# Explicit Rate Control for Anypoint Communication

Ken Yocum
*Department of Computer Science*
*University of California at San Diego*
kyocum@cs.ucsd.edu

Jeff Chase
*Department of Computer Science*
*Duke University*
chase@cs.duke.edu

***Abstract*—**

**This paper investigates explicit multiflow rate control for common forms of ensemble communication. We address rate control in the context of the Anypoint, a communication model that combines strong transport-layer guarantees with the flexibility of Anycast-style indirection. For example, Anypoint supports cluster-based services where extensible routers at the network edge route inbound requests according to pluggable service-specific redirection policies. Anypoint also supports multipoint-to-point communication in the reverse or outbound direction.**

**Our approach extends recent work on an Explicit Control Protocol (XCP) and demonstrates compelling benefits for multiflow rate control. For example, it allows us to unify flow and congestion control for multipoint traffic. The explicit rate control approach defines a set of XCP header transformations to split and merge XCP traffic flows in an Anypoint edge router. We describe a prototype XCP-enabled transport protocol and Anypoint-XCP switch implementing the split/merge transformations. Experimental results evaluate our approach in a prototype Anypoint cluster in the ModelNet emulation environment, extended with support for emulated XCP routers. The results show how Anypoint-XCP flows compete fairly with conventional XCP flows and meet a range of multiflow rate control objectives in a flexible and efficient way.**

## I. Introduction

Several factors are driving an increased interest in multiflow rate control and ensemble communication. Web clients often draw and assemble content from multiple sites to satisfy a single request [1]. And while network-layer Anycast [2] sees limited real-world deployment, many Internet content services employ the underlying communication model. For example web server farms [3] or scalable storage systems [4, 5] often use indirection to allow dynamic binding to a specific server from a set of servers that can provide a given piece of content. Similarly, environments for large-scale data-intensive computing have inspired new multiflow approaches for cluster-to-cluster communication [6] and more general multipoint-to-point communication in which multiple flows converge at a single site [7].

This paper considers multiflow rate control in the context of Anypoint [8], a transport-layer abstraction for communication with *ensemble sites* such as cluster-based network services. Anypoint is designed for transports that transfer data as rate-controlled streams of frames, such as SCTP. Extensible Anypoint switches at the ensemble edge mediate communication; an Anypoint switch represents the ensemble as a single virtual site to the outside network, and implements a redirection policy to route incoming frames to ensemble members. Figure 1 illustrates the forms of multipoint communication that occur between an ensemble and a connection "peer" or client. Section II discusses Anypoint in more detail.

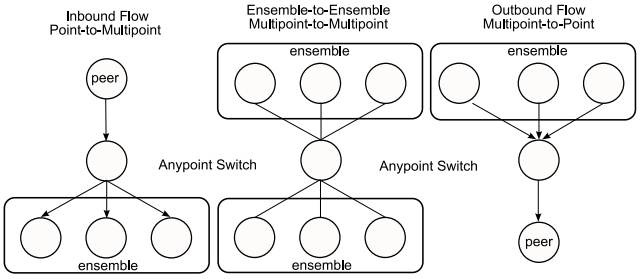


Fig. 1. The Anypoint communication model encapsulates both point-to-multipoint (Anycast) and multipoint-to-point communication. It can allow two clusters to communicate without explicit knowledge of the nodes participating in the other cluster.

Crucially, Anypoint's rate control issues are similar to the core challenges for general multipoint communication and multiflow rate control. Figure 1 illustrates the range of multipoint communication in the Anypoint model. Inbound traffic arriving at an Anypoint ensemble is distributed across the ensemble members according to a service-specific routing policy. Outbound traffic from multiple ensemble members may converge on the same client. In addition, Anypoint sites may be combined for cluster-to-cluster communication or they may cascade to form arbitrary trees to support tiered service architectures. This creates several rate control challenges that are common to other multipoint environments:

- The rate of an inbound connection to the ensemble must be limited to avoid network path and endpoint

buffer overflows. The fraction of inbound traffic destined to an ensemble member must not exceed the rate at which the member can receive and process it.
- The outbound traffic on a given Anypoint connection or session originates from multiple ensemble members (a multipoint-to-point flow). The aggregate flow to the client must not exceed the rate at which they can receive and process it. Ensemble members must allocate rates to share this capacity effectively.
- The traffic on a given Anypoint connection or session should compete fairly with other connections sharing a bottleneck link. In particular, traffic on an Anypoint connection should be *session-fair* with respect to competing connections: a connection's fair share across a bottleneck resource is independent of the number of constituent ensemble member flows.

We develop and evaluate an approach to multiflow rate control for Anypoint traffic. Our approach is based on explicit rate coordination in the Anypoint switches—the network points where the flows of a given Anypoint connection split and merge. To deliver rate feedback to the end systems, we use a derivative of the Explicit Control Protocol (XCP [9]), in which routers and switches along the network path for each flow mark the packet headers with rate signals. We recognize that though XCP has many compelling benefits, it is unlikely that XCP will be deployed in the Internet core. However it is an important and valuable research perspective to understand the full impact of such a redesign of Internet congestion control. This work extends XCP's benefits beyond those already demonstrated for point-to-point flows, and finds that XCP is a powerful basis for multipoint communication. This work makes the following contributions:

