An Architecture for Active Networking

Samrat Bhattacharjee
Kenneth L. Calvert
Ellen W. Zegura

{bobby,calvert,ewz}@cc.gatech.edu

GIT–CC–96/20

Abstract

Active networking offers a change in the usual network paradigm: from passive carrier of bits to a more general computation engine. The implementation of such a change is likely to enable radical new applications that cannot be foreseen today. Large-scale deployment, however, involves significant challenges in interoperability, security, and scalability. In this paper we define an active networking architecture in which user control the invocation of pre-defined, network-based functions through control information in packet headers.

After defining our active networking architecture, we consider a problem (namely, network congestion) that may benefit in the near-term from active networking, and thus may help justify migration to this new paradigm. Given an architecture allowing applications to exercise some control over network processing, the bandwidth allocated to each application's packets can be reduced in a manner that is tailored to the application, rather than being applied generically. Our results show that the ability to gracefully adapt to congestion makes a good case for active networking.

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280

1 Introduction

As the cost of computing power decreases, it is worthwhile to consider the benefits of adding computing to all types of systems, either to enhance services or to trade off against other costs such as time, bandwidth and storage. *Active Networking (AN)* refers to the addition of *user-controllable* computing capabilities to data networks[1]. With active networking, the network is no longer viewed as a passive mover of bits, but rather as a more general computation engine: information injected into the network may be modified, stored, or redirected as it is being transported. Obviously, such a capability opens up many exciting possibilities. However, active networking also raises a number of issues, including security, interoperability and migration strategy. All of these are influenced in large part by the active networking *architecture*, which defines the interface between the users and the capabilities provided by the network.

In this paper, we consider an approach to active networking that generalizes traditional packet switching. In our approach, users can select from an available set of functions to be computed on their data, and can supply parameters as input to those computations. The available functions are chosen and implemented by the network service provider, and support specific services; thus users are able to influence the computation of a selected function, but cannot define arbitrary functions to be computed. This approach has some benefits with respect to incremental deployment as well as security and efficiency: AN functions can be individually implemented and thoroughly tested by the service provider before deployment, and new functions can be added as they are developed.

1.1 A Near-Term Application of Active Networking

Deployment of active networking will obviously require substantial modifications to existing applications and networking infrastructure. The question, however, is whether existing applications can benefit from the placement of computational capabilities within the network. The quantity and quality of these benefits will determine the price point at which active networking becomes reasonable to deploy.

Later in the paper, we show how mechanisms consistent with this approach can help users (applications) deal with a problem that is unlikely to disappear in the near future: network congestion. The basic idea is that, instead of applying "one size fits all" congestion reduction techniques, mechanisms are placed in the network to allow each application to give advice to the network on how to reduce (by discarding or transforming) its data should congestion occur. Congestion is a prime candidate for an active networking solution, because it is specifically an intra-network event and is potentially far removed from the application.

The suitability of active networking to aid in dealing with congestion is based on two assumptions:

- Transmission bandwidth and computational power will both continue to increase, but so will application requirements for bandwidth. In particular, we assume that congestion occurs at network nodes due to bandwidth limitations, and that even

[1] In this paper we focus on packet- and cell-switched networks. We follow Tennenhouse and Wetherall [19] in adopting the term active networking; the differences between our approach and theirs are elucidated later.

congested nodes will have spare processing power (though not unlimited buffering) available.

- There will always be applications that prefer to *adapt* their behavior dynamically to match available network bandwidth, as opposed to reserving bandwidth in advance. This assumption can be justified based on the higher cost likely to be associated with reservations, and the success of the sender-adaptation model in the Internet. We further elaborate on this point in Section 3.

1.2 Roadmap and Contributions

In the next section we present a simple but general model of packet processing in a network, and show how it encompasses both traditional networks as well as the addition of more powerful computing capabilities to the network. We then describe a particular approach, in which packet headers identify and control the processing functions applied to the packet. In Section 3, we describe an active networking mechanism that supports user-controlled degradation of service under congested conditions. In Section 4 we discuss variations on this basic mechanism, and describe how applications might benefit from its use. Results of our experimental evaluation of the mechanism are presented in Section 6; the results show that the mechanism can significantly improve the utility of the network service. Section 7 offers some concluding remarks and directions for future work.

Our contributions include the following:

- A simple and general architecture for active networking, based on user control of pre-defined AN functions via control information placed in the packet header.
- Demonstration of a near-term benefit of active networking for applications that can benefit from controlled degradation of service under congestion.
- Prototype implementations of IP- and ATM-based AN architectures.
- Experimental investigation of a range of active congestion mechanisms both in contrast to and in conjunction with traditional end-to-end mechanisms for adaptation to congestion.

2 An Architecture for Active Networking

We first describe a simple model of packet processing in packet-switched networks, and then present our approach to active networking as a natural extension of that model. We then address some considerations for adding active networking to existing network architectures. Finally, we briefly describe our testbed implementation.

2.1 A Generic Model of Packet Processing

The network consists of switching *nodes*, which are connected via *links*. In this simple model, nodes don't do anything except process the packets received on their incoming

links; processing an incoming packet may result in one or more packets being transmitted on outgoing links.

More precisely, the *state* of a node comprises the following pieces:

- An *input queue* of packets. Packets received on any link are placed in the input queue.
- For each outgoing link, an *output queue* containing packets to be transmitted on that link.
- A collection of *generic state information*. This represents long-lived information maintained at the node, such as routing tables or virtual-circuit switching tables.

We postulate that each node in the network supports a particular set of functions, each of which has a unique identifier. Each packet contains a set of *headers*, which specify (i) the identifier of one or more functions to be applied to the packet; and (ii) parameters to be supplied to those functions. When the packet is processed, the function identified by each header is applied, resulting in updating of the node's state and possibly modification of the rest of the packet. Different functions affect different components of the node's state. For example, regular end-to-end data packets do not modify the routing tables, but routing protocol packets may; unicast packets result in one packet being forwarded, while multicast packets result in many. Thus we postulate that for each function identifier f, and each parameter value p for function f, there is a particular subset of the node's generic state information that is *relevant* to f and parameter p. Functions cannot modify or use parts of the node state that are not relevant.

Each node repeatedly performs the following:

```
Remove a packet M from the input queue;
while (more functions need to be applied to M):
    Let f, p be the function ID and parameter from the next header of M;
    Let g be the state component relevant to f and p;
    Invoke function f on M, with p as parameter:
        (optionally) Modify M;
        (optionally) Update g;
        (optionally) Queue messages for output;
```

Traditional networking functions can be characterized as node-processing functions in this model. For example, when an IP datagram containing destination address X is received by a router node (none of whose addresses is X) over an Ethernet, the router node examines two of its headers, which identify the "Ethernet function" and the "IP forwarding function". The Ethernet function checks the error-control code and removes the Ethernet header, while the IP forwarding function determines which outgoing link the packet should be forwarded on, and adds the appropriate link-level header for that link. Similar interpretations of the model for virtual circuit switching and other node functions, such as network management, are straightforward.

2.2 From Packet Forwarding to Active Networking

We define active networking to be extension of the set of functions that can be invoked at a network node beyond those required to simply move bits from place to place. The basic idea of our approach to active networking is the incremental addition of user-controllable functions, where each function is precisely defined and supports a specific service. For example, consider giving users the capability to make use of state storage in network nodes. This might be used in, say, a mobility-support function, where a mobile user can associate "pointers" (to its new location) with an identifier unique to that user. Packets destined for that user would contain the user's identifier and the "mobility update function" identifier, which would cause the relevant state to be retrieved, and the packet to be forwarded accordingly.[2]

In general, the introduction of new AN functions of the type we have in mind involves specification of the following:

- The *identifier* associated with the function.
- The *parameters* associated with the function, and the method of encoding them in a packet.
- The *semantics* of the function. Ideally, function semantics would be given in a standard notation such as LOTOS [6], SDL [7], or another notation developed specifically for the purpose. A *standard environment*, comprising support services such as private state storage and retrieval, access to shared state information (e.g. routing tables), message forwarding primitives, etc., would provide a foundation on which new AN functions services could be built.

