Active Networking and the End-to-End Argument

Samrat Bhattacharjee, Kenneth L. Calvert, Ellen W. Zegura*

Networking and Telecommunications Group
College of Computing
Georgia Institute of Technology, Atlanta, GA 30332-0280
{bobby,calvert,ewz}@cc.gatech.edu

Abstract

Active networking is the placement of user-controllable computing functionality in the switching nodes of a network. The end-to-end argument states that functions should be placed "in" the network only if they can be cost-effectively implemented there. We argue that active networking is a natural consequence of the end-to-end argument, because certain functions can be most effectively implemented with information that is only available inside the network. We propose a performance model for quantifying the benefit of implementing a particular functionality solely in the end system versus implementing it through a combination of end system and network support. We show how the model applies to specific services, including congestion control and reliable multicast.

1 Introduction

Discussions of the implementation of various functions in a communication network often invoke the *end-to-end argument*, an architectural principle that guides the placement of functions within a distributed system [1]. The end-to-end argument is often construed to preclude the implementation of any kind of higher-level function within a network. As such, it might seem that active networks are the antithesis of the end-to-end argument. We claim, however, that the "activation" of networks is a natural extension of this well-accepted design principle. In this paper, we argue the case for active networks in view of the end-to-end argument. We present a model that quantifies the performance benefits of placing functionality in the network, and give examples where those benefits are significant compared to solutions based on a traditional passive network service.

For the purposes of this discussion, "active networking" refers to the placement of user-controllable computing and other resources in the communication network, where they can be utilized by applications that need those capabilities [2]. An active network supports a user-network interface allowing the nodes of the network to be *programmed* by the application (user) to provide a desired functionality, such as routing. This programming might be done on a per-packet basis (as in the *capsule* approach of Tennenhouse and Wetherall [3]) or through an out-of-band signaling mechanism (e.g. injection of user-specific programming into the switch, as in Switchware [4]). The level of programmability might range from a Turing-complete programming language to a set of predefined, user-selectable functions whose behavior can be controlled through parameters. The important point is that with active networks, *the network service can be tailored to the user's requirements*. It is this aspect of active networks that relates to the end-to-end argument.

The rest of this paper is organized as follows. In the next section we review the end-to-end argument and characterize situations where applications can benefit from additional functionality in the nodes of the network. In Section 3 we present a model that quantifies the benefit to an application of network-based functionality. Section 4 considers the benefits of network support for reliable multicast transport. Section 5 shows how the model can be applied to various forms of network-based support for congestion control. Finally, Section 6 offers some concluding remarks.

2 The End-to-End Argument Revisited

The end-to-end argument "provides a rationale for moving a function upward in a layered system closer to the application that uses the function" [1]. According to the argument, a computer network, as part of the "lower layers" of a distributed system, should avoid attempting to provide functions that can be better implemented in the end systems, especially if some applications might not benefit from such functions at all. The

*This work was supported in part by NSF Careers Award MIP-9502669.

canonical example of such a function is reliable transfer. The network can go to great lengths to protect against and recover from losses in the network, but an application that requires reliability will often have to protect against other sources of error, so those efforts may be redundant. Moreover, some applications will not need the network-provided reliability, but would have to pay for it anyway.

We identify the following principles as the key ideas of the end-to-end argument as it applies to the placement of functionality in networks:

- Some services require the knowledge and help of the end-system-resident application or user to implement, and so *cannot* be implemented entirely within the network.
- If not all applications will make use of a service, it should be implemented in such a way that only those applications using it have to pay the price of supporting it in the network.
- The amount of support for any given end-to-end service in the network is an engineering tradeoff between the performance seen by the application and the cost of implementing the support.

We claim that these principles do *not* rule out support for higher-level functionality within the network. Rather, they require that the interface to such functionality be carefully designed; that costs and benefits of such support be calculated; and that the "engineering tradeoff" be carefully evaluated. The goal of this paper is to explore some of these tradeoffs in the specific context of active networking as described above.

A fundamental premise of this paper is the following:

> *Some services can best be supported or enhanced using information that is only available inside the network.*

In other words, the network may have information that is not available to the application, and the timely use of that information can significantly enhance the service seen by the application. Examples of information that is first (or only) available to the nodes of the network include:

- The time and place where congestion occurs.
- Global patterns of access to objects retrieved over the network (e.g. Web pages). In particular, the location of "hot spots", or points in the network where requests for objects are highly correlated in time and space.
- The location of packet losses within multicast distribution trees.

