Flow Labelled IP over ATM: Design and Rationale

Greg Minshall Bob Hinden Eric Hoffman Fong Ching Liaw Tom Lyon
Peter Newman [*]

Ipsilon Networks, Inc

ABSTRACT

We describe a system in which layer 2 switching is placed directly under the control of layer 3 routing protocols on a hop-by-hop basis. Specifically, ATM switching is controlled by IP. We couple each ATM switch with a general purpose computer running IP routing and management protocols. We define a default ATM virtual channel identifier (VCI) to be used for transmitting IP packets over ATM links. We then define mechanisms which allow specific *flows* to be transmitted on specific ATM VCIs. The resulting system obeys IP's semantics for routing and forwarding, and takes advantage of ATM's switching hardware to accelerate the forwarding of packets. While this system takes advantage of ATM hardware, the ATM signalling, routing, and management architecture (as specified by the ATM Forum) is replaced by the protocols and practices currently in use for IP routing and management.

Categories and Subject Descriptors

C.2 [**Computer Systems Organization**]: Computer Communication Networks; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; C.2.6 [**Computer Communication Networks**]: Internetworking

General Terms

ALGORITHMS, DESIGN, EXPERIMENTATION, MEASUREMENT, PERFORMANCE, STANDARDIZATION

Keywords

IP switching, IP, ATM, IFMP, GSMP, Ipsilon, Flow Labelled IP

1. INTRODUCTION

IP — the Internet Protocol — [29] exists as an internetwork layer protocol, allowing datagrams to flow between hosts and routers to accomplish the delivery of information to a node (host or router). During the course of IP's history, a number of different link level technologies have been used to provide *bit-pipes* between neighboring IP nodes. The earliest technology, the ARPANET, provided a non-broadcast, multi-access, network with 24 bit addresses [2]; in order to use the ARPANET, a node's ARPANET address was embedded in the low order 24 bits of the node's IP address [31]. Three-megabit Ethernet [16] had an 8 bit address; to utilize three-megabit Ethernet, a node's Ethernet address was embedded in the low order byte of a node's IP address [32]. Ten-megabit Ethernet [35] has a 48-bit Ethernet address; in order for a node to discover the Ethernet address of another node, the Address Resolution Protocol (ARP) was developed [27].

In each of the above cases, running IP over a data link involved inventing methods of mapping some of IP's functionality (addressing, mostly) into some data link specific representation.

Asynchronous Transfer Mode (ATM) is a recent data link technology [7]. For various reasons (economic and technical), people would like to run IP over ATM. As with previous data link technologies, running IP over ATM in an efficient, manageable way requires inventing some new, ATM-specific, mechanisms. There have been previous attempts to run IP over ATM [22]; these attempts, however, have been characterized by attempts to situate the IP routing function at the edges of an ATM *cloud*, utilizing other (non-IP) protocols to manage routing and links within the ATM cloud. Our choice is to design as simple a mechanism as possible that keeps IP's routing and management in control of datagram flow and adapts to topology changes. Our approach couples ATM switches with IP routing, creating an entity we term an *IP switch*. We believe that the mechanisms created to support the IP switch may have applicability beyond the initial application of using ATM as a data link technology.

In choosing the approach, we reveal one of our fundamental beliefs: that IP is the right layer to manage data communications in a network. In the local area, this means that IP controls the physical links that connect systems. In the wide area, we believe that the correct model is a separation between the IP layer, on the one hand, and a supplier of a *bit-pipe*, on the other. This belief is not subject to any empirical test. However, it to a large extent motivates and guides our work.

1.1 Terminology

In this paper, we adopt the convention of naming a node as being *upstream* or *downstream* with respect to the flow

[*]{minshall,hinden,hoffman,fong,pugs,pn}@ipsilon.com.

```
Current e-mail addresses for authors:
Greg Minshall <minshall@acm.org>
Bob Hinden <bob.hinden@nokia.com>
Eric Hoffman <yuri@tenuki.org>
Fong Ching Liaw <fongliaw@yahoo.com>
Tom Lyon <pugs@ieee.org>
Peter Newman <peter.newman@ieee.org>
```

of actual data packets (i.e., an FTP data transfer running over TCP), even when discussing control messages that may be flowing *against* the direction of data flow. Thus, if the packets in the FTP data transfer are being sent across a specific data link from node A to node B, and node B sends a control message (relating to the FTP data transfer) to node A, we say that the *downstream* node has sent a message to the *upstream* node. This is similar to the usage of these terms in the RSVP protocol [37].

2. A SHORT REVIEW OF ATM AND IP

2.1 ATM Addressing

As opposed to IP datagrams or Ethernet frames, ATM frames carry no end-to-end addressing information. An ATM frame carries two fields — Virtual Channel Identifier (VCI) and Virtual Path Indentifier (VPI) — that together identify some association (connection, circuit, whatever) between the sender and receiver. The VCI field is 16 bits; the VPI field is either 8 or 12 bits, depending on the types of interconnected nodes. We call the VPI and VCI fields in a frame the *label* for the frame.

In fact, at the hardware level — the level of ATM switches and NICs — an ATM device has no address. ATM is fundamentally a *point-to-point* technology. The fact that a frame sent over a given ATM link *on* a given label will ultimately arrive at a particular destination is not visible to the ATM hardware.[1]

ATM labels are of *local* significance, having meaning only at the (ATM) interface on which they are received.

In the standard definition of ATM, labels are *bidirectional* — cells sent from system A to system B on label X refer to the same (high-level) *conversation* as cells sent on label X from system B to system A. We do not use this convention; in our system labels are unidirectional.

2.2 ATM Frame Switching

To a first approximation, we can think of ATM as being a technology which switches frames.[2]

Conceptually, each label on an interface at an ATM switch is associated with a (possibly null) set of tuples *(outgoing interface, outgoing label)*. Switching a frame that enters an ATM switch involves looking up this set, then transmitting the frame out each interface in the set, labelled with the appropriate label. Notice that this implies that in the case of multicast the labels of the output frames are independent and may have different values. Figure 1 depicts unicast frame switching; figure 2 depicts multicast frame switching.

ATM switches guarantee that no frames will be delivered out of order or duplicated. Lost frames are allowed. A Cyclic Redundancy Check (CRC – actually provided by AAL5, see footnote 2 and section 6.6) protects frame data against corruption.

[1]We *do*, however, make use of IEEE-assigned 48-bit MAC addresses assigned to ATM switches and NICs as *local identifiers*, for the purpose of forming point-to-point adjacencies and to identify specific pieces of hardware for network configuration.

[2] In reality, ATM switches *cells*; a number of cells make up a frame — an *ATM Adaptation Layer 5* (*AAL5*) frame — a fact which is somewhat problematic, as we shall see in section 6.6.

