each end-system maintains a view of the whole network, i.e. a graph with a vertex for every node[1] and an edge between two neighbor nodes. QoS information such as delay, bandwidth, and loss rate are attached to the vertices and the edges of the view. This QoS information is flooded regularly to all end-systems to update their views. When a new application requests service from the network, the source end-system uses its current view to select a *source route* to the destination end-system that is likely to support the application's requested QoS, i.e., a sequence of node ids starting from the source end-system and ending with the destination end-system. A VC-setup message is then sent over the selected source route to try to reserve the necessary resources (bandwidth, buffer space, service priority) and establish a VC.

Typically, at every node the VC-setup message visits, a set of admission control tests are performed to decide whether the new VC, if established, can be guaranteed its requested QoS without violating the QoS guaranteed to already established VCs. At any node, if these admission tests are passed, then resources are reserved and the VC-setup message is forwarded to the next node. On the other hand, if the admission tests fail, a VC-rejected message is sent back towards the source node releasing resource reservations made by the VC-setup message, and the application request is either blocked or another source route is selected and tried. If the final admission tests at the destination node are passed, then a VC-established message is sent back towards the source node confirming resource reservations made during the forward trip of the VC-setup message. Upon receiving the VC-established message, the application can start transmitting its packets over its reserved VC. This VC is torn down and resources are released at the end of the transmission.

Clearly, the above simple routing scheme does not scale up to large networks. The storage at each end-system and the communication cost are proportional to $N \times d$, where N is the number of nodes and d is the average number of neighbors to a node.

*This work is supported in part by ARPA and Philips Labs under contract DASG60-92-0055 to Department of Computer Science, University of Maryland, and by NSF grant NCR-89-04590. The work of Cengiz Alaettinoğlu is also supported by NSF Cooperative Agreement NCR-93-21043 to USC/ISI.

[1] We refer to switches and end-systems collectively as nodes.

A traditional solution to this scaling problem is the area hierarchy used in routing protocols such as the Open Shortest Path First (OSPF) protocol [18]. The basic idea is to aggregate nodes hierarchically into areas: "close" nodes are aggregated into level 1 areas, "close" level 1 areas are aggregated into level 2 areas, and so on. An end-system maintains a view that contains the nodes in the same level 1 area, the level 1 areas in the same level 2 area, and so on. Thus an end-system maintains a smaller view than it would in the absence of hierarchy. Each area has its own QoS information derived from that of the subareas. A major problem of an area-based scheme is that aggregation results in loosing detailed link-level QoS information. This decreases the chance of the routing algorithm to choose "good" routes, i.e. routes that result in high successful VC setup rate (or equivalently high carried VC load).

Our scheme

In this paper, we present a scalable VC routing scheme that does not suffer from the problems of areas. Our scheme is based on the viewserver hierarchy we recently proposed in [4, 3] for large internetworks and evaluated for administrative policy constraints. Here, we are concerned with the support of performance/QoS requirements in large wide-area ATM-like networks, and we adapt our viewserver protocols accordingly.

In our scheme, views are not maintained by every end-system but by special switches called *viewservers*. For each viewserver, there is a subset of nodes around it, referred to as the viewserver's *precinct*. The viewserver only maintains the view of its precinct. This solves the scaling problem for storage requirement.

A viewserver can provide source routes for VCs between source and destination end-systems in its precinct. Obtaining a route between a source and a destination that are not in any single view involves accumulating the views of a sequence of viewservers. To make this process efficient, viewservers are organized hierarchically in levels, and an associated addressing structure is used. Each end-system has a set of addresses. Each *address* is a sequence of viewserver ids of decreasing levels, starting at the top level and going towards the end-system. The idea is that when the views of the viewservers in an address are merged, the merged view contains routes to the end-system from the top level viewservers.

We handle dynamic topology changes such as node/link failures and repairs, and link cost changes. Nodes detect topology changes affecting itself and neighbor nodes. Each node communicates these changes by flooding to the viewservers in a specified subset of nodes; this subset is referred to as its *flood area*. Hence, the number of packets used during flooding is proportional to the size of the flood area. This solves the scaling problem for the communication requirement.

Thus our VC routing protocol consists of two subprotocols: a *view-query protocol* between end-systems and viewservers for obtaining merged views; and a *view-update protocol* between nodes and viewservers for updating views.

