Anypoint: Extensible Transport Switching on the Edge

Kenneth G. Yocum, Darrell C. Anderson*, Jeffrey S. Chase, and Amin M. Vahdat
Department of Computer Science†
Duke University
{grant,anderson,chase,vahdat}@cs.duke.edu

Abstract

Anypoint is a new model for one-to-many communication with ensemble sites—aggregations of end nodes that appear to the external Internet as a unified site. Policies for routing Anypoint traffic are defined by application-layer plugins residing in extensible routers at the ensemble edge. Anypoint's switching functions operate at the transport layer at the granularity of transport frames. Communication over an Anypoint connection preserves end-to-end transport rate control, partial ordering, and reliable delivery. Experimental results from a host-based Anypoint prototype and an NFS storage router application show that Anypoint is a powerful technique for virtualizing and extending cluster services, and is amenable to implementation in high-speed switches. The Anypoint prototype improves storage router throughput by 29% relative to a TCP proxy.

1 Introduction

This paper presents the design and implementation of Anypoint, a new architecture for transparent communication with *ensemble sites* such as cluster-based network services. Intermediary routers at the ensemble border act to mediate communication with the ensemble, presenting it as a single virtual site to the outside network. Anypoint is implemented as a new set of functions for extensible switches.

Anypoint provides *transport switching*; it is the first general indirection approach that operates at the granularity of transport *frames*. *Transport switching* enables reliable, ordered, rate-controlled communication to the ensemble through a redirecting switch. Unlike a TCP proxy, an Anypoint switch does not terminate transport connections. Instead Anypoint switching functions transform each frame to maintain transport-layer guarantees between end nodes. Anypoint is complementary to IP-layer Internet indirection architectures such as Anycast [33] and *i3* [41].

A key goal of Anypoint is to generalize "L4-L7" server switches that support load balancing and content-aware request routing for Web server clusters. Previous work (e.g., [31, 8, 9, 18]) demonstrated the importance of content-aware request routing for Web services, and the challenges of supporting it, particularly with persistent connections [28]. Web switch architectures are limited to handle each request in a separate transport connection, or to process requests on each persistent connection serially; the former increases overheads and provides no ordering guarantees, and the latter limits concurrency and imposes head-of-line blocking for delayed packets or large requests. While these restrictions still exist for HTTP, redirection architectures that depend on them cannot extend to other services including network storage protocols, which have also been shown to benefit from content-aware request routing [5].

To overcome this challenge, we designed Anypoint for advanced IP transports with partially ordered application-level framing (ALF), as proposed by Clark and Tennenhouse over a decade ago [14]. These features are present in emerging IP transports such as SCTP [40] and DCCP [27]. We show how redirecting switches can leverage framing to enable an approach that is both more powerful and more elegant than Web switches and other solutions constrained by TCP. Our premise is that IP-based services—including network storage, general RPC-based services, and next-generation Web services using SOAP/HTTP—will migrate from TCP to these new transports. Our goal is to define a redirecting switch that accommodates pluggable indirection policies for a wide range of service protocols, not limited to HTTP over TCP. This generality is in the spirit of Active Networks [44] and subsequent proposals for extensible routers [17, 29, 38].

The contributions of this paper are to: (1) show that transport frame switching at the network edge is a powerful technique to virtualize and extend Internet services, (2) define an extensible framework and mechanisms to enable this technique, (3) present experimental results demonstrating the use of Anypoint for an NFS storage router, (4) compare the behavior of the Anypoint pro-

*D. Anderson is currently at Google.

†This work is supported in part by the U.S. National Science Foundation (CCR-00-82912, EIA-9972879, and ANI-126231), Network Appliance, and Cisco Systems, and by an equipment grant from IBM.

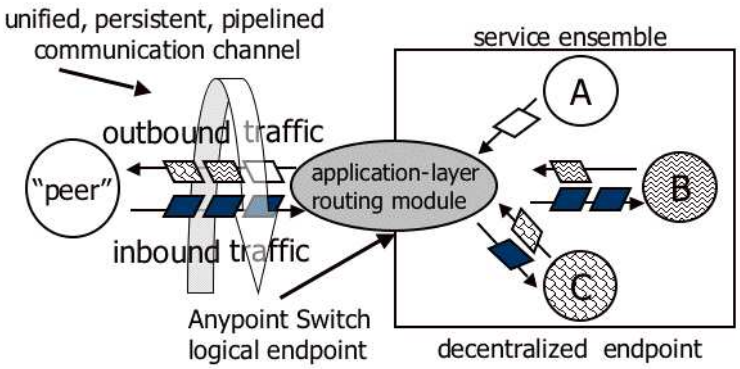

Figure 1: A connection between a peer (client) and an Anypoint server ensemble. The ensemble appears to the peer as a single virtual server; an Anypoint switch at the ensemble border transforms the packets to route the connection's traffic according to policies defined by switch plugins (Application Layer Routing Modules or ALRMs). In addition to simple load balancing, an ALRM can implement content-aware routing policies to improve service performance and robustness. In contrast to current Web switches, Anypoint enables concurrent handling of pipelined requests while preserving ordering constraints at the transport layer; thus it is useful for a general class of service protocols including IP storage.

totype to an alternative structure using application-level proxies [19, 39], and (5) explore the implications for Internet service structure, extensible switches, and IP transport protocols.

2 Motivation and Overview

Figure 1 illustrates the Anypoint abstraction. An Anypoint connection allows communication between two logical endpoints: an *ensemble* of end nodes and another *peer* IP site. In a typical use, the ensemble is a server cluster and the peer is a client interacting with it through some request/response service protocol. Many connections may be active to the same ensemble. The current ensemble membership for a connection is its *active set*. Anypoint switches direct the traffic flow between each peer and its active set under the control of the ALRM plugins. This serves four related purposes:

- **Dynamic request redirection.** Anypoint switches direct inbound requests to selected servers according to service-specific policies in the ALRM. The ALRM may consider information above the transport layer, such as client identity, client location, or the nature of the request, as well as server status and traffic conditions. Content-aware policies can optimize server cache effectiveness as well as balance load [31, 18, 5], as in the NFS storage router example in Section 3.2.

- **Server resource management.** Switch-based redirection enables flexible resource provisioning in Internet server clusters. A management interface in the switch allows the service to reassign server resources with dynamic, coordinated changes to its active sets [7, 13], without relying on the client to select a new server or refresh a DNS cache.

- **Response merging.** Anypoint supports direct delivery of content to each peer from multiple ensemble nodes [22, 5]. The peer receives the response traffic on a single connection and assembles it using ordering information inserted by lightweight translation functions at the switch. The reassembly buffer resides in the peer, not in the switch.

- **Service composition.** ALRMs can act as "wrappers" to compose or extend services. For example, an ALRM might support mirroring for an ordered multicast of request traffic across replicated servers. Since the ALRM understands the service protocol, it might distribute read traffic evenly and mirror only those requests that modify service state.

This paper focuses on the first three goals, i.e., the role of Anypoint as a basis for virtualized, manageable, scalable Internet services.