- **Multipoint explicit rate control.** The key to our approach is a new set of transformation rules applied to XCP headers within a network switch. Note that these transformations can be applied in general to any XCP-based transport, and not only to our prototype Anypoint-compatible transport protocol. For example, the transformations do not require a frame-based transport, nor any Anypoint switch functionality beyond parsing and writing XCP headers [9].
- **Rate-based flow control.** We address flow control in a unified way by treating it as a congestion control problem; receive buffer space is a shared resource subject to congestion. This *soft flow control* approach is flexible enough to meet flow control needs for multipoint communication, and our experiments confirm that it avoids the pitfalls of obvious credit-based flow control policies [8].
- **Prototype-based evaluation.** We have prototyped a complete rate-controlled Anypoint implementation (Section V). It includes kernel extensions to FreeBSD for a host-based Anypoint switch and an Anypoint-compatible transport protocol that supports application-layer framing and XCP-based congestion control. We also added support for emulating XCP-enabled routers in the ModelNet Internet emulation environment [10]. This provides a direct execution environment for evaluating our XCP rate control scheme or any general XCP implementation. We find the combination of these two point-to-point transport facilities—XCP and framing—also particularly powerful for Anypoint communication.

## II. Background

The original impetus for the Anypoint model was to generalize "L4-L7" server switches that support load balancing and content-aware request routing for Web server clusters; the objective is to define a general redirecting switch architecture that accommodates pluggable indirection policies for a wide range of service protocols, not limited to HTTP over TCP. A fundamental goal of Anypoint is to reconcile transparent Anycast-style communication with strong transport guarantees, including reliable, ordered, rate-controlled transmission.

For example, consider an Anypoint cluster implementing a request/response service, such as a network file service. Clients of the service open connections to the service in the usual manner, and transmit streams of requests. As in Anycast, the server to receive each request is selected dynamically, and the selection policy may consider the content of the request [3] such as the specific file or block requested [4]. Thus, different ensemble members may concurrently execute requests on behalf of any given client. To a client, the server cluster appears as an ordinary endpoint for a reliable, ordered transport connection, i.e., it appears to be a service running at a single host at a virtual IP address. Anypoint hides the selection policy and the structure of the ensemble from the client.

The key to Anypoint is a set of rules for maintaining state and transforming packets to implement the redirection policy in an extensible Anypoint switch. An Anypoint switch does not terminate connections, but merely transforms packets to maintain end-to-end transport protocol guarantees at the end systems. The transport protocol itself is a general-purpose protocol with no Anypoint-specific functionality, although it does require advanced features—application-layer framing and partial ordering—which exist today in transports such as SCTP. To an end system, Anypoint connections are indistinguishable from point-to-point connections using the same transport: we refer to this property as *transport equivalence*. The switch functions include sequence number translations, acknowledgment coalescing, and coordination of rate control signals

as described in this paper. Because the switch operates at the transport layer, we refer to it as a *transport switch*. It maintains per-flow soft state that is proportional to the size of the cluster.

The Anypoint transport switch is a trusted component of the service implementation, designed to operate at the edge of a server cluster (e.g., it is allowed to decrypt incoming data). The switch is extensible, supporting pluggable application-layer routing modules (ALRMS) that implement the service-specific routing policy. The ALRM determines the *indirection* schedule, or how the flow is split across ensemble endpoints. In our previous work [8] we demonstrated the Anypoint transport switching approach as a building block for scalable IP storage appliances, and compared it to alternative approaches such as TCP proxies with respect to performance, scalability, and reliability.

## III. Anypoint Rate Control

This work addresses the challenge of splitting and merging transport flows while avoiding network congestion and endpoint buffer overflows. We do this at explicit points, Anypoint switches, within the network, at the granularity of connections. An Anypoint connection consists of an *inbound flow* into the ensemble and an *outbound flow* to the connection peer. Figure 1 illustrates inbound and outbound flows. The inbound flow is split by the switch into separate flows to each ensemble member. In the other direction individual ensemble flows are merged into a single outbound flow. Here we may speak of the flow between switch and peer, or the separate flows between switch and ensemble endpoints. In the parlance of multipoint-to-point rate control, an Anypoint connection, consisting of one or more flows, is analogous to a session.

In Anypoint transport equivalence defines the fair behavior of connections. Transport equivalence implies that end nodes do not change their rate control policies to use Anypoint. Each flow abides by the same notion of rate control fairness; each is limited by the available bandwidth between the endpoint and the switch. That bandwidth is determined by the rate control policy at the endpoint, e.g., a split or merged TCP flow should remain TCP-compatible, sharing network paths fairly with other TCP connections. Thus all Anypoint flows are "session-fair", sharing bottlenecks fairly independent of the ensemble size. This is in contrast to "connection-fair" behavior, where session rates are proportional to the number of flows within them.

### A. Inbound and Outbound Flows

The inbound rate control challenge is to maximize the total inbound rate while avoiding overflowing the different network paths to each ensemble member. Observe that the peer's sliding window paces outgoing data at the rate of returning acknowledgments that open up free buffer space for new transmissions. Any outstanding data on the slowest or most congested path stops the peer from transmitting until the data is acknowledged. This limits the amount of outstanding data down all paths. Thus the optimal inbound rate is a function of the switch's indirection policy. Because the peer is unaware that its traffic is split across the ensemble, we must merge congestion information from multiple network paths in proportion to how much they are used.

In the outbound direction ensemble endpoints must coordinate to collectively achieve the appropriate capacity across the bottleneck link to the connection peer, but also share that capacity among themselves. The key challenge here is that an ensemble endpoint doesn't know how the rest of the ensemble is using the bottleneck capacity.

