

On Active Networking and Congestion

Samrat Bhattacharjee

Kenneth L. Calvert

Ellen W. Zegura

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

GIT-CC-96/02

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 and security. Less clear, perhaps, are the “immediate” benefits of such a paradigm shift, and how they might be used to justify migration towards active networking.

In this paper, we focus on the benefits of active networking with respect to a problem that is unlikely to disappear in the near future: network congestion. In particular, we consider application-specific processing of user data within the network at congested nodes. Given an architecture in which applications can specify intra-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. We consider a range of schemes for processing application data during congestion, including “unit” level dropping (a generalization of packet dropping), media transformation, and multi-stream interaction. We also present some architectural considerations for a simple approach, in which packets are labeled to indicate permitted manipulations. Our results suggest that congestion control makes a good case for active networking, enabling schemes that are not possible within the conventional view of the network.

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

1 Introduction

1.1 Active Networking and Congestion

Active networking offers a change in the usual network paradigm: from passive carrier of bits to a more general computing engine. In an active network, nodes (routers and switches) can perform computations on user data as it traverses the network. Further, applications are provided with a mechanism to select, and even program, the computation that occurs [20, 23]. In one proposal these two capabilities are integrated in a *capsule* entity: a combination of code and data that migrates through the network being executed at nodes [23].

The following examples have been cited as evidence that active networking technology is either needed or already exists in some form [23]:

- **Multicast routers**, which selectively duplicate packets before forwarding them on links.
- **Video gateways**, capable of transcoding video (i.e., changing from one representation format to another) as it passes from one part of the network to another [4].
- **Firewalls**, which selectively filter data passing into and out of an administrative domain.

In addition to these relatively incremental examples, wide-spread implementation of active networking is likely to enable radical new applications that cannot be foreseen today. Large-scale deployment is clearly not without its hurdles, however, with significant challenges in interoperability and security.

We assert that more immediate and widely useful benefits would help justify a migration towards active networking. In this paper, we focus on the benefits of active networking with respect to a problem that is unlikely to disappear in the near future: network congestion. In particular, we consider *application-specific* processing of user data within the network at congested nodes. Rather than applying congestion reduction mechanisms generically and broadly, we discuss an architecture that allows each application to specify how losses to its data should occur in a *controlled* fashion.

We note that congestion is a prime candidate for active networking, since it is specifically an intra-network event and is potentially far removed from the application. Further, the time that is required for congestion notification information to propagate back to the sender limits the speed with which an application can self-regulate to reduce congestion, or can ramp up when congestion has cleared. (One can argue quite legitimately that a gradual increase following congestion —as provided by slow start in TCP [13]— is good practice; an active network node can similarly implement gradual increase mechanisms.)

1.2 Characterizing Active Networks

Viewing the network as a computational engine admits a rather large space of possible approaches. In this section we consider some of the dimensions of this space.

As usual, the network is abstracted as a graph of nodes connected by edges. Users interact with the network by injecting units of data along with addressing information; the network conveys the data through the graph to the indicated destination. The basic tenet of active networking is that in addition to addressing information, the data may have associated information describing a *computation* to be performed using that data as some or all of its input. The class of possible computations may be characterized in several dimensions:

- **Computational power.** What class of functions can the network compute? Is the network Turing-complete? How much state is available for each computation?
- **Temporal locality.** If a computation may have state, how long does that state persist in the network? Can a computation span multiple data units?
- **Spatial locality.** Is the computation state local to a particular node, or is it distributed throughout the network?

As an example, at one extreme an active network might support arbitrary Turing-computable functions, distributed across the network, using state information that persists forever. At the other extreme, only a fixed, small set of finite-state computations can be performed on individual data units, one at a time.

The characteristics of any particular active networking approach with respect to these dimensions will necessarily be reflected in the interface between the user and the network, i.e., the *language* for specifying the computation to be performed. Many proposals for moving toward active networking tend toward the more general end of the scale [12, 23], with programs in a Turing-complete language attached to each packet. Our approach, described next, is more modest in scope, with the aim of offering a useful and feasible first step toward active networking.

1.3 Overview of Our Approach

In our approach, the network defines a finite set of functions that can be computed at an active network node. Computing these functions may involve *state* information that persists at the node, across the handling of multiple data units. Each function computation is spatially local, in the sense that state information does not move from node to node (unless it is carried in a data unit).

The interface between the user and the network is very simple: each data unit carries control information including a number identifying the function to be computed, plus a set of identifiers called *labels* that select the state information to be used (and possibly updated) in the computation. By assigning these labels to its data units in an appropriate way, the user can control the processing of its data units. As an example, the network may support a *unit-level drop* function: when one packet that specifies that function is dropped, for some time thereafter, every packet that matches some subset of its labels is also dropped. Thus, the definition of “unit” depends entirely on the labeling, and the user can control how much information is aggregated for dropping purposes.

In addition to being well-suited for functions dealing with congestion, this approach is backward-compatible: packets that don’t carry any active processing control information are not “actively processed”. On the other hand, packets that do carry the

information can be switched by non-active nodes, thereby allowing active and non-active nodes to interoperate in the same network. There are, however, situations where it is necessary for a switching node to be active-processing-aware; these are discussed later.

1.4 Roadmap

Following a discussion of related work in Section 2, we turn in Section 3 to a detailed description of our approach, including specific examples of active congestion processing. We have developed an environment for experimenting with active processing using existing network node technology (specifically, an ATM switch) with a workstation acting as an active processor. We describe this setup in Section 4 and use it in Section 5 to evaluate a variety of active processing techniques. Our schemes for processing application data during congestion include “unit” level dropping (a generalization of packet dropping), media transformation, and multi-stream interaction. In Section 6 we present a possible system architecture for an active node, including hardware and protocol architectures. We conclude with an assessment of our approach, including open issues and areas for future work. Our results suggest that congestion control makes a good case for active networking, enabling schemes that are not possible within the conventional view of the network.

2 Related Work

Tennenhouse and Wetherall [23] have outlined an architecture and set of issues for active networking. They propose an integrated approach in which *capsules* containing data and programs replace the traditional (passive) network packets. Much of their discussion focuses on the programming and security aspects of this architecture, including program scope, cross-platform execution of code, and resource usage. Active messages [24] are related to active networking in the sense that the messages contain both user data and information to specify processing at the receiver. (Specifically, each message contains an address of a user-level handler.) However, active messages are intended to optimize performance for a network of workstations or other relatively closely coupled set of processors, not a multi-hop, wide-area network.

Examples of processing that could be called “active” can be found in existing congestion control mechanisms. Recognizing the performance degradation caused by fragmentation [14], packet-level discard techniques have been explored to improve the performance of TCP/IP over ATM [5, 19]. The Partial Packet Discard strategy drops all cells in a packet that follow a dropped cell [5]; the Early Packet Discard strategy improves performance by aggressively dropping entire packets when a congestion threshold is reached [19].