Redirecting switches are controversial because the Internet architecture implicitly assumes that each IP datagram is addressed and delivered to a uniquely defined end host, running a single instance of its operating system (cf. RFC 1122 [12]). Anypoint provides a rich set of capabilities enabling an ensemble operating system and its applications to manage the ensemble as a coordinated virtual "host"—effectively a multicomputer with internal policies for handling network traffic addressed to it. Cru-

cially, the indirection hides the ensemble's internal structure from the connection peer: the peer addresses inbound traffic to a *virtual IP address* (VIP) for the ensemble and receives outbound traffic with that VIP as the source address. This use of network address translation (NAT)—which may be damaging in other contexts—is done with the guidance and consent of the ensemble applications, and is transparent to the peer. The switch does not obscure the identity of the peer from the ensemble members.

2.1 Anypoint and the Transport

Anypoint extends IP transports to decentralized (ensemble) endpoints in a way that supports the key transport properties of rate-controlled, ordered, reliable delivery. A core principle of Anypoint is *transport equivalence*. To end nodes, an Anypoint connection is equivalent to a point-to-point connection. The burden of maintaining these properties—reorder buffering and reassembly, duplicate suppression, acknowledgment, retransmission—are handled in the usual end-to-end fashion by the end nodes. This minimizes buffering and processing overhead in the switch.

As previously stated, Anypoint assumes use of transports with framing [14]. Our purpose is not to propose a new transport, but rather to construct an indirection architecture that generalizes to a class of service protocols and framed transports. A *frame* is a variable-sized, self-contained sequence of bytes sent as a unit over a transport connection; frames are semantically meaningful and independently processed at the application layer. The service protocol uses the transport API to map its requests and responses into transport frames. Depending on the transport, a frame could span multiple transport segments (packets), or a single segment might contain multiple frames. Frame boundaries are recognizable from the transport headers, so a receiver may process frames independently as segments arrive.

Framing and partial ordering enable network-level processing of transport flows, which is useful for network adapters with protocol offload or direct-access features (e.g., RDMA), as well as for virtualization switches and routers. Frames are the granularity of transformation, redirection, and merging in Anypoint. While a service protocol may send frames over TCP, TCP imposes a total order even when the service does not require it; this precludes redirecting frames concurrently to different nodes, which could violate the delivery order. To permit effective fine-grained (e.g., content-aware) redirection, the service protocol and transport must not constrain frame order unnecessarily. However, Anypoint allows the service and transport to specify partial frame orderings if necessary for correctness. SCTP is one example of a transport that supports a partial order on frames.

Because the switch performs transport-layer processing, we envision the Anypoint switches residing in a data center or enterprise, with ensemble members colocated within a single security and routing domain. For example, transport switches must access packet state beyond the IP headers, which may be encrypted (e.g., with IPsec) across the external network. While packets from a given Anypoint connection must pass through a single intermediary, they use the service's VIP address to maintain this invariant even in the presence of asymmetric routing.

3 Anypoint Services

The Anypoint architecture factors service implementations into (1) a simple, lightweight ALRM installed as a switch extension, and (2) a server component running on the ensemble nodes, which handles complex issues such as server coordination, group membership, and recovery.

3.1 ALRMs

Anypoint ALRMs are plug-in software modules that manage Anypoint traffic through the switch. Each ALRM implements frame redirection and transformation logic for a given service. ALRMs are installed through a privileged management interface.

An ALRM is an event-driven, asynchronous, deterministic, non-blocking module supporting the interface in Table 1. ALRMs affect only traffic for specific designated ports. At most one ALRM is bound to each connection. ALRMs may access only the frame data and private structures of bounded size. We intend that the processing required per frame is statically bounded, and that the ALRM's memory references are statically verifiable.

Inside the switch, common low-level functions examine transport headers to classify packets into flows (connections), pass each frame to the *redirect()* function of the ALRM bound to its flow, and merge the transformed frames into the destination transport stream for the selected target node. These functions manipulate transport headers to extract frames from incoming transport segments and pack them into outgoing segments. The ALRM itself may perform deep processing beyond the transport headers.

ALRMs must be deterministic so they do not apply inconsistent transforms or violate exactly-once delivery for retransmitted frames. The RTX handle passed to *redirect()* indexes a unique state field for each distinct frame, enabling the ALRM to cache arbitrary state for each transformed frame. When the receiver is known to have processed an ack for the frame, the switch calls *retire(*rtx)* so the ALRM may recycle this state.

Upcalls from Anypoint switch transport functions.	
redirect(frame,id,*len,src,*rtx)	Redirect a frame on connection id. If it is inbound to the ensemble, select an ensemble member to receive it.
retire(*rtx)	Retire rtx state associated with an inactive frame.
{add,rm}_conn(id)	Add a new connection with ID, or delete a connection with ID.
alrm_{init,uninit}(void)	Start up or shutdown this ALRM.

Table 1: Registered upcall interface for an Anypoint ALRM.

3.2 Example: NFS Storage Router

To illustrate the structure of Anypoint services, we outline use of Anypoint for an L7 storage router for the NFSv3 file system protocol. We refer to this system as "Slice-lite" or *Slite* because it is a simplified form of the Slice storage architecture [5, 4]. Slite presents an ensemble of standard NFS servers as a single, unified, virtual NFS server. As in Slice, the directory tree is partitioned into subtree buckets and spread across the servers as it grows. This partitioning uses a *mkdir switching* policy, with a parameter for subtree granularity. A soft-state map tracks the assignment of subtree buckets to nodes.

The Slite ALRM uses content-aware routing to direct NFS requests to the assigned servers. Slice has previously shown that NFS is amenable to virtualization using this technique because most NFS operations apply to a single content item—a directory, name entry, or file—evident from the request type and arguments. The ALRM extracts a key identifying the subtree bucket from each request's NFS file handle at a known offset, and uses it to index the map to identify the assigned server. This indirection allows dynamic rebalancing of the buckets across the ensemble; this technique is common to cluster-based Internet services [31, 21, 36, 23]. With Slite, as in Slice and these other services, fine-grained content-aware request routing improves locality of the server caches as well as balancing load. A switch-based implementation can deliver the best latency and bandwidth, both of which are critical for file services.

Slite differs from Slice in two key respects. First, Anypoint enables Slite to run the NFSv3 protocol over a reliable, rate-controlled transport; Slice is limited to NFS/UDP, which sends one request or response per packet and relies on the RPC layer for primitive rate control and retransmission. Second, Slite is easier to deploy and it can use standard NFS servers. Although both are NFS-compliant, Slite is not Posix-compliant because it does not support *readdirplus*, *link*, and *rename* operations that cross server boundaries. While still compatible with Anypoint, this coordination requires the more comprehensive Slice approach. We use Slite for simplicity.

The Slite ALRM redirects some complex operations to a designated back-end server derived from the *coordinator* in the Slice architecture. In particular, the coordinator executes namespace operations (*mkdir*, *rmdir*, *lookup*) on name entries that cross servers (*switched directories*). The coordinator uses a write-ahead intention logging protocol for failure atomicity [4]. To simplify the ALRM further, our prototype indirects *lookup* through the coordinator, which maintains an index of switched directories. These redirect cases are easy to encode in the ALRM.

s.una	CSN of the oldest unacknowledged frame.
s.last	CSN of the last frame.
s.next	LSN of the next frame.
s.hole	CSN of the oldest pending sequence hole from a source.
s.lastgap	CSN of the newest hole at the time of the last send to a sink.

