Improving the Performance of Distributed Applications Using Active Networks

Ulana Legedza, David Wetherall and John Guttag *

Software Devices and Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract

An active network allows applications to inject customized programs into network nodes. This enables faster protocol innovation by making it easier to deploy new network protocols, even over the wide area. In this paper, we argue that the ability to introduce active protocols offers important opportunities for end-to-end performance improvements of distributed applications.

We begin by describing several active protocols that provide novel network services and discussing the potential impact of these kinds of services on end-to-end application performance. We then present and analyze the performance of an active networking protocol that uses caching within the network backbone to reduce load on both servers and backbone routers.

Keywords: active networks, caching, distributed applications, networking protocols, performance.

1 Introduction

Traditionally, the function of a network has been to deliver packets from one endpoint to another. Processing within the network has been limited largely to routing, simple QOS (quality of service) schemes, and congestion control. Today, however, there is considerable interest in pushing other kinds of processing into the network. Examples include the transport-level support for wireless links of snoop-TCP [1], and the application-specific filtering of network firewalls [14].

Active networks [24, 23] take this trend to the extreme. They allow servers and clients to inject customized programs into the nodes of the network, thus interposing application-specified computation between communicating endpoints. In this manner, the entire network may be treated as part of the overall system that can be specialized to achieve application efficiency.

Of course, it is not *a priori* obvious that a programmable network is a good idea. It clearly offers increased flexibility, but at some cost. Both the advantages and the costs can be examined along three independent dimensions:

1. Ease of creating and deploying protocols,
2. Impact on network services, and
3. Impact on network performance.

In [27] we addressed the first of these issues. We described an architecture and a toolkit that facilitates the construction and deployment of active application-specific protocols. We presented evidence that it is not terribly difficult to write active protocols and that the overhead of the support mechanisms for running them (including code transfer) is not prohibitive.

In this paper, we address the other two issues by focusing not on the mechanisms needed to introduce active application-specific network protocols, but rather on the protocols themselves. Our goal is to demonstrate two things:

- There are a variety of useful network services that involve processing at intermediate nodes, and
- The use of such services can lead to better end-to-end performance for applications.

We sketch a few simple applications of active networks, and discuss qualitatively how these uses of active networks impact end-to-end application performance. This starts to make the case for application-specific protocols by suggesting the kinds of things one might do within the network, if it were easy to do so. We also give a more detailed description of an in-network caching scheme that exploits some of the capabilities of active networks.

The remainder of this paper is organized as follows. Section 2 briefly describes our active network architecture, and explains how it allows application-specific protocols to be

*{ulana,djw,guttag}@lcs.mit.edu. http://www.sds.lcs.mit.edu/. This work was supported by DARPA, monitored by the Office of Naval Research under contract No. N66001-96-C-8522, and by seed funding from Sun Microsystems Inc.

deployed. This sets the stage for Sections 3 and 4, the heart of the paper, in which we discuss how the use of active networking technology can lead to performance improvements and then present a quantitative analysis of a detailed example. We contrast our approach with related work in Section 5, and conclude in Section 6.

2 Network Architecture

The protocols we consider in this paper are developed in the context of our active network architecture, called ANTS, and its prototype implementation. These are described in detail in [27]. Here we cover only those aspects needed to understand the programming model used to develop protocols.

Our architecture is composed of a set of nodes connected by point-to-point or shared medium channels. Unlike IP, the network service provided by ANTS is not fixed – it is flexible. Different applications are able to introduce new protocols into the network by specifying the routines to be executed as programmable network nodes forward messages.

2.1 Protocols and Capsules

To make use of programmable network elements, we require a model for combining forwarding routines at individual nodes into a pattern of behavior – a protocol – that defines the processing to occur across the network as a whole. In ANTS, this is accomplished with capsules and protocols.

- At the lowest level, a *capsule* is a generalized replacement for a packet. It includes a reference to the routine used to forward it at network nodes.
- A *protocol* is a collection of related capsule types that are treated as a single unit of protection by network nodes. That is, capsules within a single protocol may share information that is not accessible by other protocols, and each protocol is presented with its own view of the network.

Thus protocols are the units by which the network as a whole is customized by applications. We reference forwarding routines from capsules by using a fingerprint (e.g., the MD5 message digest) of the routines. This scheme allows protocols to be allocated quickly and in a decentralized fashion, enabling co-operating parties to use their own protocol by mutual agreement. Further, it greatly reduces the danger of protocol spoofing because a fingerprint based on a secure hash is effectively a one-way function that can be verified without trusting external parties.

In developing protocols, two system requirements restrict the kinds of processing that may occur at programmable network elements. First, we require that network-based processing be transparent to endpoints. This is necessary so that its failure or absence (due to packet or connectivity loss) does not effect the correctness of the application. Second, we require that network-based processing be compatible with networks in which only some strategic nodes are programmable. As discussed later, many of the benefits of active protocols can be achieved by installing a few active nodes at strategic points in the network. We expect these strategic points to be located at boundaries between bandwidth-poor and bandwidth-rich parts of the network.