The presence of a cell loss priority bit in the ATM cell header allows the source to indicate to the network that some cells should be treated differently than others under congestion. Separating traffic more finely into classes allows further specialization of treatment.

While dynamic congestion control is not an explicit goal of Amir et al.'s application level video gateway [4], their techniques for transcoding can have the effect of bandwidth reduction. Their focus is on accommodating an environment with heterogeneous transmission and endstation capabilities, by converting video from one representation format to another. An implementation of a JPEG [2] to H.261 [3] transcoder can reduce the bandwidth of a video stream from 6 Mbps to 128 kbps. Amir et al. further consider some temporal and spatial bandwidth reduction techniques to use in concert with format conversion.

3 Our Approach

3.1 Motivation

Our approach is motivated by several expectations. First, active networking, in its most general (and useful) form, will require substantial changes in network architecture. To move the network in the direction of these changes, active networking must offer some benefits, even with only a partial implementation of the architecture. (We assume that functionality will not be added to end systems unless there is some benefit in doing so, and similarly that switch manufacturers and network operators will not upgrade their switches to support active networking unless there is ultimately some benefit to their customers.)

Second, transmission bandwidth and computational power will both continue to increase, but so will application requirements for bandwidth. In particular, we expect that network node congestion will be due to bandwidth limitations, and that even congested switches will have considerable processing power (but not unlimited buffering) available.

Third, 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 bandwidth and applications will have no choice. Also, reserved bandwidth is likely to cost more. Finally, as computing speeds increase, so will a sending application's ability to trade processing for transmission bandwidth in reaction to congestion in the network.

At the same time, however, it must be noted that the sender-adaptation model [13], 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. (E.g., 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.

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.

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*. In the next subsection, we consider a computational model that is general enough to support this kind of processing—as well as many other functions—but simple enough to be feasible.

3.2 Computational Model

A distinctive feature of our approach is a simple but powerful “interface” to the active network’s computational resources, which consists of (i) a set of predefined computations that can be performed by nodes on user packets, and (ii) header information in each switched packet that specifies which computation is to be performed on it. We call this header information the Active Processing Control Information (APCI). Backward compatibility with existing network protocols is achieved by arranging that non-active nodes in the network need not recognize APCI in order to switch packets, and by not requiring APCI in packets switched by active nodes. In other words, APCI and its interpretation are to some extent orthogonal to and compatible with various architectures, as discussed in Section 6.2.

The APCI has two components: an *Active Processing Function Identifier* (APFI) and an *Association Descriptor*. APFI indicates which function the active network should compute for the data unit to which it is attached. The Association Descriptor consists of a fixed number of *labels*, which are simply fixed-size octet strings, and a *selector*. The number of label fields should be large enough to encompass the maximum number of levels of hierarchy that a single computation on the data unit would ever span. For example, in TCP/IP four labels should suffice: one for the connection or application data unit, one for the transport data unit (e.g. sequence number), one for the IP datagram, and one for the datagram fragment.

In general, the computation specified by the APFI involves access to state information stored in the active processor. The purpose of the Association Descriptor is to identify state information for the packet. State is stored in an associative memory; a computation may specify that certain information is to be *bound* to a *tag*, which is computed as some function of the current packet’s labels. The selector part of the Association Descriptor determines which function of the packet’s labels are used in com-

putting a tag. A computation may also *retrieve* the information, if any, previously bound to a given tag. In some cases the only information desired is whether anything has been bound to a given tag. However, the information bound to a tag could in principle be anything — even the data unit itself, thus providing a unit reassembly mechanism.

As a simple example of a computation, consider a function that passes only the *first* packet with a given label pattern, and drops all others:

```
if anything is bound to the tag consisting of this packet's labels then
    drop this packet;
else associate 1 with the tag consisting of this packet's labels;
```

By assigning labels appropriately, this “gating” function can be applied to individual packets within a stream, or to multiple streams.

Naturally, the active node will need to reclaim state storage at some point. One possibility is to age associations, and throw them away when they reach some maximum age. The assumption is that only functions that exhibit (temporal) locality of reference will be implemented. Moreover, the functions should be defined in terms of soft state, i.e. if the needed state is missing the function can recover. In all cases, the result of active processing should be no worse than what would be observed in a passive network.

A general model of what happens to a packet when it arrives at a node might be:

1. The output destination port for the packet is computed as usual.
2. If the packet contains valid APCI, it is sent to an active processor and processing continues; otherwise, it is transmitted as usual.
3. The function specified in the APCI is computed, using the packet's association descriptor and user data as inputs.
4. If the result of the function is transformed data (e.g. reduced length), the packet's network-level header and APCI are recomputed as necessary; the nodes's state is updated as required by the specified function.
5. The (possibly modified) packet is transmitted to its next-hop node.

If the deployment of active nodes is motivated primarily by congestion control, the above model is not optimal, because most supported functions will be congestion-control functions, which need not be invoked unless congestion is present or eminent. In that case the above model should be modified so that active processing is not performed except when a node is congested; in Section 6.1 we describe a hardware architecture for this modified model.

To summarize the features of our approach:

- The label-association mechanism is quite general, and can be used to make computations affect multiple streams, data units, or mixtures of both.
- Because the set of functions is fixed, it can be heavily optimized, implemented in hardware, etc.

- Although the *set* of supported functions is fixed, the functions themselves can be anything computable.
- The active networking “service” is *best-effort*, and the supported functions should be designed with this in mind; in the worst case, the service provided with active processing of packets should be no worse than what would be received without it.
- Because only specific network-supported functions can be invoked, security is not an issue.

3.3 Example Functions

Next we consider some of the functions that might be supported by an active processor (AP) for the purpose of controlling loss in the face of congestion.

Buffering and Rate Control The most direct “translation” of sender-based adaptation to active networking is to have the AP monitor the available bandwidth and rate-control the data, buffering it and metering it out at the available rate. The obvious question to ask about this function is why it isn’t better to simply put the additional buffering it requires into the switch instead of the AP. Our answer is that in the AP, this storage can be much more flexibly used. Many of the other processing functions discussed below use this basic capability as a building block.

Unit-Level Dropping. Unit-level dropping is a natural extension of packet-level discard [5, 19]. Units that are meaningful to the application are dropped if any portion of the unit must be dropped. Note that to be most effective, this does require buffering at the AP to collect a unit so that it is clear that the “tail” will not be dropped. Thus some additional latency and buffering at the AP are traded for improvements in useful delivery.