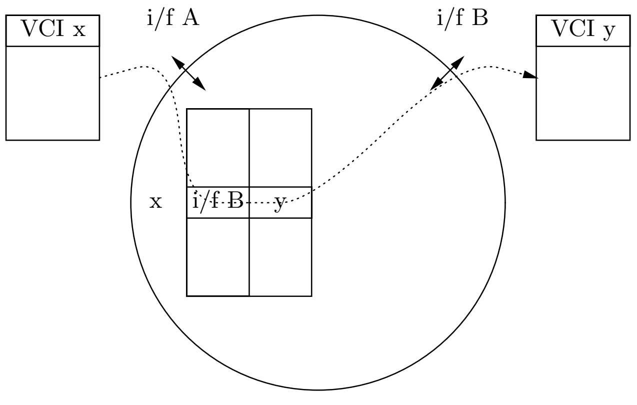

Figure 1: ATM Unicast Switching
Frames arriving on interface *i/f A* with label VCI x are transmitted on *i/f B* with label VCI y.

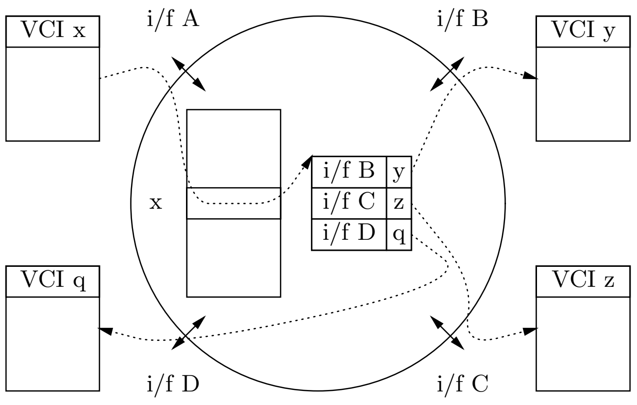

Figure 2: ATM Multicast Switching
Frames arriving on interface *i/f A* with label VCI x are transmitted on interface *i/f B* with label VCI y and on interface *i/f C* with label VCI z and on interface *i/f D* with label *VCI q*.

2.3 IP

IP enables the transmission of datagrams from a source to a destination without regard to the data link technology at either source or destination or between the two.

A fundamental characteristic of IP is the minimal coupling between nodes. Basically, a *router* node agrees to provide a *best effort* service of forwarding packets towards a destination. There is very little *a priori* knowledge necessary for such data to flow.

In general, there are very few *prior agreements* that need to be in place between two nodes before datagrams can be sent from one node to the other.

3. HOW TO RUN IP OVER ATM

Our goal is an interconnected mesh of IP routers that gets some of the performance possible by use of layer 2 switching. In particular, we would like to make *routing* decisions once per *flow* rather than once per *packet*. Additionally, we wish to retain all the features of IP that have enabled it to interoperate over so many different networks and between so many administrative domains.

In order to achieve the performance of layer 2 switching, we partition the application (of being an IP router) between the software and hardware (ATM switch). Anything that can be implemented in software can also be implemented in hardware; thus, how to decompose an application between hardware and software is not always clear at the outset. Our (not-particularly-novel) approach has been to decompose things such that the hardware concerns itself with very simple operations, the semantics and functionality of which evolve very slowly over time, while at the same time allowing the software to handle the more complex, and possibly more quickly evolving, operations in a way which allows us to preserve all of IP's semantics and manageability.

The summary of section 3 is this: we couple a general purpose computer with an ATM switch; we define a default label for transmitting IP datagrams from one IP node to another over an ATM link; we allow downstream nodes to tell upstream nodes "send packets that look like *this* on *that* label"; when packets coming in on one non-default label are being sent out on a non-default label, we can switch the packets between the incoming and outgoing interface. The node (computer and ATM switch) we refer to as an *IP switch*; the computer itself we refer to as an *IP switch controller*. Details follow...

3.1 VCI 15 — The Importance of Binding Early

There is a *well known* VPI/VCI pair (VPI 0/VCI 15) that is used when sending an IP packet to a node. During initialization, nodes set up their local ATM NICs and ports on any local ATM switches[3] to receive packets sent to this VPI/VCI and have them processed by their local IP protocol module. If because of some higher level configuration information, IP decides to send a packet to a given node connected over an ATM interface, the sending node can send the packet to VPI 0/VCI 15.

Since this default VPI/VCI is known to the code compiled into a host or router, there does not need to be any sort of configuration, directory, or other information available to a node to determine how to send an IP datagram on an ATM link. This reduces the mechanisms which need to be in place before IP can begin operation, thus simplifying the overall system. This is very similar to assigning the hexadecimal value 0x0800 as the *ethertype* for sending and receiving IP packets on Ethernet [12].

3.2 Hop-by-hop Forwarding

Specifying this default VPI/VCI allows one to build an IP network in which nodes are interconnected via ATM links. In this network, all packet forwarding is handled on a "hop-by-hop" basis. I.e., each packet is examined individually by each router in the path, and each router makes an independent determination of whether or not to forward that packet and if so, out which interface to forward it. This behaviour, which preserves the local autonomy of IP, is depicted in figure 3.

[3]It is necessary to set up mapping tables in the ATM switch such that packets coming in on VPI 0/VCI 15 on a given port arrive at the IP switch controller on a unique VPI/VCI pair.

This hop-by-hop model means that the interconnected mesh of ATM nodes behaves as just like what it *really* is — an interconnected mesh of IP routers. Note that at this level, ATM switches don't provide any *performance* advantage.

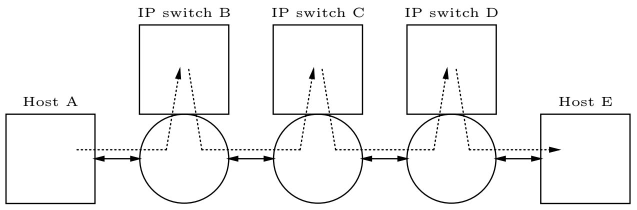

Figure 3: Hop-by-hop Forwarding
A packet sent from *Host A* to *Host E* is forwarded hop-by-hop through IP switch B, IP switch C, and IP switch D. In the diagram, the circles represent ATM switches and the boxes represent general purpose computers. Solid lines indicate physical connectivity; dotted lines indicate the path taken by a datagram.

3.3 The Flow Management Protocol

The concept of a *flow* has been used in the Internet context for a number of years [17, 18, 37, 5], normally in a fairly informal way. In our search for the optimal binding of IP to ATM, we formalize this concept. I.e., we define a means of precisely specifying which packets are, and are not, part of a given flow; this allows one node to tell another node "packets that look like *this* should be treated like *this*".

The protocol which adjacent nodes use to talk about flows is known as the *Ipsilon Flow Management Protocol* (*IFMP*) [21]. An IFMP *flow* consists of a *flow type*, which specifies which bits of the IP header (and, possibly, beyond) are inspected to determine membership, as well as a *flow identifier*, which specifies the actual values of the bits specified by the flow type. Flow types are pre-defined; flow identifiers are consed up as needed by examining the relevant fields of passing packets.