Table 2: **Endpoint table entry**. For an outbound flow, s is a source, and the fields pertain to frames transmitted by s. For an inbound flow, s is a sink, and the fields pertain to frames received by s.

4 Inside the Anypoint Switch

This section outlines the transport switching functions within an Anypoint switch. Without loss of generality we consider the traffic on a single connection. This traffic consists of two partially ordered flows of frames passing through the switch: one flow *inbound* to the ensemble and one flow *outbound* from the ensemble. The n ensemble members in the connection's active set are *sources* for the outbound flow and *sinks* for the inbound flow.

We make the following assumptions about the transport. The transport senders on the end nodes mark the transmitted frames for each flow with *frame sequence numbers* (FSNs) that are unique within the flow, monotonically increasing, and consecutive. FSNs are the basis for reliable at-most-once delivery. Frames arriving at a receiver are delivered in FSN order, and the receiver uses FSNs to generate cumulative acks in the return flow. The transport limits the number of outstanding unacknowledged frames to a frame flow window w.

frame.csn	CSN for this frame.
frame.lsn	LSN for this frame.
frame.source	Source node ID (outbound).
frame.sink	Sink node ID (inbound).
frame.ack	Has receiver acknowledged?
frame.hole	Is this frame a pending hole?
frame.link	CSN of next chain entry.

Table 3: **Frame ring entry.**

The *transport equivalence* property means that end nodes do not distinguish Anypoint connections from point-to-point connections using the same transport. This implies that each participating node views the frames from each flow in a local FSN space, since it believes (at the transport layer) that it is the exclusive owner of its end of the connection. Since there is only one connection peer, it defines a global FSN space of *connection sequence numbers* (CSNs) for both flows in the connection. The switch's sequencing functions translate between the end nodes' FSN spaces—the *local sequence numbers* (LSNs) understood by each ensemble member and CSNs understood by the peer.

The Anypoint switch transforms the frames flowing through it to split inbound traffic across the LSN spaces of the n sinks, and to merge outbound traffic from the LSN spaces of the n sources into the peer's CSN space. Correct translation of the sequence numbers is the key to extending the transport's end-to-end ordering, duplicate suppression, acknowledgment, and retransmission functions to Anypoint connections because *it enables reassembly buffering and retransmission at the end nodes rather than in the switch*. The switch also coordinates the transport mechanisms for ordering and reliable delivery, and propagates rate control signals to ensure that each Anypoint connection behaves correctly with respect to flow control and congestion.

4.1 Per-Flow State

The Anypoint switch maintains per-flow control state proportional to the number of unacknowledged frames. This state consists of a *frame ring*—a circular queue of w frame entries—and an *endpoint table* with an entry for each active set member, as shown in Figure 2. Table 2 summarizes the state in each endpoint table entry. The frame ring contains an entry for each active frame in the sliding window ranging from the oldest active frame ($flow.left$) to the highest numbered frame ($flow.left + w$) eligible for transmission into the frame flow window. Entries become active as new frames arrive, and inactive as the left edge of the window passes them.

Every frame has a unique CSN; the ring entry for a frame

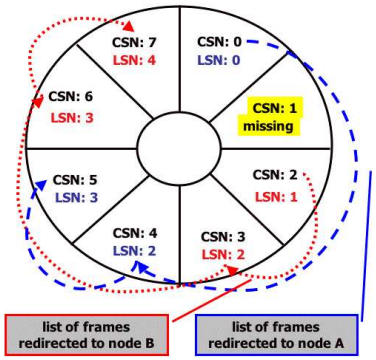

Figure 2: The Anypoint switch has one frame ring per flow, each of size w. In this example the ALRM redirects inbound frames to either sink A or sink B. The network has dropped CSN 1, so the switch inserts a gap of length 1 into each sink's LSN space. When CSN 1 arrives, the ALRM may direct the switch to deliver it to either A or B.

flow.left	CSN of the oldest frame for which the sender is not known to have received an ack.
flow.una	CSN of the oldest frame for which the receiver is not known to have sent an ack.
flow.next	CSN of this flow's next frame.
lastgap	CSN of the newest hole (inbound).
hole	CSN of the oldest hole.

Table 4: **Per-flow state variables.**

may be retrieved in constant time by indexing from the frame's CSN in the obvious fashion. Table 3 summarizes the frame ring entry, and Table 4 summarizes per-flow state variables maintained to index the frame ring. CSNs are also used to link frame entries in *frame chains* in CSN order. For outbound flows, n frame chains link the frames from each source s, including the holes originating from s. The chain for each source s is rooted in $s.una$. For inbound flows, the n frame chains link the active frames destined for each sink, with a separate chain for pending sequence holes (see below). Every active frame is linked into exactly one chain.

Given these data structures, many aspects of state maintenance for the endpoint table, frame ring, and flow variables are straightforward. The discussion below focuses on the more interesting aspects.

4.2 Sequencing and Acknowledgments

The switch translates frame sequence numbers between CSNs and LSNs as frames pass through it. This is trivial

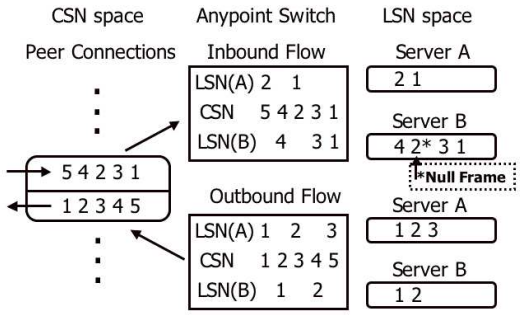

Figure 3: The Anypoint switch translates between the CSN space of the peer connection and each server's LSN space. The network reorders frames 2 and 3 on the *inbound* flow; a gap is inserted into server B's LSN space to maintain ordering if frame 2 is redirected to server B. However, frame 2 is redirected to server A, and a null frame is sent to server B.

in the common case of frames arriving in order: if the flow is outbound (merge), assign CSN $flow.next$; if it is inbound (split), assign LSN $sink.next$ for the frame's destination sink. Figure 3 shows resequencing between CSNs and LSNs for a single peer connection to two servers.

The frame ring and chains allow efficient handling of acks. For inbound flows, the sinks encode their outgoing acks as LSNs. To convert an LSN ack from $sink$ to a CSN for the peer, the switch traverses the frame chain for $sink$ from $sink.una$ and examines each frame's lsn to determine if the ack covers it. The ack passed through to the peer is the CSN of the oldest frame not acknowledged by any sink s: $min(s.una)$. For outbound flows, on receiving a CSN ack from the peer, the switch traverses the frame ring from $flow.una$ to identify frames covered by the ack and update their $source.una$. The LSN ack passed through to each source is its $source.una$.