Evaluation

In this paper, we compare our viewserver-based VC routing scheme to the simple scheme under real-time workload using VC-level simulation. We do not compare our scheme against other scalable schemes. This is because of lack of time, and because precise definitions of the hierarchies in these schemes are not available. For example, to do a fair evaluation of an area-based scheme, we need precise guidelines for how to group nodes into areas, and how to derive QoS information of an area from that of its subareas.

In our simulation model, we define network topologies, QoS requirements, viewserver hierarchies, and evaluation measures. Our evaluation measures are the amount of memory required at the end-systems, the amount of time needed to construct a path[2], the carried VC load, and the VC blocking probability. We use network topologies each of size 2764 nodes. Our results indicate that our viewserver-based VC routing scheme performs close to or better than the simple scheme in terms of VC carried load and blocking probability over a wide range of workload. It also reduces the amount of memory requirement by up to two order of magnitude.

Organization of the paper

Section 2 discusses recent approaches to real-time VC routing. In Section 3, we present the view-query protocol for static network conditions, that is, assuming all links and nodes of the network remain operational. Due to space reasons, we omit the view-update protocol that handles topology changes and also the protocol specifications; they can be found in [2]. In Section 4, we present our evaluation model. Our results are presented in Section 5. Section 6 concludes the paper.

2 Related Work

In this section, we discuss VC routing schemes recently proposed for real-time QoS networks.[3] Most of these schemes are based on the link-state full-view approach described in Section 1 (e.g. [7, 1]). Generally, a cost function is defined in terms of the QoS information in the view. The route selection algorithm favors short paths with minimum cost.

VC routing schemes based on the path-vector approach have also been proposed [14]. The scheme in

[2] We use the terms route and path interchangeably.

[3] We refer the reader to [20, 7] for a good survey on many other routing schemes.

[14] provides two kinds of routes: pre-computed and on-demand. Calculating on-demand routes involves broadcasting a special packet over all candidate paths, which can be exponentially expensive.

The link-state approach is often proposed and favored over the path-vector approach in QoS architectures for several reasons [17]. An obvious reason is simplicity and complete control of the source over QoS route selection.

The above VC routing schemes do not scale well to large QoS networks in terms of storage and communication requirements. Several approaches to achieve scaling exist. The most common approach is the area hierarchy described in Section 1.

Other approaches to scaling include the link-vector approach [6] and the landmark hierarchy [24]. A thorough study of enforcing QoS and policy constraints with these two approaches has not been done.

A number of VC routing schemes have also been designed for networks that use the Virtual Path (VP) concept [16, 15]. This VP concept has been proposed to achieve scaling and simplify network management and control. Typically, routing schemes that only consider paths with one VP and two VPs are used. However, it is not clear how to install VPs [9]. Also, the static allocation of resources to VPs can result in resource under-utilization and higher blocking probabilities [16]. One approach to solve this problem is to install fewer VPs and thus use longer paths (i.e. consisting of more than two VPs). Another approach is to dynamically allocate resources to VPs [19]. It is not clear, however, how these allocations could be varied effectively without causing massive instability[4]. This is a hard problem and beyond the scope of this paper. In this paper, we are interested in *general* network topologies, where the shortest paths can be of arbitrary hop length and the overhead of routing protocols is of much concern.

Finally, we should point out that extensive effort is currently underway to fully specify and standardize VC routing schemes for the future integrated services Internet and ATM networks [11].

3 Viewserver Hierarchy Query Protocol

In this section, we present our scheme for static network conditions, that is, all links and nodes remain operational. The dynamic case is presented in [2].

Conventions: Each node has a unique id. `NodeIds` denotes the set of node-ids. For a node u, we use $nodeid(u)$ to denote the id of u. $NodeNeighbors(u)$ denotes the set of ids of the neighbors of u.

[4] Instability could result because of the strong interaction between the VP allocation and routing/admission control.

In our protocol, a node u uses two kinds of sends. The first kind has the form "Send(m) to v", where m is the message being sent and v is the destination-id. Here, nodes u and v are neighbors, and the message is sent over the physical link (u, v). If the link is down, we assume that the packet is dropped.

The second kind of send has the form "Send(m) to v using sr", where m and v are as above and sr is a source route between u and v. We assume that as long as there is a sequence of up links connecting the nodes in sr, the message is delivered to v. This requires a transport protocol support such as TCP [21].