Media Transformation. Active networks can do more than simple intelligent dropping of data when congestion occurs. A particularly powerful capability is the *transformation* of data at a congestion point, into a form which reduces the bandwidth but preserves 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. In general, this reduced-bandwidth form is not obtained by simply dropping units from the higher-bandwidth form.

The importance of graceful degradation is well recognized, and indeed is supported by techniques such as layered encoding of images and video [2, 11, 16]. 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.

The media transformations that we have in mind may offer more fine-grained control over bandwidth and image quality than layered encoding. They could work well

in conjunction with layered encoding to tailor the transformed stream to the desired bandwidth. 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 [9, 22, 17], and decoding and recoding at higher quantization.

Multi-stream Interaction. As alluded to 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.

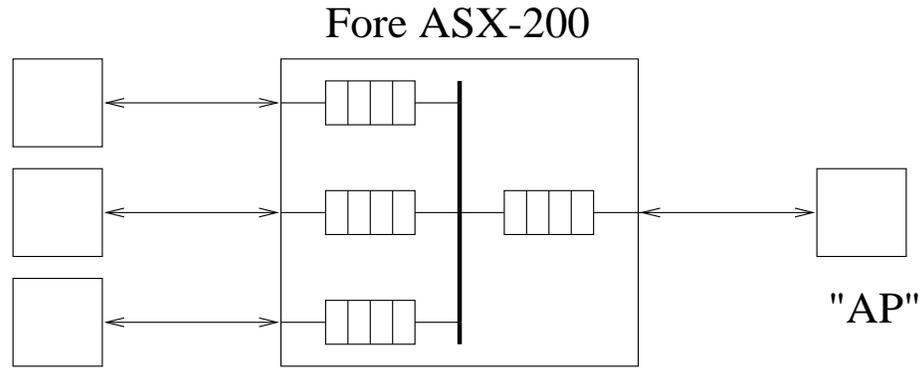
MPEG compressed video is separated into three streams, one for each type of frame (I, P, B). This would enable the quality of service specification for each stream to be tailored to the stream type; since the three frames differ significantly in their compression and importance, this could improve overall 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 Experimental Setup

4.1 Overview

We have emulated an active node hardware architecture (as described in Section 6.1) using a Fore ASX-200 ATM switch with four attached Sparcstations. Specifically, we have designated one of the endstations to be an active processing element, while using the other three endstations to act as sources and destinations of network traffic. We have controlled the transmission of data streams to allow selected diversion of streams



Endstations

Figure 1: Experimental Configuration

through the active processing element before being forwarded to the proper output. Figure 1 depicts this setup.

We note that our emulation has one limitation that may not be present in an integrated implementation of active processing with the network node hardware. This limitation is on the speed of the pipe into the processing node. In our emulation, the speed is restricted to the speed of the external link from the ATM switch to the processing node. Cells must traverse the output buffer before being transmitted over the fiber link to the processing node. In an integrated implementation, the active processor would likely have direct access to the cells in its buffer, allowing data transfer to the active processor at a faster rate than the external link rate. In a bus-based switch architecture, the active processor could receive data at n times the external link rate.

In order to calibrate/translate our results to those of an integrated implementation, we intentionally “slow” the links from the source to the switch and from the switch to the destination. Thus, the link into the active processor appears to be faster, relatively speaking.

4.2 Details

It is important to understand some of the operational details of the Fore ASX-200 switch [1], as they have constrained and dictated portions of our implementation.

Network Device Characteristics. Each VC is characterized at circuit setup time by peak and mean data rate, and burst capacity. Using these parameters, the Fore ASX-200 maintains an internal leaky bucket for the connection, marking or dropping non-compliant cells, depending on the configuration. These parameters directly translate to the amount of buffering the switch reserves for a particular VC. The SBA-200e NICs used in the experiments heed only the peak cell rate per VC, and always transmit all available data at the peak cell rate.

Source Characteristics. Given the network characteristics as defined above, it is possible to model a wide range of sources. In order to model congestion, we set our VCs with peak cell rate strictly greater than the mean cell rate. As the network interface always transmits at peak rate, this leads to cells being dropped at the switch.

In order to model interesting source characteristics, we control the source transmission rate through the use of a pair of parameters, p and *MaxWait*. After each network write, sources generate a random number x in the range $[0..1]$. If $x < p$, then the source generates another random number y in the range $[0..1]$, and waits (does not transmit) for $y \times \text{MaxWait}$ amount of time. By varying these probability parameters and the amount of wait time, we can model bursty sources of traffic.

5 Results

We have studied three forms of active processing: 1) unit-level dropping, where the units are contiguous data from the same stream, 2) MPEG frame and group-of-picture dropping, and 3) multi-stream interactions of both *conjunctive* and *disjunctive* forms. We calibrate the performance of these APs by comparing to performance with no AP at all, and also to a Null AP, which brings data into the active processor and immediately retransmits it to the receiver ¹.

With the exception of the Null AP, all of the APs we have considered conform to the same generic processing architecture. In the generic architecture, an AP accepts data from one or more sources. The AP maintains a working buffer for each source, used to assemble incoming data into application-specific units. The units will often comprise multiple packets or cells; the working buffer provides space for the assembly process. Once a complete unit has been assembled, processing is performed. If the processing results in any data, the AP transfers this to its output queue and rate controls the transmission into the switch, sending at the rate which the destination can accept data. Losses may occur at the AP output queue if the buffer overflows; however, the rate control prevents any losses from occurring in the switch between the AP and the destination.

5.1 Unit-Level Dropping

The *Unit-Level Drop* (ULD) AP is an example of the generic architecture, where the processing is null. The AP provides a buffer for the data, does unit-level dropping, and smooths the traffic to the destination. In addition to source traffic and congestion conditions, the performance of this AP will be affected by two parameters: the amount of buffering at the AP and the size of the application units. In Figure 2 we vary the application unit size on the x -axis, and plot the fraction of transmitted application units that are received intact. (That is, with no piece missing). The different curves correspond to various amounts of AP buffering, expressed as a multiple of the application unit size. Our base unit size is 4092 bytes; thus a unit size of two corresponds to 8184 bytes. We also plot the performance of no AP, running the application stream directly through the Fore switch.

¹Interestingly, the performance of the Null AP can be better than no AP, since the extra read/write into the buffer provides enough delay to allow the switch buffers to drain more effectively.

