An Architecture for Active Networking

Samrat Bhattacharjee, Kenneth L. Calvert, Ellen W. Zegura
Networking and Telecommunications Group
College of Computing, Georgia Institute of Technology,
Atlanta, GA 30332.
email: {bobby,calvert,ewz}@cc.gatech.edu

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 users 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.

Keywords

active networks, congestion control, MPEG, early packet discard

1 INTRODUCTION

As the cost of computing power decreases, it is worthwhile to consider the benefits of adding computing to various 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*. 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*,

*In this paper we focus on packet- and cell-switched networks. We follow Tennenhouse and Wetherall [7] in adopting the term active networking; the differences between our approach and theirs are elucidated later.

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.

2 AN ARCHITECTURE FOR ACTIVE NETWORKING

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. 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 the datagram 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.

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.

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 [3], SDL [4], 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 this 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 imple-

ment the functionality in a 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.

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 [7] 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.

2.3 An AN Implementation

We have implemented several AN functions in an IP routing architecture. We defined an "Active Processing" IP option `IPOPT_AP`, which specifies a particular AN function in IP routers. The `IPOPT_AP` option is a tuple of the form $\langle f, m, \alpha \rangle$. The identifier 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 simulate congestion by defining a "virtual router memory limit." 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.

3 PROGRAMMABLE CONGESTION CONTROL

3.1 Operating Model

A *flow*, for our purposes, is a sequence of packets all having the same source and destination, from the point of view of a node somewhere in the network. Thus a flow might consist of packets traveling between a single pair of endpoints, or it might be the aggregation of a set of lower-level flows. We assume that a flow is identified by a label of some kind in the network protocol header.

Generically, programmable congestion control operates as follows: *Based*

on triggers that indicate congestion control should take place, flow state is examined for advice about how to reduce quantity of data.

The important components of this generic model are the *triggers* responsible for initiating congestion control, the *flow state* that contains the specific advice for this flow, and the *reduction techniques* defined by the network and made available to the users. An important feature of this model is its consistency with traditional best-effort service. That is, a flow provides *advice* about what to do with its data. The network node is not required to take the advice, and may apply generic bandwidth reduction techniques.

In its most powerful and general form, congestion control might reduce bandwidth requirements by applying *transformations* to reduce the quantity of data at a congestion point. Such transformations will be application-specific and computationally complex (not to mention the significant issues they raise regarding interaction with end-to-end mechanisms such as error detection and sequence numbering). We therefore focus on the special case of *intelligent discard* of data. Using a natural extension of packet-level discard [1], we allow applications to define units based on application semantics, with aim of discarding the entire unit if any portion must be discarded. It should be noted that for this scheme to be most effective, buffering is required to collect application data units to ensure that the "tail" will not be dropped. Further, the end-to-end reliability mechanisms (if any) affect the optimal definition of a unit. 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 unnecessary retransmission.

Given that bandwidth reduction will occur by discarding units, a question arises as to which unit (within a flow) to discard. In the most simple case, there is no choice: when the congestion indication trigger occurs, a fixed unit (typically the one currently being processed) is subject to discard. More efficient network behavior is possible, however, if we expand the set of data from which we choose to discard. We consider congestion control advice that indicates how to make such a choice. One can think of this advice as indicating priority or some other policy by which to discriminate across data in the same flow. Making use of this advice clearly requires that the network node have access to a collection of data within a single flow. Thus, these mechanisms will involve storing and manipulating flow data before it leaves the node, e.g., while sitting in a per-flow queue from which packets are periodically selected for output by a scheduling mechanism.

3.2 Example Application and Mechanisms

To illustrate specific mechanisms and evaluate performance, we focus on the use of congestion control advice to improve the best-effort service provided to 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.