In our view of AN, addition of a new function to a network node would be the responsibility of the network service provider. As with current networks, once a function is specified, each provider or vendor would be free to implement the functionality in manner consistent with the specification. This approach corresponds roughly to the way new features are deployed in the public switched telephone network today: users have the option of provisioning various features implemented and deployed by the service provider. (In the telephone network, these features go under the title "Advanced Intelligent Network".)

There is another approach to extending the set of available functions at a node: adding a single very powerful (Turing-complete) function, which interprets its parameter as a program and then executes that program with the packet and the relevant state as inputs. This single function thus extends the set of functions computable at a node to include all computable functions. The "active capsule" approach of Wetherall and Tennenhouse [19] is based upon this idea. In this approach, the interface between the user and the active network is a programming language, with well-defined syntax and semantics.[3] All other functions are subsumed by the "interpreter" function: traditional packet-forwarding services are obtained by including a "standard" routing program in the packet.

The differences between these two approaches ("individual specialized functions" vs. "single Turing-complete function") are analogous to the differences between adding

[2] Clearly there are limits on the amount of per-user that can be maintained at a node; for scalability, the state repository would function as a cache, with information aged and removed periodically.

[3] Wetherall and Tennenhouse are using "safe Tcl" initially.

individual features to HTML and defining the Java language. In the former case the semantics of each feature must be considered, including its interactions with all other features; in the latter case the set of all possible programs must be considered in defining the language and its interface to the rest of the system. It may be easier to address concerns about security and efficiency for a single function as opposed to an entire language (cf. Java). In addition, with our approach it is possible to deploy active networking *incrementally*, beginning with simple AN functions that provide immediate benefits for users and providers, and expanding the set over time. Later in this paper, we describe functions that can provide such near-term benefits.

2.3 Implementation Considerations

Implementing the above-described approach to active networking requires some means of encoding information about the AN functions that are to be applied to a packet, as well as the parameters of those functions. Doing this in a backward-compatible manner means placing this *AN control information* in one or more *headers* in the packet. The proper location for this information in the packet is determined by two factors: how the network node will locate the information in the packets it switches, and how the control information will be inserted by the end system.

One possibility is to put the AN control information in the same header used to switch the packet, i.e. in the same place as the control information for "traditional" network functions. However, this doesn't work for ATM or other technologies where the switched unit is too small to accomodate additional the overhead of the function ID and parameter information, which may be relatively large. An alternative is to define a "generic" location for the control information, sufficiently high in the protocol stack that the additional overhead is not prohibitive, but low enough to allow its location by switching nodes without too much knowledge of higher-level protocols. In particular, the information can be placed after the header of the first relay layer whose data units are large enough to handle the additional overhead. For example, in a network in which ATM is used as the end-to-end transport, the control information would follow the ATM Adaptation Layer header. This implies that an active ATM node must reassemble AAL Data Units before it can examine their AN control information. This adds latency to the processing of those data units.

If the Internet Protocol (version 4 or version 6) is being used as the end-to-end switching layer, the AN control information can be treated as a "sublayer" or optional header, as is done for example with the headers used to implement the IP security architecture [2, 3].

What if multiple network layers are in use? For example, many IP internets today include ATM as a subnet technology, but not all traffic those ATM networks is IP traffic. In this scenario, ATM switches handle a mixture of traffic, of which some carries an IP header while the rest has only ATM headers. IP switches, on the other hand, deal only with IP datagrams, some of which are transmitted over ATM subnets. The ideal solution is for IP datagrams to have the AN control information following the IP header, and for ATM-only traffic to have it following the AAL header. This implies that ATM active nodes have to look in *two* places for control information: following the AAL

header in non-IP traffic, and following the IP header for IP traffic. This requirement is reasonable *provided* the ATM-AP does not have to do reassembly of the IP datagram as well as the AAL data unit, i.e. there is one IP datagram (or fragment) per AAL-PDU.

Placement of AN control information in the packet by the end system is complicated by the fact that the information may depend on multiple protocol layers. For example, the control information for an intelligent discard mechanism (Section 4) should identify the various levels of data units to which the packet payload belongs: a block number in a file, a TCP sequence number, an IP datagram identifier. (Feldmeier's "chunk" labeling scheme [9] is an example of a possible encoding for this information.) The question is how to aggregate this information in a single location in a packet, especially if the end system's protocol implementation is layered. Ideally, the AN control information will originate with the application itself, because it can best determine which, if any, AN functions are appropriate for its data.[4] As the data unit progresses down through the protocol stack, each protocol may add to the AN control information it receives from the layer above. The question then becomes where to place the information in the packet. If this is done by the first appropriate layer (e.g. IP or AAL) that receives it from the layer above, the rule stated in the previous subsection will be satisfied, i.e. the control information will follow the header of the highest-level end-to-end relay layer.

2.4 An AN Implementation

We have implemented several AN functions in an IP routing architecture. We defined an "Active Processing" IP option `IPOPT_AP`, which is specifies a particular AN function in IP routers. The `IPOPT_AP` option is a tuple of the form $\langle f, m, \alpha \rangle$. f identifies the active processing function and m is used to retrieve cached state associated with the current flow. α is a variable (possibly null) sequence of arguments to the function f. We augmented an existing SunOS IP kernel to support an in-kernel active processing router. We implement an associative memory function M, which can be used to retrieve state (e.g. queues) associated with the current flow. We define an AN function dispatch table Δ, such that $\Delta[i]$ is the address of the ith active processing function. We modified the IP option processor to execute $\Delta[f](M(m), \alpha)$ for each datagram that specifies an AN IP option. We further modified the IP option processor to notify IP not to further process datagrams with an active processing header. In order to simulate congestion, we define a "virtual router memory limit", and the datagram discard routine is initiated as this virtual router memory is exhausted. To let the router queues grow, we service the router queues at predefined output rates – this is used to simulate various output link rates.

We have also implemented a functionally similar ATM active network node, comprising a switch and an attached processor [4]. Circuits are routed through the switch to the attached "active processor", where the AN functions are implemented in user space; after processing, data is sent back through the switch and on to the destination. In this case, the AN header is just part of the user data instead of an option header. A similar virtual memory and queue service strategy are used in the ATM version as well. Details of our experiments using AAL5 as the adaptation layer, and a null transport layer can be found in [4].

[4] To allow for this, the network user interface could be extended to enable the application to specify AN control information on a per-application-data-unit basis, say as a parameter to the "send" primitive.

3 A Mechanism for Graceful Degradation under Congestion

We now turn to the application of active networking to provide graceful, user-controlled degradation under congestion. We begin by discussing the advantages and problems with traditional sender-adaptation, and we conclude that active networking can maintain the basic advantages while alleviating two significant problems. We then present three broad categories of active adaptation methods. In the remainder of the paper we expand on the Intelligent Discard category, by providing and evaluating several detailed example mechanisms.

3.1 Adaptable Applications

We expect that there will always be applications that prefer to *adapt* their behavior dynamically to match available network bandwidth, as opposed to reserving bandwidth in advance. This expectation is based on several observations. One is that there will be times when the network will reject requests for reserved bandwidth and applications will have to use a non-reserved service or go away. Also, reserved bandwidth is likely to cost more. Further, the recent advances in understanding the self-similar nature of traffic [13, 20, 14] indicate that effective congestion control is likely to require longer time scale methods (e.g., source adaptation and admission control) rather than shorter time scale methods (e.g., adding more buffering).