To implement both kind of sends, we assume there is a reserved VC on each link for sending routing, signaling and control messages [5]. This also ensures that routing messages do not degrade the QoS seen by applications.

Views and Viewservers

Views are maintained by special nodes called *viewservers*. Each viewserver has a *precinct*, which is a set of nodes around the viewserver. A viewserver maintains a *view*, consisting of the nodes in its precinct, links between these nodes and links outgoing from the precinct[5]. Formally, a viewserver x maintains the following:

$Precinct_x \subseteq$ `NodeIds`. Nodes whose view is maintained.

$View_x$. View of x.

$$= \{\langle u,\ timestamp,\ expirytime,\ \{\langle v,\ cost\rangle : v \in NodeNeighbors(u)\}\rangle : u \in Precinct_x\}$$

The intention of $View_x$ is to obtain source routes between nodes in $Precinct_x$. Hence, the choice of nodes to include in $Precinct_x$ and the choice of links to include in $View_x$ are not arbitrary. $Precinct_x$ and $View_x$ must be connected; that is, between any two nodes in $Precinct_x$, there should be a path in $View_x$. Note that $View_x$ can contain links to nodes outside $Precinct_x$. We say that a node u is *in the view* of a viewserver x, if either u is in the precinct of x, or $View_x$ has a link from a node in the precinct of x to node u. Note that the precincts and views of different viewservers can be overlapping, identical or disjoint.

For a link (u, v) in the view of a viewserver x, $View_x$ stores a *cost*. The cost of the link (u, v) equals a vector of values if the link is known to be up; each cost value estimates how expensive it is to cross the link according to some QoS criteria such as delay, throughput, loss rate, etc. The cost equals ∞ if the link is known to be down. The cost of a link changes with time (see [2]). The view also includes *timestamp* and *expirytime* fields which are described in [2].

[5] Not all the links need to be included.

Viewserver Hierarchy

For scaling reasons, we cannot have one large view. Thus, obtaining a source route between a source and a destination which are far away, involves accumulating views of a sequence of viewservers. To keep this process efficient, we organize viewservers hierarchically. More precisely, each viewserver is assigned a hierarchy level from $0, 1, \ldots$, with 0 being the top level in the hierarchy. A parent-child relationship between viewservers is defined as follows:

1. Every level i viewserver, $i > 0$, has a parent viewserver whose level is less than i.
2. If viewserver x is a parent of viewserver y then x's precinct contains y and y's precinct contains x.
3. The precinct of a top level viewserver contains all other top level viewservers.

In the hierarchy, a parent can have many children and a child can have many parents. We extend the range of the parent-child relationship to ordinary nodes; that is, if $Precinct_x$ contains the node u, we say that u is a child of x, and x is a parent of u. We assume that there is at least one parent viewserver for each node.

For a node u, an address is defined to be a sequence $\langle x_0, x_1, \ldots, x_t \rangle$ such that x_i for $i < t$ is a viewserver-id, x_0 is a top level viewserver-id, x_t is the id of u, and x_i is a parent of x_{i+1}. A node may have many addresses since the parent-child relationship is many-to-many. If a source node wants to establish a VC to a destination node, it first queries the name servers to obtain a set of addresses for the destination[6]. Second, it queries viewservers to obtain an accumulated view containing both itself and the destination node (it can reach its parent viewservers by using fixed source routes to them). Then, it chooses a feasible source route from this accumulated view and initiates the VC setup protocol on this path.

Figure 1 illustrates a viewserver hierarchy.

View-Query Protocol: Obtaining Source Routes

We now describe how a source route is obtained.

We want a sequence of viewservers whose merged views contains both the source and the destination nodes. Addresses provide a way to obtain such a sequence, by first going up in the viewserver hierarchy starting from the source node and then going down in the viewserver hierarchy towards the destination node. More precisely, let $\langle s_0, \ldots, s_t \rangle$ be an address of the source, and $\langle d_0, \ldots, d_l \rangle$ be an address of the destination. Then, the sequence $\langle s_{t-1}, \ldots, s_0, d_0, \ldots, d_{l-1} \rangle$ meets our requirements. In fact, going up all the way in the hierarchy to top level viewservers may not be necessary. We can stop going up at a viewserver s_i if there is a viewserver $d_j, j < l$, in the view of s_i (one special case is where $s_i = d_j$).