On the other hand, applications may have information that is needed by the network in order to fully optimize performance. Examples of this type of information include:

- The existence of dependencies among application data units, e.g. some are useless if others are not received.
- Variations in importance of data units, including whether to retransmit if lost.
- Whether or not it is acceptable to service a request using cached data.

Thus, to optimize performance, it is desirable to combine application and network information.

Classically, the end-to-end argument views the network as a monolithic entity that provides a single type or quality of service to all users. For example, the debate about reliable service assumed the network would support a single paradigm for all users: either reliable or best-effort transport. Active networks allow users to increase the likelihood that the service offered by the network will be useful to them, by providing an interface that supports multiple (or programmable) services.

There are costs associated with such a flexible interface, and they affect all of the network's users whether they take advantage of active network support or not. The (monetary) cost of *providing* the interface, though likely to be significant, is paid once and can be amortized over all users for a period of time. The performance cost of *using* the interface should vary with application; this is the end-to-end argument. For example, applications that need only a best-effort datagram delivery service should not suffer reduced performance because of the increased flexibility of the interface. On the other hand, any performance penalty for customizing network behavior (e.g. signalling overhead, or taking packets off the "fast path") must be more than offset by improved end-to-end performance delivered to the ultimate users. These performance costs will be determined largely by the design of the user-network interface. While we believe it is possible to design an interface consistent with these implications of the end-to-end argument, presentation of such a design is beyond the scope of this paper. We therefore do not further consider the interface-related performance costs of an active network.

3 A Model for Service Location

In this section, we quantify the engineering tradeoff that must be made when deciding where to locate service implementation. Specifically, we develop a generic model that gives the expected performance under two design options. In the first option, the service is achieved by implementation exclusively in the end-systems. In the second option, the service is achieved through a combination of implementation at the end-systems and in the network. The equations that we develop are implied, to an extent, by the text in the Saltzer paper [1]. We demonstrate the model using the reliable file transfer example from Saltzer.

3.1 Model Development

To be clear, we first state the assumptions that are made by our model. Some of these assumptions are relaxed in the refinements to the model described in Section 3.3.

- There exist applications that have a need for the service in question. The purpose of the model is to evaluate options for the *location* of the implementation, not to comment on whether the service should be offered.

- The implementation of the service in question can be distributed in two ways: exclusively in the end-systems, or in a combination of the end-systems and the network. Two particular designs for the implementation of the service are being compared: Design X is exclusively an end-system design; Design C is a combined end-system and network design.

- The designs will be compared by evaluating the *expected performance* for each design. The complex part of the analysis is encapsulated in the expected performance variables (the T's) of the model. This will be evident when the model is applied to particular examples.

- The "network" is treated monolithically. If the service is implemented in the network, then all network nodes (switches and routers) are assumed to cooperate.

- In the combined Design C, the network support for the service is *boolean*, in the sense that it either accomplishes the service or it does not. The parameter p_n expresses the probability that the network support accomplishes the service. In essence, p_n is a measure of the effectiveness of the network support.

The parameters of the model are listed in Table 1. The expected performance under Design C, T_C, is a function of the other model parameters. We have two possibilities: (1) the service is achieved by the network support or (2) the service is not achieved by the network, and the end-system must provide the service. Thus the expected performance is given by:

Exclusively End-system (Design X)	
T_X	Expected performance
Combined End-system and Network (Design C)	
T_C	Expected performance
p_n	Pr{network support accomplishes service}
T_E	Expected performance, end-system version
T_N	Expected performance, network version

Table 1: Parameters of performance model

$$T_C = (1 - p_n)T_E + p_n T_N$$

3.2 Example: Reliable Data Transfer

Saltzer et al. motivate the end-to-end argument with the example of point-to-point reliable data transfer. We demonstrate our model on the reliable data transfer example, though we simplify the details somewhat for clarity of exposition.

The premise is that data may be corrupted in various ways, with sources of corruption both in the endsystems and within the network. Two designs are proposed. In Design X, the endsystems implement a mechanism (e.g., checksum) to check whether the data received is corrupted. If corruption is detected, the receiver requests a retransmission from the sender. In Design C, the network implements some ability to detect and correct corruption errors that are caused by the network, while the data is en-route.