The specific components of the programmable congestion control are implemented as follows:

- **Source-attached advice.** We consider mechanisms in which the source identifies "units" such that the unit will be discarded if any portion of the unit must be dropped.
  The **Partial Packet Discard** mechanism is for baseline comparisons; it defines each IP fragment to be a unit. Fragments are discarded if they cannot be buffered in the output queue.
  Our **Frame Level Discard (EMD)** 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. We maintain state for each frame that is being discarded or buffered, and use this state information to decide, in constant time, to buffer or discard a particular datagram.
  We further consider a mechanism that identifies dependencies between units. Our **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.
- **Choice among units.** We consider one policy for making choices amongst units. When an I frame is too large to be accommodated in the output queue, and the queue contains P and B frames such that their combined sizes are greater than that of the I frame, then such P, B frames are discarded, and the I frame transmitted.
- **Triggers.** Finally, we consider two types of triggers. In the first, we detect and respond to congestion only when data arrives that cannot fit in the output queue. All three mechanisms mentioned above use this trigger. We also consider an "early" trigger, following the Early Packet Discard of Romanow and Floyd [6]. This trigger detects and responds to congestion when the output queue occupancy exceeds a certain threshold. Our **Early MPEG Discard** mechanism combines the GOP Level Discard with an early congestion trigger.

4 EXPERIMENTAL METHODOLOGY

4.1 Topology and Data streams

The experimental topology is shown in Figure 1. The bandwidth of the link between the source and the router, and between the congestion producing host

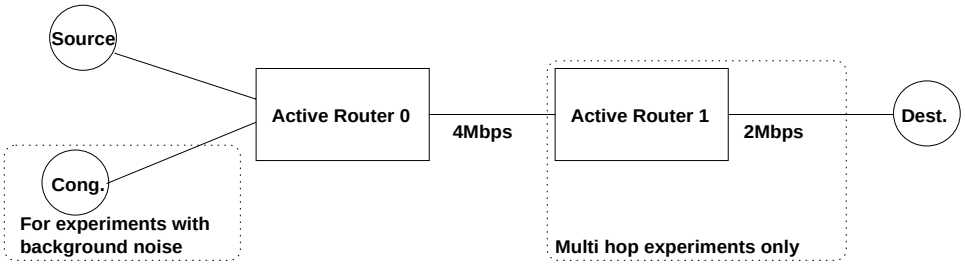

Figure 1 Experimental topology

MPEG stream	Avg. Frame (bytes)	Avg. I Frame (bytes)	Avg. GOP (bytes)	Avg. Y SNR
B0	8177	20222	81782	21.00
F0	9256	36755	92568	23.05
M0	8970	36272	89713	23.86
T0	4174	18623	41750	25.60

Table 1 MPEG stream statistics

and the router is varied in the experiments. The source host runs Solaris 2.5.1. All other hosts run SunOS 4.1.3; the routers executing our modified SunOS 4.1.3 kernel. The physical connectivity is provided by 155 Mbps ATM links. The output queues of the routers 0, 1 have 48K, 64K of memory, respectively.

Four different MPEG streams were subjected to the same tests. Each stream consisted of 120 frames in 12 GOPs. Particulars about the streams are provided in Table 1. For experiments with background traffic, the routers were congested by replaying packet traces with average cumulative bandwidth of 2.15 Mbps.

4.2 Metrics

We evaluate the results of our experiments using the following criteria.

- **SNR of received stream.** Under congestion, IP packets carrying the MPEG data may be discarded at intermediate routers. We evaluate the received stream by computing the signal-to-noise ratio (SNR) of the received stream – 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 YUV files which were used to create the transmitted stream.

- **Fraction of I Frames received.** The fraction of I frames that are received provide a good measure of the quality of the resultant MPEG stream.
- **Fraction of bytes that were eventually discarded at the receiver.** Due to the inter-dependent frame structure of MPEG, frames can only be decoded if all other frames that they depend on have also been received. We measure the fraction of data that is received but cannot be used due to incomplete frames, or missing related frames.

4.3 Destination Feedback Mechanism

Our active mechanisms can be used in conjunction with 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 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. 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.