In our initial system, we have defined two major flow types with the not-very-imaginative names of *type 1* and *type 2*. These two flow types are shown in Figure 4.

A type 2 flow consists of all packets between a given pair of hosts matching in certain other IP header fields.

A type 1 flow has the same characteristics as a type 2 flow, with the addition of the first 32 bits of the transport payload. In the case of TCP and UDP, this logically means that the TCP and/or UDP source and destination port numbers are included in the flow specification. This has the effect of binding a flow to a specific conversation between a specific pair of hosts.

3.4 Use Redirects

Having defined flows, we define in IFMP the ability for a node to send a *redirect*, similar to ICMP Redirects [28], upstream requesting the upstream node to start sending packets belonging to that flow over a non-default label. When the upstream node receives such a redirect, it may choose to send all future packets in the redirected flow to the label contained in the redirect message.

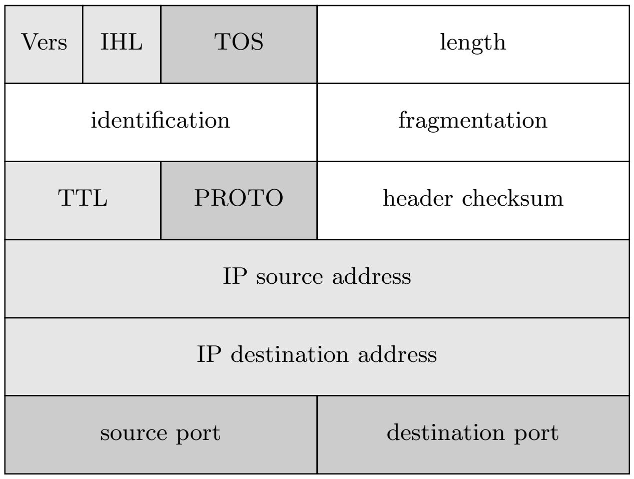

Figure 4: Flow Definitions
The above represents an IPv4 header along with the first four bytes of a TCP or UDP header. All the grey areas are used to define Type 1 flows; only the light grey areas are used to define Type 2 flows.

Redirect messages are *advisory* — the upstream node (the receiver of the redirect) is free to ignore a redirect message.[4] After having received and accepted a redirect message, an upstream node is free to "forget" the redirect at any time.

As a result of the advisory nature of redirects, there is a minimum amount of time specified between the sending of a redirect for a flow and the resending of a redirect for the same flow (nominally one second).

If, after having sent or received a redirect for a flow, higher-level entities on a router (such as IP routing processes) direct that packets for that flow should be sent out a different interface, the node will send them out the new interface, *not* out the interface over which the previous redirect had been sent or received.

3.4.1 Receiver initiated

As noted above, one of the reasons for the success of the connectionless, datagram model used by IP is the relatively weak coupling between adjacent network nodes. This decoupling allows for substantial autonomy for nodes, allowing them to make decisions based on local policy, and still provide the basic internetwork service. If we envision large portions of the Internet using our new data link, it is imperative that we retain this decoupling.

There are two decisions associated with redirecting: 1) whether or not to redirect; 2) which label to use for the redirection.

If we required that a node obey a redirect — i.e., if we made the redirects *mandatory* — we would be reducing the autonomy of the nodes. By making the redirects *advisory*, we allow the node receiving the redirect to make use of local policy to decide whether or not to obey the redirect. This goes in the direction of retaining local autonomy. Thus, effectively, both neighbors need to agree to redirect before the redirection happens.

From a purely theoretical point of view, either neighbor (upstream or downstream) could choose the label to be used for sending redirected packets. However, in IFMP, we have opted for having the downstream node choose the label and send the redirection message. There are several reasons for this decision.

First, we made a decision that an upstream node should never send a packet on a non-default label unless the upstream node knew (to a very high degree of probability) that the downstream receiver is aware of what flow the upstream node associated with the label. Thus, even if the upstream node chose the label, it might be useful to have at least *one* message per redirection sequence from the downstream node to the upstream node. On the other hand, by having the downstream node choose the label and send the advisory redirect message to the upstream node, we are able to reduce the redirection sequence to that single message; i.e., the redirection message is not explicitly acknowledged (to protect against the loss of the redirect message, the downstream node is able to monitor packets coming in on the default label in order to detect that the upstream node has or has not accepted the redirect[5]).

Second, in at least the case of today's ATM hardware (adapter cards and switches), it is common that the ability to *receive* on a label is a more precious resource than the ability to transmit on a label. (There are, of course, exceptions to this.) Thus, very pragmatically, having the receiver choose the label makes sense.

To take care of the case where the upstream node is only able to transmit packets within a certain range of labels, we have provided a *label range* message which is sent to the downstream node, informing it of the particulars.

3.5 *Get Switched*

If a given router has redirected a flow, by sending a redirect message to its upstream neighbor (for that flow), and has likewise received a redirect message for the same flow from its downstream neighbor, the router can begin to *switch* (rather than hop-by-hop forward) packets belonging to the flow. To do this, the router instructs its internal ATM switch to take frames arriving on the interface from the upstream neighbor on the redirected upstream label and transmit them on the interface to the downstream neighbor on the redirected downstream label. Figure 5 shows the exchange of messages which causes this switching to occur; figure 6 shows the resulting data flow.

3.6 Robustness

In addition to trying to preserve local autonomy, another goal of our design has been to produce a protocol that is robust in the face of network and/or node failure. While we strive to avoid introducing inconsistent state in the network, we have also been guided by the principle that it should be possible to bound the amount of time that bad or inconsistent state exists in the network [33]. The specific hazard our design introduces in the network is that an upstream node

[4]And, of course, the downstream node need not send redirects; the redirection mechanism is simply an optional performance enhancement, and data communications proceeds if one or both ends does not support it.

[5]Notice that, because of the advisory nature of the redirects, continued reception on the default label does *not* necessarily mean that the redirect message was lost; the upstream node could have decided, for reasons of its own, to ignore the redirect message.

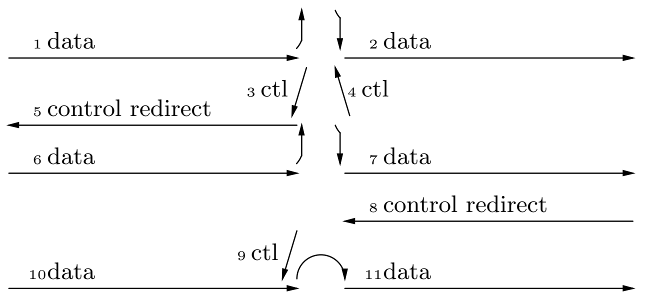

Figure 5: The IFMP *Dance*
Datagrams 1, 6, and 10 are sent from the upstream node; datagrams 2, 7, and 11 are being forwarded to the downstream node. Message 3 is a control message from the IP switch controller to the ATM switch, setting up a remapping; message 4 acknowledges message 3. Datagram 5 is an IFMP redirect sent to the upstream node; datagram 8 is an IFMP redirect received from the downstream node. Message 9 is a control message to the ATM switch, remapping the incoming label to the outgoing label; message 9 is an unacknowledged message. In this example, it is assumed that datagram 1 is forwarded via standard IP hop-by-hop forwarding; datagram 6 is forwarded via so-called *VCx forwarding*; and datagram 10 is being forwarded by IP switching.