[6] Querying the name servers can be done in the same way as is done currently in the Internet.

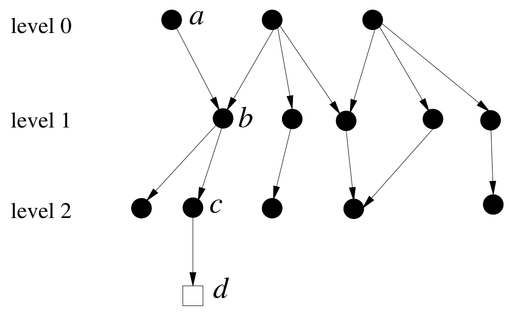

Figure 1: Viewserver hierarchy. A black circle represents a viewserver. A box represents an end-system. Arrows indicate the parent-child relationship. An address of an end-system is a directed path on this graph, starting from a top level viewserver and ending with the end-system. For example, an address for end-system d is $a.b.c.d$.

The view-query protocol uses two message types:

- (`RequestView`, *s_address*, *d_address*) where *s_address* and *d_address* are the addresses for the source and the destination respectively. A `RequestView` message is sent by a source node to obtain an accumulated view containing both the source and the destination nodes. When a viewserver receives a `RequestView` message, it either sends back its view or forwards this request to another viewserver.
- (`ReplyView`, *s_address*, *d_address*, *accumview*) where *s_address* and *d_address* are as above and *accumview* is the accumulated view. A `ReplyView` message is sent by a viewserver to the source or to another viewserver closer to the source. The *accumview* field in a `ReplyView` message equals the union of the views of the viewservers the message has visited.

We now describe the view-query protocol in more detail. To establish a VC to a destination node, the source node sends a `RequestView` packet containing the source and the destination addresses to its parent in the source address.

Upon receiving a `RequestView` packet, a viewserver x checks if the destination node is in its precinct[7]. If it is, x sends back its view in a `ReplyView` packet. If it is not, x forwards the request packet to another viewserver as follows: x checks whether any viewserver in the destination address is in its view. If there is such a viewserver, x sends the `RequestView` packet to the last such one in the destination address. Otherwise x

[7] Even though the destination can be in the view of x, its QoS characteristics is not in the view if it is not in the precinct of x.

is a viewserver in the source address, and it sends the packet to its parent in the source address.

When a viewserver x receives a `ReplyView` packet, it merges its view to the accumulated view in the packet. Then it sends the `ReplyView` packet towards the source node in the same way it would send a `RequestView` packet towards the destination node (i.e. the roles of the source address and the destination address are interchanged).

When the source receives a `ReplyView` packet, it chooses a feasible path using the *accumview* in the packet. If it does not find a feasible path, it can try again using a different source and/or destination addresses. Note that the source does not have to throw away the previous accumulated views; it can merge them all into a richer accumulated view. In fact, it is easy to change the protocol so that the source can also obtain views of individual viewservers to make the accumulated view even richer. Once a feasible source route is found, the source node initiates the VC setup protocol. Figure 2 illustrates the query protocol showing an example merged view.

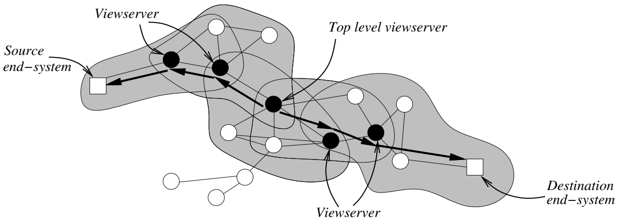

Figure 2: Viewserver-query protocol. A circle represents a switch. A box represents an end-system. Black circles represent viewservers. The shaded area around a viewserver is its view. Straight arrows indicate the viewserver addresses for the source and the destination end-systems. Views of the five viewservers in the addresses are merged. The merged view contains paths from the source end-system to the destination end-system.

Above we have described one possible way of obtaining the accumulated views. There are various other possibilities; see [2].

4 Evaluation

In this section, we present the parameters of our simulation model. We use this model to compare our viewserver-based VC routing protocols to the simple approach. The results obtained are presented in Section 5.

Network Parameters

We model a campus network which consists of a campus backbone subnetwork and several department subnetworks. The backbone network consists of backbone switches and backbone links.