To use the model, we must supply the performance parameters based on the application and the methods for implementing functionality. For this example, we use the expected transfer time as the performance metric. For Design X, let t_X denote the time for the receiver to request the data, the data to be sent through the network, and the receiver to check the integrity of the data. If the check reveals an error, a request will be made to retransmit the data. This process will repeat until the data is received without error. We assume that the probability of an error on each transmission is independent and given by probability p. Thus,

$$\begin{aligned} T_X &= \sum_{i=1}^{\infty} Pr\{i \text{ transmissions}\} i t_X \\ &= t_X \sum_{i=1}^{\infty} i(1-p)p^{i-1} = t_X/(1-p) \end{aligned}$$

For Design C, let t_C denote the time for the receiver to request the data, the data to be sent through the network, and the receiver to check the integrity of the data. We expect $t_C > t_X$, since in Design C the network is doing some reliability checks while the data is sent, thus adding time to the transfer. We will say that the network "accomplishes reliable transfer" if (1) there is no error in transmission or (2) there is an error that can be corrected by the network version of reliability. Let p denote the probability of an error in transmission; let q denote the probability that the network can correct the error, given that there is an error. Then,

$$p_n T_N = (1 - p + pq)t_C$$

The expected performance with the end-system version covers the case when there is an error in the initial transmission that the network cannot correct. At least one retransmission is required; retransmissions will occur until there is either no error, or an error that the network can correct. For a given retransmission, the probability that it is successful is $1 - p + pq$; the probability that it is not successful is $p(1 - q)$. We assume that each retransmission is independent. Then,

$$\begin{aligned} T_E &= t_C + \sum_{i=1}^{\infty} Pr\{i \text{ retransmissions}\} i t_C \\ &= t_C + t_C \sum_{i=1}^{\infty} i(1 - p + pq)(p(1 - q))^{i-1} \\ &= t_C(1 + 1/(1 - p + pq)) \end{aligned}$$

Combining the performance values,

$$\begin{aligned} T_C &= (1 - p + pq)t_C \\ &\quad + p(1 - q)t_C(1 + 1/(1 - p + pq)) \end{aligned}$$

3.3 Refinements to the Model

The model makes several assumptions about the combined Design C that do not hold for every service implementation and networking environment. For example, the network support for a service may not be boolean; the network may be able to accomplish some useful portion of the service, with the end-system left the task of completing the service. In this case, the model must enumerate the possibilities for partial network functionality and the corresponding performance. The expected performance for Design C will have the form:

$$T_C = \sum_i p_i T_{C,i},$$

where i indexes over all possible partial network functionalities, p_i denotes the probability that the network accomplishes the particular partial functionality and $T_{C,i}$ denotes the expected performance for this combination of network and end-system collaboration. The basic model represents the special case of two "partial" network functionalities: either completely accomplishing the function or not accomplishing the function. By similar methods, we can also relax the assumption that the network is monolithic, to account for multi-node networks, with possibly heterogeneous behavior at the different nodes.

4 Example: Reliable Multicast

Reliable delivery of multicast data is an example of an application that can potentially benefit (via enhanced performance) from network-based functionality. The "traditional" IP multicast service hides from its users the details of the routing topology and the number and location of receivers [5]. For unreliable multicast, this approach makes sense and allows scaling to larger applications; however, there are inherent problems in using this model when it is desired to deliver data to all receivers *reliably*.

In general, only a subset of receivers will correctly receive any particular data unit, due to losses on some links of the multicast routing tree; lost data units must then be retransmitted or recovered in some other way. Traditional point-to-point reliable transfer protocols do not generalize well to point-to-multipoint, because they require the sender to maintain per-receiver state to keep track of who has received each data unit; this limits scalability for obvious reasons. This has led to various proposals for diluting the amount of state required [6, 7]. For example, NACK-based protocols reduce the amount of Sender state required, but introduce the NACK-implosion problem, and result in unnecessary retransmissions being sent to some or all receivers.

A common approach is to spread the responsibility for multicast retransmission among the *Receivers*, by constructing an explicit "tree" of receivers on top of the network-level multicast service (e.g., [8]) In effect, this subdivides the multicast group into smaller groups according to the parent-child relationships in this tree. For such grouping to be efficient, out-of-band information about the network topology is required so that nodes that are "near" each other in the underlying topology are grouped together. The point is that the information needed for efficient retransmission of lost data —namely the location of losses within the network— is hidden by the IP multicast service interface.