At the same time, however, it must be noted that the sender-adaptation model [12], which has worked so well in the Internet, presents a couple of well-known challenges. The first is the time interval required for the sender to detect congestion, adapt to bring losses under control (e.g. by reducing transmission rate in the case of TCP, or adjusting coding parameters in the case of continuous media), and have the controlled-loss data propagate to the receiver. During this interval, the receiver experiences uncontrolled loss, resulting in a reduction in quality of service that "magnifies" the actual bandwidth reduction. (For example, if a portion of each of several application data units is lost, each entire data unit may become useless to the receiver.) As transmission and application bandwidths increase, this problem is exacerbated because propagation delays remain constant[5].

The other well-known challenge of sender adaptation is detecting an *increase* in available bandwidth. This problem, which is worse for continuous-media applications, arises in best-effort networks because loss is the only mechanism for determining available bandwidth. Thus, for example, if a sender adapts to congestion by changing to a lossier encoding, it must detect the easing of congestion by periodically reducing compression and waiting for feedback from the receiver. In the case of long-lived congestion, this dooms the receiver to periodic episodes of uncontrolled loss.

In view of the foregoing, we conclude that a useful application of active networking is to move those adaptations a sender might make into the network itself, in order to solve both of the problems just noted. The general objective is thus to ensure that, as far as possible, *losses occur in a controlled and application-specific manner.*

[5] As an example, consider an application transmitting at the available bandwidth of 100 Kbps, with a round-trip delay of 30 milliseconds. If the available bandwidth is reduced by 20%, and the sender adapts immediately, the amount of uncontrolled loss is around 75 bytes. However, if the same sender is transmitting at 1 Gbps, the uncontrolled loss when the bandwidth is reduced by 20% is 750 Kilobytes.

Note that the Available Bit Rate (ABR) service being defined for ATM [5] is also intended to relieve the two problems noted above. ABR reduces the time required for the sender to detect congestion by using resource management (RM) cells to convey congestion information. With RM cells, the time to notify the sender of congestion is reduced to the path delay from the point of congestion to the sender[6]. ABR addresses the problem of detecting an increase in available bandwidth by providing explicit rate feedback information. Switches can specify the allowed sender rate, based on calculations of current load and fair-share per sender. Thus ABR attempts to improve the sender-adaptation process, while maintaining the adaptation at the sender.

We envision that our active congestion control might be used either (1) in conjunction with, or (2) exclusive from, sender adaptation methods like ABR, depending on the application and the duration of the congestion. Some applications will generate traffic from dumb devices (e.g., video cameras) that are incapable of performing their own adaptation, and can benefit from the (shared) adaptation processing capability at active nodes. If congestion is severe and long lived, sending applications that can adapt will probably want to do so, in addition to utilizing intra-network active congestion control. In Section 6 we examine the performance of active congestion control both with and without sender rate adaptation.

The importance of graceful degradation is well recognized, and indeed is supported by techniques such as layered encoding of images and video [11, 10, 15]. In layered encoding, the image is coded into successive levels. Reconstruction is possible (albeit at lower quality) even if one or more lower level components are missing. Layered encoding provides a fairly coarse level of control over the bandwidth of the stream, based on the number of levels transmitted.

3.2 Active Adaptation Methods

Active adaptation can be performed in many ways. We consider three broad categories:

Intelligent Discard. Intelligent discard is a natural extension of packet-level discard [1, 17]. Units are defined based on application semantics, and the entire unit is discarded if any portion must be discarded. Note that to be most effective, this does require buffering at the network node to collect a unit to ensure that the "tail" will not be dropped. Thus some additional latency and buffering at the node are traded for improvements in useful delivery.

Clearly, unit-level discard interacts with end-to-end reliability mechanisms. The most benefit derives from dropping units that are equivalent to the reliable retransmission units. Dropping a unit that is smaller or larger may result in retransmission of data has already been received.

Media Transformation. Whereas intelligent discard relies on out-of-band information (generic or user-supplied) to determine unit boundaries, active networks armed with information about the encoding of the user information can go further, and *transform* user data at a congestion point. Such a transformation should reduce bandwidth

[6]Note, however, that this delay can be considerable, if the point of congestion is far removed from the sender, or the path back to the sender is itself congested.

requirements while preserving as much useful information as possible. In essence, this may allow the active node to create the form of the data which the application would have created, had it known about the bandwidth limitations encountered at the congestion point. A number of techniques are suitable for on-the-fly transformation of MPEG and JPEG data, including selective discard of discrete cosine transform (DCT) coefficients [8, 16, 18], and decoding and recoding at higher quantization.

Multi-stream Interaction. In the architecture described earlier, active networks are capable of operating on units that have been defined to include data from multiple streams. Consider the following examples of multi-stream interactions:

- Multimedia playout.

  Two streams that are to be played out together (e.g., video and audio, text and graphics) are carried on separate streams. This would arise if the feeds for the streams are in different locations, or if the streams are stored in different locations (e.g., media repositories that can be used to create multimedia presentations). If a segment is lost from one, it would be helpful if the other is elevated in importance so that the end user maintains some information.

- Sensors.

  Sensors monitoring some critical system (e.g., medical patient condition or home security) independently report to a central monitoring station. Again, if data is lost from one sensor, the information from the others increases in importance.

- Video striping.

  MPEG compressed video is separated into three streams, one for each type of frame (I, P, B). This would enable the quality of service specification for each stream to be tailored to the stream type; since the three frames differ significantly in their compression and importance, this could improve overall performance. Since P frames may depend on certain I frames and B frames may depend on certain P and I frames, the reconstruction and playout process must have all dependent frames. Thus, a loss of an I or P frame should trigger loss of any P or B frames that depend on the lost frame.

In the first two examples, a "unit" comprises data from two or more streams, and the data from one stream is elevated in importance if data from another stream is dropped. This contrasts with the last example, where the unit is not considered complete unless all components from all streams have been received.

4 Intelligent Discard Mechanisms

For the remainder of the paper, we focus on the Intelligent Discard category. We give a generic description of the type of application that can benefit from Intelligent Discard, and show that MPEG is a specific example. We discuss a set of metrics that can be used

to evaluate the performance of our congestion control mechanisms. We then present four active congestion mechanisms for handling MPEG.

For our purposes, the important feature of an MPEG stream is that it consists of a sequence of frames of three types: I, P and B. Coding dependencies exist between the frames, causing P and B-frames to possibly require other frames in order to be properly decoded. Each I-frame plus the following P and B frames forms a group of pictures (GOP), which can be decoded independently of the other frames.

4.1 Assumptions about Applications

Two features of an application are important for determining whether it can benefit from an Intelligent Discard mechanism.

- Units

  The application must have one or more "units" that are significant. These units come in two forms: (1) without dependencies, where the loss of any part of the unit renders the entire unit useless, and (2) with dependencies, where the loss of certain parts of the unit will render the entire unit useless. (Of course, whether the unit is useless depends on application semantics and end-to-end reliability mechansims. We discuss our reliability assumptions below.)

  For MPEG, we can define two units: an MPEG frame (I, P or B), and an MPEG GOP. MPEG frames are an example of a unit without dependencies, since loss of any part of a frame may render the entire frame useless, while MPEG GOPs are an example of a unit with dependencies. We assume that loss of the I-frame in a GOP renders the P and B-frames in the GOP useless. Note that this is a slight simplification; some P and B-frames may be coded independent of the I-frame.