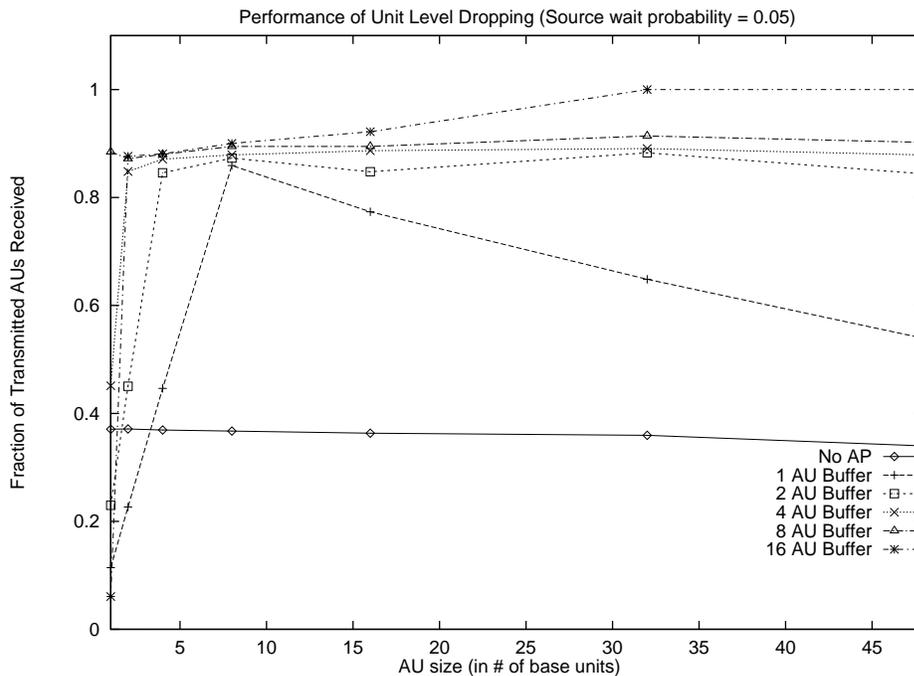


Figure 2: Performance of Unit-Level Dropping, Source Rate: 32Mbps, Destination Acceptance Rate: 20Mbps (mean)

The salient features of this plot include the following. Without an AP, approximately 36% of the units are properly received, regardless of the unit size. The performance with no AP is actually better than with an AP for the smallest application unit size and small buffers at the AP. This is caused by contention for the limited output queue space and the AP rate shaping, which will cause data losses when the input is bursty. The ULD AP is capable of preserving nearly 90% of the application units, provided the amount of buffering is at least four times the application unit size.

5.2 MPEG Streams

MPEG streams present an interesting opportunity to consider a special case of (basic) unit-level dropping, and a more complex dropping mechanism that reflects the inherent dependencies in the data stream. 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. We have augmented an existing MPEG decoding tool [15] to delimit frames as application units. We further mark each frame with its type, I, P or B.

We consider two types of MPEG active processing. Our *MPEG Frame Dropping* (MPEG-FD) AP performs frame level dropping of MPEG streams. This is unit-level dropping with variable length units. To reflect some of the dependency structure in

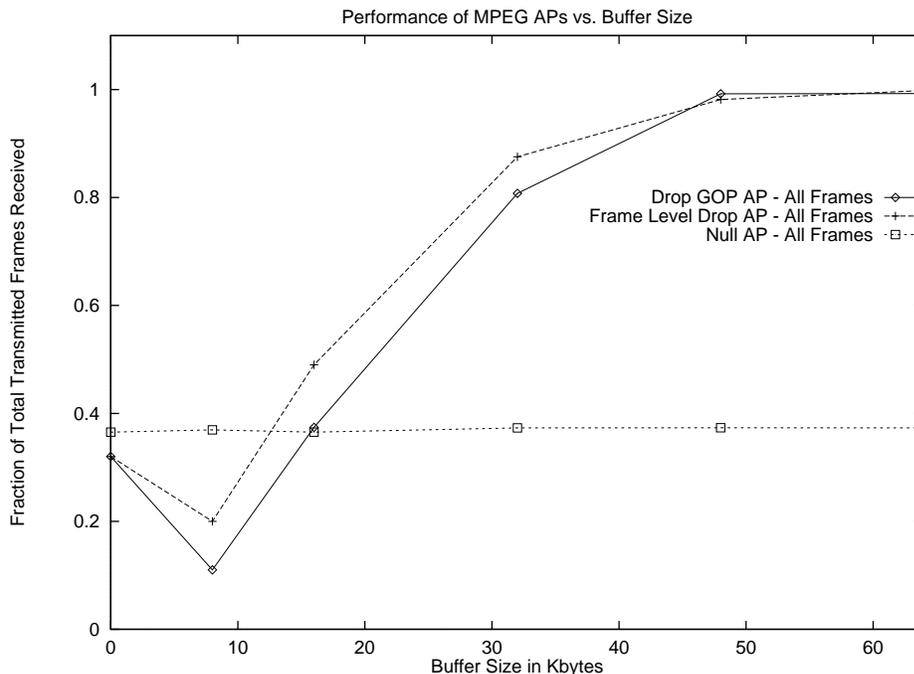


Figure 3: MPEG Performance - All Frames, Source Rate: 32Mbps, Destination Acceptance Rate: 20Mbps (mean)

MPEG, we also have an *MPEG Group-of-Pictures Dropping* (MPEG-GOP) AP, which drops an entire GOP if it is not able to transmit the I-frame. (Note that a more precise dropping scheme is possible, namely one which is aware of the exact dependencies amongst the frames, and triggers a drop of all (transitively) dependent frames whenever a frame must be dropped.) Ramanathan et al. explore similar discard mechanisms to improve video transmission [18].

We evaluate the performance of the MPEG APs using two metrics. First, we measure the fraction of transmitted frames that were received, regardless of frame type. These results are given in Figure 3, and include a curve for a Null AP. The buffer size is varied on the x -axis. We note that the frames have an average size of about 8 Kbytes; until the buffer size is about twice the average frame size, the Null AP is the best performer. As buffer size increases, the MPEG APs approach 100% of the data successfully received, while the Null AP is constant at about 36%. Interestingly, the Frame dropping AP outperforms the GOP dropping AP on this metric. This does not necessarily imply a better perceived quality of service at the receiver, however, because many of the frames received with the Frame dropping AP are dependent on frames that were *not* received.

We also evaluate the performance based on the fraction of the I-frames received. These are the most important frames, since receiving an I-frame will preserve some portion of its GOP, while losing an I-frame destroys most (and often all) of the GOP. These results are shown in Figure 4, and indicate that the GOP dropping AP is better than the frame dropping AP on this metric, particularly if buffer space is limited.