5 RESULTS

5.1 Fixed Rate Sources

In Figure 2, we consider the simplest case of a single congested node, a fixed rate source, and no feedback from the destination. On the x-axis, we vary the fixed source rate from 2 Mbps to 24 Mbps, corresponding to increasing overload on the 4 Mbps link between the router and the destination. On the y-axis we plot the average value of the signal-to-noise ratio for the Y component of the MPEG data. Each curve corresponds to a different trigger/advice mechanism, as described in Section 3.2.

This plot is primarily included so that the reader has a baseline to keep in mind when we consider other combinations of feedback, background traffic and multi-node experiments.

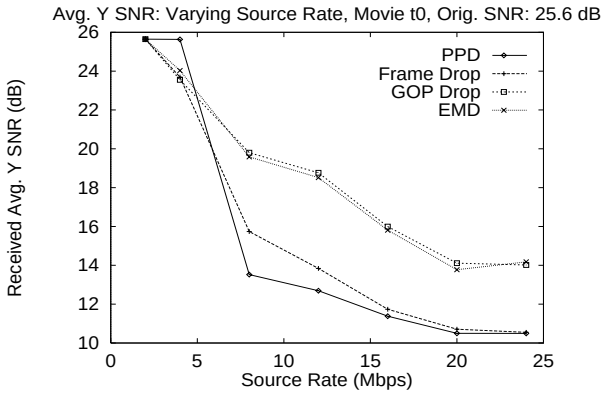

Figure 2 Avg. Y SNR at destination, with fixed source rate

5.2 Sources with Destination Feedback

We consider cases in which the destination provides feedback to the source as described in section 4.3. The end nodes implement the flow control algorithm with the following parameters. 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.

In Figure 3, the GOP level discard and EMD mechanisms consistently maintain an average SNR above 21 dB, while transmitting nearly 80% of the I-frames (Figure 4). No data transmitted via GOP level discard, or EMD is discarded at the receiver (Figure 5). Due to oscillations in the source rate, and indiscriminate discard at the intermediate router, the average SNR maintained by the PPD algorithm is 5—10 dB less in all cases (compared to GOP discard, and EMD). Both the PPD and the Frame level discard algorithm transmit data that is discarded at the receiver, thereby wasting network resources. The Frame level discard algorithm only transmits complete frames, thus, the data discarded at the receiver is due to the transmission of undecodable frames. In case of PPD, data is also discarded due to receipt of partial frames*.

5.3 Addition of Background Traffic

The addition of background traffic leads to further congestion of the output link at the router. Under this scenario, with choice enabled, the GOP level

*The MTU for our experiments was 4096 bytes. The losses due to individual frame fragmentation would be greater if the MTU had been 1500 or 576 bytes.