### B. Flow Control

In a unicast scenario, flow control is a simple contract between a sender and a receiver to manage endpoint resources (buffering) of a single flow. However, outbound flows must share the peer's buffer among the ensemble, and an inbound flow may consume receive buffering at any ensemble endpoint. Assuming credit-based flow control (e.g., TCP window advertisements), the switch could split the peer's available credit evenly across the ensemble to control the outbound flow. But then actively transmitting endpoints may starve as quiescent endpoints sit with surplus credits. Similarly, the switch could throttle an inbound connection to the slowest ensemble endpoint. Both "strict" policies avoid buffer overruns but limit the ability of the switch to optimally manage receiver buffers.

We overcome these limitations through *soft flow control*, an optimistic rate-based flow control detailed in Section IV-A that allows dynamic allocations. For comparison, consider the behavior of multiple uncoordinated TCP connections when the peer's receive buffer space is constrained. These connections can match the throughput of SFC only by optimistically over committing the receiver's memory to each of the connections; this is the default policy in most TCP implementations, but then the receiver must drop packets if all senders transmit simultaneously. In effect, SFC reallocates receive buffer memory among the senders according to their demand, similarly to congestion management on a shared link.

### C. A Transport Switching Rate Control Architecture

For the switch to coordinate rates, we make two requirements: the transport carries explicit rate information in its headers, and the transport employs receiver-based

congestion control. We assume that transport headers carry $rate$, the current transmit rate, and $fdbk$, the future sending rate. The receiver uses the returning $fdbk$ to adjust its transmission rate. Protocols that fit this model include TFRC and XCP [11, 9]. The switch coordinates outbound and inbound flows by transforming packet headers, merging outbound flow headers and splitting inbound flow headers.

In this and the next sections we use the following terminology. For any given connection, the achieved throughput to/from the peer is $\lambda_p$, and the achieved throughput to/from each ensemble endpoint is $\lambda_i$. There are $N$ ensemble endpoints. For inbound flows we define $\beta_i$, the receive ratio for ensemble member $i$, as the ratio of traffic directed to that endpoint, $\frac{\lambda_i}{\lambda_p}$.

There are four switch transforms. Each transform ensures that $fdbk$ and $rate$ are consistent with the actual rate of the flow along that path and that the resulting flow is transport equivalent. The first two modify $rate$ and $fdbk$ from source to sink (data flow), and the second two modify $fdbk$ from sink to source (acknowledgment flow). For each flow there is a merge (outbound) and split (inbound) operation. The switch merges data headers as they arrive from the ensemble. The merged data flow represents the collective desires of the $N$ ensemble members as $rate = \sum_{i=1}^{N} rate_i$ and $fdbk = \sum_{i=1}^{N} fdbk_i$. In contrast we split a data flow header by multiplying $rate$ and $fdbk$ by $\beta_i$, and this maintains $\lambda_i = \lambda_p * \beta_i$.

The acknowledgment flow returns $fdbk$ to the source(s). The merge transform cannot simply sum $fdbk$ from each ensemble member because it allows unused paths to increase the input rate, potentially overflowing used paths. The $fdbk_i$ returning from each path $k$ indicates a rate increase or decrease. We know that $\lambda_p = \frac{\lambda_i}{\beta_i}$ and thus the change in the peer's rate is proportional to the change in $i$'s rate ($\Delta_p = \frac{\Delta_i}{\beta_i}$). Thus for any particular $i$, a $fdbk$ of $\frac{fdbk_i}{\beta_i}$ achieves the rate change down path $i$. Finally, we find the rate change along the most constrained inbound path by taking the minimum across all paths $i$ (Equation 1).

$$fdbk = \underset{\forall i}{\text{Min}}(\frac{fdbk_i}{\beta_i}) \quad (1)$$

The last acknowledgment flow transform splits the returning feedback among the ensemble sources. The transform operates in an analogous fashion to AIMD controllers in standard TCP. It spreads a rate increase evenly among all ensemble sources, and it reduces ensemble rates in proportion to their contribution to the total outbound rate. Thus, for ensemble member $i$, positive feedback is $fdbk/N$, and negative feedback is $fdbk * \frac{rate_i}{rate}$.

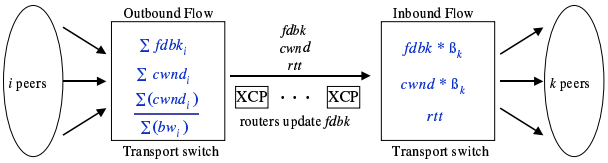


Fig. 2. A transport switch modifies XCP headers for outbound and inbound data flows. This diagram shows two transport switches moderating communication between two clusters. An outbound flow for the first transport switch becomes the inbound flow for the second. In this direction the switch updates all three XCP header fields: $cwnd$, $rtt$, and $fdbk$.

## IV. Transport Switching with XCP

The eXplicit Control Protocol (XCP) [9] addresses many of the issues with traditional point-to-point transports in the Anypoint context. XCP is designed for high bandwidth-delay networks, virtually eliminates congestion loss, is independent of RTT, runs the network at a high level of efficiency, and removes the onus of inferring congestion from secondhand end-host observations. By eliminating congestion loss, XCP obviates the need for mechanisms like fast retransmit and loss differentiation in contexts where link-layer loss is rare. Critically, explicit rate control allows a network switch to dynamically coordinate flow rates, and is compatible with our rate control architecture.