The difficult sequencing cases involve reordered or dropped packets, which create sequence holes. To allow ordered, reliable delivery at the receiver, the switch must pass any sequence holes through to the receiver's sequence space. For example, if an **outbound** frame's LSN $frame.lsn > source.next$ for its source, then one or more frames from that source are delayed. The switch passes the gap through to the peer by assigning the frame CSN $flow.next + (frame.lsn - source.next)$, marking the intervening frame entries starting at CSN $flow.next$ as holes from that source, and updating $flow.next$ and $source.next$. On the other hand, if $frame.lsn < source.next$, then this is a delayed frame that fills an existing hole. To identify the frame's CSN, follow the source's frame chain forward from $source.hole$ to locate the hole. The switch then places the CSN in the frame before forwarding it to the the peer.

The more difficult case occurs when an **inbound** frame is delayed. Since the switch does not know which sink will receive a given delayed inbound frame, it must reserve an LSN in the local space for any $sink$ that receives a frame numbered after a pending hole. For each inbound frame, assign the LSN as $sink.next + h$, where h is the number of pending holes created since the last frame sent to the destination $sink$. The common case $h = 0$ is easily recognized with a check that $sink.lastgap = flow.lastgap$. Else h is the number of pending hole frames f with $f.csn > sink.last$; these are found by traversing the frame chain rooted at $flow.hole$. Note that each pending hole is visited exactly once for each destination sink that receives a frame while the hole is pending; inbound holes create at worst $O(n)$ extra work within the switch.

When an inbound frame destined for $sink$ arrives to fill a hole at CSN i, traverse the frame ring forward from i, visiting each entry. For each sink s, let f be the entry for the first non-hole frame encountered that was destined for s (if any frames were sent to s since the sequence gap opened at i). If $s = sink$, then redirect frame i to s. Else, send a null frame (*redirect patch*) to s to fill the gap in its local sequence space (see Figure 3 for an example). In each case, the LSN for the sent frame is $f.lsn - h$, where h is the number of holes encountered before f in the traversal, including the hole at i. If no frame for $sink$ is encountered before the end of the traversal (CSN $flow.next$), then redirect i to $sink$ with LSN $sink.next$. Again, the number of redirect patches per hole grows with the number of frames arriving while the hole is pending, and is bounded by $n - 1$ and w.

4.3 Rate Control

This section examines the role of the Anypoint switch in coordinating rate control. This discussion considers the traffic from the switch's viewpoint: because the switch must control each endpoint's rate independently, it views the traffic to or from each endpoint (the n active set members plus the peer) as distinct flows. (This contrasts with the previous section, in which the peer's view of a connection is a pair of flows, one inbound to the ensemble and one outbound from the ensemble.) The switch merges a *fan in* of n flows—one from each ensemble member—into the connection's outbound flow to the peer, and splits the peer's inbound flow into a *fan out* of n flows. As flows split and merge, the switch propagates rate control signals to avoid overflowing any receiver or network path. Transport equivalence implies that end nodes do not change their rate control policies to use Anypoint; the end nodes are not aware that a split or merge is occurring. Instead, it is the switch's responsibility to transform and coordinate these signals to induce the correct local behavior from the sources and produce the desired global outcome.

The switch observes rate control signals flowing through it, and can determine if forwarding a frame violates rate limits to the receiver. It can also send rate control signals to any sender. *Flow control* signals proactively limit the rate of the source; we assume that the transport allows a receiver to rate-limit a source by advertising a flow window. A switch may manipulate these windows to suit its needs [25]. *Congestion* signals cause a sender to reactively reduce its rate. For example, if the switch drops a packet, a TCP-friendly sender interprets the event as congestion in the usual fashion.

The policy choice is to determine how the switch uses these rate control signals to respond to observed conditions. But the switch cannot predict how the ALRM will route inbound traffic to the ensemble sinks, or what portion of the bandwidth back to the peer will be needed for each source. In either direction, it may optimistically oversubscribe the windows, conservatively rate-limit senders to avoid any overflow, or select any point on the continuum between these extremes. For example, for inbound traffic it may optimistically advertise the sum of the active set flow windows to the peer, or conservatively advertise the minimum window from any sink. For outbound traffic, it may advertise the peer's full window to each ensemble source, partition it evenly among the sources, or overcommit it to an arbitrary degree.

The conservative approaches may limit connection throughput, while the optimistic approaches may cause the switch to overflow a receiver or network path, forcing it to drop packets. A dropped inbound packet induces the peer to throttle its sending rate to the entire ensemble, even if just one sink overflows. The peer's inability to distinguish among ensemble nodes is fundamental to the Anypoint model; we accept it because we assume that the network and memory within the ensemble are well-provisioned in the common case, and aggregate throughput is more important than bandwidth from the peers when the ensemble is overcommitted. For outbound traffic, congestion on the path to the peer results in a lazy throttling of individual sources in the usual fasion.

4.4 Discussion

The switch mechanisms described in this section illustrate several key points about the Anypoint architecture. Most importantly, *transport equivalence* says that Anypoint does not affect the transport connection semantics perceived by the end nodes. This architectural choice yields several benefits:

- End nodes use the same transport code for point-to-point and Anypoint connections, and do not distinguish between them. All Anypoint-specific functions are local to the switch.

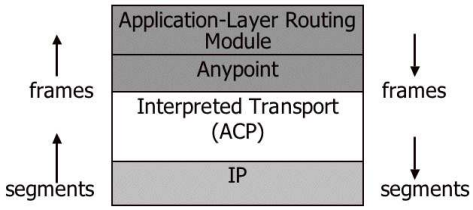

Figure 4: The switch extracts frames from incoming network segments and packs them into outgoing segments.

- Transport equivalence helps to ensure that Anypoint is compatible with all clients and servers implementing the transport, conforms to congestion conventions, and composes well with other components of the Internet architecture. For example, given appropriate ALRM support, both ends of an Anypoint connection could be ensembles, or Anypoint ensembles could nest arbitrarily in a hierarchy.

- Transport functions for sequencing and reliable at-most-once delivery continue to operate in an end-to-end fashion. The Anypoint switch never buffers data for rate control or transport reassembly; its role is to transform frames to coordinate these functions at the end nodes. The switch buffers packet data only as needed for port queues and frame assembly. This improves scalability of the switch architecture.

Although the switch maintains per-flow control state, it is bounded by the flow window w. Because acknowledgments and buffering remain end-to-end in Anypoint, a failed switch does not lose user data. However connections maintained by the failed switch must be re-initiated for the service to recover a failed session. Support for session recovery is common in new service protocols including iSCSI and DAFS [16].

Note also that no mutable state is shared across connections within an Anypoint switch. Thus an Anypoint switch design could spread frame processing load across processors at each external switch port. An ensemble may also partition communication traffic from different peers across multiple switches to further improve scalability.

5 Anypoint Prototype

We prototyped a host-based Anypoint switch as a set of kernel extensions to FreeBSD, implementing the transport switching mechanisms and ALRM interface. The prototype consists of 2080 lines of C code. We also implemented Anypoint ALRMs for the Slite service described in Section 3.2 and a simple clustered counter service for microbenchmarking.