- Reliability

  As mentioned earlier, unit-level discard is clearly not independent of the end-to-end reliability mechanisms. If an end-to-end reliability mechanism is in place, either at the transport or application layer, then the most effective unit to discard is the unit corresponding to the reliable retransmission unit. For applications running over TCP/IP, IP Packet Discard is most reasonable.

  For MPEG, we consider IP Packet Discard as a point of comparison, however we focus primarily on discard of larger units. Thus, our results apply to applications that implement their own reliability (e.g., over UDP) or are unreliable (e.g., a video stream from a dumb device).

4.2 Metrics

We evaluate the congestion control mechanisms using three metrics. (While these are stated in terms specific to MPEG, they can easily be generalized to any application with units as defined above.)

- Fraction of I-frames received

From the receiving application perspective, an important metric is the fraction of the data sent that is received in a form that is "useful."

For MPEG, we define a frame to be "useful" if and only if the frame is received in its entirety, and any frames that it depends upon for decoding are also received in their entirety. According to this definition, we should measure the number of useful frames divided by the number of frames sent. An alternative is to measure the number of useful I-frames divided by the number of I-frames sent. We elect the latter since our intelligent discard algorithms are designed to give preference to I-frames, and we are interested in evaluating how well they achieve this end. An active algorithm could easily be designed to optimize the former, however this would require communicating the frame dependencies; our current algorithms do not include this complexity.

- Fraction of capacity wasted

  To assess the efficiency of network usage, we measure the fraction of the link capacity (from the active node to the receiver) that is used to send data that is ultimately not useful to the receiver. Specifically, we count the number of bytes received that are useless (i.e., they belong to a P or B frame whose I frame was discarded), and divide by the total number of bytes received. As noted earlier, this is a slight simplification, since it is possible for a P or B frame to be independent of the corresponding I-frame.

- Transmission latency

  We record the time that each datagram is successfully delivered (i.e., in order, without errors) to the receiver application. The difference in time between the last datagram and first datagram corresponds to the end-to-end transmission time, and is affected by the sender rate, losses and retransmissions, and the active network processing.

  We are particularly interested in verifying that the active processing methods do not impose substantial increases in end-to-end latency.

4.3 MPEG Discard Algorithms

We have implemented a variety of frame discard algorithms (each of which take $O(1)$ time per datagram) as active processing functions. The details of the frame header structure and the individual algorithms are provided in Appendix I.

- **Partial Packet Discard** This is a simply an implementation of IP routing. Datagrams are discarded if they cannot be buffered (though the memory limit is imposed by the virtual router memory). This is also the simplest algorithm to implement as it does not require deciphering the frame header structure.

- **Frame Level Discard** This is an extension of the partial packet discard algorithm in which a datagram is buffered if and only if its corresponding frame can be buffered in its entirety. We maintain state for each frame that is being discarded

or buffered, and using this state information, we can decide to buffer or discard a particular datagram in constant time.

- **Group of Picture (GOP) Level Discard** We augment the frame level discard algorithm to maintain 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.

- **Early MPEG Discard** The early MPEG drop algorithm initiates an *early* discard if the fraction of memory exhausted is greater than a certain predefined fraction of total memory. Otherwise, this algorithm is identical to the GOP level discard algorithm. Note that this generalizes the Early Packet Discard scheme of Romanow and Floyd [17].

5 Evaluation Methodology

Given the above set of active congestion control mechanisms, we can ask a number of questions about performance.

- How well do the different mechanisms perform, on their own, *without* end-to-end adaptation?

  We evaluate this by applying each mechanism to a fixed rate, non-adaptive source. We vary the rate at which the source is fixed, and evaluate performance using the metrics described above.

- How does the performance compare to end-to-end adaptation?

  To explore the role of end-to-end source adaptation, we implemented a simple form of flow control over UDP. The flow control mechanism has three parameters: the feedback resolution F, the reaction rate R and the source increment S. The mechanism operates 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.

- How do the mechanisms perform when used *in conjunction* with end-to-end adaptation?

  We evaluate this by combining the simple flow control with the active congestion mechanisms. In other words, the router implements the active mechanisms, and the sender adapts its rate based on receiver feedback.

- What is the effect of background traffic?

  We include background traffic of two forms in our system. One form is IP trace data, collected from a backbone campus network at Georgia Tech and at the San Diego Supercomputer Center. This data has average rate approximately 2.5 Mbps.

The other form of background traffic is a stream of MPEG frames representing a multiplexing of a number of individual MPEG streams. This data has average rate just over 2 Mbps.

The background traffic can be subjected to active congestion control mechanisms or not. We examine the performance for both cases.

6 Results

6.1 Experimental Setup

In all of our experiments, we have two machines sending data to our active router. One machine serves as the source under test, the other provides aggregate background traffic. The connections were over ATM links, and as such, there was no network interference.

[Specifically, the active router was implemented on a Sparcstation 20, running SunOS 4.1.3. The destination and the background traffic generator machines were Sparcstation 20s, both running SunOS 4.1.3 as well. The source under test was run on a Sparcstaton 5, running Solaris 2.5. All communication was over a Fore ATM LAN, running at 155Mbps. For both the test and the background sources, the active router was made the default route to reach the destination.]

The active router receives the two streams of data, buffers and processes the data according to the various active congestion mechanisms, and transmits to a receiving machine. To allow the link from the router to the receiver to experience congestion, we intentionally "slow" the transmission, imposing rate control at a level of 4 Mbps from router to receiver. We restrict our router to 48 Kbytes of virtual memory, allowing buffer overflow to occur fairly quickly once the outgoing link is saturated. Each data point represents the average of 10 runs, where a single run consists of 50 Mbytes of MPEG data sent from the source under test. We calculated the 95% confidence intervals, but we do not include them as they are extremely tight in all cases.

6.2 Fixed Rate Sources

We begin by considering the simplest case: the source operates at a fixed rate with no background traffic. Figure 1 shows the fraction of I-frames received for each of the four mechanisms as the (fixed) source rate is increased. As expected, the performance of the PPD scheme falls off drastically when the source rate exceeds the link rate of 4 Mbps. The Frame Discard mechanism also performs poorly when the link is overloaded, though the performance degrades more slowly than PPD; Frame Discard gives no priority to I-frames and thus drops proportionately more I-frames as the source overloads the link.

GOP Discard and Early MPEG Discard (EMD) are far more successful at delivering useful data to the receiver, with nearly half the I-frames getting through even with source rate of six times the link capacity. The size of the router virtual memory and the EMD threshold of 0.9 combine to make EMD quite similar in operation to GOP Discard. Only 4.8 Kbytes of buffer space remain when the threshold is reached; this is generally not sufficient to hold an entire I-frame. A more conservative threshold or a larger router memory would make the differences between EMD and GOP Discard more significant.