XCP details are found in [9], and here we give an overview as it pertains to Anypoint. XCP endpoints are window based; the size of the congestion window determines the transmission rate. XCP requires support from core routers along the network path between source and destination. Each router adjusts individual flow rates to match the total arrival rate to the outgoing link's capacity. XCP routers have separate controllers for efficiency and fairness. The efficiency controller determines the available capacity on the outgoing link, and the fairness controller allocates that capacity across the current flows. XCP packet headers contain sufficient information to run both controllers without maintaining per-flow state in the routers. The routers update rate information in the packet headers as they travel from source to sink.

XCP headers contain three fields: the round-trip time ($rtt$), the sender's current congestion window ($cwnd$), and a $fdbk$ field initialized to the sender's desired rate increase (in bytes). To control the sender's congestion window, XCP routers along the network path update $fdbk$ according to the read-only fields $cwnd$ and $rtt$. Returning acknowledgments carry $fdbk$ to the sender; the sender sets its $cwnd$ to $\text{Max}(cwnd + fdbk, s)$, where $s$ is the packet size. Note that the bottleneck XCP router determines the feedback returned to the source: XCP routers only decrease $fdbk$.

The goal of the switch transforms in Section III is to make sure that the rate control information in the headers is consistent before and after the switch. This is distinct from XCP controllers who share a link among multiple point-to-point transport flows. A transport switch splits an inbound XCP flow into $N$ transport-equivalent XCP flows, where $N$ is the ensemble size. In the reverse direction, it merges ensemble flows into a unified XCP outbound flow.

XCP switch transforms have two differences with respect to those outlined in the previous section. First we must additionally transform $rtt$, and, second, rate information is carried as a window. All three XCP header fields are modified for transport-equivalent XCP data flows, while the returning acknowledgment flow transform only modifies $fdbk$. Figure 2 shows two Anypoint transport switches moderating communication between two clusters. From left to right, a flow is outbound at the first switch, and then inbound at the second switch. Transport equivalence allows us to consider the outbound and inbound flows independently; each transport switch can be replaced by a single endpoint. We assume $N$ ensemble members, and that all routers are XCP-enabled.

Figure 2 shows the outbound and inbound data flow transforms. Merged flows represent the ensemble's collective rate and rate increase, while a split flow is a fraction of the original. The transport-equivalent outbound header has $cwnd = \sum_{i=1}^{N} cwnd_i$, $fdbk = \sum_{i=1}^{N} fdbk_i$, and $rtt = \frac{cwnd}{bw}$ where $bw = \sum_{i=1}^{N} (\frac{cwnd_i}{rtt_i})$ is the total outbound flow bandwidth. Each inbound header is adjusted by the receive ratios, $\beta_k$ (Section III-A), for each ensemble member $k$. The transform splits the incoming header in proportion to these ratios: $cwnd_k = cwnd * \beta_k$ and $fdbk_k = fdbk * \beta_k$. The round-trip time is $rtt_k = \frac{cwnd_k}{bw_k} = \frac{cwnd*\beta_k}{bw*\beta_k} = rtt$.

Figure 3 shows the acknowledgment flow returning $fdbk$ to the source(s). There are two differences with respect to the transforms in Section III-C. First, $fdbk_i$ is proportional to $rtt_i$: $fdbk_i = \Delta bw_i * rtt_i = \frac{fdbk}{N} * \frac{rtt_i}{rtt}$. Second we must not return a $fdbk$ that is greater than any upstream XCP bottleneck. The final feedback along any path is the minimum of $fdbk_{src}$, the last feedback from the source, and the new feedback from the transforms.

There are three important observations. First, the transforms maintain transport equivalence when $N = 1$; XCP headers pass through the switch unmodified. Second, inbound data and ack flow transforms depend upon timely and accurate updates of the receive ratios. A stale ratio will either constrain the inbound flow or allow router queue overflows. Section VI-B empirically shows that the inbound and outbound flows are stable even with varying ensemble link capacities and latencies (Figure 8). While it is future work to formally show stability, note that the transforms maintain consistency between the rate control information in the packet headers and the actual rate of the flow.

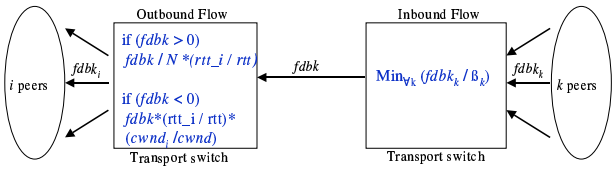


Fig. 3. Unlike XCP routers, transport switches merge and split $fdbk$ as it returns to the source.

## A. Rate-based Flow Control

Recall that strict credit-based flow control policies limit performance (Section III-B) because endpoint buffers are shared across flows in an Anypoint connection. Our approach here is rate-based flow control, or *soft* flow control (SFC), where endpoints advertise receive *rates*. We treat transport-layer receive queues in an identical fashion to XCP router queues, and install an XCP controller at the end host. We slightly modify the efficiency controller to probe for the appropriate receive rate to avoid buffer overflows. [1] By leveraging XCP, the $fdbk$ in XCP packets represent either a bottleneck in the network or at the receiving end host. Note that XCP-enabled routers aren't required for SFC. This approach adds negligible overhead at the endpoints as XCP controllers were designed for the stringent CPU requirements of high-speed routers [9].