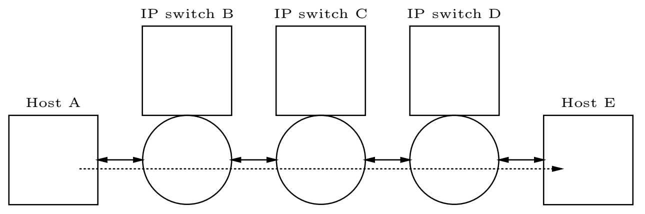

Figure 6: IP Switching
In this figure, datagrams from *Host A* to *Host E* travel through IP switch B, IP switch C, and IP switch D without the need for hop-by-hop forwarding (compare with figure 3).

may be sending a given flow on a label unbeknownst to the downstream node. There are two subcases: first where the downstream node thinks a different flow is being transmitted on the label; second where the downstream node thinks no flow is being transmitted on the label. In the first case, packets sent on the label will be mis-routed and corrupted; in the second case, packets will be dropped at the downstream node.

In order to reduce this failure from being introduced into the network, we have designed a simple *adjacency protocol* within IFMP. This protocol enables a node to be sure that its neighbor is the same instance running on the same node. Figure 7 is the simple state diagram for this protocol; it intentionally resembles the "top half" of the state diagram for TCP [30]. IFMP redirects are only sent or processed when in ESTABLISHED state; when leaving ESTABLISHED state, all redirection state for that link is cleared.[6]

[6]There is a minor detail in the case where IFMP packets, themselves, are sent over a redirected label. In this case, messages *to* a (new) instance of a (new) peer may not be received at the peer, since the peer is not listening to the redirected label. For this reason, IFMP packets are always sent on the default label.

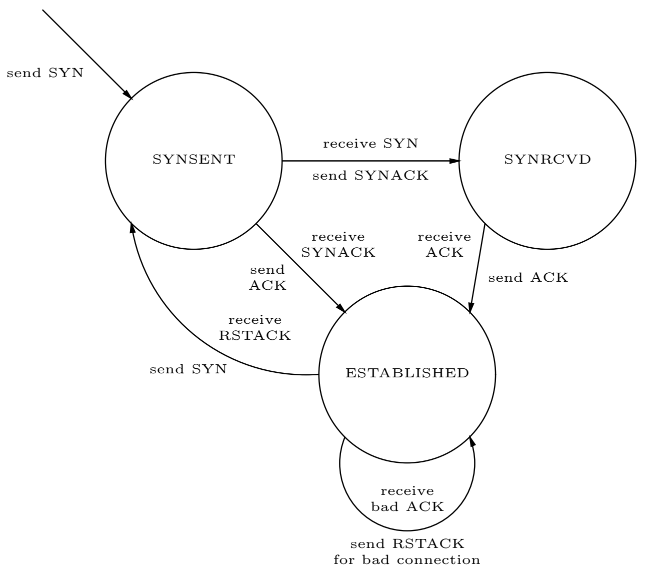

Figure 7: IFMP Adjacency Protocol State Diagram
This state diagram informally documents the state transitions in forming an IFMP adjacency between neighboring IP hosts and/or routers.

In addition, redirection messages are sent with a monotonically increasing sequence number. This allows the message receiver to process messages in a correct order. (The IFMP specification is somewhat silent on what a "correct order" means; basically, messages may be skipped (there is no retransmission), but if message i is processed after message j, but $i < j$, then message j must be re-processed after processing message i.)

We recognize that inconsistent state may develop occasionally between nodes, even with the barriers we have put up to prevent such an occurrence. We attempt to limit the lifetime of such bad or inconsistent state in the network by associating *lifetimes* with redirection messages. After the lifetime expires, upstream nodes are required to stop sending messages on the redirected label. Lifetimes are expressed in units of one second in an 8-bit field. Typically, lifetimes are on the order of one or two minutes; these values correspond to the results reported by Claffy, et al. [5].

In addition to the lifetime mechanism, there is a *reclaim* message which is part of the redirection protocol. This allows a downstream node to request that an upstream node quit using a given label. A *reclaim ACK* from the upstream node confirms the reclaim, allowing the downstream node to reuse that label for another flow.

4. CONTROLLING THE ATM SWITCH

The interface between the IP switch controller, which is running IP routing and management protocols, and the ATM switch is a simple protocol known as the General Switch Management Protocol (*GSMP*) [20]. This is a fairly lightweight

Connection Management Messages	
Add branch	add one output branch for a VCI
Delete branch	delete one output branch for a VCI
Delete tree	delete all output branches for a VCI
Verify tree	consistency check (as GSMPis unreliable)
Delete all	delete all output branches on a port
Move branch	move the destination port of an output branch
Port Management Messages	
Bring up	set a port's administrative state to *in service*
Take down	set a port's administrative state to *out of service*
Internal loopback	set the port to internal loopback
External loopback	set the port to external loopback
Bothway loopback	set the port to both internal and external loopback
Reset input port	reset an input port
Reset event flags	flow control for Event Messages
Statistics Messages	
VC activity	check for traffic activity on a VCI
Port statistics	get per-port cell (and frame, if available) counters
VC statistics	get per-VCI cell (and frame, if available) counters
Configuration	
Switch configuration	learn the identity of the switch
Port configuration	learn the type and status of a single switch port
All ports configuration	learn the type and status of all the ports on the switch
Event Messages	
Port enabled	a port is physically ready to send/receive
Port disabled	a port is no longer physically ready to send/receive
Invalid VPI/VCI	one or more cells with an invalid VCI have been received at a port
New port	a new port has been recognized
Dead port	a previous port has disappeared

Figure 8: A brief overview of GSMP facilities

protocol designed to allow the control of almost any ATM switch from an attached system. GSMP allows the controlling system to: determine the ATM switch's configuration and enumerate its interfaces; gather simple per-port and per-VCI statistics; and set up the tables that map between incoming interface and VCI and outgoing interface(s) and VCI(s). GSMP also includes a simple, unacknowledged, event reporting message to inform the IP switch controller when exceptions occur. Figure 8 gives an overview of GSMPś functionality.

GSMP turns out to be easy to implement in many switches: we've implemented it on three quite different switches from different manufacturers, taking between 500 and 2500 lines of C code in the implementations.

Implementing our system on top of GSMP has allowed us to decouple the IP switch controller from the attached ATM switch, which allows engineering to proceed in parallel on the two systems. The decoupling we achieve by using GSMP is also useful in allowing different IP switch controller and ATM switch combinations over time, allowing a site to install different sets of hardware as needed to handle evolving workloads.