5.1 ACP Transport

The Anypoint model can apply to a range of transports with the properties defined previously. In particular, we believe that Anypoint is compatible with SCTP. However, SCTP is a new protocol with complex features unrelated to Anypoint, and SCTP implementations are not yet fully mature. To experiment with Anypoint, we implemented a simple framed transport with a few hundred lines of code by reusing the FreeBSD TCP implementation, whose behavior is stable and reasonably well understood. We refer to this as Anypoint Control Protocol (ACP), although its functions are not Anypoint-specific.

ACP adds a shim with framing support based on a subset of the expired upper-layer framing (TUF) proposal [10]. New code at the kernel socket layer supports ACP sockets using a UDP-like message interface (*sendto, recvfrom*). Each message is sent as a frame, and frame boundaries are preserved at the receiver; this made it easy to run NFS over ACP for the Slite experiments.

An ACP segment is identical to a TCP segment, except that ACP adds one or more framing headers to each segment's data to partition it into an integral number of frames with consecutive FSNs as defined in Section 4. Like TCP, ACP preserves the send order for all of a connection's frames routed to a given end node. However, ACP does not specify an order among frames to or from different ensemble nodes. ACP differs from TCP primarily in that ACP is not defined to require this ordering. ACP's mechanisms for reliable delivery and rate control are indistinguishable from TCP (4.4 BSD Reno).

ACP segments are *self-describing*. The first frame header in each segment is aligned with the segment header so that an Anypoint switch can recognize frames even when segments arrive out of order. Each frame header contains a length field giving the offset of the next frame in the segment, if any. Our testbed network uses 9000-byte "Jumbo" Ethernet packets, and the maximum ACP frame size is 8936 bytes, which is less than the Maximum Segment Size (MSS). The ACP prototype never splits application frames across segments, although it may combine multiple frames in a single segment when space allows. In practice, this means that ACP often sends short segments, but it also frees the prototype from the need to "chunk" or reassemble frames across segment boundaries. Our ACP prototype does not support path MTU changes.

5.2 Switch Prototype

Figure 4 shows the protocol stack at the switch. Arriving IP packets are interpreted by a *veneer* layer that recognizes the transport, classifies flows, and extracts frames. The Anypoint layer maintains connection-related state, active set membership, and ALRM bindings as described in the previous sections. The TCP-derived ACP receiver uses TCP byte sequence numbers for reordering, and ignores FSNs. The switch also translates byte sequence numbers and caches them in the frame ring. It uses this state to identify the frames covered by acks, which are encoded as byte sequence numbers rather than FSNs.

The switch prototype validates TCP checksums for incoming ACP segments and recomputes a fresh checksum from scratch for transformed outgoing segments, using network cards with TCP checksum offloading. Redirect patches are carried in each ACP segment as a monotonically increasing sequence number. This field, updated by the switch for inbound flows, indicates that all data with sequence numbers less than this number may be delivered to the application. The switch has limited support for dynamic changes to the active set; it can remove failed servers, but it cannot add servers.

6 Experimental Results

This section presents results from our host-based Anypoint/ACP prototype and the Slite/NFS server cluster. The experiments explore the overhead and bandwidth the host-based Anypoint switch, frame processing costs, memory requirements, interactions with TCP rate control, and scaling and response time of Slite/NFS.

We also compare behavior of the Anypoint switch with an alternative service structure using a *redirecting proxy* that terminates incoming client connections and relay traffic over connections maintained between the proxy and the servers. Our proxies are implemented at the application level for TCP or UDP. The TCP proxy uses a blocking *select* to relay data between the peer and ensemble.

The Anypoint testbed consists of Dell PowerEdge 4400s with 733 MHz Pentium-III CPUs and 256 MB RAM, running FreeBSD 4.4. Each node has an Alteon Gigabit Ethernet NIC with hardware checkum offloading, connected to an Extreme Summit 7i switch. Unless stated otherwise, our microbenchmark tests use 4KB transport frames, 128 KB socket buffers, 9KB (Jumbo) segment/MTU sizes, and delayed acks. The Anypoint switch uses a frame window w of 384 entries. Each Slite NFS server is fitted with eight 18 GB 10,000 RPM Seagate Cheetah drives over two dual-channel Ultra-160 SCSI controllers.

6.1 Memory and CPU Overheads

First we explore the raw performance of the Anypoint switch and compare its CPU and memory utilization to a TCP proxy. The microbenchmark suite consists of simple user-level programs that open TCP or ACP sockets.

Figure 5 shows the peak aggregate throughput for 8 inbound streams (outbound results are identical) for the

Anypoint switch and TCP proxy. The active set for each stream is a single ensemble server; no merging or splitting occurs. We vary the frame size from 8KB to 1KB (2KB decrements), increasing the number of frames per segment on the x-axis. With 8KB frames, the TCP proxy's aggregate bandwidth is CPU-limited at 63MB/s, while the Anypoint switch is NIC-limited at 106MB/s. To factor out copying costs in the user-level proxy, the *Anypoint copy* lines show the performance of the Anypoint switch with two copies added to the critical path. Even with the copying, avoiding full termination in Anypoint yields an average 29% improvement in peak bandwidth. The declining bandwidth with increasing frames per segment quantifies the effect of per-frame processing costs.

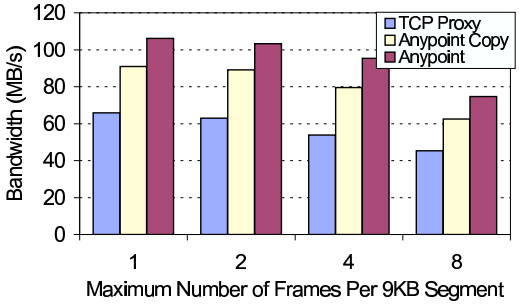

Figure 5: Aggregate bandwidth through the Anypoint switch and TCP proxy with increasing number of frames per segment.

Figure 6 shows the memory overhead of the TCP proxy and Anypoint switch for 8 simultaneous inbound single-server connections as the round trip time increases between client and ensemble (using dummynet [35]). Anypoint memory usage is determined by the flow window w and is independent of the number of servers active per connection. In contrast, a TCP proxy's memory usage scales with the aggregate bandwidth-delay product (BDP) for outbound flows. For inbound flows, proxy memory usage is inversely proportional to the BDP because an increasing share of the flow window is in transit in the network rather than buffered at the proxy.

Next we investigate splitting and merging of a single connection. Figure 7 shows throughput for a single connection that is either outbound (merged) or inbound (split) from/to four servers. The ALRM round robins the inbound 4KB frame stream across the ensemble. Throughput for inbound and outbound flows is nearly identical. As the round-trip time (RTT) and BDP increase, the outbound bandwidth to the peer is limited by the peer's receive window for both Anypoint and the TCP proxy. The Anypoint switch conservatively splits the peer's flow window evenly among the servers, each of which achieves the same share of the outbound bandwidth. For the inbound case, the TCP proxy is limited by its own receive window as BDP increases. In contrast, Anypoint's inbound bandwidth is limited by its conservative flow window advertisement, which is the the minimum of the server's advertised windows.