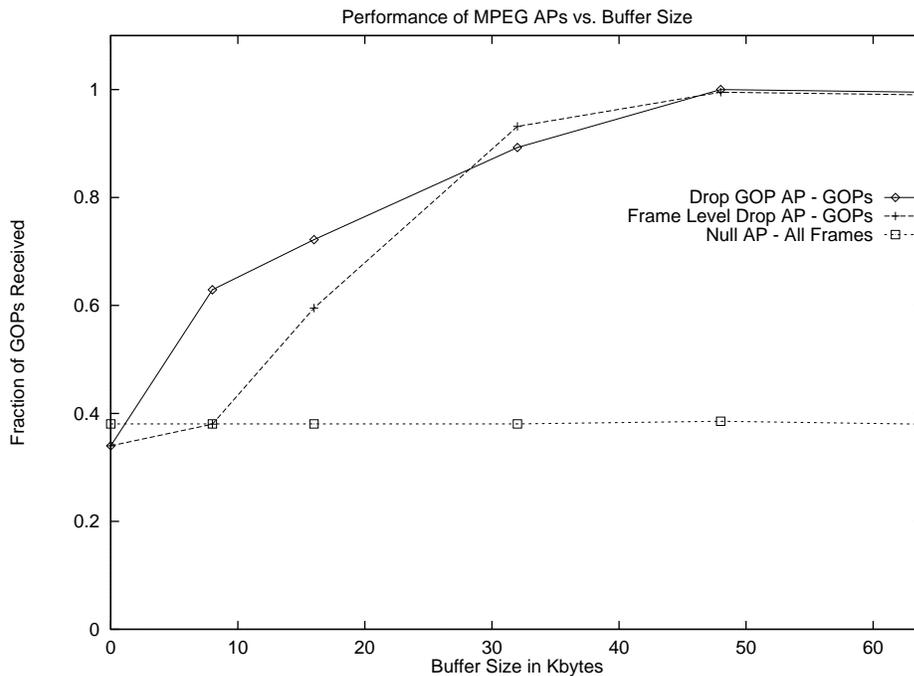


Figure 4: MPEG Performance - I-Frames, Source Rate: 32Mbps, Destination Acceptance Rate: 20Mbps (mean)

5.3 Multi-Stream Interactions

We now consider the case of multi-stream interaction, in which a *logical application unit* is composed of data from two or more streams. We use the term “mates” to denote a set of application units that form a logical application unit. We are interested in two forms of multi-stream interactions:

1. Conjunctive association: Multiple streams are defined to be *conjunctively associated* if the logical application unit is useful iff *all* of its components are correctly received.
2. Disjunctive association: Multiple streams are defined to be *disjunctively associated* if the logical application unit is useful iff *at least one* of its components is correctly received.

We have considered several options for the management of the AP output queue for multiple streams. One option is to create separate queues, each containing application units from a single source, with a control algorithm to efficiently track which multi-stream sources are completely assembled and ready to send to the destination. We implemented a second option, in which the sources share a single output queue. This scheme may cause the AP to transmit complete application units but fragmented logical application units. Below, we consider several methods for reducing this fragmentation.

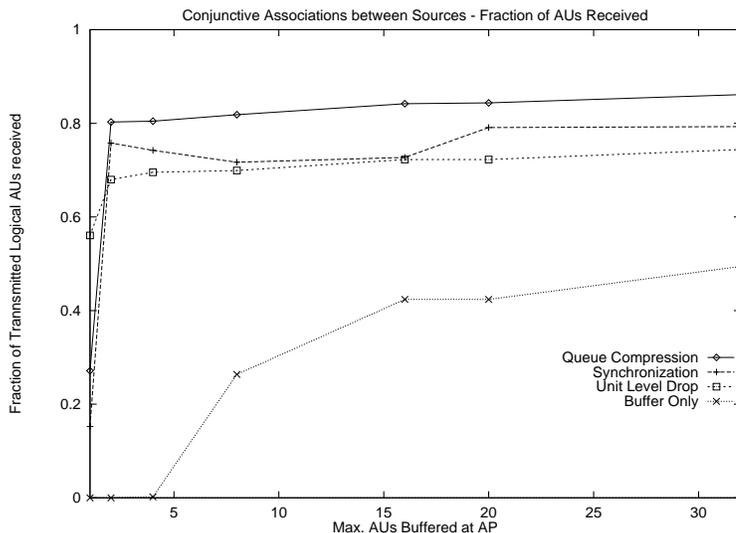


Figure 5: Conjunctive Association - Fraction Received, Two Sources: 16Mbps each, Destination Acceptance Rate: 22Mbps (mean)

5.3.1 Conjunctive Association

We have implemented two AP algorithms exploiting the synchronous nature of conjunctive streams. In our experiments, we limited the number of streams to two and sent 16 Mbytes on each stream. The first *Sync* AP tries to “synchronize” the two streams within the AP. Specifically, it keeps a history list of application unit identifiers which have been dropped at the AP. When a unit arrives whose mate is in the history list, it is dropped as well, and the entry is deleted from the history list.

The performance of the Sync AP can be improved in certain cases. Consider the case in which the individual sources are naturally relatively synchronized. In this case, when an application unit is dropped, there is a fairly high probability that its mate is still in the AP output queue. With the synchronization algorithm, the mate would be sent, although the logical application unit is known to be corrupted. The *Queue Compression* AP remedies this by checking the output queue for mates of units that it is about to drop. If it finds a mate, the mate is dropped as well.

Multi-stream APs are evaluated in two ways. First, we consider the fraction of logical application units which were received intact. Figure 5 shows this fraction for the conjunctive streams and the two APs. For reference, we also include the unit-level dropping AP, and a *Buffer Only* AP. The Buffer Only AP provides buffering and rate control, but it does not assemble the incoming data into units. The results using this AP give an indication of the improvement that could be obtained by simply adding more buffering to a network node, and foregoing any sort of processing. As the figure indicates, more buffering is helpful, however the APs that do some data processing can double the fraction of logical application units received.

To compare the processing APs, we note that these two source streams are naturally relatively synchronized, thus the unit-level dropping AP has performance which

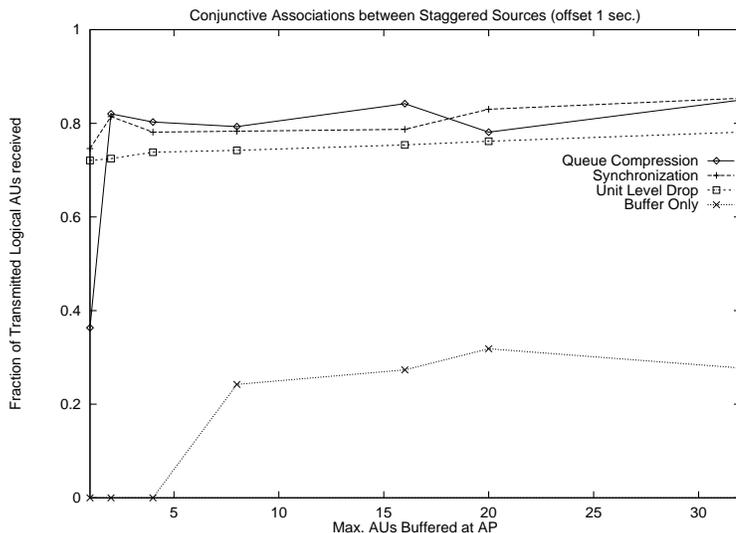


Figure 6: Conjunctive Association - Fraction Received, Two Sources: 16Mbps each, Destination Acceptance Rate: 22Mbps (mean)

is comparable to the Sync AP; the mates nearly always arrive at the AP close to one another, and thus tend to see congestion at similar times. The Queue Compression AP achieves a 10 to 15% better success rate for logical units.