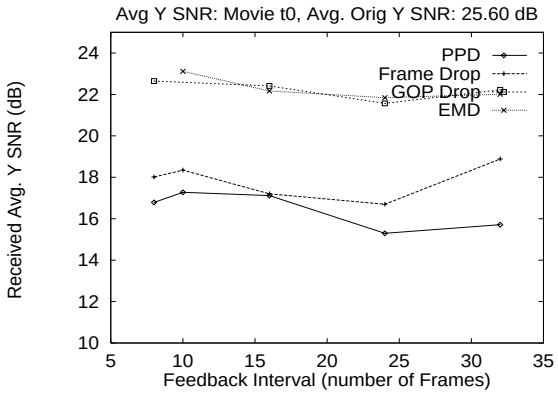

Figure 3 Avg. Y SNR at destination providing feedback

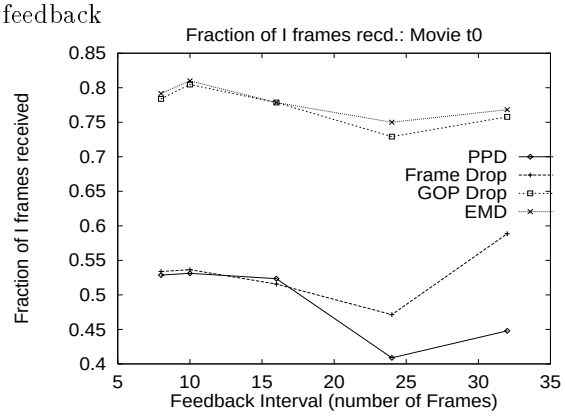

Figure 4 Fraction of I frames received at destination

discard and EMD mechanisms are able to maintain an average SNR of 19—22 dB (Figure 6). Under the same circumstances, the average SNR maintained by PPD varies between 10—17 dB. The GOP discard and EMD mechanisms are able to transmit 70—80% of the I frames, while 5—50% of the I frames are transmitted using PPD (not shown). Between 30 and 95% of the data transmitted by PPD is discarded at the receiver. Interestingly, a larger fraction of data transmitted by the Frame level discard algorithm is discarded in this case (with background traffic) than without (not shown). This is due to the smaller amount of buffering available to the MPEG stream (because of the background traffic). Thus, the choice policy has fewer frames to work with, and in many cases P and B frames are transmitted, as discarding them would

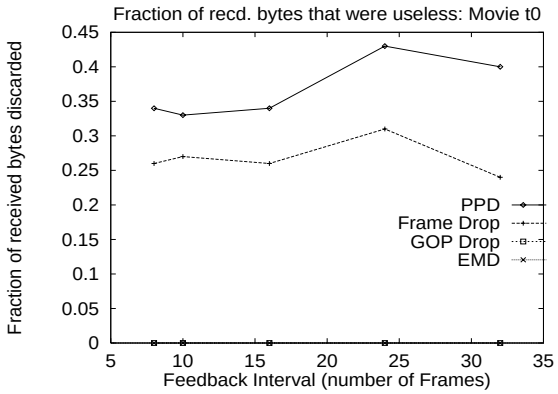

Figure 5 Fraction of received bytes discarded at destination

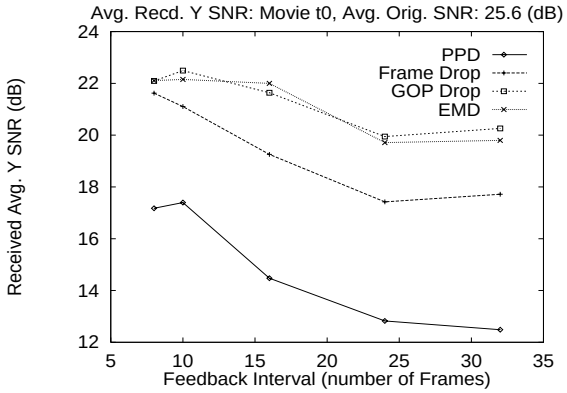

Figure 6 SNR at dest. (with background traffic, and feedback)

not create enough space to transmit another I frame. Note that in these cases, the GOP level discard and the EMD mechanisms do not transmit the P and B frames, and thus conserve network resources.

5.4 Multi-Node Experiments

We extend the topology to include one additional router. Figure 7 shows the fraction of I frames received after the MPEG stream encounters two congested routers. The output link of the first router is restricted to 4 Mbps, and the

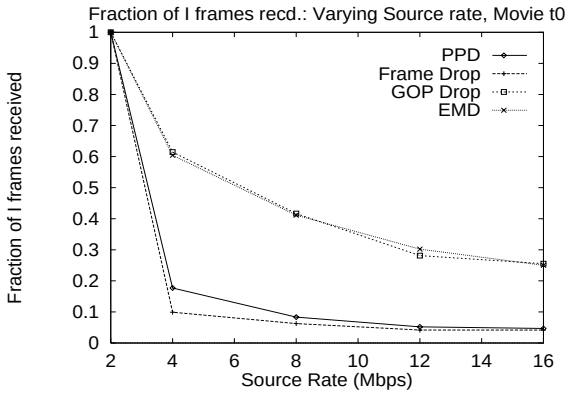

Figure 7 Fraction of I frames received (multi-AN hops, no feedback)