The SFC controller attempts to find the current drain rate of the transport receive queue. Like XCP, we run the efficiency controller every epoch $d$, the average round-trip time, to determine how much to change the sender's transmission rate (converted into a window adjustment) during the next epoch. The original XCP efficiency controller calculates the change in the sender's rate using the queue's drain rate, $\lambda_{target}$, which is fixed to the link's capacity. In SFC the drain rate is a function of time, and we need to determine the function $\lambda_{target}(t)$ that does not overflow the receive buffers. It depends on the application, OS process scheduling, and transport protocol (out-of-order data does not drain in TCP).

The challenge is a controller that respects both window limits and application limits. Each epoch we measure the current buffer occupancy $Q$, the application's receive rate $\lambda_{app}$, and the current input rate $\lambda_{in}$ to calculate a new $\lambda_{target}$. We use an AIMD controller with five cases.

The first case is an idle state that resets our estimation to one TCP segment when $\lambda_{app} = 0$ and $Q \geq 0$. The

[1] Here, because there is only one flow, the fairness controller does not have to *shuffle* bandwidth across flows to converge to fairness [9].

second case halves the rate if packets were dropped at the receiver. The third case decreases the rate by $\gamma$% if our queue exceeds a high-water mark, $decrthresh$. The fourth case avoids probing if the current input rate is less than 50% of the target. And the fifth case grows the target by two segments when the queue is below a low-water mark, $incrthresh$, and the application is draining faster than data is arriving ($\lambda_{app} > \lambda_{in}$). We set $incrthresh$, $decrthresh$, and $\gamma$ to $1/3$, $1/2$, and 0.1, respectively. We measure $Q$ as an average of the running queue length and its maximum during the interval.

## V. Prototypes

To evaluate our approach, we extended a prototype of an Anypoint system with support for explicit multiflow rate control, and explored its behavior across a range of scenarios. To allow controlled, reproducible experiments using wide-area network topologies, we ran our experiments over the ModelNet [10] emulation system. ModelNet is a large scale network emulator that supports direct execution of distributed system prototypes over emulated wide-area network topologies: the ModelNet software intercepts packets and subjects them to hop-by-hop delays and losses of a specified target topology, including router queuing delays.

The prototype consists of a host-based Anypoint switch and an IP-based transport protocol, both implemented as a set of kernel extensions to FreeBSD. The Anypoint prototype (minus the rate control features) is described in detail in a previous paper [8]. We briefly summarize the key aspects of our design, then focus on the new support for multiflow rate control based on XCP and split/merge transformation rules for XCP headers, as described in Section IV.

As discussed in Section II, Anypoint assumes a reliable transport protocol with application-layer framing and partial ordering (e.g., SCTP). In our previous work 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 the Anypoint Compatible Protocol (ACP), although its functions are not Anypoint-specific. ACP uses a kernel socket-layer shim with framing support based on a subset of the TCP upper-layer framing (TUF) proposal, based on a now-expired IETF draft. Like TCP, ACP preserves the send order for frames delivered to a given end node node. However, ACP does not define the delivery order among frames routed to different ensemble nodes.

Our previous Anypoint/ACP implementation uses conventional rate control mechanisms derived from the TCP-Reno (4.4 BSD) implementation on which it is based. The end systems advertise windows for flow control, and congestion control is based on New Reno with delayed acknowledgments, fast retransmit, and fast recovery. The Anypoint transport switch used a strict flow control policy: for each connection, divide the flow windows for outbound traffic evenly among the ensemble for the connection, and advertise the smallest window in the ensemble to the connection peer.

The new support for multipoint rate control consists of three components:

- **XCP transport implementation** consists of a small amount of code added to ACP to read/write XCP headers and adjust the transmit rate. The transport includes optional support for XCP-based soft flow control (SFC), as described in Section IV-A.
- **XCP router emulation** consists of extensions to the ModelNet [10] Internet emulation environment to support XCP functions in the emulated routers. We added about 200 lines of code in a ModelNet plug-in module for a router queue discipline, based on Katabi's XCP simulation code for *ns*. Together with the end-system support for the ACP-XCP transport, it constitutes a complete XCP prototype for emulation experiments.
- **XCP-based rate control for Anypoint** consists of about 170 new lines of code in the Anypoint transport switch to support XCP congestion control. For Anypoint connections, the transport switch uses the XCP header transforms defined in Section IV for splitting inbound traffic and merging outbound traffic. It uses a simple histogram to compute the receive ratios, $\beta_k$.

## VI. Experimental Results

We conducted several experiments to explore the effectiveness of XCP-based rate control for Anypoint communication. Our methodology for all experiments is direct execution of Anypoint/ACP ensemble clusters under synthetic load from one or more clients. In these experiments the testbed consists of Dell PowerEdge 1650 (single 1.4GHz PIII) with integrated Intel Pro/1000 gigabit Ethernet NICs connected by a Cisco Catalyst 4000.

We ran these experiments over wide-area network topologies using the ModelNet emulation environment [10]. The experiments use variants of the network topology shown in Figure 4, varying characteristics such as the latency, bandwidth, and/or cross-traffic on the transit link. There is a single peer connecting to a service with an ensemble of four servers ($N = 4$). Link latencies are 1ms unless otherwise noted, and the service uses 1KB frame sizes.

Unless stated otherwise, all socket buffers are set to

200KB, and each transmitting source sends a continuous flow of 1KB frames to the receiver. Each data point is the average throughput over a 20 second interval.