2.2 Active Nodes

A key difficulty in designing an active network is to allow nodes to execute user-defined programs while preventing unwanted interactions. Not only must the network protect itself from runaway protocols, but it must offer co-existing protocols a consistent view of the network and allocate resources between them.

Our approach is to execute protocols within a restricted environment that limits their access to shared resources. *Active nodes* are the programmable network elements that play this role in our architecture. They export a set of primitives for use by application-defined processing routines, which combine these primitives using the control constructs of a programming language. They also supply the resources shared between protocols and enforce constraints on how these resources may be used as protocols are executed.

When a capsule arrives at a node, its associated processing routine is run to completion. The routine processes the payload of the capsule and initiates any further actions, e.g., forwarding, that are necessary. During capsule processing, active nodes are responsible for the integrity of the network and handle any errors that arise. We associate with each capsule a resource limit that functions as a generalized TTL (Time-To-Live) field. This limit is carried with the capsule and decremented by nodes as resources (bandwidth, processing time, and memory) are consumed. When it reaches zero, the capsule is discarded.

Active nodes also implement a mechanism to propagate code to where it is needed. Our approach has been to couple the transfer of code with the transfer of data as an in-band function. We have designed a scheme that is suited to flows, i.e., sequences of capsules that follow the same path and require the same processing. At end-systems, applications may begin to use a new protocol at any time by registering the code definition at their local node. As capsules travel through nodes of the network, a lightweight protocol is used to transfer the capsule programs incrementally from one node to the next, caching them for future use along the way.

3 Sample Applications

There are many ways to take advantage of computation within a network. In this paper, we concentrate on four general mechanisms: fusion, fission, caching, and delegation. A fusing node (e.g., a filter) has the potential of forwarding fewer packets than it receives. A fissioning node has the potential of forwarding more packets than it receives. A proxy node performs a task delegated to it by another node.

In this section we describe how these techniques can be used by different applications. We first sketch three examples of customized active protocols, and then discuss their performance benefits.

3.1 Active Reliable Multicast

Multicast protocols provide a point-to-group communication facility; a multicast protocol is reliable if it continues to try to deliver information until it is received by all members of the group.

Providing an efficient and scalable reliable multicast service over a wide-area network is a difficult problem; it has been a topic of considerable recent interest in the networking community [13, 15]. The key challenges include: managing bandwidth utilization of bottleneck links, not overloading the sender with retransmission requests, and keeping latency of retransmissions low. At the level of the implementation, these challenges translate into finding mechanisms for preventing NACK (negative acknowledgment) implosion, distributing responsibility for sending retransmissions, and limiting the delivery scope of retransmitted packets.

In existing end-to-end approaches [13, 15], considerable effort is made to control NACK implosion and distribute responsibility for repair, but at the cost of increased retransmission latency [13] and/or aggregate bandwidth utilization [15]. None of these approaches offer a particularly attractive scoping technique. Most of the difficulty stems from not having a good way to implement a hierarchy using only endpoint nodes. As a result, new research in "end-to-end" approaches is exploring the potential improvements to be gained by expanding the services provided by network routers [21].

In another paper in these proceedings [17], Lehman presents an active reliable multicast algorithm, ARM, that takes fuller advantage of the opportunities afforded by processing within the network. Here, we briefly outline some of the techniques used, which illustrate several ways of exploiting active nodes.

The protocol takes advantage of the capabilities of active nodes to combine reliable multicast processing with the multicast data distribution tree itself. Active nodes perform the following processing on multicast, NACK and retransmission packets passing through them:

- Duplicate NACKs are suppressed by checking to see if another NACK with the same sequence number has been forwarded recently. This prevents implosion at the source.
- A limited amount of multicast data is cached and retransmitted downstream when a NACK is intercepted. This reduces the bandwidth usage and latency of retransmissions.
- Missing data packets are detected by checking for gaps in the multicast sequence numbers; when a gap is found a NACK packet is generated and sent towards the source. This reduces the latency of retransmissions since intermediate network nodes both detect loss earlier than the receivers that experience it and are closer to the needed data.

3.2 Online Auctions

A server running a live online auction collects and processes client bids for each available item. This server also responds to requests for the current price of an item. Because of the network delay experienced by a packet responding to such a query, its information may be out of date by the time it reaches a client, possibly causing the client to submit a bid that is too low to beat the current going price. Thus, unlike auctioneers in traditional auctions, the auction server is likely to receive bids that are too low and must be rejected.

Current implementations of such servers [10, 20] perform all bid processing at the server. Our active protocol filters out low bids in the network, before they reach the server. When the server senses that it is heavily loaded, it activates the filters and periodically updates them with the current price of the popular item. The filtering active nodes drop bids not greater than this price and send bid rejection notices to the appropriate clients. This frees up server resources for processing competitive bids.

3.3 Mixing Sensor Data

Imagine a situation in which geographically dispersed sensors and receivers are connected over a data network. The sensors are continuously collecting large amounts of information that must be combined for one or more receivers. These sensors could be microphones collecting audio signals, antennas collecting radio signals, devices measuring emissions of pollutants, etc.