Each department network consists of a hub switch and several non-hub switches. Each non-hub switch has a link to the department's hub switch. And the department's hub switch has a link to one of the backbone switches. A non-hub switch can have links to other non-hub switches in the same department, to non-hub switches in other departments, or to backbone switches.

End-systems are connected to non-hub switches. An example network topology is shown in Figure 3.

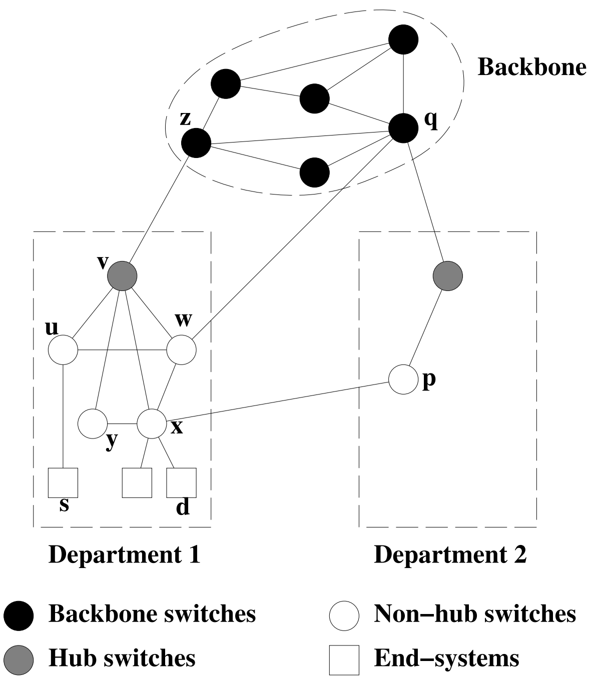

Figure 3: An example network topology.

In our topology, there are 8 backbone switches and 32 backbone links. There are 16 departments. There is one hub-switch in each department. There is a total of 240 non-hub switches randomly assigned to different departments. There are 2500 end-systems which are randomly connected to non-hub switches. Thus, we have a total of 2764 nodes.

In addition to the links connecting non-hub switches to the hub switches and hub switches to the backbone switches, there are 720 links from non-hub switches to non-hub switches in the same department, there are 128 links from non-hub switches to non-hub switches in different departments, and there are 64 links from non-hub switches to backbone switches.

The end-points of each link are chosen randomly. However, we make sure that the backbone network is connected; and there is a link from node u to node v iff there is a link from node v to node u.

Each link has a total of C units of bandwidth.

QoS and Workload Parameters

In our evaluation model, we assume that a VC requires the reservation of a certain amount of bandwidth that is enough to ensure an acceptable QoS for the application. This reservation amount can be thought of either as the peak transmission rate of the VC or its "effective bandwidth" [13] varying between the peak and average transmission rates.

VC setup requests arrive to the network according to a Poisson process of rate λ, each requiring one unit of bandwidth. Each VC, once it is successfully setup, has a lifetime of exponential duration with mean $1/\mu$. The source and the destination end-systems of a VC are chosen randomly.

An arriving VC is admitted to the network if at least one feasible path between its source and destination end-systems is found by the routing protocol, where a feasible path is one that has links with non-zero available capacity. From the set of feasible paths, a minimum hop path is used to establish the VC; one unit of bandwidth is allocated on each of its links for the lifetime of the VC. On the other hand, if a feasible path is not found, then the arriving VC is blocked and lost.

We assume that the available link capacities in the views of the viewservers are updated instantaneously whenever a VC is admitted to the network or terminates.

Viewserver Hierarchy Schemes

We have evaluated our viewserver protocol for several different viewserver hierarchies and query methods. We next describe the different viewserver schemes evaluated. Please refer to Figure 3 in the following discussion.

The first viewserver scheme is referred to as **base**. Each switch is a viewserver. A viewserver's precinct consist of itself and the neighboring nodes. The links in the viewserver's view consist of the links between the nodes in the precinct, and links outgoing from nodes in the precinct to nodes not in the precinct. For example, the precinct of viewserver u consists of nodes u, v, w, s.

As for the viewserver hierarchy, a backbone switch is a level 0 viewserver, a hub switch is a level 1 viewserver and a non-hub switch is a level 2 viewserver. Parent of a hub switch viewserver is the backbone switch viewserver it is connected to. Parent of a non-hub switch viewserver is the hub switch viewserver in its department. Parent of an end-system is the non-hub switch viewserver it is connected to.