5. BRIEF INTERLUDE

5.1 Why Do This?

While at first glance it may appear that in adopting IP as our control protocol, we are substantially increasing the computational load at every ATM switch. However, there is no reason to believe that the computational requirements for doing IP routing and forwarding is any higher than those required for other approaches which make use of ATM using *native* ATM signalling.[7]

By using IP as the control protocol, each ATM link becomes an IP point-to-point link. By doing this, IP is able to manage each of these links, and has cognizance of their existence.

As one example of the advantage of doing this, efficient support for IP multicast [8], which is tricky to do using conventional approaches to ATM, works without requiring any additional effort in our approach. As different downstream peers send redirect messages upstream, they are connected, within the ATM switch, to the source interface which is receiving the multicast packets from further upstream; down-

[7]It is questionable whether ATM to the desktop is an economically, technically, or practical solution in the face of the fairly rapid development of desktop Ethernet technology; for this reason, we have engineered our system with corporate and/or campus backbone connectivity in mind.

stream peers which have not sent redirect messages receive the multicast packets via hop-by-hop forwarding on the default label.[8]

5.2 Why Should This Perform Well?

Our claim is that using the ATM switch to speed up forwarding rates will have a positive effect on performance. But, this is clearly dependent on the traffic workload; if all the packets that come through are *singletons*, i.e., packets from single-packet flows, the system will spend all its time doing hop-by-hop forwarding, and the ATM switch will not help performance at all.

In previous work [22], using a fairly simple algorithm for deciding which flows to switch and which flows not to switch, our *simulator* was able to switch over 80% of the packets, accounting for over 90% of the bytes, setting up less than 120 flows per second, when running off of a trace of data from the Internet. Thus, there is reason to believe that a majority of the packets are from a restricted number of flows.

6. THORNY DETAILS

6.1 Secure Switching

We are building a system which allows very fast IP switching between ATM nodes. One very common use for IP routers is to install a *firewall* [4] between *administrative domains*. We have tried to engineer our protocol design such that, quite often, firewalls can be built such that most packets through the firewall are switched rather than routed.

Basically, the idea is this: IP packets for a flow are routed until the firewall software is convinced that subsequent IP packets in the flow are *safe*. At that point, the firewall software allows the flow to be switched.

This approach does not, by any means, cover all the ways a site may want to control traffic transiting a firewall. However, it does deal with a fairly large subset of firewall policies in a very efficient manner.

6.1.1 Wire representation

As part of our effort to make it possible to convince oneself of the security of switching IP packets in a flow through a firewall, we have changed the *wire representation* used in the various flow types. In general, those header fields which are involved in the flow *definition* are *not* sent across the data link. This is possible because senders do not transmit packets on a given label until the receiver knows which flow is being transmitted on that label. Since the receiver has that knowledge, it is able to *reconstitute* the packet (by supplying the missing header fields).

The reason for elliding the header fields has to do with trust. For example, if an upstream node has initiated a flow to the corporate public FTP server which has been switched, it can then dally a bit, and then inject packets (to the corporate billing server, say) into the flow. If the entire header were sent across the firewall, it is possible that because of a programming *error* at the terminus of the flow, the invalid packets are routed to the very machine the firewall was trying to protect. By elliding the header fields, we prevent such an attack from taking place.

Needless to say, there is a significant downside to our wire representation. First off, if a node receives a packet on an "unknown" label, it is unable to do anything with the packet (since it cannot reconstitute the original packet). Second, if a packet sent on a non-default label is captured "on the wire" (by a program like *tcpdump(1)* [14], say), there is "outside" knowledge that needs to be applied before interpreting the packet.[9]

6.2 TTL

The goal of our design is to be able to use IP switching to forward packets while, at the same time, remaining subservient to IP routing for determining paths through the network and obeying all the rest of the IP semantics. One of the basic requirements of IP is that the Time To Live (TTL) field of the IP header in a packet be decremented at each node and, if it reaches zero, the packet be discarded (and an ICMP Time Exceeded error message [28] be returned to the source of the packet).

In the case where a flow is being switched, there is no process at intermediate switches decrementing the TTL and checking for a zero value. Thus, we need to add a bit of complexity to our system to make sure that packets flow only as far as they should and that error are generated. To accomplish this, our flow identifiers in redirection messages include a specific TTL value (i.e., packets which are the same in all fields except the TTL values are considered to be part of two distinct flows). This ensures that a packet with a TTL of zero will never be switched through a node. The price of this is an increase in the number of flows created.

6.3 IP Header Checksum

A complication is that the IP specification [29] requires a router to check the IP header checksum at each hop. We do not do this for packets which are being switched through a node. We do not provide a long justification for this practice; we merely note that the "next generation" of IP, IPv6 [9], does not even *include* a header checksum, supposedly because many current high-speed IP routers update, but do not check, the IP header checksum [13]. (In IPv6, the lack of an Internet header checksum has led to the specification that *transport* protocols, such as TCP and UDP, *must* include checksums which cover the so-called pseudo-header to protect applications against mis-delivery of datagrams; this same protection is provided in our system since the ultimate recipients of datagrams *do* check the IP header checksum, in addition to whatever transport-layer checking they may do.)

We do, however, require that each node that modifies the IP header checksum do so in an *error preserving* way, i.e., *not* by recomputing the checksum, but by *updating* it in such a way that correct checksums are taken to correct checksums and (more importantly) incorrect checksums are taken to incorrect checksums. This means that at the next downstream node that checks the checksum (possibly the ultimate destination) will detect any errors that may have crept into the IP header during its transmission.

[8]There *is* an issue of the support provided by the previous generation (and some current generation) of ATM switches for multicast in general – it has tended to be added as an afterthought. Multicast is difficult to implement in many designs, but we hope that many ATM designers understand the importance of strong support of multicast.

[9]In actual fact, we reconstitute the packet *before* handing the packet to tcpdump(1).

6.4 TTL and Header Checksum

There is a link between our wire representation, IP switching, and the IP header checksum. The node which needs to convert between our wire representation and the "canonical" IP representation needs to include the correct value for the TTL field in the canonical representation. However, this node does not know what the value for the TTL field was when the header checksum was computed. This is because the header checksum *may* have been computed by its immediate upstream neighbor, in which case the value of the TTL field was the value known to the node; however, if the immediate upstream neighbor had switched the packet from one of *its* upstream neighbors, the value of the TTL field when the header checksum was computed will have been greater than the value associated with the incoming label.

In order to deal with this, we logically *subtract* the value of the TTL field from the header checksum in a packet at the point where the packet is being converted from canonical IP representation to our link-specific wire representation. The node which is converting back to the canonical IP representation then *adds* the expected value for the TTL field (i.e., the value associated with the flow associated with the label on which the packet has been received). The effect of this, assuming no errors are introduced in the transmission path, is that the header checksum contains the value it would have contained had the packet been forwarded hop-by-hop (and the TTL decremented and the checksum updated) by all the upstream nodes. Since we perform the subtraction and addition in an *error preserving* way (as mentioned above), the effect of errors introduced into the IP header in the transmission path is that an incorrect header checksum will be be produced, which will cause the packet to be rejected the next time the header checksum is checked.