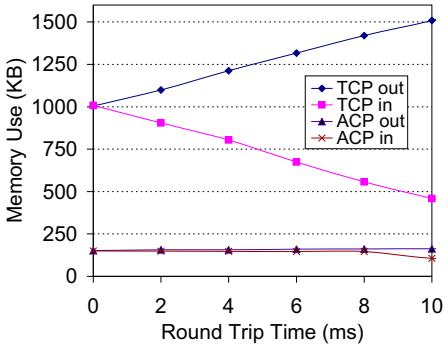

Figure 6: Total memory usage of a TCP proxy versus an Anypoint switch as a function of round-trip time. There are 8 simultaneous connections.

The Anypoint inbound and outbound flows achieve lower throughput than the TCP proxy at 2ms RTT. This effect is evident even with one server per connection. It occurs because the TCP proxy acknowledges data immediately, even before forwarding it to the receiver. This trades end-to-end reliable delivery and proxy buffer memory for improved bandwidth in this case. This effect diminishes with increasing RTT.

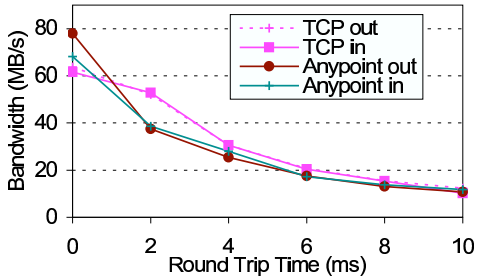

Figure 7: Bandwidth for single inbound or outbound connections with an ensemble of four servers.

6.2 Layer-4 Informed ALRMs

We now explore integrating layer-4 information into ALRMs with *speed-sensitive steering*, which always selects servers with open flow windows for inbound frames instead of using strict round robin, balancing inbound traffic across the ensemble more effectively. This allows the Anypoint switch to optimistically advertise the *sum* of the ensemble's flow windows to the peer.

We now compare the inbound throughput of a speed-sensitive steering ALRM versus the TCP proxy. In this experiment there are four servers, server socket sizes

(flow windows) are 32KB, and the MTU is 1500 bytes. One server incurs a variable delay between processing frames. Figure 8 shows throughput as this delay increases to 0.75ms. The TCP proxy gates the receive rate of every server to the slowest server. But the Anypoint connection can take advantage of excess capacity at the other servers and maintain high throughput during load imbalances.

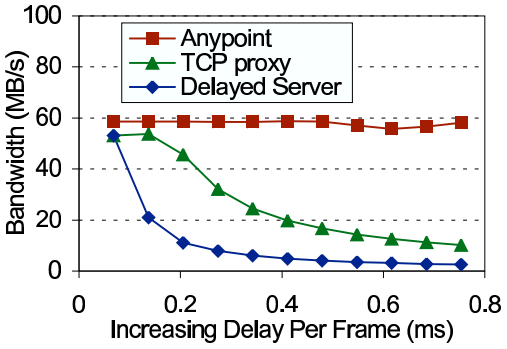

Figure 8: With speed-sensitive steering, the ALRM redirects inbound frames to servers with sufficient capacity to handle them. Here one server incurs an increasing CPU delay to process a frame. The TCP proxy gates the receive rate of each server to that of the bottlenecked server.

6.3 Observations

Anypoint offers three advantages relative to the TCP proxy:

- **Efficiency.** Memory use is bounded independent of traffic rates, and scales with the number of connections independent of the number of active servers. The Anypoint switch also avoids processing overheads to terminate the protocol.
- **End-to-end guarantees.** Anypoint offers end-to-end reliability. In contrast, a proxy acks data that it has not yet delivered to the receiving end node. Also, note that the proxy's delivery order is the same as Anypoint's, which is not the ordering specified by its transport (TCP).
- **Layer integration.** Anypoint allows a continuum of redirection policies that consider Layer 4 state, e.g., for speed-sensitive steering during periods of unbalanced load.

TCP splicing is one technique to reduce the runtime overheads for a proxy [19], and is amenable to switch-based implementations. This technique is related to Anypoint's sequence number translations to short-circuit protocol processing. However, the Anypoint transport model is fundamentally different.

Interestingly, inbound Anypoint flows in our prototype may slow down relative to a TCP proxy as the ensemble size grows, due to an interaction between the transport's congestion control and acknowledgments from the ensemble. The Anypoint switch merges acks from the ensemble nodes and sends cumulative acks to the peer. If the servers return acks out of order, the switch must delay them to avoid inciting a fast-recovery reaction on the peer, causing it to reduce the congestion window (TCP Reno and later presume that duplicate acknowledgments indicate lost data). Delaying these acks can negatively impact the acknowledgment clocking, lowering throughput.

After these experiments we can make a number of observations about desirable features for Anypoint-compatible transports:

- **Explicit rate control.** Outbound Anypoint flows should share link bottlenecks in a TCP-friendly manner. An outbound flow is an aggregate of n ensemble sources; in our prototype this flow is likely to be more aggressive than a competing TCP flow. To ensure fairness, the switch must coordinate ensemble sources through explicit congestion control signals.
- **Selective acks (SACK).** The transport must divorce congestion behavior from reliability. Triple-duplicate cumulative acks are common and meaningless as congestion indicators for Anypoint communication.
- **Flexible flow control.** The switch can optimistically or conservatively manage the flow windows as described in Section 4.3. If the switch conservatively distributes the peer's advertised receive window across the ensemble sources, it should be able to revoke unused window allocations and redistribute them to active sources. Alternatively, ensemble members could bid for the peer's receive window by advertising to the switch the amount of data they wish to send.

6.4 NFS Storage Router

In this section we evaluate the performance of the Slite NFS storage router. We compare an Anypoint Slite storage router (described in detail in Section 3.2) to user and kernel-level proxy alternatives.

The kernel-level (fastpath) configurations employ an Anypoint ALRM module to redirect frames. The ALRM sends most frames directly to servers; some require additional processing and go to the *coordinator*, which is just another server to the Anypoint layer. This category uses either an ACP or UDP transport. The UDP fastpath uses a simple kernel hook to intercept UDP packets and advertise them via the Anypoint interface to the Slite ALRM.