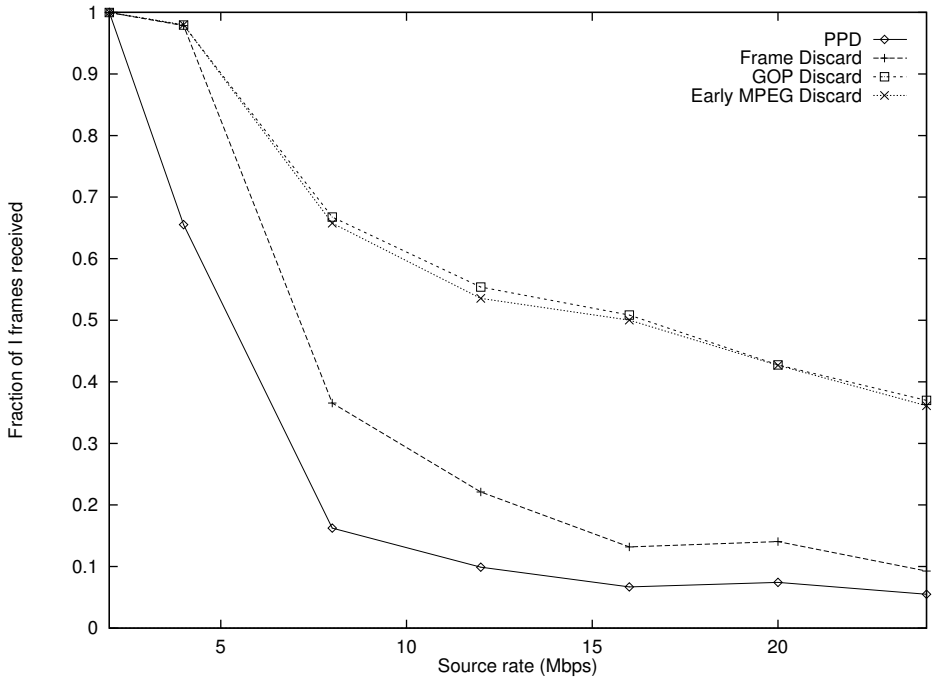

Figure 1: Fixed Source: Useful Data Received

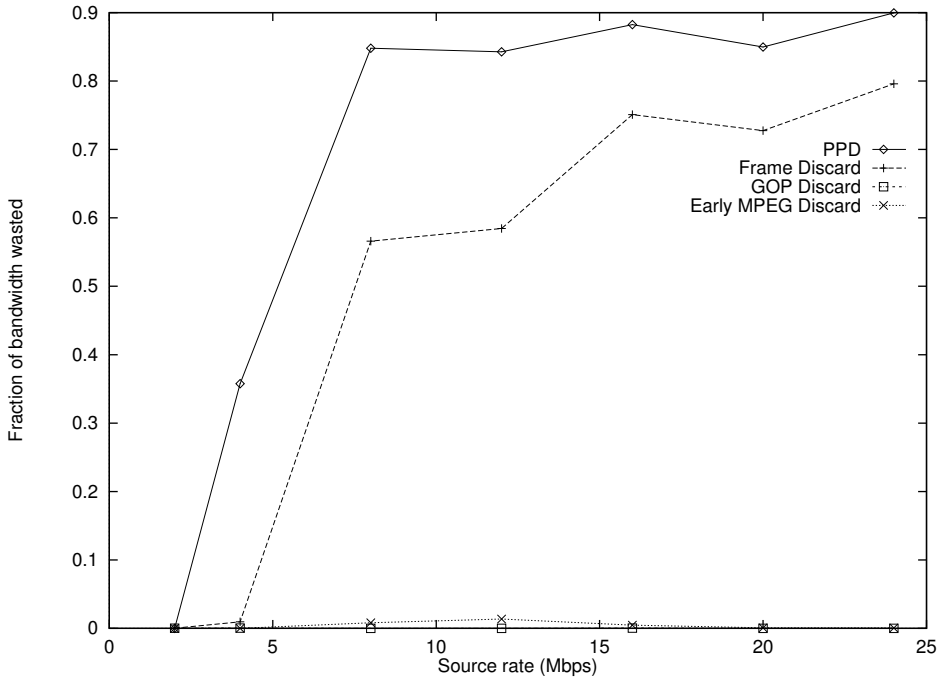

Figure 2: Fixed Source: Wasted Data Received

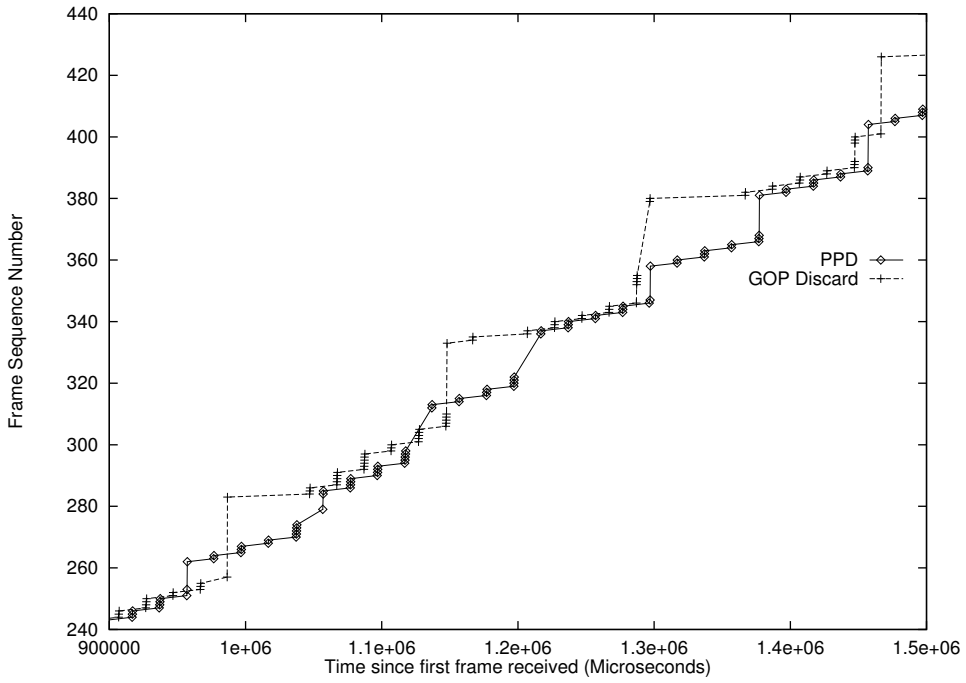

Figure 3: Fixed Source: Timing of Received Datagrams

In Figure 2 we plot the fraction of capacity wasted by sending bytes that are not useful at the receiver. This reveals that PPD and Frame Discard make extremely ineffective use of the link bandwidth, transmitting large amounts of data that the receiver will not be able to use. The GOP Discard and EMD mechanisms are both able to keep the wasted bandwidth negligible, even under heavy source load. The slight increase in wasted data for EMD under moderate source rates is likely due to the activity of other users on the destination machine.

Considerable detail about the system behavior over time is represented by Figure 3 which plots the time that datagrams with sequence numbers from 240 to 440 are received. For simplicity, we include only the PPD and GOP Discard results at a source rate of 8 Mbps. Note that one can easily discern the points at which GOPs are discarded, corresponding to a (nearly) vertical line segment indicating a range of unreceived datagram sequence numbers. PPD contains periods of dropping that occur more frequently, but tend to last for a shorter period of time.

6.3 Sources with Feedback

We next consider sources that adapt their rate based on end-to-end feedback. Two questions are of interest: (1) how does end-to-end feedback *compare to* active congestion mechanisms and (2) how does end-to-end feedback operate *in conjunction with* active congestion mechanisms.