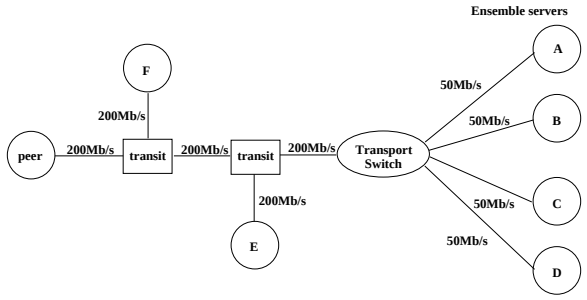


Fig. 4. A representative network topology used as the basis for experiments with Anypoint rate control.

## A. Soft Flow Control

We begin by evaluating the effectiveness of soft flow control (SFC). SFC treats receive buffer allocation as a congestion management problem, as an alternative to conventional window-based or credit-based flow control. However, note that SFC is optimistic and may drop packets. We show that when flow control is needed SFC converges quickly to the maximum sending rate.

*1) Point-to-Point Behavior of SFC:* The first experiments (Figure 5) illustrate the delivered throughput of SFC in simple point-point scenarios with a direct link between two hosts. This experiment is designed to stress the transport endpoint mechanisms in isolation, without the complexity of multipoint communication, network congestion, or switch interactions. We vary the link speed, round-trip time, and receiver buffer space.

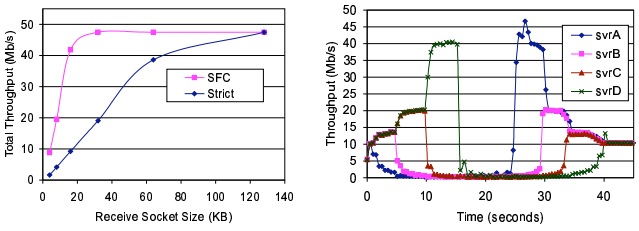


Fig. 6. Using the topology in 4 these graphs compare strict and soft flow control as data is sent to (left graph) and from (right graph) four servers.

The left graph of Figure 5 shows the throughput effect of increasing the per-frame CPU service demand of the receiving application. As the processing demand increases, throughput becomes limited by the rate at which data drains from the receiver's buffer. XCP-based soft flow control is as effective as credit-based flow control for both 10 ms and 100 ms RTTs. Note that both approaches are limited by the receive buffer size in the 100 ms case with low per-frame service demand.

The middle graph of Figure 5 shows the effect of network latency on delivered throughput in buffer-limited scenarios, i.e., when the receive buffer size is inadequate to transmit data as fast as the network can carry it or the receiver can consume it. Receive buffer size increases on the x-axis, allowing higher throughput; the limited buffer sizes affect throughput more severely with higher RTTs. The interesting behavior occurs with a shorter (10 ms) RTT, in which the SFC controller sometimes under utilizes the available capacity, yielding lower average throughput than credit-based flow control. These cases should be rare if end systems have adequate memory.

The right graph of Figure 5 evaluates agility of flow control when the per-frame processing demand oscillates between 100 and 750 microseconds in a square wave pattern. SFC closely follows the available receive rate, but buffer overflows and data loss occur when the consumption rate drops; the SFC controller requires a round-trip time to respond.

While SFC is effective in these scenarios, the current implementation can benefit from continued tuning and development. Our current SFC controller responds conservatively to packet loss with a multiplicative decrease in the sending rate, as in TCP Reno and the reference code for XCP. For this reason, it is tuned to avoid overshoot by increasing the sending rate conservatively as buffer space becomes available.

*2) Ensemble Behavior of SFC:* To show how soft flow control can improve fairness and efficiency for multipoint communication, we conducted several experiments using a four-server ensemble with a network topology based on Figure 4 where the transit link bandwidth is 50Mb/s. The graphs in Figure 6 report peak aggregate throughput for an Anypoint connection to the ensemble in various scenarios; inbound flows are split evenly across the ensemble. The graphs compare throughput for SFC and strict credit-based flow control (see Section III-B).

The left graph in Figure 6 shows the effect on peak inbound throughput of reducing the socket buffer size of one of the ensemble servers below the standard 200KB to the size given on the x-axis. The strict transform advertises the minimum window to the peer, gating the flow to the smallest window. In contrast, the SFC approach delivers aggregate throughput, $\lambda_p$, that follows $\frac{S_{min}}{\beta_{min}}$, where $S_{min}$ is defined by the receive window limited server and $\beta_{min}$ is 0.25 (Section III-A). Note that limiting the application's receive rate, by increasing the application's per-frame CPU demand, is analogous to a slow ensemble link. In this case our experiments (removed for brevity) show that soft flow control policies provide no advantage over strict