A straightforward design is for the network to passively forward each packet of the input streams to each receiver. Each receiver would then do its own mixing.

An alternative is to use fusion to do some of the mixing within the network, as suggested in [28]. If multiple input signals pass through the same internal node at approximately the same time, that node can mix the signals. If the mixed signal is smaller than the sum of its constituents, this will reduce the total network traffic. It also reduces the bandwidth and processing needed at the end nodes.

Now, suppose that each sensor is sending its signal to multiple receivers. If each signal is being multicast to the same set of receivers, this combines with mixing in a straightforward way.

If the sets of destination nodes are not identical, a more complex protocol is needed. Consider a situation in which an interior node received signal *S1* destined for nodes *R1, R2, R3, R4*; signal *S2* destined for nodes *R1, R2*; and signal *S3* destined for nodes *R3, R4*. It might be productive to first use fission to split *S1* into two multicast messages and then fusion to mix the resulting signals with *S2* and *S3*.

3.4 Performance

The remainder of this section (and most of the next section) is devoted to evaluating the overall impact of network-based processing on the performance of applications. Traditional network performance measures, such as throughput (bits or packets per second) and packet latency, are aimed at evaluating the performance of the network rather than of the applications. However, there are cases in which network performance is not positively correlated with application performance.

An active network can perform operations that can cause fewer packets to be sent or delivered and packets to experience longer latency. While these effects would appear to degrade performance measured by throughput and individual packet latencies, they may actually result in improved application performance because of reduced demand for bandwidth at endpoints, reduced network congestion, etc. Therefore, performance must be evaluated in terms of application-specific metrics.

We first consider application-specific notions of throughput. In the multicast example, cached retransmission reduces the server's throughput in terms of packets sent per second, but increases the number of packets per second successfully received at clients. In the active online auction, the relevant measure could be either the number of bids processed per second or, perhaps, the total number of *winning* bids processed per second. Both analysis and preliminary experiments with this application indicate that the active implementation will increase both these measures. For the sensor application, mixing in the network allows an increase in each sensor's sample rate or in the number of sources from which the server can receive and process signals. Preliminary simulation results support this intuition. All of these improvements in throughput are brought about by the parallelism resulting from the delegation of application functionality (mixing, bid rejection, retransmission, etc.) to internal network nodes.

All active processing slows down packets at least somewhat, but can more than make up for it by improving the latency of application-level operations. Caching in the network, as in the auction example, can reduce the latency of data accesses when the server is busy. When network nodes in the auction application reject low bids, they inform the "losing" end nodes more quickly than could the overloaded (and farther away) server. The fission performed in the multicast and signal mixing example will clearly deliver data to most destinations more quickly than multiple unicasts.

The cost of these performance improvements is the increased consumption of computational and storage resources in the network, which may slow down other network traffic traveling through the busy active nodes. However, this competing traffic could also benefit from active processing. Because the active processing in all of the example applications reduces the application's bandwidth utilization in some regions of the network, other traffic will benefit from the resulting reduction in congestion-related loss and delays.

Sometimes, doing work within the network also reduces the total amount of work that needs to be done by an application. Consider the mixing example. The active network implementation offers the opportunity to reduce the work done at end nodes by more than it increases the work within the network, i.e., there is a reduction in total work as well as in endpoint work. Consider a situation in which *N* sources send signals to *M* destinations. If each end node does all of its own mixing, the work, summed over all end nodes, is proportional to *N*M*. In the best case, by mixing pairs of signals within the network the end nodes can be completely freed of the need to mix signals. Furthermore, the total amount of mixing done can be reduced – in the best case to *N*.

The degree to which intra-network processing improves performance depends on where in the network it is deployed. In all our examples, placing processing near a bottleneck link is likely to decrease delay and loss due to congestion. In the signal mixing and distribution example, bandwidth utilization is decreased the most when fission is performed as late as possible and fusion as early as possible. In the auction application, filters should be far enough away from the server to turn back low bids as early as possible, but close enough to the server to get reasonably up to date price information.

4 Caching Within the Network

In the previous section, we briefly looked at three active protocols and discussed the kinds of performance improvements that might be attained using the techniques they incorporate. In this section, we take a much closer look at the performance of a single active protocol that performs intra-network caching of rapidly changing data.

We first outline an application for which such caching offers the potential of significant performance improvements, then describe an active protocol that supports that application, and finally evaluate that protocol using simulation results that compare it to the conventional approach to Web caching.

4.1 The Application

Consider a server that supplies rapidly changing information to a variety of clients with different needs. Military planning systems, airline flight status systems, and stock quotation systems are examples of this kind of service. Obtaining fast up-to-date information from such such systems is most important during crisis periods (e.g., wars, snowstorms, market crashes) which are likely to coincide with peak load on servers and the network.

Superficially, this looks like the problem that Web caching already addresses. However, conventional approaches to caching don't suffice in this context. First of all, to avoid the problem of shipping stale data, today's caches do not store rapidly changing information such as stock quotes. This means that long delays and server overload will persist during periods of high demand.