We use only one address for each end-system. The viewserver-address of an end-system is the concatenation of four ids. Thus, the address of s is $z.v.u.s$. Similarly, the address of d is $z.v.x.d$. To obtain a route between s and d, it suffices to obtain views of viewservers u, v, x.

The second viewserver scheme is referred to as **base-QT** (where the QT stands for "query up to top"). It is identical to *base* except that during the query protocol all the viewservers in the source and the destination addresses are queried. That is, to obtain a route between s and d, the views of u, v, x, z are obtained.

The third viewserver scheme is referred to as **vertex-extension**. It is identical to *base* except that viewserver precincts are extended as follows: Let P denote the precinct of a viewserver in the *base* scheme. For each node u in P, if there is a link from node u to node v and v is not in P, node v is added to the precinct; among v's links, only the ones to nodes in P are added to the view. In the example, nodes z, y, x, q are added to the precinct of u, but outgoing links of these nodes to other nodes are not included (e.g. (x, p) and (z, q) are not included). The advantage of this scheme is that even though it increases the precinct size by a factor of d (where d is the average number of neighbors to a node), it increases the number of links stored in the view by a factor less than 2.

The fourth viewserver scheme is referred to as **vertex-extension-QT**. It is identical to *vertex-extension* except that during the query protocol all the viewservers in the source and the destination addresses are queried.

5 Numerical Results

We studied several networks [2]. Here we present results for only one network, referred to as Network 1. The parameters of the topology are as given in Section 4. The link capacity C is taken to be 20 [7], i.e. a link is capable of carrying 20 VCs simultaneously.

Our evaluation measures were computed for a (randomly chosen but fixed) set of 100,000 VC setup requests. Table 1 lists for each viewserver scheme (1) the minimum, average and maximum of the precinct sizes (in number of nodes), (2) the minimum, average and maximum of the merged view sizes (in number of nodes), and (3) the minimum, average and maximum of the number of viewservers queried.

The precinct size indicates the memory requirement at a viewserver. More precisely, the memory requirement at a viewserver is $O(\text{precinct size} \times d)$, except for the *vertex-extension* and *vertex-extension-QT* schemes. In these schemes, the memory requirement

Scheme	Precinct Size	Merged View Size	No. of Viewservers Queried
base	5 / 16.32 / 28	4 / 56.46 / 81	1 / 5.49 / 6
base-QT	5 / 16.32 / 28	27 / 59.96 / 81	6 / 6.00 / 6
vertex-extension	22 / 88.11 / 288	14 / 155.86 / 199	1 / 5.49 / 6
vertex-extension-QT	22 / 88.11 / 288	113 / 163.28 / 199	6 / 6.00 / 6

Table 1: Precinct sizes, merged view sizes, and number of viewservers queried for Network 1.

is increased by a factor less than two. Hence these schemes have the same order of viewserver memory requirement as the *base* and *base-QT* schemes.

The merged view size indicates the memory requirement at a source end-system during the query protocol; i.e. the memory requirement at a source end-system is $O(\text{merged view size} \times d)$ except for the *vertex-extension* and *vertex-extension-QT* schemes. Note that the source end-system does not need to store information about end-systems other than itself and the destination. The numbers in Table 1 take advantage of this.

The number of viewservers queried indicates the communication time required to obtain the merged view at the source end-system. Hence, the "real-time" communication time required to obtain the merged view at a source is slightly more than one round-trip time between the source and the destination.

As is apparent from Table 1, using a *QT* scheme increases the merged view size by about 6%, and the number of viewservers queried by about 9%. Using the *vertex-extension* scheme increases the merged view size by about 3 times (note that the amount of actual memory needed increases only by a factor less than 2).