The source implements the flow control over UDP described earlier in section 5. We

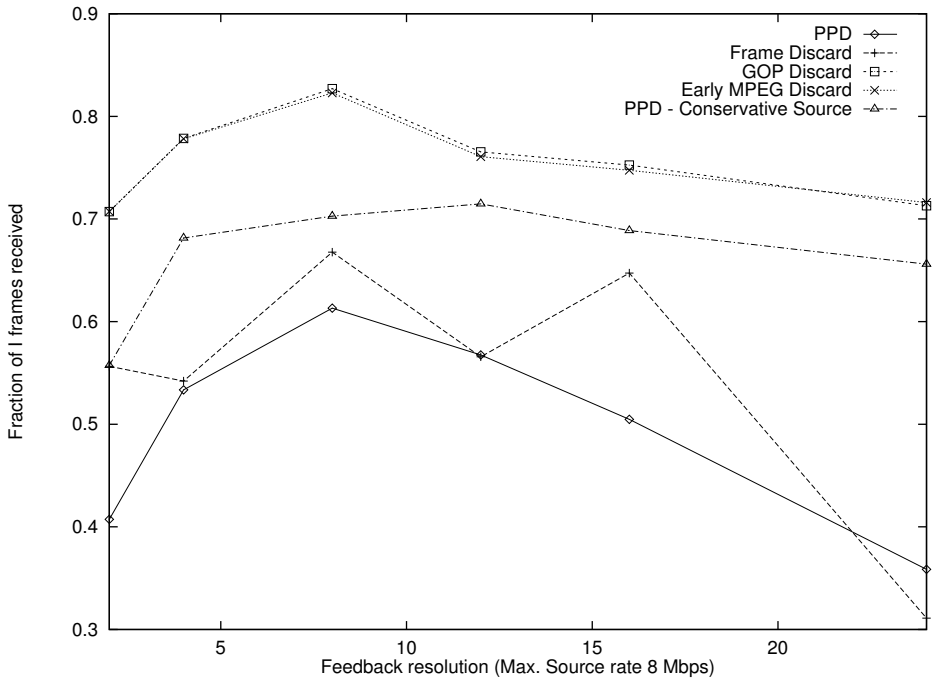

Figure 4: With Feedback: Useful Data Received

initialize the source rate to 1 Mbps and constrain the source rate to a maximum of 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. Thus, the number of frames that must be received without error before the source increases its rate is $F\lfloor 40/F \rfloor$. Figure 4 shows the fraction of I-frames received as the feedback resolution F is varied from 2 to 32 frames.

In general, the number of I-frames received decreases as the feedback interval increases. This is because the source is better able to match the link rate when it gets feedback more often. This behavior does not hold when the feedback interval is small. An examination of the source behavior shows that the source is oscillating between a rate which slightly overloads the link (causing drops) and a rate that slightly underloads the link. We would expect about half of the frames to be received; the more sophisticated discard methods do a better job of insuring that the frames received are I-frames.

The Frame Discard method exhibits interesting behavior, with performance that varies widely and non-monotonically. Closer examination of the data reveals that when the feedback resolution is 12 frames, the average throughput rate is higher than with resolution of 8 or 16 frames. (For example, a representative run at resolution 12 had average throughput 3.0 Mbps, while the througput was 2.87 Mbps and 2.84 Mbps respectively for resolution 8 and 16.) The source is more aggressive and the router tends to have a relatively full buffer, thus while smaller P and B-frames may fit, larger I-frames are often discarded for lack of buffer space.

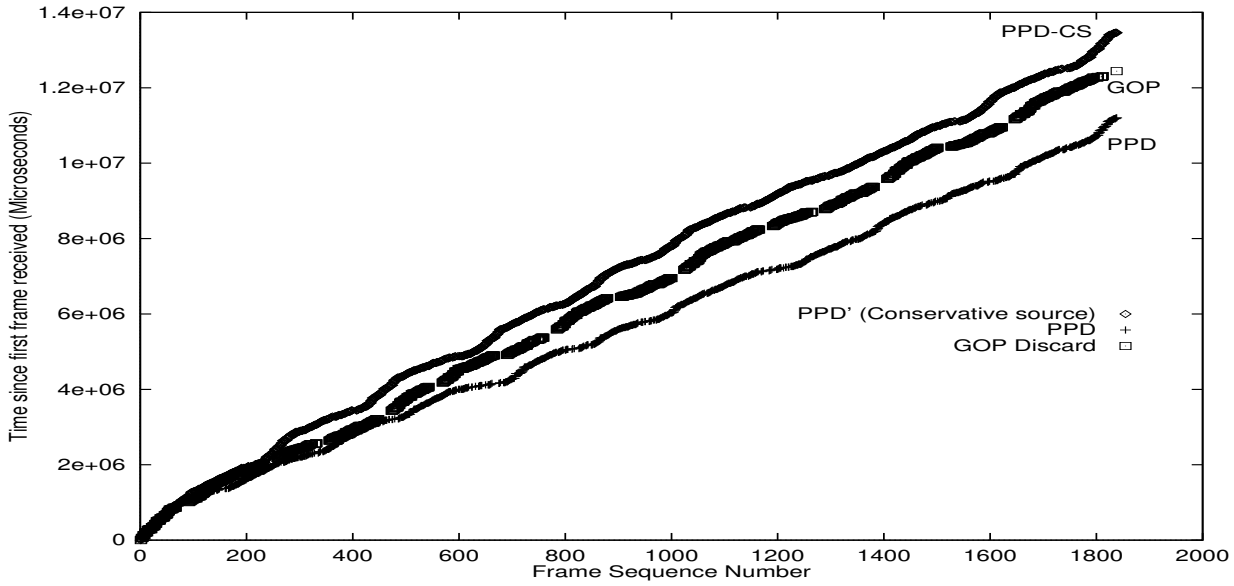

Figure 5: With Feedback: Timing of Received Datagrams

All of the methods do a better job of getting I-frames through when the active mechanisms are used in conjunction with end-to-end adaptation with a moderate feedback resolution. Even with coarse feedback (e.g., $F = 24$) the source adaptation in combination with the GOP Discard and EMD mechanisms is sufficient to maintain over 70% of I-frames received.

We can compare end-to-end adaptation (operating alone) to active congestion mechanisms (operating alone) by comparing the PPD curve in Figure 4 to the performance in Figure 1 when the source rate is 8 Mbps. The maximum performance for PPD is approximately 60% of I-frames received, while the GOP Discard and EMD mechanisms achieve about 65% of I-frames received. PPD requires frequent feedback to achieve this result; further, the latency between receiver and sender in our experimental setup is quite small. Less frequent feedback or longer delay on the feedback path are likely to degrade PPD performance.

Figure 5 gives the timing for the datagrams received, for the GOP Discard mechanism and two variations on PPD. In the basic PPD result, the source adaptation parameters are identical to the adaptation for the GOP Discard mechanism. The feedback (and thus the instantaneous source rate) differs, of course, since the PPD and GOP Discard drop packets differently. As a result, the PPD transmits data more aggressively and completes its transmission earlier than with GOP Discard.

To provide a more fair comparison, we change the reaction rate R of a PPD source (called the conservative source) to $\lfloor 80/F \rfloor$ where F is the feedback rate. Now, the conservative source is favored since it takes longer than the GOP Discard for transmission [7]. The PPD-CS curve shows the timing for the datagrams received with the modified source adaptation. The fraction of I-frames received does increase when the source transmits more conservatively, as indicated by the PPD-Conservative Source curve in

[7] In a truly fair comparison, both the PPD and GOP schemes would take the same total amount of time. However, this is difficult to achieve since the total transmission time cannot be precisely controlled.