Second, the granularity of objects stored in Web caches (i.e., entire Web pages) is inappropriate for this application. Different clients will want different kinds and combinations of data; although there may be a lot of cross-client overlap in the data requested, there may be little cross-client overlap in the pages actually delivered. The likelihood of a cache hit will be small.

In an active network, the caching strategy can be customized to deal with these needs. In the next subsection we describe one way of doing this. For concreteness, we describe an active protocol designed to support a stock quote server.

4.2 The Protocol

Quotes are cached at active nodes as they travel from the server (or a cache) back to a client. Subsequent client requests are intercepted at the nodes where the local cache is checked to determine whether the desired quotes are available. If so, the quotes are sent to the client, where they are assembled into a viewable format. If not, the request is forwarded on towards the server. The caches are easily found because they lie directly in the paths followed by requests traveling towards the server. We believe this approach of intra-network caching will prove to be more efficient (in terms of latency and bandwidth utilization) than one which requires requests to be repeatedly redirected to caches located at network edges/endpoints, as well as less complex since no additional routes need be calculated.

In order to avoid obtaining stale data, requests also specify a client-controlled degree of currency of the desired information. This allows each client to trade off response-time and currency as appropriate. For example, a financial planner may prefer to have instantaneous access to large amount of slightly out-of-date information, while a trader may prefer to wait for more up-to-date information. To implement this, each cached quote has a timestamp associated with it that indicates the time the quote was issued by the server. When a request capsule arrives at an active node, the quote cache is actually checked for a suitably *current* version of the desired quote.

Our protocol caches information at network nodes using a small grain size, i.e., on a per quote basis. For simplicity, we assume that requests for multiple items are partitioned into separate requests, each of which fits within a single packet. The results are then assembled at the client. This enables more requests to hit in the cache, no matter what combination of stocks is requested.

In our current implementation, each active node stores in its cache each quote that it forwards. A node never forwards a quote that is less current than one for that stock stored in its cache and the quotes stay in the cache until they are replaced by a more current quote. Because this protocol consumes storage resources rather greedily, in future implementations we may incorporate a more efficient space management scheme such as that described in [4].

The request and quote response capsules of our protocol were implemented using the primitives supplied by ANTS. The capsule processing routines capture the algorithms in a straightforward way.

This caching scheme has several potential benefits:

- It can decrease the latency observed by clients,
- It can decrease the traffic at routers, and
- It can decrease the load on the server.

The extent to which these potential benefits are realized depend upon the degree to which the network is successful in directing requests for data to active nodes that happen to have a current copy of the data in their cache. Typically, requests for data are routed along a greedy (or shortest path) to the source for the data. This is not necessarily an optimal strategy. It is simple to construct examples where routing data along a shortest path will cause the request to avoid the active nodes which are most likely to have a copy of the data in their caches.

We are currently investigating algorithms and protocols for path selection and caching in active networks that (to the extent possible) will maximize the chances that a request for data will be routed along a path that is likely to quickly intersect a cache with a copy of the data. However, the results in this paper are all based on a version of the protocol that uses shortest path routing.

4.3 Simulation Results

Caching within local client sites (client site caching) is already a well-accepted and widely used mechanism with clear benefits. Given this, we assume client site caches large enough to hold all stocks and study the additional value of caching within the network. We do this with a series of simulations.

There are several key parameters in our simulations:

- *T*, the network topology. We used GT-ITM [5, 6] to generate a variety of random internetwork topologies. Each topology is a transit-stub network [30] supporting 1000 end nodes. The end nodes are organized into sites (of average size six) that are connected to the backbone. With an average of four sites per backbone node, the backbone consists of 40 nodes that are randomly connected with average degree of 3.5. In all of our simulations each node that connects a site to the backbone has caching enabled, and all caches are large enough to hold all stocks. When simulating our active protocol, all nodes in the backbone act as caches as well.
- *N*, the number of clients sending requests to a single server. We vary the number of clients from 100 to 1000, by hundreds. To simulate fewer than 1000 clients, we deactivate a fraction of the end nodes in each topology.
- *U*, the number (universe) of stocks for which quotes are available. Increasing this relative to the request rate of clients tends to decrease the number of cache hits because it decreases the likelihood that multiple clients will request the same stock within a currency period. Decreasing it tends to diminish the value of caching within the network relative to caching at client sites. In our simulations we look at a universe containing 100 stocks. Even this small universe gives us a relatively small maximum ratio of clients to stocks ($N/U = 10$), which would tend to underestimate the utility of our caching scheme. For reasons of practicality (i.e., simulation time), our values for U and N are lower than what one might expect to be the case in a real system. However, we believe that for the purpose of our experiments, their individual values do not matter so much as the value of their ratio N/U.
- *C*, the period of time for which clients consider a quote to be current. For simplicity, in our simulations we assume this is the same for all clients. As *C* increases, the number of cache hits in both network and client caches tends to increase. In all our simulations we use the same currency (1 second), since varying currency has similar effects to varying the rate of requests.
- *R*, the rate of requests sent by each client. We assume that each client makes *R* independent requests a second. These requests are chosen at random from *U* stocks. For a fixed currency, the hit rate in both network and client caches tends to go up with the request rate. In our simulations, each client generates one request per second ($R = 1$).