4.1 Applying the Model

It follows from the foregoing that a benefit (in the form of reduced latency) should accrue from the placement of buffering and retransmission functionality in the network to support reliable multicast. We will apply the model of Section 3 to the performance (i.e. latency) seen by a multicast Receiver in recovering from a lost packet. With network support, when a packet is lost, it is retransmitted from the first node upstream of where the loss occurred. The request and the retransmitted data thus travel the *minimum* possible distance. In topology-unaware schemes, where retransmission is always handled by end-system Receivers, the request and response have to travel an additional distance to reach an end-system supporting the recovery function.[1]

To quantify the performance benefit of network support for reliable multicast, we need to determine the latency experienced by a Receiver between the time of transmitting the request and the time of receiving the retransmitted packet, with and without network support. In quantifying this latency, we make the following simplifying assumptions:

- the latency is determined by the number of hops traveled by the request and retransmitted packets;
- neither the retransmission request nor the retransmitted packet are lost.

In Design X, only end-systems participate in the recovery; the "active" Design C involves the end-systems as well as the interior nodes of the network.

Note that in Design C, the network nodes have to do more work to forward *every* packet transmitted by the Sender, namely buffering a copy for some period of time. This may tend to increase the end-to-end delay seen by Receivers for packets that are *not* lost, compared to Design X. This performance differential must be taken into account (along with the probability of a lost packet) when comparing the overall performance of Design X and Design C; we do not model it here, however, as it depends strongly on the way each design is implemented. We do note that recent studies of MBONE traffic show that the likelihood of any particular multicast packet being delivered correctly to all receivers is surprisingly small, on the order of 20% [9].

4.2 Loss Recovery Approaches

When a multicast packet is lost in the network, we refer to that portion of the multicast routing tree that does not receive the packet as the *loss tree*. The router at the root of this tree, where the loss occurred, is referred to as the *loss node*. In both Design X and Design C, a Receiver noticing the loss sends a retransmission request toward the Sender.[2] In Design C, the "active" approach, this request stops when it reaches the first node upstream of the loss tree; that node retransmits the lost packet to the requesting Receiver, provided it still has a buffered copy. [3] If the upstream node does not have a buffered copy of the requested packet, it forwards the request upstream toward the Sender.

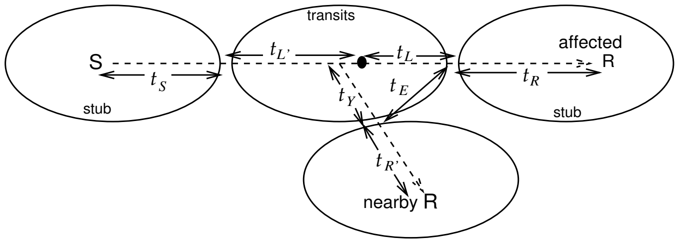

Figure 1: Stub and transit domains and distances

In Design X, the end-system-only approach, the request message is directed —by unspecified means— from the loss node to a "nearby" Receiver that is not in the loss tree; that Receiver then retransmits the lost packet to the affected Receiver.

4.3 Structure of Recovery Paths

To quantify the reduced recovery latency of Design C, we model the Internet as a collection of *routing domains*, each of which is a connected set of nodes such that the path between two nodes in the same domain does not include any node outside that domain. We define two classes of routing domains: *stub* domains, which carry only traffic originating from or destined for some node in the domain; and *transit* domains, which may carry traffic that does not originate or terminate within the domain. A network node in a stub domain is called a *stub node*, and a node in a transit domain is a *transit node*. We make the (reasonable) assumption that end-systems connect only to stub nodes, and also that there is a single transit domain connecting all stub domains. For the purposes of our model, the path followed by a packet from a multicast Sender to any Receiver consists of a sequence of one or more stub nodes, followed by zero or more transit nodes, followed by one or more stub nodes, as illustrated in Figure 1.

[1] Papadopoulos et al [7] have recently proposed an extension to router functionality to support reliable multicast. Their proposal stops short of placing the retransmission function in the router nodes themselves, but rather offers a service requiring less state, and allowing end nodes to handle the retransmission functions more efficiently. The service permits packets to be efficiently directed toward "nearby" receivers.