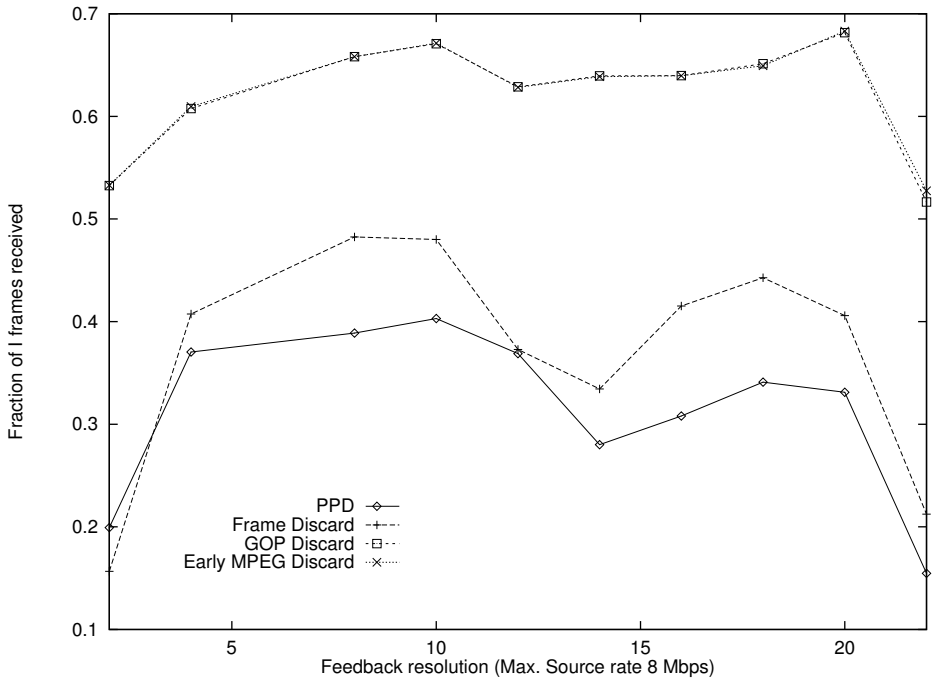

Figure 6: With Feedback and Background: Useful Data Received

Figure 4.

6.4 Effect of Background Traffic

When background traffic is added, the sources see a more congested link. Figure 6 shows the fraction of I-frames received for each mechanism, with source feedback and background traffic generated from the traces described earlier. The feedback interval is varied on the x-axis. Note that the more sophisticated congestion mechanisms perform even better relative to the PPD and Frame Discard under congestion. The more congested the link, the greater the value in having controlled losses. The PPD and Frame Discard schemes exhibit similar non-monotonic behavior, caused by the fact that some feedback resolutions result in an aggressive source, as noted earlier.

A similar result can be observed for sources without adaptation; the fraction of I-frames received drops more sharply for PPD and Frame Discard under congestion.

6.5 Let's All AN

Finally, we consider a "friendly" environment in which the background traffic is subject to the EMD mechanism. The background traffic used for this example is MPEG streams. We again vary the treatment of the the source traffic.

Figure 7 shows the fraction of I-frames received when the source operates at a fixed rate, given on the x-axis. The performance of all of the methods is the best seen so far,

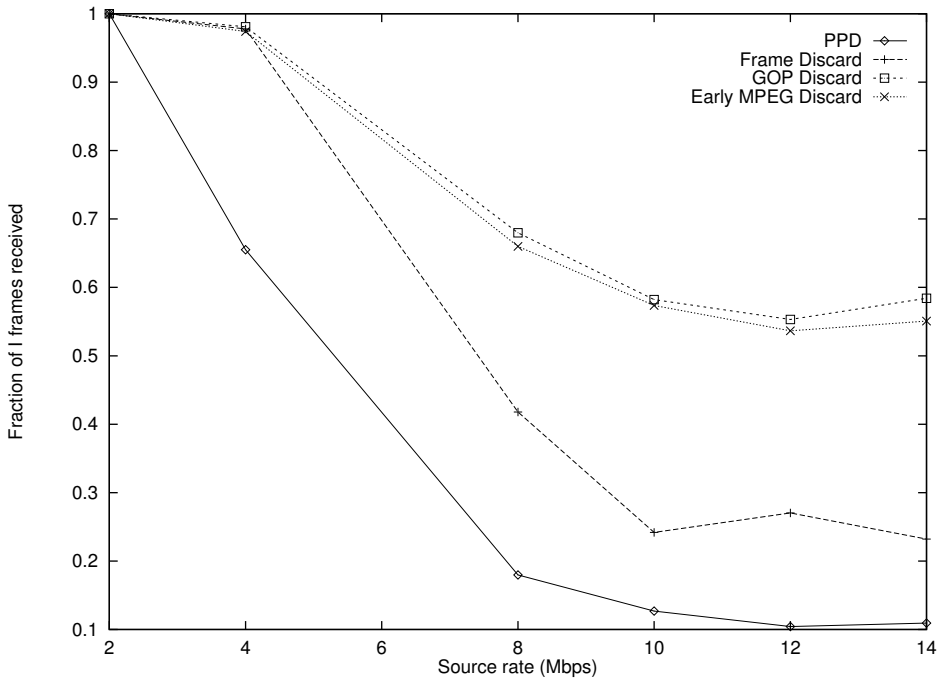

Figure 7: Fixed Rate and EMD Background: Useful Data Received

since the background traffic is controlled by the EMD mechanism. The GOP Discard and EMD mechanisms achieve nearly 60% of I-frames received, even when source rate increases. PPD shows little improvement and Frame Discard eventually perform poorly, though they degrade more slowly when the background traffic is well behaved.

In Figure 8 we combine all components: the sources adapt based on end-to-end feedback and the background traffic is subject to EMD processing. The PPD and Frame Discard mechanisms actually improve more than the GOP Discard and EMD mechanisms since they have more to gain when the background traffic is well-behaved. Note that the GOP Discard and EMD mechanisms can achieve nearly 75-80% of I-frames received in this optimized environment.

7 Conclusions

The premise of active networking is that users can benefit from enhanced network functionality. We have presented a simple approach to AN in packet-switched networks, in which users can control the application of a set of supported network-based functions to their data. Our approach permits new functions to be developed and deployed over time, and is backward compatible in that not all users need be aware of the active functionality in the network, and not all nodes need support the same set of functions. In the context of this approach, we have implemented several intelligent discard functions, which support user control of service degradation in the face of congestion. We have

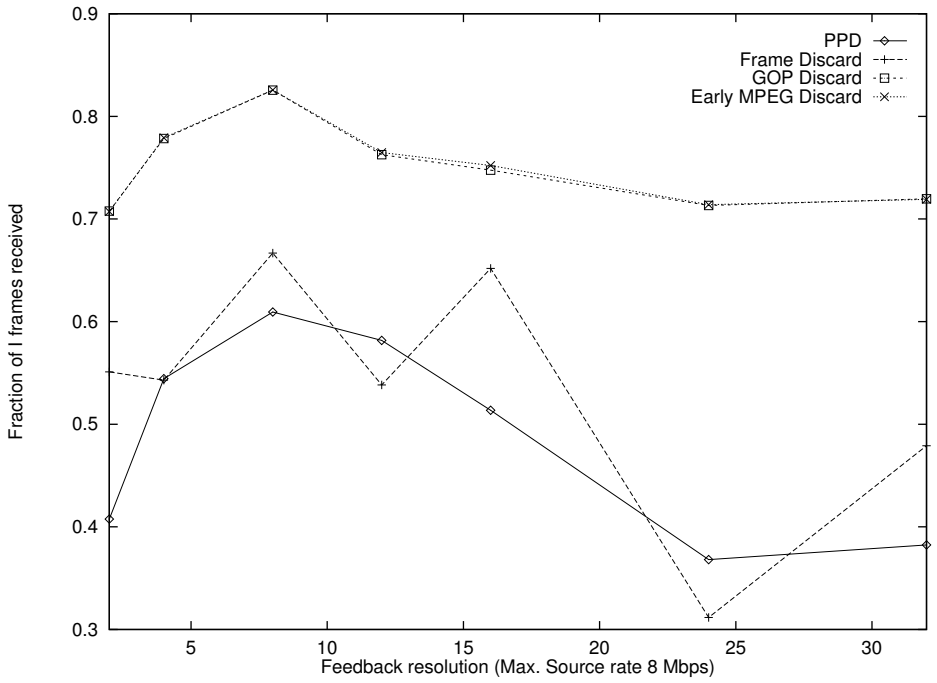

Figure 8: With Feedback and EMD Background:Useful Data Received

investigated the effectiveness of the discard functions under a variety of network and source behaviors. Our results show that for a certain class of applications —those whose application data can be classified according to utility, and which do not use a lower-level protocol for reliability— the intelligent discard mechanism can significantly improve the utility of data delivered to the receiver, even under severe congestion.