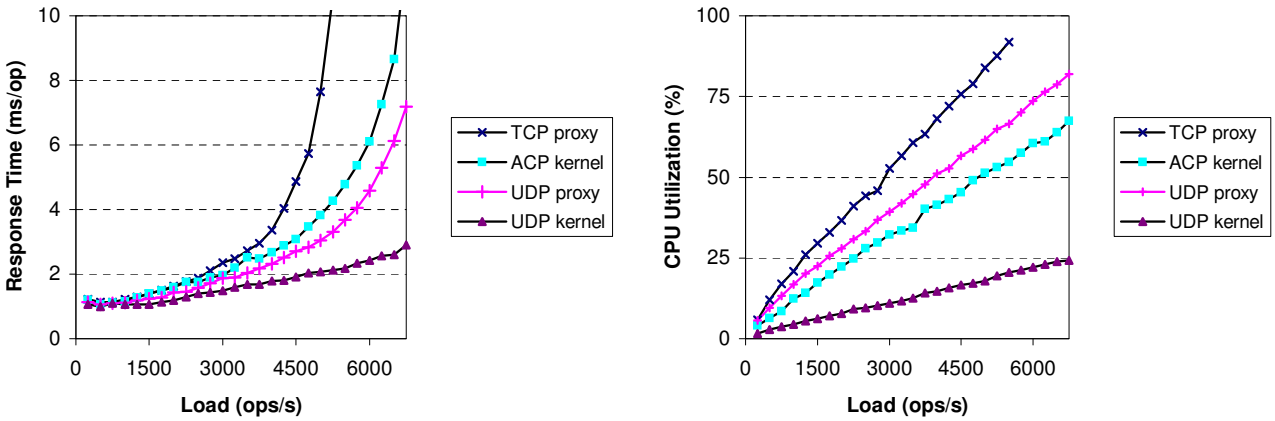

Figure 9: Slite latency and switch CPU utilizations as a function of offered load for varying intermediary configurations.

We use one coordinator proxy and four back-end NFS servers, exporting independent drives for a total of 32 NFS volumes (unified into a single logical volume). All machines run FreeBSD 4.4. Delayed acks are disabled to get good performance from the NFS RPC protocol over TCP, and the system is configured to use 8KB transport frames.

Figure 9 plots results from these four configurations, using the Fstress NFS benchmark [3] configured to generate a Web server's file system load. We measure average operation response time and CPU utilization at the switch with increasing request rate, plotted on the left and right, respectively. The TCP proxy has the lowest peak throughput and becomes CPU saturated at 5500 ops/s. The Anypoint switch shows a 29% improvement in throughput relative to the TCP proxy.

Note that the user-level UDP proxy gives better response times than the Anypoint switch, even though the Anypoint CPU overhead (shown in the right graph of Figure 9) is lower at any given request rate. We believe Anypoint/ACP is limited by the interaction between cumulative acks and congestion control described in the previous section.

7 Related Work

There is a large body of work on incorporating architecturally correct support for indirection into the Internet. The design alternatives span layers of the Internet architecture and all levels of its implementation. Indirection may apply to names at any layer (e.g., URNs or URLs, domain names, IP addresses, or abstract names [43, 1]). A given request may route through multiple intermediate nodes, as in hierarchies, meshes, or overlays for Web content caching [46]. Request routing policies may reside in application-level intermediaries (e.g., Web proxy caches), DNS name servers (e.g., Akamai and other CDNs), in the clients [45] or servers [8, 9] themselves, or in the network switches or routers.

IP-layer indirection. Anycast [33] and *i3* [41] support best-effort packet delivery with indirection at the IP layer. Anypoint operates at the transport layer and above for a related but different purpose. While Anycast is useful for binding to service sites across the wide area, Anypoint enables stateful, reliable, and congestion-aware connections to an ensemble at a logical site.

Web switches and persistent connections. Anypoint is similar in goals and concept to L4-L7 Web server switches, which implement server load balancing (SLB) and/or content-based routing (CBR) for Web server ensembles. The benefits of Web server traffic management are well-established from research studies (e.g., [31]) and commercial experience. Anypoint offers a more powerful mechanism for handling *persistent connnections*, as in HTTP 1.1, in which a single transport connection carries a stream of requests to simplify congestion control and amortize connection setup costs [28]. Web switches supporting HTTP 1.1 may route all requests on a given connection to a single server, or else they use *connection handoff* [8, 32, 37] to migrate the connection to a different server between requests. The first approach does not support CBR and the second forces the server ensemble to process and respond to the requests in strict sequence, as mandated by HTTP 1.1. Anypoint enables independent, pipelined processing of the requests, to generalize redirection to a broader class of service protocols that do not impose this constraint. This is important for fine-grained requests, e.g., storage protocols such as iSCSI or NFS, or RPC protocols such as RMI.

Redirecting proxies. Section 6 compares Anypoint to redirecting *proxies* that terminate incoming client connections and relay traffic over connections maintained between the proxy and the servers. Proxies do not preserve end-to-end transport semantics because the proxy may ac-

knowledge data before delivering it to the receiver, and this data is lost if the proxy fails. Moreover, the proxy incurs a high runtime overhead to perform full protocol processing for both connection endpoints and high memory overhead to buffer connection data. TCP splicing is one technique to reduce the runtime overheads [19, 15, 39], and is amenable to switch-based implementations [6]. Anypoint uses related techniques to short-circuit protocol processing and rewrite packets to map among multiple sequence spaces. However, the Anypoint model is fundamentally different: an Anypoint intermediary does not terminate transport connections or otherwise interfere with end-to-end transport functions.

Active Networks and extensible routers. Anypoint is related to Active Networks [44], in which dynamic packets carry behavior (*capsules*) that execute on routers. Like Active Networks, the Anypoint switch architecture defines a powerful new capability for extending switches and routers through simple, clean abstractions, meeting needs that are currently served in an ad hoc fashion. However, in contrast to capsules, Anypoint extensions—ALRMs—are installed by an authorized configuration tool, are relatively constrained in their actions, and have known, bounded resource and interface requirements. Thus they are more closely related to *switchlets* in the SwitchWare architecture [2]; Anypoint enables such extensions to perform application-layer functions while preserving transport semantics. Another extensible router architecture is Click [29]; Anypoint could run as a new set of transport-layer and application-layer modules within the Click framework. Anypoint does not require or benefit from the richer interfaces of the NodeOS [34] framework for extensible routers, which supports multiple execution environments similar to those offered in a general-purpose operating system.

Group communication and multicast. Anypoint is similar to group communication in that it supports communication with a dynamic ensemble addressed in a unified way. However, the basic Anypoint abstraction is less powerful in that it supports an indirect frame unicast rather than ordered multicast. The Anypoint framework is powerful enough to allow an ALRM to implement ordered multicast using an approach similar to Amoeba [24], in which a single intermediary for each connection acts as a sequencer.

Scalable Reliable Multicast(SRM) [20] is similar to an Anypoint intermediary in that it manipulates sequence numbers and is based on application-level framing. However, SRM and Anypoint have different goals: Anypoint enables application-layer routing while SRM does not, and Anypoint is not limited to a multicast model.

Traffic shaping and congestion management. One function of an Anypoint intermediary is to control sending rates to preserve the traffic balance between multiple senders and receivers. The technique of rewriting transport-layer flow window advertisements is due to Packeteer [25]. Anypoint connections aggregate communication from the ensemble to the client, and can be viewed as a multipoint-to-point session. The issue of maintaining fairness both among ACP connections and among ensemble members on an ACP connection is analagous to work on session fairness across the Internet [26]. Anypoint is similar to transport-layer bandwidth reservation schemes [42] in its use of per-flow state at the network edge. It is also related to schemes for coordinating congestion control across multiple flows with the same source or destination, when those schemes are applied in intermediaries [11, 30].

Multihoming. SCTP includes support for *multihoming*, which is similar to Anypoint in that a single connection may deliver traffic to a site through multiple IP endpoints. Anypoint differs from multihoming in that it does not require these multiple IP endpoints to coordinate through shared memory, and it uses multiple IP endpoints concurrently for the same connection.