[2] We consider only a single Receiver, though in general more than one Receiver is affected. Either design could incorporate mechanisms to control the number of retransmission requests originated from the loss tree.

[3] In either design, the retransmitted message might be either unicast or multicast, depending on the protocol. The choice should have no effect on the latency perceived by the requesting Receiver, though it may affect other measures of performance of a reliable multicast protocol, such as the number of redundant messages.

We view each loss and recovery to be a random event affecting a single Receiver, and quantify the expected performance of each approach (T_C and T_X) in terms of the number of hops traversed by the request and retransmitted packet for a lost packet in each design. For brevity of exposition, we make the following additional assumptions:

- losses are uniformly distributed along the path connecting the exit node from the Sender's stub domain and the entry node of the affected Receiver's stub domain (a recent study found that losses in this part of the path account for a substantial majority of the losses in the MBONE [9]);
- all stub domains are topologically similar and have exactly one connection to the transit domain;
- the Sender's stub domain contains no other Receivers;
- in Design C, if the first node upstream of a loss does not have a copy of the requested packet, the request goes all the way to the Sender.

We define the performance in terms of the following quantities (refer to Figure 1): t_R is the expected number of hops between the affected Receiver and the exit from the Receiver's stub domain; t_L is the expected number of hops between the entrance to the affected Receiver's stub domain and the loss node. For Design X, we also have t_Y, the expected number of hops between the loss node and the entrance to the "nearby" Receiver's domain, $t_{R'}$, the expected number of hops between the nearby Receiver and its stub domain entrance, and t_E, the expected number of hops between the entrances to the affected and nearby Receivers' stub domains. (Note that reasonable routing topologies will satisfy a triangle inequality, so that $t_E + t_Y \geq t_L$.)

Also, in Design C: $t_{L'}$ is the number of hops between the Sender's stub domain entrance and the loss node, t_S is the number of hops between the Sender and the entrance to the Sender's stub domain, and p_n is the probability that the node upstream of the loss node has a buffered copy of the lost message when the request arrives.

Note that the t's are all primarily determined by the topology of the network and the location of the loss. The distance to a nearby Receiver's domain, t_E, is determined by topology *and* the average "density" of Receivers in the network, and thus should decrease as multicast group size increases (assuming Receivers are distributed throughout the network uniformly). The probability p_n can be made as large as desired, but only by increasing the cost of implementing the protocol.

4.4 Performance

For Design X, the request travels from the affected Receiver out of its stub domain, to the loss node, and then to the nearby Receiver. The retransmitted packet travels from the nearby Receiver out of its stub domain, to the entrance to the affected Receiver's stub, and then to the affected Receiver. Thus:

$$T_X = t_R + t_L + t_Y + 2t_{R'} + t_E + t_R$$

For Design C, the request similarly travels from the affected Receiver to the next node upstream from the loss node, at which point we consider two cases. If the upstream node has a copy of the packet (probability p_N), it retransmits the packet, which follows the same path back, and the total number of hops is given by:

$$T_N = 2(t_R + t_L + 1)$$

Otherwise, the request travels all the way to the Sender, and the retransmitted packet travels all the way back to the affected Receiver, giving a hop count of

$$T_E = 2(t_R + t_L + t_{L'} + t_S)$$

Thus we have:

$$T_C = 2p_n(t_R + t_L + 1) + 2(1 - p_n)(t_R + t_L + t_{L'} + t_S)$$

Assumptions made earlier about the topology and loss distributions imply that $t_R = t_{R'} = t_S$ and $t_L = t_{L'}$, and we can rewrite:

$$\begin{aligned} T_X &= 4t_R + (t_L + t_Y + t_E) \\ T_C &= 4t_R + 4t_L - 2p_n(t_R + t_L - 1) \end{aligned}$$

It follows that, to achieve $T_C < T_X$, we require

$$p_n > \frac{3t_L - t_Y - t_E}{2(t_R + t_L - 1)}$$

Note that t_Y in the numerator decreases with Receiver density, so as groups get larger, p_n must increase in order for Design C to remain superior.