The processing and storage resource requirements of the intelligent discard mechanism discussed here are comparatively modest. Future work will investigate AN functions supporting other kinds of services (e.g. multicast, mobility) and their resource requirements. Other avenues for possible investigation include formal specification of an interface between AN functions and shared data structures (e.g. routing tables), and specification of an "AN API" (Application Programming Interface).

Appendix I – MPEG Discard Algorithm Details

Frame Header Structure We divide each MPEG frame into IP datagrams each carrying a maximum of 4096 bytes of user data. The MPEG data is prepended by a frame header of the following form – ⟨*frame sequence number, frame type (I, P, or B frame), offset, total frame length*⟩. For MPEG frames of size greater than 4096 bytes, the *offset* specifies the relative position of the current IP datagram from the beginning of the frame.

The Frame Discard Algorithms Given the frame header structure, we show an $O(1)$ implementation for each of the active processing functions described in Section 4.3. None of these functions require any arguments. It should be noted that in the algorithm discussions below, we assume that datagrams arrive in order[8], and that each program variable is local to each flow.

- **Partial Packet Discard** As mentioned before, the PPD algorithm does not decipher the frame level header.

  algorithm PARTIAL PACKET DISCARD
  $\forall$ active datagram d
  if Buffering of datagram overflows router memory
  Discard d
  else
  Enqueue d in output buffer
  end PARTIAL PACKET DISCARD

- **Frame Level Discard** In this algorithm, state is maintianed for each frame in form of either the *DiscardFrame* or *AcceptFrame* variable. Frames that cannot be accomodated in the output buffer are identified using the frame sequence number stored in *DiscardFrame* and each datagram corresponding to such frames are discarded. Also, a similar fast path is maintained for frames that have been accepted so that subsequent datagram may be accepted with only one check (of the *AcceptFrame* variable).

  algorithm FRAME LEVEL DISCARD
  $\forall$ active datagram d, **let** $h = \langle sq,\ type,\ offset,\ total\_len \rangle$ be the frame header
  if $AcceptFrame == h.sq$
  Enqueue d in output buffer
  else
  if $DiscardFrame == h.sq$
  Discard d
  else if Buffering of frame overflows router memory
  $DiscardFrame \leftarrow h.sq$; Discard d
  else
  $AcceptFrame \leftarrow h.sq$; Enqueue d in output buffer
  end FRAME LEVEL DISCARD

  The frame level discard algorithm is initialized for each flow by setting the value of
  $AcceptFrame$, and $DiscardFrame$ to null.

- **Group of Picture Level Discard** We augment the frame level discard algorithm to maintain state about the type of frame discarded. The $IFrameDropped$ variable indicates whether the I frame for the current GOP has been dropped, and if true, then entire GOP is dropped.

[8] The algorithms can be extended to maintain enough state to handle out of sequence datagrams.

algorithm GOP LEVEL DISCARD
$\forall$ active datagram d, **let** $h = \langle sq,\ type,\ offset,\ total\_len \rangle$ be the frame header
if $AcceptFrame == h.sq$
Enqueue d in output buffer
else if $DiscardFrame == h.sq$
Discard d
else if $IFrameDropped == True$
if $h.type \neq I$
$DiscardFrame \leftarrow h.sq$; Discard d
else
$IFrameDropped \leftarrow False$
else if Buffering of frame overflows router memory
$DiscardFrame \leftarrow h.sq$; Discard d
if $h.type == I$
$IFrameDropped \leftarrow True$
else
$AcceptFrame \leftarrow h.sq$; Enqueue d in output buffer
end GOP LEVEL DISCARD

The GOP level discard algorithm is initiated by setting the values of $AcceptFrame$, and $DiscardFrame$ to null, and $IFrameDropped$ to $False$.

- **Early MPEG Discard** The early MPEG drop algorithm is identical to the GOP LEVEL DISCARD except for the addition of one more clause that initiates the *early* discard if the fraction of memory exhausted is greater than a certain predefined fraction ($Threshold$) of total memory.

algorithm EARLY MPEG DISCARD
$\forall$ activedatagram d, **let** $h = \langle sq,\ type,\ offset,\ total\_len \rangle$ be the frame header
...
if Buffer occupancy is greater than $Threshold$
if $type \neq I$
$DiscardFrame \leftarrow h.sq$; Discard d
else
if $DiscardFrame == h.sq$
...
end EARLY MPEG DISCARD

The initialization is identical to the group level discard algorithm.

8 Acknowledgements

The trace data from Georgia Tech was collected by Steve Klivansky and Scott Register. The trace data from the San Diego Supercomputer Center was collected by Hans Werner-Braun. Amarnath Mukherjee provided us with access and expertise regarding the traces.

References

[1] G. Armitage and K. Adans. Packet reassembly during cell loss. *IEEE Network Magazine*, September 1993.

[2] Ran Atkinson. IP authentication header. *RFC 1826*, August 1995.

[3] Ran Atkinson. IP encapsulating security payload. *RFC 1827*, August 1995.

[4] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura. On active networking and congestion. Technical Report GIT-CC-96-02, College of Computing, Georgia Institute of Technology, 1996.

[5] F. Bonomi and K. Fendick. The rate-based flow control framework for the available bit rate ATM service. *IEEE Network Magazine*, March/April 1995.

[6] E. Brinksma. An introduction to lotos. In *Proc. 7th IFIP WG 6.1 Workshop on Protocol Specification, Testing and Verification*, 1987.

[7] CCITT. Recommendations z.101 to z.110. Blue Book, 1988.

[8] A. Eleftheriadis and D. Anastassiou. Constained and general dynamic rate shaping of compressed digital video. In *IEEE International Conference on Image Processing*, Washington, D.C., October 1995.

[9] D. Feldmeier. A data-labelling technique for high-performance protocol processing and its consequences. In *ACM SIGCOMM '93*, 1993.

[10] International Organisation for Standardisation. Generic coding of moving pictures and associated audio. ISO/IEC/JTC1/SC29/WG-11, March 1993.

[11] ITU-T. ISO DIS 10918-1 Digital compression and coding of continuous-tone still images (JPEG). Recommendation T.81.

[12] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, 1988.

[13] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of Ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, pages 1–15, February 1994.

[14] N. Likhanov, B. Tsybakov, and N. Georganas. Analysis of an ATM buffer with self-similar (fractal) input traffic. In *IEEE Infocom '95*, 1995.

[15] S. McCanne and M. Vetterli. Joint source/channel coding for multicast packet video. In *IEEE International Conference on Image Processing*, October 1995.

[16] D. G. Morrison, M. E. Nilsson, and M. Ghanbari. Reduction of the bit-rate of compressed video in its coded form. In *Sixth International Workshop on Packet Video*, Portland, OR, September 1994.

[17] A. Romanow and S. Floyd. Dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas in Communications*, May 1995.

[18] H. Sun, W. Kwok, and J. Zdepski. Architectures for MPEG compressed bitstream scaling. In *IEEE International Conference on Image Processing*, Washington, D.C., October 1995.

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

[20] W. Willinger, M. Taqqu, R. Sherman, and D. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. In *ACM Sigcomm '95*, pages 100–113, 1995.