6.5 Fragmentation

When Mogul and Kent wrote their paper *Fragmentation Considered Harmful* [15], they didn't know the half!

Remember that our flow type 1 includes (TCP or UDP) source and destination port numbers. Unfortunately, fragmented IP datagrams carry these port numbers *only* in the first fragment of the datagram. We *could* send subsequent fragments on a type 2 flow, or on the default flow, but this seems non-optimal. In particular, we would like to avoid introducing any more re-ordering than necessary for packets in a given flow; sending packets for the same flow on different labels risks introducing systematic reordering (of NFS traffic, say) for the lifetime of the flow.

As an implementation (not a protocol) issue, we keep track of the identification field from the IP header of recently seen packets that are part of a given flow. If a subsequent packet arrives that contains other than the first fragment of a datagram, we compare the identification field in the fragment with our list of recently seen identification fields. If we find a match, we send the datagram on the type 1 flow associated with the recently seen identification field.

6.6 Packet Shredding

From a data networking point of view, one of the features of ATM that presents difficulties is the fact that the unit of transmission at the data link layer (cells) is not the same as that at the internetwork layer (packets). In particular, the unit of *loss* at the two layers is different, and this presents problems [34].

The protocol that reassembles packets from cells, AAL5, detects the last cell of a frame via a bit in this last cell of the frame. If the last cell in frame N is dropped, then frames N and N+1 are *spliced* and are taken by the reassembly process to be one frame. Most likely, the CRC which AAL5 uses to protect against errors will detect this corruption, and so both frames N and N+1 will be dropped [25].[10]

Our system is sensitive to these errors in the following way: when we decide to start or stop switching the *packets* in a given flow, we do this by directing the local ATM switch to change the mapping for *cells* received over the incoming interface/label. This remapping occurs on a *cell* boundary. If this cell where the remapping takes place is other than the first cell in a packet, one or two packets will be effectively "shredded".

Additionally, when we remap a flow, there is the possibility of a short burst of out of order packets being delivered to the ultimate destination. For example, if the flow was being processed hop-by-hop, and is then switched, the "earlier" hop-by-hop, packets may well be overtaken by later, switched, packets.

6.7 Cell Interleaving

If cells from more than one packet are interleaved over the same VCI, the reassembling AAL5 will think that each cell is from the same packet, and so will reassemble a "Frankenstein's monster" of a packet (which, again, will likely be discarded as a result of AAL5's protective CRC [25]).

This introduces a scalability problem. If a given router is a *junction* point for a number of flows from upstream, all of which will follow the same path going downstream, these flows cannot be combined by switching, but must be presented to the router and, thus, combined into a single flow by intervention on a packet-by-packet basis by the router.

6.8 Address Resolution/Location

One advantage of a shared, broadcast subnet, such as Ethernet, is that a host on the subnet can be located by broadcasting an address resolution request [27]. While the *exact* same issue does not arise in our scheme, something similar does happen. In an Ethernet network, often it is possible to unplug a system from one port (in an Ethernet switch, say) and into another port (of the same switch) without needing to reconfigure any part of the system. In order to provide the ability for the same sort of flexibility in our system, the IFMP adjacency protocol (see section 3.6) allows systems to advertise their configured IP address(es) to their peers. By coupling this with a routing protocol (such as OSPF [19]) which supports so-called *host routes* a certain amount of dynamism in terms of physical connectivity is supported.

7. OUR IMPLEMENTATION

We have implemented two separate systems — one host and one IP switch — using the approach discussed in this paper. Both systems are based on a 4.4BSD-based kernel (FreeBSD 2.0.5).

The IP switch can control an ATM switch and/or connect to an Ethernet, FDDI, or other, local area network. For the IP switch, we have heavily modified the lower levels of the

[10]It is interesting to note that had AAL5 been designed with both an "end of frame" bit and a "beginning of frame" bit, only frame N would have been dropped.

kernel, both for performance and for the funcitonality we require. Figure 9 is a representation of the internal structure of the IP switch.

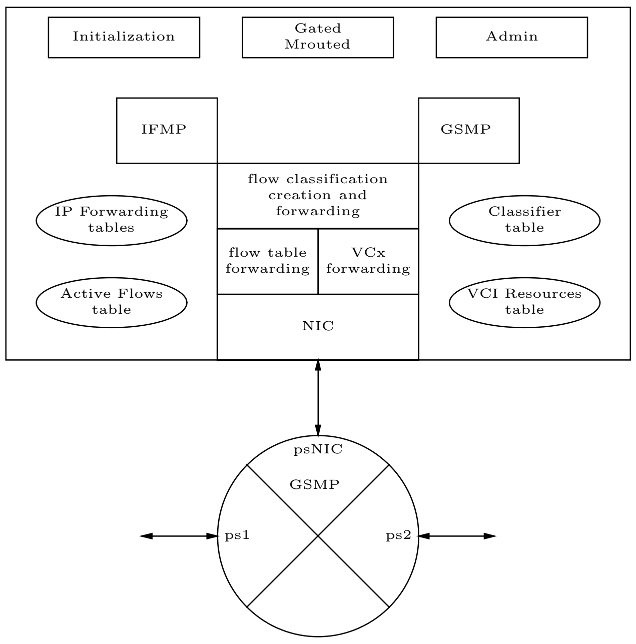

Figure 9: IP switch Internal Structure

At initialization, the default label (VPI 0/VCI 15) on each external interface is mapped to deliver frames through the *psNIC* interface to the IP switch controller. Packets received on one of these default labels are first looked up in the *Active Flows Table* (AFT). If no entry is found in the AFT, the packet is classified and (in the normal case) forwarding information is determined all of which is stored in a new entry in the AFT. As a byproduct of this, a note is recorded in the entry in the IP forwarding table used to determine the fowarding information; should this entry change in the future, the entry in the AFT will be updated. Succeeding packets in the same flow will be discovered in the AFT which caches the forwarding information, allowing the packet to be forwarded without any recourse to the IP forwarding table. When the classifier determines that this flow should be redirected to a specific label, an IFMP packet will be sent upstream (see Figure 5). When a packet arrives on a specific label, the specific entry in the AFT (previously associated with the specific label) is accessed directly, and the previously cached forwarding decision is replayed; in this case (known as *VCx forwarding*) the *lookup* in the AFT is bypassed.

For the host, we constrained ourselves to only add a (somewhat large) device driver to the system, and otherwise not modify the kernel or any other part of the system in any way. The host implementation, which talks over a standard ATM card, looks similar to the implementation in the IP switch; the most notable difference is that the host implementation has no notion of the IP forwarding table (and, thus, there is no need for the dependency mechanism). We are in the process of porting the host implementation to other hardware/software platforms.