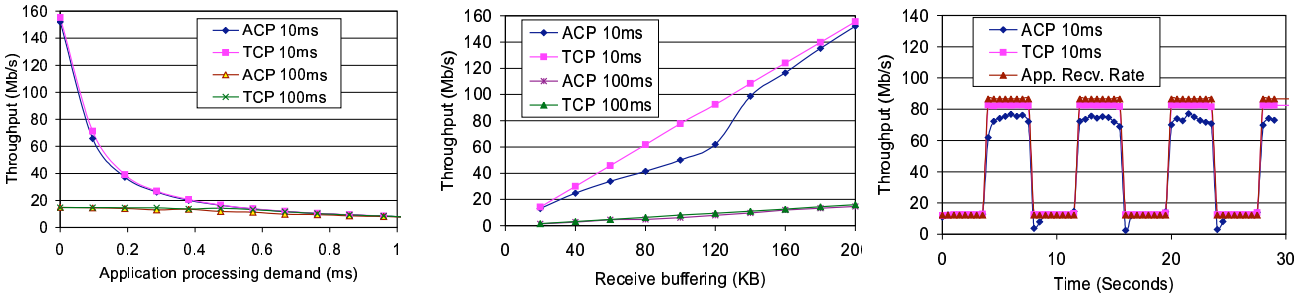


Fig. 5. Behavior of Soft Flow Control (labeled ACP) versus credit-based flow control (labeled TCP) in simple direct-connect scenarios. Its delivered throughput is similar across a range of bandwidth-delay products, consumer speeds, and receive buffer (socket) sizes. However, SFC is less agile.

flow control.

The right graph in Figure 6 illustrates that SFC allocates the peer's buffer space fairly across the ensemble servers for traffic in the outbound direction. In this experiment, the peer's receive window (67KB) limits the aggregate outbound traffic to 40Mb/s. Initially each server transmits at 10 Mb/s, but one of the active servers stops transmitting after each five-second interval. SFC quickly allows the remaining senders to increase their rates to fill the idle capacity. Beginning at time 25, one of the inactive servers resumes transmitting after each five-second interval. Again, SFC allocates the outbound flow window fairly among the active servers. In contrast, strict flow control (not shown) constrains each server to 25% of the peer's receive capacity, even when unused capacity exists; strict flow control fully utilizes the receiver's capacity only when all servers are transmitting simultaneously.

## B. Anypoint-XCP Rate Control

We now evaluate the fairness of Anypoint-XCP congestion control with respect to bandwidth sharing on wide-area transit links. These experiments focus on fair sharing of transit bandwidth among Anypoint and non-Anypoint connections, using variants of the network topology in Figure 4. These experiments use a fully XCP-enabled network, including emulated XCP routers and Anypoint-XCP transform functionality in the transport switch.

We first compare the behavior of Anypoint-XCP with ur coordinated TCP-like connections (ACP-Reno) for traffi outbound from the ensemble, in which each ensembl server transmits at its maximum rate to the connectio peer. These experiments constrain the transit link banc width to 20 Mb/s so that it becomes the bottleneck. W also introduce cross-traffic flows on the transit link (fro node E to node F in Figure 4), adding an additional cross-traffic flow after each 10-second interval. The graphs show the aggregate outbound throughput of the Anypoint connection along with each of the cross-traffic flows. All link latencies are 5 ms.

As a baseline, Figure 7 shows the behavior of standard TCP rate control mechanisms (ACP-Reno) for outbound flows. The uncoordinated outbound ensemble flows compete on an equal basis with other flows for bandwidth on the transit link. The cross-traffic flows share bandwidth fairly amongst themselves, modulo the effects of TCP slow start and fast retransmit/recovery. However, the outbound traffic from the ensemble consumes a transit-link bandwidth share proportionate to the ensemble size. Thus TCP and ACP-Reno are not session-fair in Anypoint scenarios for outbound (multipoint-to-point) traffic. This behavior is well understood for uncoordinated TCP flows, and the behavior is similar in this experiment with Anypoint communication without new mechanisms for multiflow rate control.

In contrast, Figure 7 shows the behavior for the same experiment using the XCP-enabled transport (ACP-XCP) for all traffic. In this case, transmission rates converge rapidly, and bandwidth on the bottleneck transit link is shared fairly among all flows. In particular, the multipoint-to-point traffic of the Anypoint connection shares bandwidth fairly with the cross-traffic flows: Anypoint with ACP-XCP is session-fair. The top line gives the aggregate throughput across the bottleneck transit link, showing that Anypoint-XCP uses the available link bandwidth efficiently. The results for similar experiments in the inbound direction are almost identical, and we omit them.

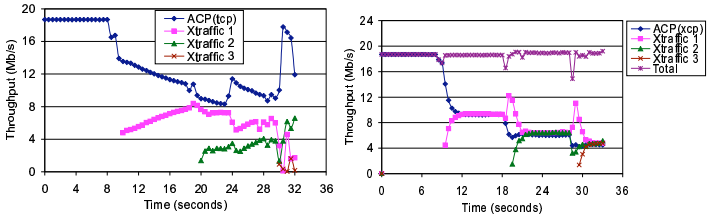


Fig. 7. These two graphs compare the behavior of an outbound Anypoint connection using either standard TCP congestion control mechanisms (ACP-Reno) (left) or coordinated XCP rate control (right). The bottleneck transit link is shared with other cross-traffic ("Xtraffic") flows.

Last we study the impact of asymmetric RTTs on the stability of the Anypoint-XCP transforms. The experiments in Figure 8 use 3 ensemble nodes connected by 10Mb/s links, while the RTT to one server increases from 15 to 68ms. Small, transient fluctuations occur when we dynamically change the link's latency via the sysctl interface on the ModelNet emulator node. The inbound rate remains unchanged until the flow becomes send-window limited at an RTT of 54ms. At that point the flow achieves it's window-limited rate, where it was previously limited by the sum of the outbound links. The outbound rate never changes because each ensemble member's window is sufficient for its portion of the total flow.