4.5 Scaling

Design C above requires that network nodes buffer packets for each reliable multicast "flow". A factor of n scaleup in the number of flows supportable by the network can be achieved simply by storing the information required to recover at every mth node, instead of every node. This can be accounted for in the model through the parameter p_n. There are now $m + 1$ cases: m cases where some network node has a buffered copy of the desired packet, and one case where no node has a copy. Given that some node has a copy, with probability $1/m$ the first node upstream of the loss has it;

with probability $1/m$ the second node upstream has it, etc. The expected increase in T_N is approximately $(m-1)/2$, as long as m is small enough to guarantee that *some* node has a copy before the request reaches the Sender.

5 Example: Congestion Control

Another application that can potentially benefit from a combination of end-system and network processing is congestion control [10]. We claim that the best-effort service provided to adaptive applications can be enhanced by allowing applications some control over the way their packets are processed in network switches when they encounter congestion. Instead of applying "one size fits all" congestion reduction techniques, mechanisms can be placed in the network to allow packet processing —e.g. discarding or transforming— to proceed according to advice supplied by the application. The observation is that the application knows *how* to adapt to congestion, while the network knows *when and where* adaptation is needed.

In this section, we evaluate the performance of several congestion control mechanisms, one that involves no special network assistance to the application, and three that are combined application and network mechanisms.

Generically, application specific congestion control operates as follows:

> Based on triggers that indicate congestion control should take place, flow state — specifically installed by the application — is examined for advice about how to reduce quantity of data.

Advice about a particular flow is contained in the flow packets themselves and may be stored at the node. We assume the source attaches the application-specific advice to the packets in the flow.

5.1 Best-Effort MPEG Delivery

To illustrate application specific congestion control we focus on the use of congestion control advice to improve the best-effort service provided to MPEG video.While we use MPEG as an example, the techniques can easily be generalized to any application which has application-layer units of various types, and/or dependencies that span across units. The specific MPEG congestion control mechanisms that we examine are as follows:

The **Partial Packet Discard (PPD)** mechanism is the exclusively end-system design, in the sense that the network provides no special assistance to help improve the best-effort delivery of MPEG. PPD defines each IP packet to be a unit; datagrams are discarded if they cannot be buffered in the output queue.

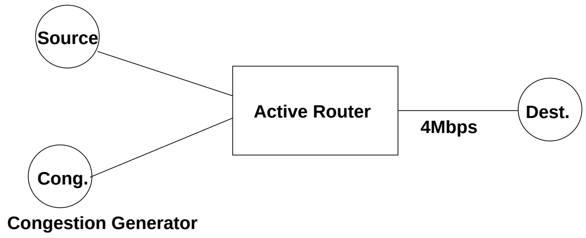

Figure 2: Emulation topology

Three designs are considered that combine end-system knowledge with network discarding policy. The **Static Priority Discard** mechanism establishes a generic two level priority within the data stream. For MPEG streams, datagrams carrying I-frames are considered to be higher priority than all other datagrams. The **Frame Level Discard** mechanism defines a unit to be an MPEG frame. The advice given is to queue a datagram if and only if its corresponding frame can be queued in its entirety. The **Group of Picture (GOP) Level Discard** maintains state about the type of frame discarded. In case an I-frame has been discarded, we discard the corresponding P and B frames as well. Thus, the GOP level discard mechanism is the application-specific case in which information about both priority and dependencies amongst application specific data units is available to the network. Detailed descriptions of the mechanisms and the experiments can be found in [10].

The experimental topology is shown in Figure 2. The bandwidth on the links between the source and the router, and between the congestion generator and the router, are large enough so that these links are never congested. The bandwidth on the link from the router to the destination is 4 Mbps, thus any combination of source and background traffic that exceeds 4 Mbps will result in congestion into the destination. The congestion generator continually replays packet traces with average cumulative bandwidth of 2.1 Mbps.

To compare the end-system only design (PPD) to the three combined designs (Static Priority Discard, Frame Level Discard and GOP Level Discard) we must define a reasonable performance metric. We consider two metrics. The first is the fraction of data that is received but cannot be used due to partial or not decodable frames. This useless data places a burden on the receiver and the network. The second metric is more oriented towards application performance. We measure the average signal-to-noise ratio (SNR) for the received stream; in cases when frames are dropped, we use the last correctly decoded frame as a substitute for the missing frame.