The above measures show the memory and time requirements of our protocols. They clearly indicate the savings in storage over the simple approach as manifested by the smaller view sizes. To answer whether the viewserver hierarchy finds many feasible paths, other evaluation measures such as the carried VC load and the percent VC blocking are of interest. They are defined as follows:

- *Carried VC load* is the average number of VCs carried by the network.
- *Percent VC blocking* is the percentage of VC setup requests that are blocked due to the fact that a feasible path is not found.[8]

In our experiments, we keep the average VC lifetime $(1/\mu)$ fixed at 15000 and vary the arrival rate of VC setup requests (λ). Figure 4 shows the carried VC load versus λ for the simple approach and the viewserver schemes. Figure 5 shows the percent VC blocking versus λ. At low values of λ, all the viewserver schemes are very close to the simple approach. At moderate values of λ, the *base* and *base-QT* schemes perform badly. The *vertex-extension* and *vertex-extension-QT* schemes are still very close to the simple approach (only 3.4% less carried VC load). Note that the performance of the viewserver schemes can be further improved by trying more viewserver addresses.

Surprisingly, at high values of λ, all the viewserver schemes perform better than the simple approach. At $\lambda = 0.5$, the network with the *base* scheme carries about 30% higher load than the simple approach. This is an interesting result. Our explanation is as follows. Elsewhere [3], we have found that when the viewserver schemes can not find an existing feasible path, this path is usually very long (more than 11 hops). This causes our viewserver hierarchy protocols to reject VCs that are admitted by the simple approach over long paths. The use of long paths for VCs is undesirable since it ties up resources at more intermediate nodes, which can be used to admit many shorter length VCs. Observe that this produces the same effect obtained with the well-known "trunk reservation" mechanism [23], giving priority to short length VCs over long ones.

In conclusion, we recommend the *vertex-extension* scheme as it performs close to or better than all other schemes in terms of VC carried load and blocking probability over a wide range of workload. Note that for all viewserver schemes, adding *QT* yields slightly further improvement. However this is not always the case [2].

6 Conclusions

We presented a hierarchical VC routing protocol for ATM-like networks. Our protocol satisfies QoS constraints, adapts to dynamic topology changes, and scales well to large number of nodes.

In all the viewserver schemes we studied, each switch is a viewserver. In practice, not all switches need to be viewservers.

A drawback of our protocol is that to obtain a source route for a VC, views are merged at (or prior to) the VC setup, thereby increasing the setup time. This drawback is not unique to our scheme [10, 17, 8, 12]. Reference [4] describes several ways, including cacheing and replication, to reduce the setup overhead and improve performance.

References

[1] H. Ahmadi, J. Chen, and R. Guerin. Dynamic Routing and Call Control in High-Speed Integrated Networks. In Proc. *Workshop on Systems Engineering*

[8] Recall that we assume a blocked VC setup request is cleared (i.e. lost).

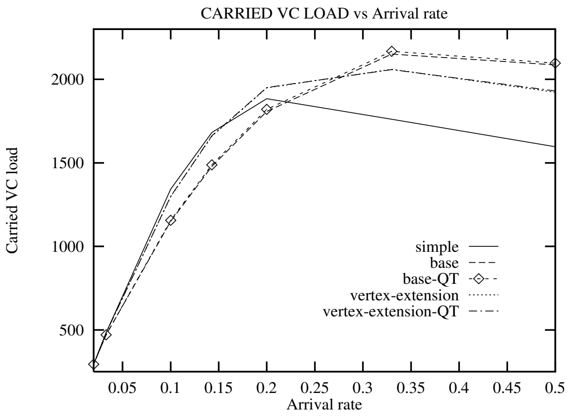

Figure 4: Carried VC load versus arrival rate for Network 1.

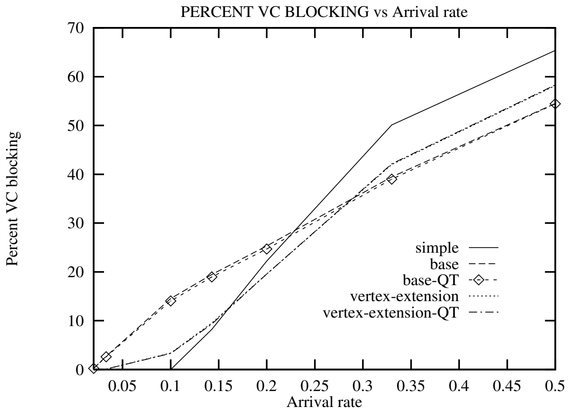

Figure 5: Percent VC blocking versus arrival rate for Network 1.

and Traffic Engineering, ITC'13, pages 19–26, Copenhagen, Denmark, June 1991.