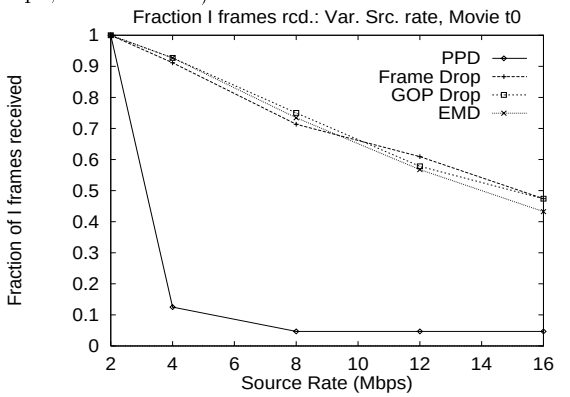

Figure 8 Fraction of I frames received (multi-AN hops, Choice policy enabled)

output link of the second router is restricted to 2 Mbps. As expected, 30—50% more I frames were received using GOP level discard, and EMD. A similar increase was noted for the received signal level. The drop in fraction of I frames transmitted (and SNR) is more gradual for the GOP level discard, and EMD mechanisms. It should be noted that between 70—80% of the data transmitted by the Frame level discard, and PPD mechanisms were eventually discarded by the receiver.

When we add the choice policy to the same scenario (Figure 8), the mechanisms other than PPD transmit a higher fraction of the I frames. A similar increase is noted in the received signal level as well. The greatest impact is on the Frame level discard algorithm — which in some cases, can transmit

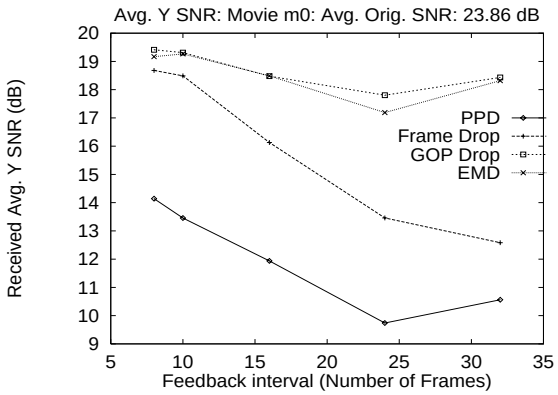

Figure 9 Avg. Y SNR at destination (multi-AN hops, dest. feedback)

over 80% more useful data using the queue manipulation techniques. Also, less than 10% of data transmitted by the Frame level discard algorithm is eventually discarded at the receiver.

In Figure 9 we consider a multi-node experiment, with destination feedback, with the choice policy enabled. Within favorable feedback intervals (feedback per 4-12 frames), the mechanisms other than PPD sustain an average SNR of 18–19 dB. Using the same feedback mechanism, the average SNR maintained by the PPD mechanism varies between 7—14 dB. Once again, more than 80% of the data transmitted by PPD is discarded at the receiver.

5.5 Detailed Evaluation of Transmitted Streams

(a) Time varying SNR

In Figure 10, we consider a specific example, and evaluate the received data on a frame-by-frame basis. The destination provides feedback (as described in section 4.3) every 10 frames, and the stream passes through two routers. The plots above the x-axis represent the SNR of the received frames. The points below the x-axis represent indices of complete frames received. Under GOP discard, 163 of 220 total frames are received, while under PPD, 148 total frames were received. However, the GOP discard mechanism was able to transmit 22 of the possible 23 I Frames, while PPD was able to transmit only 12 I Frames. From Figure 10, it is clear that PPD is susceptible to periods of catastrophic losses (as seen in indices [60–120], and [160–230]). During the same period, GOP discard experiences heavy frame losses as well; but is able to transmit the crucial I frames (except for frame index 100), thus preserving some picture quality. This leads to a visibly graceful degradation of the signal (indices [90–110]) – and recovery from signal loss is much quicker than PPD (e.g indices [190–210]).