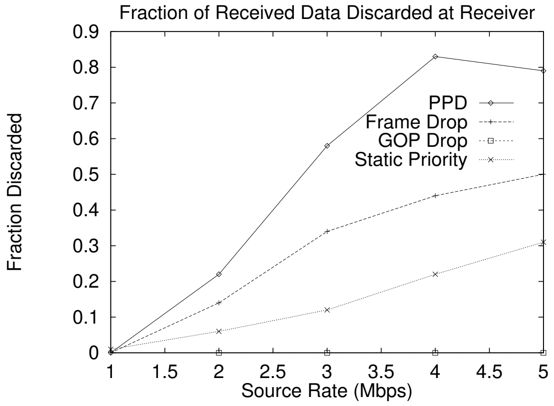

Figure 3: Fraction of received data discarded

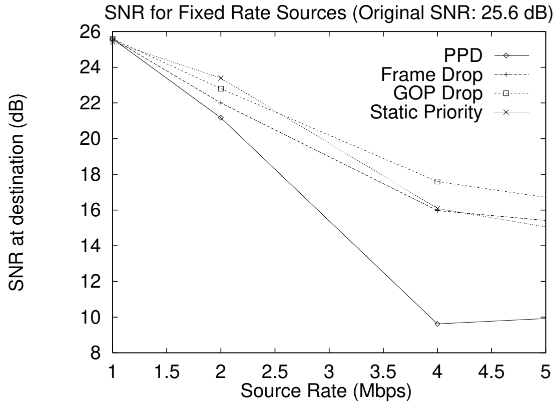

Figure 4: SNR of received MPEG stream

5.2 Use of Model: Discarded Data

Figure 3 shows the fraction of received data discarded at the receiver as the transmission rate of the source is varied. Remarkably, the GOP Level Discard mechanism does not transmit *any* useless data. When the source rate exceeds 2 Mbps, thus congesting the link to the destination, the percentage of data discarded under PPD increases rapidly, from 20% to 85%. Frame Level Discard also performs poorly, with the percentage of data discarded reaching 50%. Interestingly, the Static Priority scheme does relatively well until the link to the destination is more severely loaded: the percentage of data discarded is under 10% until the source rate exceeds 3 Mbps.

We can take the data from Figure 3 and view it in light of the model of Section 3. Let $D_{P,k}$ denote the expected fraction of data discarded at the receiver under the end-system only design (PPD) at a given source rate of k Mbps. Let the performance of the end-system design at a source rate of k Mbps be $T_{P,k}$. Define $T_{P,k}$ to be $1 - D_{P,k}$. When the source rate is 2 Mbps, performance of PPD $T_{P,2}$=0.78, but as the source rate is increased, the expected performance rapidly decreases. When the source rate is increased to 4 Mbps, the performance $T_{P,4}$ decreases to 0.17.

The other three mechanisms are examples in which there is a varying degree of co-operation between the end-systems and the network. These cases correspond to three different design choices for the combined end-system and network (Design C). Let $D_{S,k}$, $D_{F,k}$, $D_{G,k}$ denote the expected fraction of data discarded at the receiver under the Static Priority, Frame Level Discard, and GOP Level Discard design choices, respectively at a source rate of k Mbps. We can define the performance at source rate k Mbps of each of the three designs similar to the performance in the PPD case, i.e.

$$T_{i,k} = 1 - D_{i,k}, \quad i = \{S, F, G\}$$

$T_{G,k}$ attains optimal performance for all source rates. The performance of Static Priority at a source rate of 2 Mbps $T_{S,2}$ is 0.94, but decreases to 0.78 as the source rate increases to 4 Mbps. At a source rate of 2 Mbps, the performance of Frame Level Discard is 0.86, and decreases to 0.56 as the source rate is increased to 4 Mbps.

5.3 Use of Model: SNR

The discarded data metric may not capture everything of interest about the performance of the congestion control mechanisms. In particular, a metric that is more closely tied to application performance is likely to be of interest. We compute the signal-to-noise ratio (SNR) of the decoded stream at the receiver; in cases when frames are dropped, we use the last correctly decoded frame as a substitute for the dropped frame. The SNR is computed with respect to the original uncompressed (YUV) files which were used to create the transmitted MPEG stream.