We used a modified version of the *ns* [19] simulator to simulate our protocol. In order to keep this preliminary analysis simple, and to avoid having to make unsubstantiated assumptions about the relative throughputs of active and non-active routers, we simulate a high capacity network in which there are no congestion-related losses.

We performed two sets of experiments. In the first set, we varied the number of clients from 100 to 1000 and measured server load. This was done for both active router caching and client site only caching schemes. Results were averaged over 10 random topologies.

In the second set, we used 1000 clients (the maximum load seen in the first set of experiments), and measured router load and round-trip request hop counts. Again, this was done for both active router and client cite only caching schemes. Here, results were averaged over 100 topologies.

4.3.1 Load on Server

In times of great interest, the server can become a bottleneck. Not only are there more client requests, but if the network is congested lost packets introduce retransmission overhead at the server as well.

Figure 1 plots the arrival rate of requests at the server against the number of clients. The graph compares the case where all backbone nodes are active with the case in which no backbone nodes are active. In both cases, caching occurs at client sites.

As seen in the graph, our experiments highlight a range of cases in which client site caches are not effective. The server load is decreased by very little because there are few duplicate requests generated by each site's clients per currency interval. When the sites are larger, the currency interval longer, or the request rate higher, then client site caches may be effective. When this is not so (as is considered here) then our active protocol may be a better choice.

The graph shows that the impact of network caching on server load grows with the number of clients. As the number of clients grow, the total number of misses seen across all client site caches grows. In the non-active scenario, all of

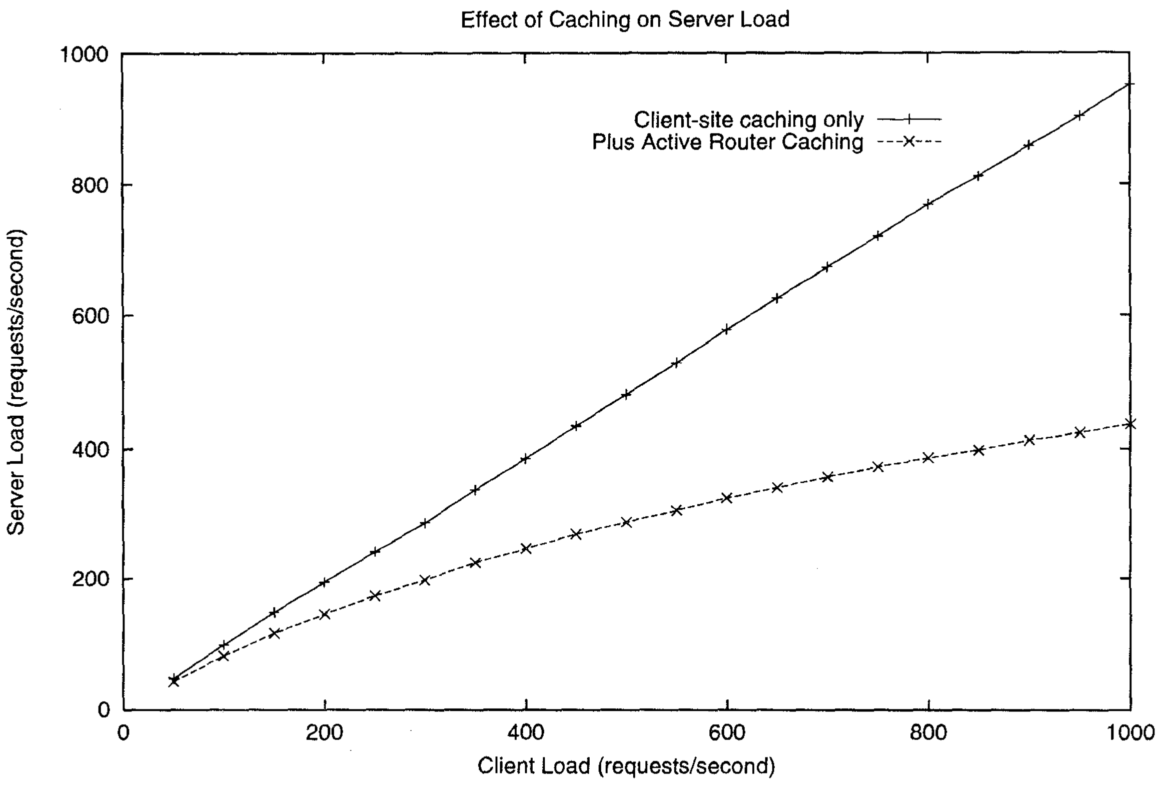

Figure 1: Server Load

these misses result in server accesses. With active router caching, many of these misses are intercepted and serviced by the network caches, thus reducing server load. In the case where N=1000, we see a decrease in server load of 54%. Caches shared between sites are clearly beneficial.