8 Conclusion

Anypoint is the first architecture to enable switching at the granularity of transport frames in extensible routers at the edge of the network. This approach allows service-specific application plugins (ALRMs), residing in the router, to coordinate request/response flows to and from the multiple nodes in an ensemble. These plugins may support dynamic request redirection, response merging from multiple servers, and other extensions for network services based on a partially ordered, framed IP transport.

Anypoint provides transport-layer guarantees including partial ordering, rate control, and reliable at-most-once delivery without the overhead to terminate the transport protocol in the switch. Experimental results with a host-based Anypoint prototype show that Anypoint is a powerful mechanism for traffic management and virtualization in server clusters. We present results from an Anypoint switch under various network conditions, showing that buffering overheads in the Anypoint intermediary are significantly lower than an application-level proxy. Results from an Anypoint-based NFS storage router show that Anypoint supports scalable services transparently to clients.

9 Acknowledgments

We thank the anonymous reviewers and our shepherd, Peter Honeyman, for helpful critiques and suggestions.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, December 1999.

[2] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare active network architecture. *IEEE Network Special Issue on Active and Controllable Networks*, 12(3):37–45, May/June 1998.

[3] D. Anderson and J. Chase. Fstress: A flexible network file service benchmark. Technical Report CS-2002-01, Duke University Department of Computer Science, January 2002.

[4] D. C. Anderson and J. S. Chase. Failure-atomic file access in an interposed network storage system. *Cluster Computing: The Journal of Networks, Software Tools and Applications*, 2002. To Appear.

[5] D. C. Anderson, J. S. Chase, and A. M. Vahdat. Interposed request routing for scalable network storage. *ACM Transactions on Computer Systems (TOCS) special issue: selected papers from the Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000*, December 2001.

[6] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha. Design, implementation and performance of a content-based switch. In *Proceedings of IEEE Infocom 2000*, March 2000.

[7] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Oceano - SLA based management of a computing utility. In *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[8] M. Aron, P. Druschel, and W. Zwaenepoel. Efficient support for P-HTTP in cluster-based Web servers. In *Proceedings of USENIX'99 Technical Conference*, 1999.

[9] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable content-aware request distribution in cluster-based network servers. In *Proceedings of USENIX Technical Conference*, June 2000.

[10] S. Bailey, J. Chase, J. Pinkerton, A. Romanow, C. Sapuntzakis, J. Wendt, and J. Williams. Internet Engineering Task Force, Internet draft: TCP ULP Framing Protocol (TUF), November 2001.

[11] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, September 1999.

[12] B. Braden. Internet Engineering Task Force, Network Working Group, RFC 1122: Requirements for Internet hosts – communication layers, 1989.

[13] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)*, pages 103–116, October 2001.

[14] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of SIGCOMM Symposium on Communications Architectures and Protocols*, pages 200–208, Philadelphia, PA, 1990.

[15] A. Cohen, S. Rangarajan, and H. Slye. The performance of TCP splicing for URL-aware redirection. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS)*, October 1999.

[16] M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, and M. Wittle. The Direct Access File System. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, March 2003.

[17] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. Router plugins – a modular and extensible software framework for modern high performance integrated services routers. In *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, September 1998.

[18] R. P. Doyle, J. S. Chase, S. Gadde, and A. M. Vahdat. The trickle-down effect: Web caching and server request distribution. *Computer Communications: Selected Papers from the Sixth International Workshop on Web Caching and Content Delivery (WCW)*, 25(4):345–356, March 2002.

[19] K. Fall and J. Pasquale. Exploiting in-kernel data paths to improve I/O throughput and CPU availability. In *Proceedings of USENIX Technical Conference*, pages 327–334, January 1993.

[20] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, 1997.

[21] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)*, Saint-Malo, France, October 1997.

[22] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 97)*, June 1997.

[23] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for Internet service construction. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 319–332, October 2000.

[24] M. F. Kaashoek and A. S. Tanenbaum. Group communication in the Amoeba distributed operating system.

In *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS)*, pages 220–230, May 1991.

[25] S. Karandikar, S. Kalyanaraman, P. Bagal, and B. Packer. TCP rate control. *ACM SIGCOMM Computer Communication Review*, 30(1), January 2000.

[26] P. Karbhari, E. W. Zegura, and M. H. Ammar. Multipoint-to-point session fairness in the Internet. In *Proceedings of IEEE Infocom*, 2003.

[27] E. Kohler, M. Handley, S. Floyd, and J. Padhye. Internet Engineering Task Force, Internet draft: Datagram Congestion Control Protocol(DCCP), November 2001.

[28] J. Mogul. The case for persistent HTTP connections. In *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, pages 299–313, September 1995.

[29] R. Morris, E. Kohler, J. Jannotti, and M. Kaashoek. The Click modular router. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 217–231, December 1999.

[30] D. Ott and K. Mayer-Patel. A mechanism for TCP-friendly transport-level protocol coordination. In *Proceedings of USENIX Technologies Conference*, June 2002.

[31] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenopoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[32] A. E. Papathanasiou and E. V. Hensbergen. KNITS: Switch-based connection handoff. In *Proceedings of IEEE Infocom*, June 2002.

[33] C. Partridge, T. Mendez, and W. Milliken. Internet Engineering Task Force, RFC 1546: Host anycasting service, November 1993.

[34] L. Peterson, Y. Gottlieb, S. Schwab, S. Rho, M. Hibler, P. Tullmann, J. Lepreau, and J. Hartman. An OS interface for active routers. *IEEE Journal on Selected Areas in Communications*, 19(3):473–487, March 2001.

[35] L. Rizzo. Dummynet: A simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, 27(1):31–41, January 1997.

[36] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, availability and performance in Porcupine: a highly scalable cluster-based mail service. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 1–15, Kiawah Island, December 1999.

[37] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan. Fine-grained failover using connection migration. In *Proceedings of the 3rd USENIX symposium on Internet Technologies and Systems (USITS)*, March 2001.

[38] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *18th ACM Symposium on Operating Systems Principles*, October 2001.

[39] O. Spatscheck, J. Hansen, J. Hartman, and L. Peterson. Optimizing TCP forwarder performance. *IEEE/ACM Transactions on Networking*, 2(8):146–157, 2000.

[40] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, and L. Zhang. Internet Engineering Task Force, RFC 2960: Stream Control Transmission Protocol, October 2000.

[41] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of the ACM SIGCOMM 2002 Conference on Communications Architectures and Protocols*, August 2002.

[42] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: a scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, September 1998.

[43] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active names: Flexible location and transport of wide-area resources. In *Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS)*, October 1999.

[44] D. Wetherall. Active network vision and reality: Lessons from a capsule-based system. In *Proceedings of 17th ACM Symposium on Operating System Principles (SOSP)*, December 1999.

[45] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler. Using smart clients to build scalable services. In *Proceedings of USENIX Technical Conference*, January 1997.

[46] L. Zhang, S. Floyd, and V. Jacobson. Adaptive web caching. In *Proceedings of the NLANR Web Cache Workshop*, June 1997.