We are currently using a small network (consisting of 5 IP switch systems in series) as the operational network for about seventeen members of our group.

8. FUTURES

8.1 Flow Granularity

Our current protocol specifies relatively fine-grained flows; the coarsest is approximately host-to-host while the finest grained flow is approximately application to application. We feel this level of granularity matches well with the needs of routers "lower down" in the Internet infrastructure (at the corporate/campus backbone and/or departmental/branch office level). However, for routing higher up in the infrastructure (within the backbone of the Internet, say), fine-grained flows do not scale as the numbers are probably too high. We will be exploring what is required in coarseness to provide service at that level of the Internet.

8.2 *Frameization*

As mentioned above, the fact that we are conceptually working at the level of packets, but ATM works at the level of cells, causes certain problems. One possible solution to this class of problems is to make switches more aware of packets. For example, the shredding problem introduced in section 6.6 could be solved by having switches execute remapping operations only on frame boundaries. Similarly, switches could (try to) transmit a complete frame's worth of cells before transmitting any other cells.

We could also cause host adapters to transmit all the cells from one frame before transmitting cells from another frame.

Notice that none of these modifications would necessarily eliminate the reordering discussed in section 6.6.

8.3 Flows as Manageable Objects

We have defined type 1 and type 2 flows as a mechanism to allow us to use hardware (ATM hardware, in this case) to speed up forwarding by switching rather than routing. Having defined flows in this way, however, we expose an opportunity to have network administrators specify policies for different classes of flows. A simple firewall, for example, would say that "packets that look like that should be directed to /dev/null". Or, different amounts of system resources could be allocated to different classes of flows.

8.4 QOS

With what we have implemented in IFMP and GSMP our switching path looks very much like a very simple, single class, tail drop router (as does our forwarding path in the IP switch controller, for that matter). If the ATM switch implements an algorithm like RED [10], a better version of this single class router is available.

However, it is quite often the case that an administrator would like to give different service levels to different classes of flows. Examples of these different service levels are well known [11, 6].

The current forwarding model implemented in the ATM switch is very straightforward, and therefore was quite easy to define (so easy, that it is hard to claim we even defined it, as it was really already there in the ATM switch). With a few very simple operations on the switch (basically, map label X to label Y), we are able to have the switch do exactly as IP routing would have done. This is primarily because the IP routing semantics are so loose: "send this packet out that interface to that next hop"

It would be desirable to support different service levels in the IP switch. For example, the RSVP protocol [37] could be used to communicate desired service levels to the IP switch. However, if we allow different service levels, we will need to modify GSMP in order to specify some simple operations on ATM switches which allow the various policies flowing out of the different service levels to be implemented. Our suspicion is that defining such a set is going to be difficult.

8.5 Layer 2 Switches

This paper has discussed IP switching running on ATM switches. However, our methods are equally appropriate for other point-to-point switching technologies, such as frame relay [3].

8.6 IFMP on Shared Multicast Media

Above, we discussed the reasons we have chosen to make IFMP redirection driven by the downstream node. Having made that decision, we are faced with how to generalize IFMP to work on a shared multicast media (such as some sort of hack to run over Ethernet by overloading some field in the Ethernet header to carry label information). The problem here is that if want to be able to redirect multicast packets, then *all* the receivers need to agree on which labels are valid, and the binding between labels and flows. Concepts such as the "designated router" of OSPF [19] and IS-IS [23] may prove valuable here, but we have not as yet done much work in this area.

8.7 Virtual Paths

ATM offers a facility known as *virtual paths* which are coarser than virtual circuits. Basically, one virtual path consists of a number of virtual circuits. Our current system does not provide support for virtual paths, but we have the beginnings of what such support might look like.

We would provide virtual path support to connect IP switches across a public (or private) non-IFMP ATM network. The non-IFMP ATM network would be provisioned to offer a virtual path between a pair of IP switches. Labels exchanged by IFMP would be relative to the configured virtual path. One physical link to the non-IFMP ATM network could be configured with a number of virtual paths, each terminating at a different IP switch. The IP switch would treat each such virtual path as a separate logical interface.

This would allow for packets to be switched from ingress point to egress point with no handling by forwarding code (except during flow setup and route transitions). However, packets would enter and leave the non-IFMP ATM network multiple times. (Clearly, locating an IP switch inside the public ATM network would be preferable.)

9. RELATED WORK

The traditional approach to running IP over ATM has been to layer a new addressing scheme on top of ATM, thus *creating* problems which our approach completely avoids. This approach has also tended to lead to viewing ATM as the cloud, with IP routers interfacing from the edges of this cloud. For a good summary, including a full set of references, of this work, see [1]; [22] provides a fuller critique of the traditional approach.

A number of projects have focussed in on the concept of using ATM as a backplane- or bus-replacement within a router itself. Such a project is that of Parulkar, et al [26], which designs a system to integrate IP and ATM. In this design, the line cards process IP datagrams, using the ATM switch/backplane to query special *routing cards* for forwarding information and then for forwarding the packet to the output line card. Integral in the design are two types of interfaces: line cards which connect to hosts or "conventional" routers, and special router interfaces which connect to other IP/ATM routers. The second type of interface uses a mechanism to bind individual flows to specialized VCIs to allow ATM-level switching of datagrams.

ATM typically runs over SONET. One approach to running IP over ATM is to bypass ATM completely by running IP directly over SONET [36]. This approach allows for straight-forward interconnection of systems via SONET lines, but doesn't provide for any hardware-assisted speedup of forwarding.

The BBN Multigigabit router project [24] is building a router which is in many ways similar in its *internal* architecture to our system (as well as being similar in many ways to [26] mentioned above). A general purpose computer runs routing protocols and builds up forwarding tables; this computer is connected to interface cards and forwarding engines via a PCI bus. Communication between interface cards and forwarding engines is via an internal, 50 Gigabit/second crossbar switch.

The approach we have seen which is closest to ours is formally unpublished work of Masataka Ohta (Tokyo Institute of Technology), and Hiroshi Esaki and Ken-ichi Nagami (both of Toshiba Corporation), and related, similarly formally unpublished work of Yukinori Goto (of the Nara Institute of Science and Technology).[11] The Ohta, et al, work is known as "Conventional IP over ATM"; the work of Goto is known as "Session Identity Notification Protocol". The Ohta work proposes treating ATM switches as routers, which is quite similar to our work. Ohta proposes using QoS signalling protocols (such as RSVP) to set up flow-specific VCIs. There is some discussion in Ohta's work of issues with respect to TTLs. The Goto work involves a mechanism for binding a specific flow to a VCI, and is again very similar to our work. The difference in the binding mechanism is primarily that Goto proposes having the *upstream* node announce its intention to send traffic on a certain VCI and then commence sending. We feel that it is preferable to have the downstream node make the redirection decision (for reasons discussed in section 3.4.1).