Figure 4 shows the SNR of the received MPEG stream as the source rate is varied. No data is lost at a transmission rate of 1 Mbps in Figure 4 and as such the SNR at the receiver is equal to the transmitted SNR in all cases (25.6 dB). Under *marginal* congestion (source rate 2 Mbps), there is higher signal degradation under PPD (21.1 dB) compared to all other schemes ($>$ 22 dB). However as the severity of congestion increases, the PPD fares poorly (10.2 dB at source rate of 4 Mbps). Both the Frame Level Discard and the Static Priority Discard schemes perform similarly ($<$ 16.0 dB) under severe congestion (source rate $>$ 4 Mbps). The application specific GOP Level Discard scheme performs best under similarly severe congestion (SNR $>$ 17.6 dB.)

5.4 Alternate End-system Design

Finally, we consider a more sophisticated scheme at the end-systems to help achieve high quality service. In particular, we add a simple form of end-to-end feedback to allow the source to adapt to the available bandwidth. This is done with flow control over UDP as follows. The flow control mechanism has three parameters: the feedback resolution F, the reaction rate R and the source increment S. The mechanism operates on the principles of linear increase and exponential decrease as follows. The receiver sends feedback whenever it determines that at least F frames have been transmitted since it last sent feedback. This determination is made by receiving the last frame in the set of F, or receiving a frame from a later set (based on sequence number). If all F frames were received correctly, the receiver sends an ACK, otherwise it sends a NACK. For each NACK, the sender cuts its rate in half. Upon receiving R consecutive ACKs, the sender increases its rate by S.

In our experiments, the source rate is initialized to 1 Mbps, and constrained to less than 8 Mbps. The source increment S is 2 Mbps, and the reaction rate R is $\lfloor 40/F \rfloor$, where F is the feedback resolution and is set to 8.

Under these conditions, the PPD and Static Priority mechanisms waste 35% and 20% of the link bandwidth respectively. The Frame Level Discards wastes 9% of the link bandwidth by transmitting packets that are useless at the receiver. Again, link utilization is optimal under GOP Discard, and no data transmitted by the GOP Level Discard mechanism is discarded at the receiver.

With feedback, the PPD and Priority schemes provide a SNR of 17.1 dB and 17.5 dB respectively. The Frame Level Discard is able to provide a SNR of 21.6 dB, while the GOP Level Discard mechanism provides SNR above 22 dB.

6 Concluding Remarks

In this paper, we have argued that active networking —in particular the provision of a generic service interface that can be programmed and customized by individual users— is consistent with, and even suggested by, the end-to-end argument. We have enumerated the costs associated with providing such a generic active networking interface, and presented a model for quantifying performance gains using active networking techniques. We have shown how the model can be applied with two services, reliable multicast and application-specific congestion control. In both cases, the active networking approach utilizes information that is only available inside the network, and outperforms the end-to-end solutions.

As future work, we are considering extensions to model that will relax some of the assumptions of the original model, and quantifying the benefits to other applications (like self-organizing network caching) due to active networking.

References

[1] H. Saltzer, D. P. Reed, and D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computing Systems*, vol. 2, no. 4, 1984.

[2] D. Tennenhouse, J. Smith, and G. M. W. Sincoskie, D. Wetherall, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, 1997.

[3] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," in *Multimedia Computing and Networking '96*, January 1996.

[4] J. Smith, "Switchware: Accelerating network evolution," Tech. Rep. MS-CIS-96-38, CIS Dept., University of Pennsylvania, 1996.

[5] T. Pusateri, "Distance Vector Multicast Routing Protocol," Internet Draft, Internet Engineering Task Force, 1997. Work in progress.

[6] J. C. Lin and S. Paul, "RMTP: a reliable multicast transport protocol," in *Proceedings of Infocom*, Mar. 1996.

[7] C. Papadopoulos, G. Parulkar, and G. Varghese, "An error control scheme for large-scale multicast applications." Available at http://dworkin.wustl.edu/ christos/PostScriptDocs/current.ps.Z.

[8] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang, "Reliable Multicast Framework for Lightweight Sessions and Application Level Framing," in *SIGCOMM*, (Cambridge, Massachusetts), Sept. 1995.

[9] M. Yajnik, J. Kurose, and D. Towsley, "Correlation in the MBone multicast network," in *IEEE Global Internet Conference*, (London), November 1996.

[10] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "An Architecture for Active Networking," in *Proceedings of High Performance Networking 97*, 1997.