To examine the effect of the relative synchronization, we repeated the experiment with the sources staggered, one offset from the other by one second. These results are shown in Figure 6, and indicate that the Buffer Only AP performs worse under staggered sources, while the other schemes perform about the same or better.

In the case of conjunctive streams, the switch has the possibility of wasting bandwidth by sending a unit whose mate either has been or will eventually be dropped. Given our (strict) definition of the usefulness of the data, any such unit simply wastes network bandwidth. Our second metric determines the fraction of data sent which is wasted data. As illustrated in Figure 7, the value of the Queue Compression scheme is more clear based on this metric, achieving no wasted bandwidth once the buffer size exceeds 15 kbytes.

5.3.2 Disjunctive Association

Upon reflection, there is a trivial algorithm that is always optimal (with respect to the network bandwidth usage) in cases of disjunctive association between multiple streams – the AP should drop all but the stream that consumes the least bandwidth. However, this algorithm may not be the most satisfactory upon the evaluation of the resulting stream at the destination. Instead, we have designed an algorithm that tries to send all the application units that are possible, except in cases when a unit was dropped due to buffer overflow. Upon buffer overflow, the algorithm tries to exploit the disjunctive nature of the stream and send the mate of the dropped unit.

When an application unit u cannot be sent, the *Dynamic Priority Elevation* (DPE)

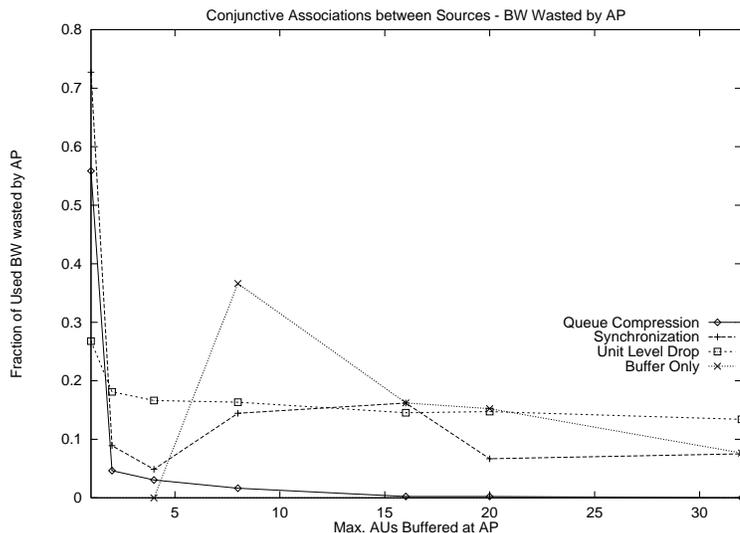


Figure 7: Conjunctive Association - Bandwidth Wasted, Two Sources: 16Mbps each, Destination Acceptance Rate: 22Mbps (mean)

AP searches a history list of units that have been dropped. If the mate of u is *not* in the history list, then the unit identifier is added to the list and u is dropped. If the mate is in the history list, then the AP elevates the priority of u , by attempting to send it at the expense of another unit. Specifically, it searches the units in the (full) output queue and attempts to find one which is not in the history list. If such a unit can be found, it is removed from the output queue, dropped, and its identifier is added to the history list. The output queue now has space for u .

Figure 8 shows the performance of the DPE AP compared to unit-level dropping and the Buffer Only AP. The close comparison of the DPE and unit-level dropping schemes is an indication of the processing requirements of the DPE algorithm. Since this algorithm must do significant searching and matching of unit identifiers, we cannot run the DPE AP experiments at a source rate that would cause the ULD AP to perform poorly. It should be noted that in all cases the DPE AP lost both units of a logical AU, it was because the DPE AP could not service its input queues at the minimum required rate. Thus, the DPE AP could not deliver the AUs because it never received them.

6 System Architecture

6.1 Hardware Architecture

The hardware architecture of an active network node will need to provide substantial processing capabilities and easy access between processing functions and node inputs and outputs. Figure 9 depicts an active node architecture with a routing core, connecting node inputs and outputs and active processing elements. Data entering the node can be sent directly to the proper output or, in the case of a congested output link, to the appropriate (first) active processor. A Congestion Toggle (CT) receives information

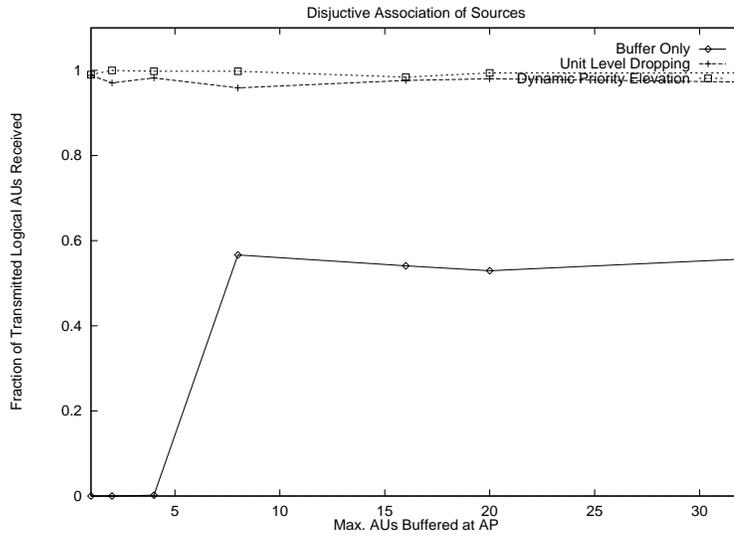


Figure 8: Disjunctive Association, Two Sources: 16Mbps each, Destination Acceptance Rate: 22Mbps (mean)