10. ACKNOWLEDGEMENTS

Http://www.research.att.com/biblio.html, the network bibliography maintained by Henning Schulzrinne, was extremely helpful in the writing of this paper. Thanks to K. Claffy and L. Huston.

11. REFERENCES

[1] A. Alles. ATM internetworking, May 1995. http://www.cisco.com/warp/public/614/12.html.

[11] As of this writing, both the Ohta and Goto papers have only been published as Internet Drafts, an ephemeral publication.

[2] BBN Inc. Specifications for the interconnection of a host and an IMP. Report 1822, Bolt Beranek and Newman, Inc., December 1983.

[3] T. Bradley, C. Brown, and A. Malis. Multiprotocol interconnect over frame relay. Request for Comments (Draft Standard) RFC 1490, Internet Engineering Task Force, July 1993, ftp://ds.internic.net/rfc/rfc1490.txt. (Obsoletes RFC1294).

[4] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: repelling the wily hacker*. Addison-Wesley, Reading, Massachusetts, 1994.

[5] Kimberly C. Claffy, Hans-Werner Braun, and George C. Polyzos. A parameterizable methodology for Internet traffic profiling. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995, ftp://ftp.sdsc.edu/pub/sdsc/anr/papers/flows.ps.Z.

[6] David D. Clark, Scott Shenker, and Lixia Zhang. Supporting real-time applications in an integrated services packet network: architecture and mechanism. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 14–26, Baltimore, Maryland, August 1992. ACM, http://ana-www.lcs.mit.edu/anaweb/pdf-papers/csz.pdf. *Computer Communication Review*, Volume 22, Number 4.

[7] Martin de Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN*. Ellis Horwood, Chichester, England, 1991.

[8] S. Deering. Host extensions for IP multicasting. Request for Comments (Standard) STD 5, RFC 1112, Internet Engineering Task Force, August 1989, ftp://ds.internic.net/rfc/rfc1112.txt. (Obsoletes RFC0988).

[9] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. Request for Comments (Proposed Standard) RFC 1883, Internet Engineering Task Force, January 1996, ftp://ds.internic.net/rfc/rfc1883.txt.

[10] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993, http://www-nrg.ee.lbl.gov/floyd/red.html.

[11] Sally Floyd and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):22, August 1995, ftp://ftp.ee.lbl.gov/papers/link.ps.Z.

[12] Charles Hornig. A standard for the transmission of ip datagrams over ethernet networks. RFC 894, Internet Engineering Task Force, April 1984, ftp://ds.internic.net/rfc/rfc894.txt.

[13] Christian Huitema. *IPv6: The new Internet Protocol*. Prentice Hall, Upper Saddle River, New Jersey, 1996.

[14] Van Jacobson et al. *tcpdump(1), BPF*, 1990, ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[15] Christopher A. Kent and Jeffrey C. Mogul. Fragmentation considered harmful. *ACM Computer Communication Review*, 17(5):390–401, 1987. SIGCOMM '87 Workshop.

[16] Robert M. Metcalfe and David R. Boggs. Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395–404, July 1976.

[17] D. Mills and H. Braun. The NSFNET backbone network. *ACM Computer Communication Review*, 17(5), August 1987. SIGCOMM '87 Workshop.

[18] David L. Mills. The Fuzzball. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 115–122, Stanford, California, August 1988. ACM. also in *Computer Communication Review* 18 (4), Aug. 1988.

[19] J. Moy. OSPF version 2. Request for Comments (Draft Standard) RFC 1583, Internet Engineering Task Force, March 1994, ftp://ds.internic.net/rfc/rfc1583.ps. (Obsoletes RFC1247).

[20] P. Newman, W. L. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. General switch management protocol specification version 1.0. Technical report, Ipsilon and Sprint, February 1996, http://www.ipsilon.com.

[21] P. Newman, W. L. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall. Ipsilon flow management protocol specification for ipv4 version 1.0. Technical report, Ipsilon and Sprint, February 1996, http://www.ipsilon.com.

[22] P. Newman, T. Lyon, and G. Minshall. Flow labelled ip: A connectionless approach to atm. In *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, page 1, April 1996, http://www.ipsilon.com/ pn/papers/infocom96.ps.

[23] D. Oran. OSI IS-IS intra-domain routing protocol. Request for Comments (Informational) RFC 1142, Internet Engineering Task Force, December 1991, ftp://ds.internic.net/rfc/rfc1142.ps.

[24] Craig Partridge. private communication, March 1996.

[25] Craig Partridge, Jim Hughes, and Jonathan Stone. Performance of checksums and CRCs over real data. In *SIGCOMM Symposium on Communications Architectures and Protocols*, page 9, Cambridge, Massachusetts, September 1995.

[26] Guru Parulkar, Douglas C. Schmidt, and Jonathan Turner. IP/ATM: A strategy for integrating IP with ATM. In *SIGCOMM Symposium on Communications Architectures and Protocols*, page 10, Cambridge, Massachusetts, September 1995.

[27] D. Plummer. Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware. Request for Comments (Standard) RFC 826, Internet Engineering Task Force, November 1982, ftp://ds.internic.net/rfc/rfc826.txt.

[28] J. Postel. Internet control message protocol. Request for Comments (Standard) STD 5, RFC 792, Internet Engineering Task Force, September 1981, ftp://ds.internic.net/rfc/rfc792.txt. (Obsoletes RFC0777).

[29] J. Postel. Internet protocol. Request for Comments (Standard) RFC 791, Internet Engineering Task Force, September 1981, ftp://ds.internic.net/rfc/rfc791.txt. (Obsoletes RFC0760).

[30] J. Postel. Transmission control protocol. Request for Comments (Standard) STD 7, RFC 793, Internet

Engineering Task Force, September 1981, ftp://ds.internic.net/rfc/rfc793.txt.

[31] Jon Postel. Address mappings. RFC 796, Internet Engineering Task Force, September 1981, ftp://ds.internic.net/rfc/rfc796.txt.

[32] Jon Postel. A standard for the transmission of ip datagrams over experimental ethernet networks. RFC 895, Internet Engineering Task Force, April 1984, ftp://ds.internic.net/rfc/rfc895.txt.

[33] Yakov Rekhter. private communication, January 1996.

[34] A. Romanow and S. Floyd. The dynamics of TCP traffic over ATM networks. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 79–88, London, UK, September 1994.

[35] J. F. Shoch. An introduction to the ethernet specification. *ACM Computer Communication Review*, 11(3), July 1981.

[36] W. Simpson. PPP over SONET/SDH. Request for Comments (Proposed Standard) RFC 1619, Internet Engineering Task Force, May 1994, ftp://ds.internic.net/rfc/rfc1619.txt.

[37] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP: a new resource ReSerVation protocol. *IEEE Network*, 7(5):8–18, September 1993, ftp://parcftp.xerox.com/pub/net-research/rsvp.ps.Z.