Ultimately, we would like to compare intra-network caching to other systems of shared caches (such as [7, 26, 31]) that locate their caches at network endpoints. However, it is not yet clear which endpoint caching scheme is the most effective. For the moment, we only observe that if our simple scheme were extended so as to relocate each network cache to its nearest network endpoint, both latency and bandwidth utilization would increase as a result of the longer paths requests would have to travel.

4.3.2 Load on Routers

Figure 2 plots the cumulative distribution of router load on the backbone. Only the top portion of the plot is shown, indicating the load on the busiest routers, which are near the server. (Because we simulate only a single server, the bulk of the routers are uninteresting as they handle a low volume of traffic near clients). The graph shows that the impact of network caching is substantial for these busy routers. The peak demand is reduced by 50%. In addition, the total network load summed across all routers is lowered 35% by caching within the network from 7419.5 to 4845.33 requests/second.

These results have some bearing on how fast an active router needs to relative to a non-active router. Suppose, for example, at peak load, an active router must handle 1/x the amount of traffic that a conventional router must handle. If a conventional router is x times faster than an active router, then each can handle the same peak load. Our graph shows that x is approximately 2 for the top 5% of the routers and 1.5 over the aggregate load.

4.3.3 Latency at Clients

At this time, we do not have a credible model of the difference in the time required to pass through active and inactive routers. Therefore, we are unable to make any quantitative statements about latency. However, we can show that in-network caching does reduce the average number of hops taken before clients receive a response to a request. Figure 3 shows the distribution of round-trip hop counts for both scenarios (in-network caching and client site only). The graph shows a clear shift towards shorter round trips in the active caching experiments. The average round trip is reduced by 18% from 9.9 hops to 8.1 hops.

These results follow directly from the fact that, as discussed

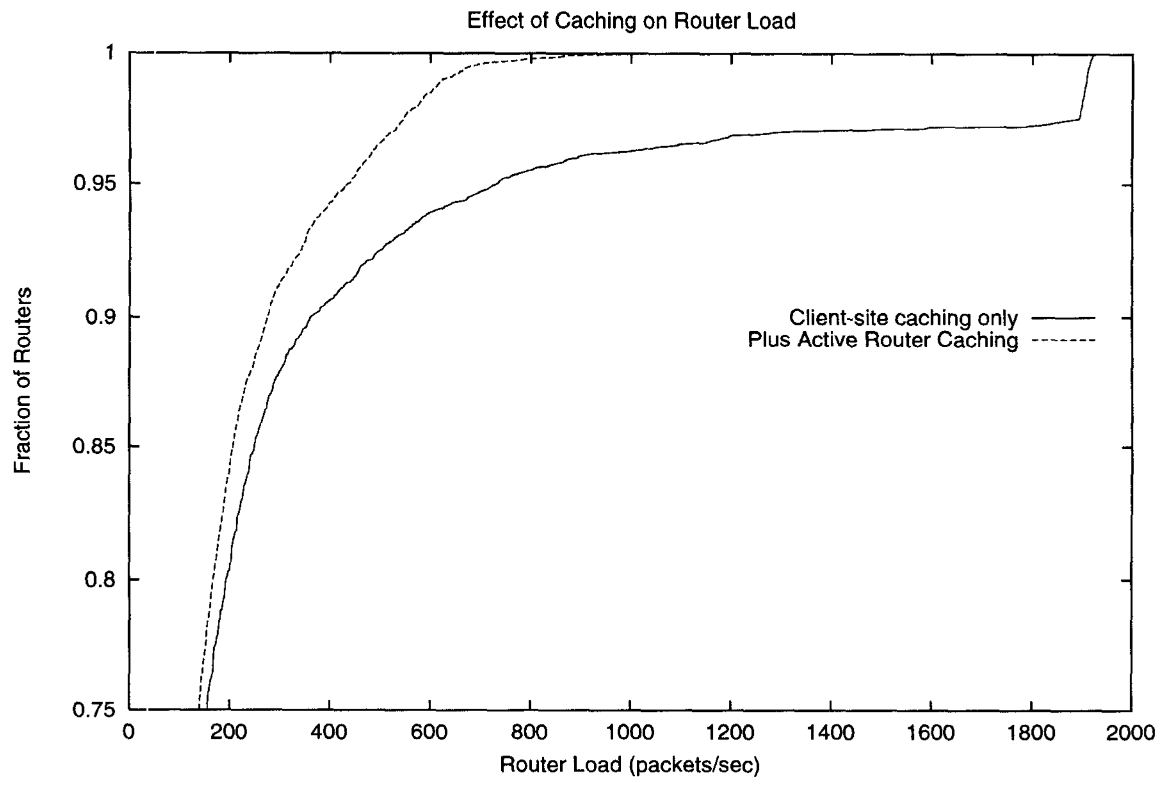

Figure 2: Router Load

above, fewer requests need to make the roundtrip from the client to the server and back. However, the net effect on latency will depend on link delays, server processing times, router processing times and router load.