[2] C. Alaettinoglu, I. Matta, and A.U. Shankar. A Scalable Virtual Circuit Routing Scheme for ATM Networks. Technical Report CS-TR-3360, Department of Computer Science, University of Maryland, College Park, MD 20742, October 1994.

[3] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol and its Evaluation. Technical Report UMIACS-TR-93-98, CS-TR-3151, Department of Computer Science, University of Maryland, College Park, October 1993. Earlier version CS-TR-3033, February 1993.

[4] C. Alaettinoğlu and A. U. Shankar. Viewserver Hierarchy: A New Inter-Domain Routing Protocol. In Proc. *IEEE INFOCOM '94*, Toronto, Canada, June 1994.

[5] A. Alles. ATM in Private Networking: A Tutorial. Hughes LAN Systems, 1993.

[6] J. Behrens and J.J. Garcia-Luna-Aceves. Distributed, Scalable Routing Based on Link-State Vectors. In Proc. *ACM SIGCOMM '94*, pages 136–147, September 1994.

[7] L. Breslau, D. Estrin, and L. Zhang. A Simulation Study of Adaptive Source Routing in Integrated Services Networks. Available by anonymous ftp at catarina.usc.edu:pub/breslau, September 1993.

[8] J. N. Chiappa. A New IP Routing and Addressing Architecture. Big-Internet mailing list., 1992. Available by anonymous ftp from `munnari.oz.au:big-internet/list-archive`.

[9] I. Chlamtac, A. Faragó, and T. Zhang. Optimizing the System of Virtual Paths. *IEEE/ACM Transactions on Networking*, 2(6):581–587, December 1994.

[10] D.D. Clark. Policy routing in Internet protocols. Request for Comment RFC-1102, Network Information Center, May 1989.

[11] R. Coltun and M. Sosa. VC Routing Criteria. Internet Draft, March 1993.

[12] D. Estrin, Y. Rekhter, and S. Hotz. Scalable Inter-Domain Routing Architecture. In Proc. *ACM SIGCOMM '92*, pages 40–52, Baltimore, Maryland, August 1992.

[13] R. Guerin, H. Ahmadi, and M. Naghshineh. Equivalent Capacity and its Application to Bandwidth Allocation in High-Speed Networks. *IEEE J. Select. Areas Commun.*, SAC-9(7):968–981, September 1991.

[14] A. Guillen, R. Kia, and B. Sales. An Architecture for Virtual Circuit/QoS Routing. In Proc. *IEEE International Conference on Network Protocols '93*, pages 80–87, San Francisco, California, October 1993.

[15] S. Gupta, K. Ross, and M. ElZarki. Routing in Virtual Path Based ATM Networks. In Proc. *GLOBECOM '92*, pages 571–575, 1992.

[16] R-H. Hwang, J. Kurose, and D. Towsley. MDP Routing in ATM Networks Using Virtual Path Concept. In Proc. *IEEE INFOCOM*, pages 1509–1517, Toronto, Ontario, Canada, June 1994.

[17] M. Lepp and M. Steenstrup. An Architecture for Inter-Domain Policy Routing. Internet Draft. Available from the authors., June 1992.

[18] J. Moy. OSPF Version 2. RFC 1247, Network Information Center, SRI International, July 1991.

[19] S. Ohta and K. Sato. Dynamic Bandwidth Control of the Virtual Path in an Asynchronous Transfer Mode Network. *IEEE Transactions on Communications*, 40(7):1239–1247, July 1992.

[20] C. Parris and D. Ferrari. A Dynamic Connection Management Scheme for Guaranteed Performance Services in Packet-Switching Integrated Services Networks. Technical Report TR-93-005, International Computer Science Institute, Berkeley, California, January 1993.

[21] J. Postel. Transmission Control Protocol: DARPA Internet Program Protocol Specification. Request for Comment RFC-793, Network Information Center, SRI International, 1981.

[22] M. Prycker. *Asynchronous Transfer Mode - Solution for Broadband ISDN*. Ellis Horwood, 1991.

[23] S. Sibal and A. DeSimone. Controlling Alternate Routing in General-Mesh Packet Flow Networks. In Proc. *ACM SIGCOMM '94*, pages 168–179, September 1994.

[24] P. F. Tsuchiya. The Landmark Hierarchy:A New Hierarchy For Routing In Very Large Networks. In Proc. *ACM SIGCOMM '88*, August 1988.