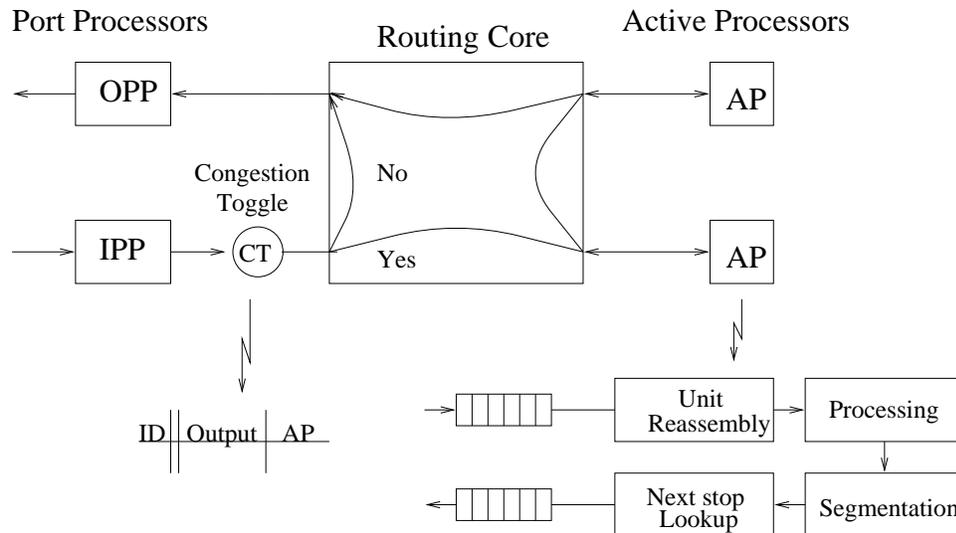


Figure 9: Hardware Architecture

from congestion monitors, and maintains a table indicating where to send data during normal and congested states. The “Yes” path in the diagram indicates data flow during congestion, while the “No” path is the normal route.

The active processing elements may be specialized hardware, tailored to perform particular active node processing, or they may be general purpose processors. An individual processing element may consist of multiple processors and/or pipelining to enable fast computation. This computation power is a shared resource, accessible by the data from any user connection that passes through this network node, thus greater resources may be justified than in a non-shared location. A data unit that enters the network node may pass through multiple processing elements before being routed to one or more network outputs. The routing core allows data to move between processing elements at high speed.

We can get a rough estimate on the (general-purpose) processing capability required by an active processor by noting that the aggregate input bandwidth limits the rate at which data must be processed. Consider a 16 by 16 switch with 155 Mbps links, and assume all input traffic from 15 of the links is bound for the same output. Thus the offered load from those links must be reduced by a factor of 15. Assume that this can be achieved by transforming the data, at a cost of i instructions per input bit. Then an upper bound on the aggregate processing bandwidth required (ignoring overhead) is $2.325i$ Giga-instructions per second. Note that data transformations are likely to be the most processing-intensive form of bandwidth-reduction method.

6.2 Protocol Architecture

In our approach, the AP *only* processes packets containing Active Processing Control Information; this header, when present, is always provided by the originating end-system. In this section we consider the placement of APCI in packets. The two main questions are where to put it, and how to get it there.

6.2.1 Location of the Active Processing Control Header

The obvious place to put the APCI is in the same header used to switch the packet. This is unacceptable, however, for at least two reasons. First, it doesn’t work for ATM or any other technology where the switched unit is too small to accommodate additional the overhead of the APCI. And second, it is not backward-compatible, requiring that all network protocols become “active-aware”. On the other hand, it does not seem possible to completely isolate the active networking architecture from the network protocol, for reasons that will be discussed below.

An alternative to placing APCI in the network header itself is to define a “generic” location for the APCI function, sufficiently high in the protocol stack that the additional overhead is not prohibitive, but sufficiently low to allow its location by switching nodes without too much knowledge of higher-level protocols. In particular, we propose to place the APCI function just above 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 APCI function would be atop the ATM Adaptation Layer, so that the APCI itself would immediately follow the AAL header in an AAL Protocol

Data Unit. This implies that an ATM-AP must first reassemble AAL Data Units before it can examine the label header. While this adds latency, recall that it does not occur for every data unit. (This is the approach we implemented in our experimental setup.)

If the Internet Protocol (version 4 or version 6) is being used as the end-to-end switching layer, we recommend that the APCI be treated as a “sublayer” or optional header, as is done for example with the headers used to implement the IP security architecture [6, 7]. At a node where active processing takes place, the switch examines packets for APCI and processes those in which it is present.

A very likely scenario is one in which IP is used with subnet technologies that include ATM, but not all traffic on the ATM network is IP traffic. In this scenario, ATM switches handle a mixture of traffic, of which some carries an IP header while the rest is VBR or CBR traffic such as voice or video that stays on the ATM network. IP switches, on the other hand, deal only with IP datagrams, some of which are transmitted over ATM subnets. In such a scenario, the optimal solution is for IP datagrams to have the APCI following the IP header, and ATM-only traffic to have it following the AAL header. Thus, ATM APs have to look in *two* places for APCI information: following the AAL header in non-IP traffic, and following the IP header for IP traffic. This implies that ATM APs need to know enough about the IP header format to be able to find the APCI. 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.²

Although the network layer processing in switches can for the most part remain ignorant of active processing, there is one case where it must know about and interpret the APCI. If anything (e.g. fragmentation) causes a restructuring of the packet while it is in transit through the network, the network layer must adjust the APCI so it remains consistent. However, fragmentation has been avoided as much as possible for some time [14], and the philosophy of *Application Layer Framing* [8] has gained wide acceptance. Where ALF prevails, the network tries to preserve the structure of the data units created by the application, and the network layer can be mostly independent of active networking.

6.2.2 Attaching the APCI

The Active Processing control information may depend on multiple protocol layers. For example, the APCI might include labels that correspond to a block number in a file, a TCP sequence number, and an IP datagram identifier. The question arises how to aggregate this information in a single location in a packet, especially within a layered encapsulation model. We first observe that ideally the APCI will originate with the application itself, because it can best determine which, if any, active processing functions are appropriate for its data. To allow for this, the user interface must be extended to enable the application to pass APCI in to the protocol stack on a per-application-data-unit basis, presumably as a parameter to the “send” primitive. Along with the APFI, the application can also pass one or more values to be placed in specific label fields in the APCI.

²The same approach would apply to any other switched technology used as a subnet by IP, e.g. XTP [21]. Similarly, when some other network layer tunnels through IP, the argument would apply with IP in the role of ATM.