With active nodes, the reduced load on the server has the potential to reduce latency when the server itself is a source of delay. Similarly, the reduced load on routers can reduce congestion-related latency. Since fewer packets would be dropped, long retransmission delays would be avoided. In future work, we hope to produce quantitative data that demonstrates this effect.

5 Related Work

In this section, we describe how our work relates to previous research on programmable networks and customization of network services.

Work at Georgia Tech investigates the benefits of supporting application-specific congestion control mechanisms (e.g., selective discard, lossless compression, transcoding) within the network [3]. They find that when transferring MPEG frames, selective discard improves useful throughput during periods of congestion. Though their experiments did not address the impact of this processing on other network traffic, their results are consistent with ours, as are their conclusions about the utility of application-specific processing within the network.

Other current projects building programmable networks are Switchware [22], whose Active Bridge demonstrates the benefits of active networking in terms of enhanced functionality rather than improved performance, and Netscript [29], which focuses on management tasks.

Our work has been influenced by the philosophy of Application Level Framing (ALF) [9], a design guideline that includes the semantics of the application in the design of its transport protocol. It argues that the roles of application and network must be matched for efficient processing. Other instantiations of this philosophy include configurable protocol systems, notably the x-kernel [16] and protocol boosters [18].

Programmable network services are also supported by extensible operating systems technology [11, 8, 2]. For example, ASHs [25] allow user-defined handlers to be run by the kernel in response to packet arrivals, while Plexus [12] allows application-specific communication protocols to be incorporated into the kernel. These systems offer, in the context of a single node, some of the same kinds of opportunities offered by active networks.

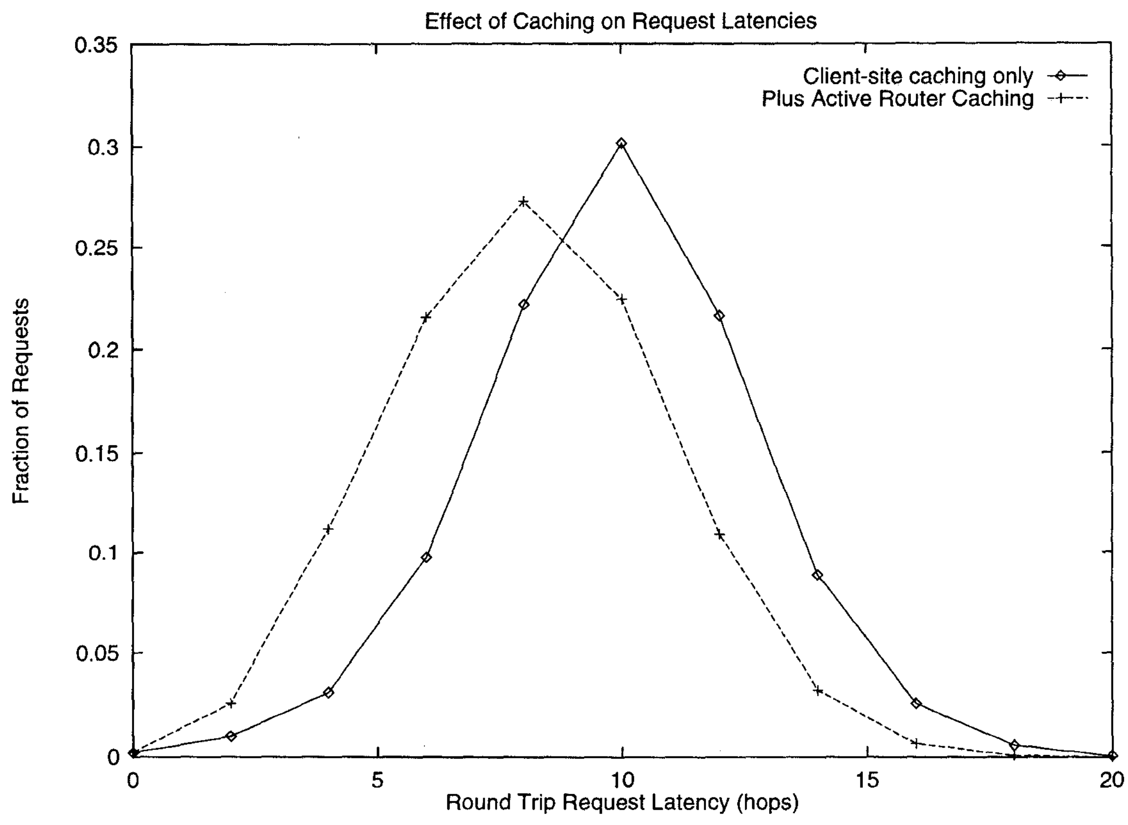

Figure 3: Round-trip hop counts

6 Conclusion

As we said in Section 1 the goal of the work reported here was to demonstrate that:

1. There are a variety of useful network services that involve processing at intermediate nodes, and
2. The use of such services can lead to better end-to-end performance for applications.

In support of these hypotheses, we presented three protocols supporting novel network services: reliable multicast, online auctions, and mixing sensor data. For each, we briefly discussed how it could be used to improve the performance of applications. We then presented a detailed simulation analysis of the performance of an active caching protocol.