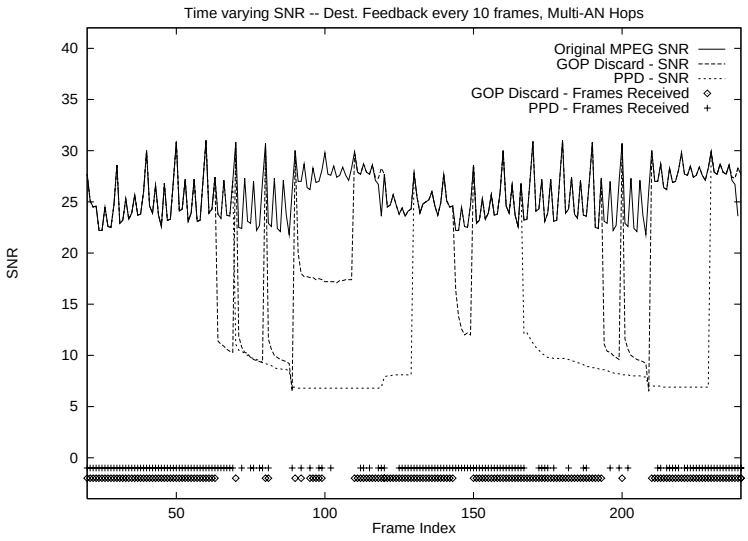

Figure 10 Per Frame SNR of received stream

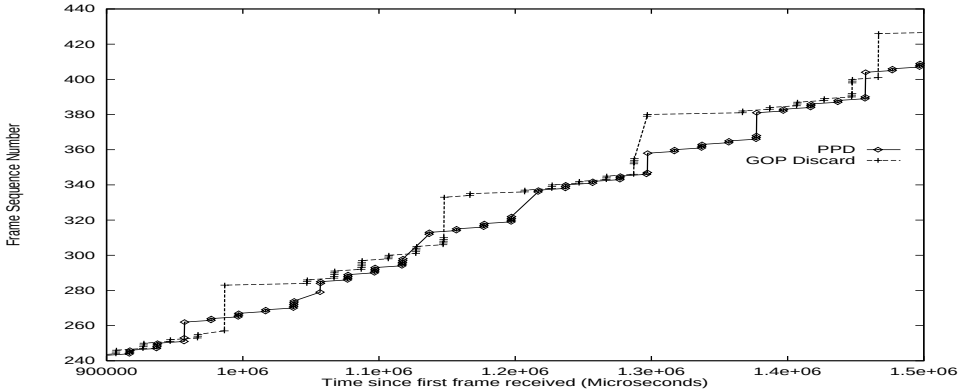

Figure 11 Fixed Rate Source: Timing of received datagrams

(b) Transmission Latency

In Figure 11, we consider the delay added by the processing. The source rate is set to 8 Mbps. Considerable detail about the system behavior over time is represented by Figure 11 which plots the time elapsed (in microseconds) since the first datagram is received for datagrams with sequence numbers from 240 to 440. 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 discard that occur more frequently, but tend to last for a shorter period of time. An important point to note is that the mechanisms do not introduce any additional buffering delay (compared to PPD). As IP packets are pro-

cessed, an $O(1)$ check is made whether this packet can be transmitted, or not. Frames, or GOPs, are not reassembled in the routers; this would have introduced buffering delays. Delays, if any, are introduced during the computation phase of the packet acceptance algorithm. Under the choice policy, this computation time may include time taken to search the queue. The search time is $O(Output\ Queue\ Size/Packet\ Size)$. However, the only packets that encounter this delay are packets that would otherwise been discarded. This is because the choice is only activated when a packet — specifically an I frame — cannot be accommodated in the output queue.

6 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.

7 ACKNOWLEDGEMENTS

The trace data used was collected by Steve Klivansky, Scott Register, Hans Werner-Braun, and Amarnath Mukherjee. David Haynes implemented the signal-to-noise ratio computation.

REFERENCES

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

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

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

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

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

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

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