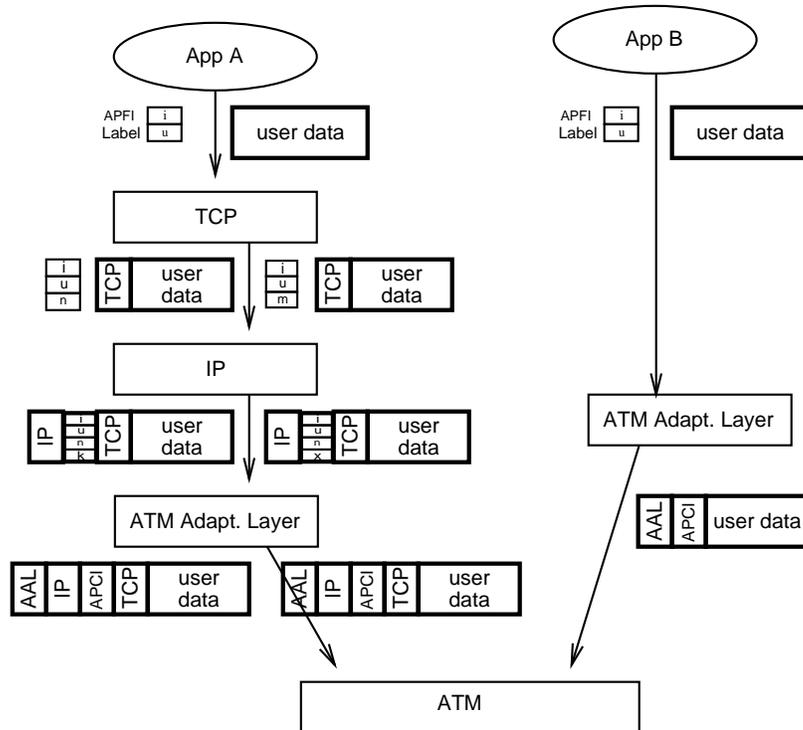


Figure 10: Attachment of APCI in sending end system

Note, however, that it is not *necessary* that the application make this determination explicitly; for certain applications and certain AP functions, the underlying protocol stack might select the function and attach the appropriate APCI. For example, TCP might attach an APCI that marks the TCP segment and selects the ULD function. In this way applications can get some of the benefits of active networking without having to be modified.

Below the user (application) interface, each layer must be modified to add the appropriate label value to the APCI it receives from the layer above. The question then becomes when to attach the aggregated APCI information to 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 APCI will follow the header of the highest-level end-to-end relay layer. Figure 10 shows an example, in which two types of traffic are generated. Application A uses the TCP/IP stack, while B runs directly atop an ATM Adaptation Layer. Data units from application A have the APCI attached by IP; the AAL receives no APCI from IP. B's data units have APCI attached by the AAL.

As the data unit progresses down through the stack, each layer adds its label to the APCI; this information can be used to encode the structure of the data at all levels, as described in Feldmeier's "chunk" labeling scheme [10].

At the receiver, the APCI must be recognized by the appropriate layer. In the simplest case, it is discarded. For example, with IPv6, it might be treated as an unrecognized option header. In the case of IPv4 or an AAL, however, it will need to be treated as an explicit (but trivial) sublayer. Another possibility is to pass the set of labels to each receiving protocol entity on the way up through the stack. As with "chunks", this information may facilitate processing at the receiver, especially when units are reordered in transit.

7 Conclusions

Viewing the network as a general computation engine enables an exciting set of capabilities and applications. Implementation and deployment of this view clearly involves challenges along many fronts, from security to interoperability to performance. We have taken a modest step towards understanding the capabilities that an active network might provide with regard to controlling congestion. In particular, we have explored mechanisms that would allow bandwidth reduction to occur in a manner that preserves as much application-level useful data as possible. We have outlined a possible hardware architecture for an active network node, and we have described a protocol architecture to allow applications to specify computation to occur in the network. Using existing networking technology, we have implemented a range of active mechanisms and evaluated their performance.

A number of issues require further consideration. If the applications or end-system protocols include congestion control or adaptation, then these mechanisms will interact with active processing in ways that we have yet to fully explore. In the face of long term congestion, source adaptation will likely be required, otherwise uncontrolled loss may still occur. For shorter term congestion, mechanisms in the network will be

better able to quickly adapt and recover. We have also not yet considered fairness issues regarding the service and performance received by various connections. Since our bandwidth reduction techniques are more extreme than arbitrary cell or packet drops, applications may experience unequal short-term performance degradation. Provisions will be required over the long term to ensure fair service.

In future work, we plan to augment our existing mechanisms, in particular to include media transformation and additional multi-stream interactions. We also plan to experiment with a wider-area setup, allowing more complex multi-active-node processing.

References

- [1] *ForeRunner ASX-200, software version 3.4.x edition.*
- [2] ISO DIS 10918-1 Digital compression and coding of continuous-tone still images (JPEG). CCITT Recommendation T.81.
- [3] Video codec for audiovisual services at p*64kb/s. ITU-T Recommendation H.261.
- [4] E. Amir, W. McCanne, and H. Zhang. An application level video gateway. In *ACM Multimedia '95*, 1995.
- [5] G. Armitage and K. Adans. Packet reassembly during cell loss. *IEEE Network Magazine*, September 1993.
- [6] Ran Atkinson. Ip authentication header. *RFC 1826*, August 1995.
- [7] Ran Atkinson. Ip encapsulating security payload. *RFC 1827*, August 1995.
- [8] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM '90*, 1990.
- [9] 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.
- [10] D. Feldmeier. A data-labelling technique for high-performance protocol processing and its consequences. In *ACM SIGCOMM '93*, 1993.
- [11] International Organisation for Standardisation. Generic coding of moving pictures and associated audio. ISO/IEC/JTC1/SC29/WG-11, March 1993.
- [12] J. Gosling and H. McGilton. The Java language environment: A White paper. Sun Microsystems, 1995.
- [13] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, 1988.
- [14] C. Kent and J. Mogul. Fragmentation considered harmful. In *ACM SIGCOMM '87*, August 1987.
- [15] Philip Lougher. Mpegutil, 1995.

- [16] S. McCanne and M. Vetterli. Joint source/channel coding for multicast packet video. In *IEEE International Conference on Image Processing*, October 1995.
- [17] 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.
- [18] S. Ramanathan, P. Rangan, and H. Vin. Frame-induced packet discarding: an efficient strategy for video networking. In *Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, November 1993.
- [19] A. Romanow and S. Floyd. Dynamics of TCP traffic over ATM networks. *IEEE Journal on Selected Areas in Communications*, May 1995.
- [20] W. David Sincoskie. Development of the U.S. national information infrastructure. Keynote address, International Conference on Computer Communications and Networks (ICCCN'95), September 1995.
- [21] W. T. Strayer, B. F. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley, 1992.
- [22] 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.
- [23] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. In *Multimedia Computing and Networking '96*, January 1996.
- [24] T. von Eicken et al. Active messages: A mechanism for integrated communication and computation. In *19th International Symposium on Computer Architecture*, 1992.