The results were encouraging. Our qualitative analysis of the novel network services presented in Section 3 strongly suggests that these protocols have the potential of improving the performance of the applications that use them while simultaneously reducing total network traffic. The simulations reported on in Section 4 were equally encouraging. First, through the use of a simple active protocol, we were able to support the caching of (heretofore considered uncacheable) rapidly changing data. Second, the protocol effected performance improvements in the form of reduced load on the server and routers, and shorter round-trip hop counts.

The experience and results described in this paper lead us to conclude that active network protocols can indeed improve application performance.

Acknowledgments

This work has benefited greatly from the help of our colleagues. In particular, we wish to acknowledge: Li Lehman, for her work on reliable multicast; Mike Ismert, for performance modeling; David Murphy, for his work on a node implementation; and Steve Garland and David Tennenhouse for their overall guidance. We would also like to thank the INFOCOM referees, whose useful comments led to several major changes in this paper.

References

[1] H. Balakrishnan et al. A comparison of mechanisms for improving TCP performance over wireless links. In *SIGCOMM 1996*, Stanford, CA, August 1996.

[2] B. Bershad et al. Extensibility, Safety and Performance in the SPIN Operating System. In *15th Symp. on Operating Systems Principles*, Dec. 1995.

[3] S. Bhattacharjee et al. On Active Networking and Congestion. Technical Report GIT-CC-96/02, College of Computing, Georgia Institute of Technology, 1996.

[4] S. Bhattacharjee et al. Self-organizing wide-area network caches. In *INFOCOM'98*, 1998.

[5] K. Calvert and E. Zegura. Georgia Tech Internetwork Topology Models (GT-ITM). Georgia Tech College of Computing. Software on-line: http://www.cc.gatech.edu/fac/Ellen.Zegura/graphs.html.

[6] K. L. Calvert et al. Modeling Internet Topology. In *IEEE Communications*, June 1997.

[7] A. Chankuntod et al. A hierarchical internet object cache. In *Proceedings of 1996 USENIX*, 1996.

[8] D. R. Cheriton and K. J. Duda. A caching model of operating system functionality. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, 1994.

[9] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *SIGCOMM '90*, 1990.

[10] eBay Inc. AuctionWeb server. http://www.ebay.com/.

[11] D. R. Engler et al. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *15th Symp. on Operating Systems Principles*, 1995.

[12] M. E. Fiuczynski and B. N. Bershad. An extensible protocol architecture for application-specific networking. In *Proceedings of the 1996 Winter USENIX Conference*, 1996.

[13] S. Floyd et al. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *ACM SIGCOMM'95*, 1995.

[14] M. Greenwald et al. Designing an academic firewall: Policy, practice and experience with SURF. In *Proceedings of the 1996 Symposium on Network and Distributed Systems Security*, San Diego, CA, 1996.

[15] H. W. Holbrock et al. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *SIGCOMM'95*. ACM, 1995.

[16] N. C. Hutchinson and L. L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, Jan 1991.

[17] L.-W. Lehman et al. Active Reliable Multicast. In *INFOCOM'98*, 1998.

[18] A. Mallet et al. Operating System Support for Protocol Boosters. Technical Report MS-CIS-96-13, CIS Dept., Univ. of Pennsylvania, 1996.

[19] S. McCanne and S. Floyd. The LBNL network simulator. Lawrence Berkeley Laboratory. Software on-line: http://www-nrg.ee.lbl.gov/ns/.

[20] ONSALE Inc. ONSALE web server. http://www.onsale.com/.

[21] C. Papadopoulos et al. An error control scheme for large-scale multicast applications. Unpublished manuscript from Washington Univ., St. Louis (http://www.ccrc.wustl.edu/ christos), 1997.

[22] J. Smith et al. SwitchWare Accelerating Network Evolution. Technical Report MS-CIS-96-38, CIS Dept., Univ. of Pennsylvannia, May 1996.

[23] D. Tennenhouse et al. A Survey of Active Network Research. *IEEE Communications Magazine*, 1997.

[24] D. L. Tennenhouse and D. Wetherall. Towards an active network architecture. In *Multimedia Computing and Networking 96*, San Jose, CA, Jan 1996.

[25] D. A. Wallach et al. ASHs: Application-specific handlers for high-performance messaging. In *SIGCOMM '96*. ACM, 1996.

[26] Z. Wang and J. Crowcroft. Cachemesh: A distributed cache system for world wide web. NLANR Web Caching Workshop, June 1997.

[27] D. Wetherall et al. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *OPENARCH'98*, 1998.

[28] N. Yeadon. *Quality of service for multimedia communications.* PhD thesis, Lancaster University, May 1996.

[29] Y. Yemini and S. da Silva. Towards Programmable Networks. In *IFIP/IEEE Intl. Workshop on Distributed Systems: Operations and Management*, 1996.

[30] E. W. Zegura et al. How to Model an Internetwork. In *INFOCOM'96*. IEEE, 1996.

[31] L. Zhang et al. Adaptive web caching. NLANR Web Caching Workshop, June 1997.