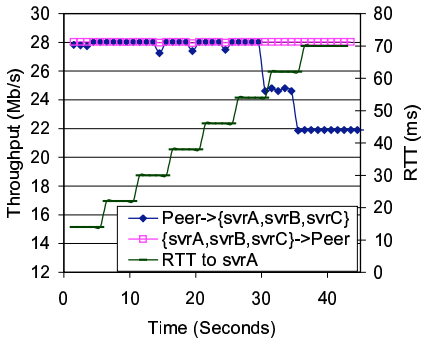


Fig. 8. Anypoint-XCP transforms are stable under asymmetric RTTs to ensemble endpoints (10Mb/s links) for inbound and outbound flows.

### C. Anypoint Connection Fairness

In the following experiments we verify that Anypoint flows are session fair with respect to one another. Here there are two simultaneous connections to the ensemble. The leftmost graph in Figure 9 illustrates the experiment: Connection 1 to server A and connection 2 to server A and B. We increase the transit link bandwidth to the peer every 5 seconds by 2Mb/s. Figure 9 shows that the inbound flows are fair across the transit link until it reaches 14Mb/s. At that point the first connection is limited to half of server A's path, 5Mb/s. Eventually the second connection is limited by its indirection schedule, $\frac{S_{min}}{\beta_{min}} = \frac{5}{0.5}$, to 10Mb/s.

The rightmost graph in Figure 9 shows the total throughput of both connections as they are outbound (multipoint-to-point) from the ensemble. Again, when the transit link capacity reaches 14Mb/s, the first flow is limited by its fair share, 5Mb/s, along server A's link. However, the second connection continues to increase throughput as its flow from server B is unconstrained.

### D. Cascaded Anypoint

Multiple, cascading switches, can be used to compose tiered services within a single data center or grid site. Here we show that a cascade of three switches produces nearly identical flow behavior as a single reduced topology with a single switch. Recall from Section III-A that inbound flows are send-window limited by the maximum RTT to any ensemble member. We construct the reduced topology in Figure 10 by simply summing the link latencies along the longest path in the cascade. Each switch splits the inbound flow evenly among its ensemble, and we measure throughput as the delay on link $c1$ is increased from 2ms to 16ms. The middle graph shows that the cascade flow receives nearly identically inbound throughput as the flow across the reduced topology. The middle graph shows that each flow in the cascade achieves a third of its peer's flow.

### E. Related Work

Multipoint-to-point rate control has been explored in the context of ATM available bit rate (ATM-ABR) networks and, more recently, in IP networks [1]. Like XCP, ATM-ABR networks allow switches to return explicit rate information to senders. The work in this area uses various algorithms to merge rate information for multipoint-to-point flows [12, 13]. We use a related approach, merging and splitting rate information within switches, but do so in the context of IP-based transport protocols.

Inbound flows exhibit many of the same challenges as rate-controlled multicast [14, 15]: avoiding per-receiver state in the sender, maximizing bandwidth to each receiver, and achieving a fair share of bandwidth across bottleneck links. However inbound flows differ in that they send disjoint data to each ensemble member.

Last, note that endpoints within an Anypoint ensemble are not co-located at a single host. This is in contrast to systems that share congestion information across transport-layer flows at a single host [16, 17]. Our notion of session fairness is similar to systems that calculate a TCP fair-share between edge-routers for cluster-to-cluster communication [18], however our solution automatically partitions the share across the ensemble in a max-min fair allocation.

## VII. Conclusion

This paper presents a unified scheme for multiflow rate control that combines flow and congestion control for Anypoint communication. We present results which show that XCP offers an elegant solution to multiflow rate control. Lightweight XCP transforms in edge switches can provide session fairness for Anypoint connections. Connections share bottlenecks between switch and peer fairly among all XCP streams, and allow each ensemble member fair access to the total outbound bandwidth. Additionally, this work demonstrates how two point-to-point transport features, framing and explicit rate control,

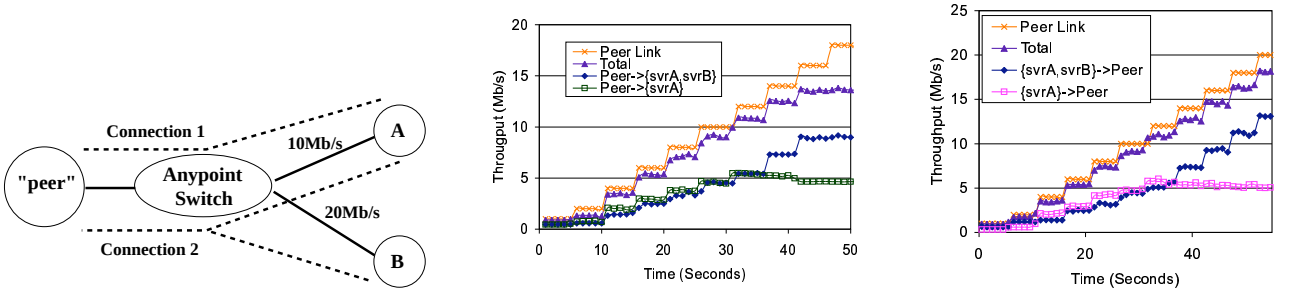


Fig. 9. Here two simultaneous connections access the ensemble, as the transit link ("Peer Link") to the peer increases in capacity every 5 seconds. Both inbound connections (center figure) and outbound connections (on right) are fair across the shared link to peer until it reaches 14Mb/s. At that point Connection 1 is limited by its fair share to server A.

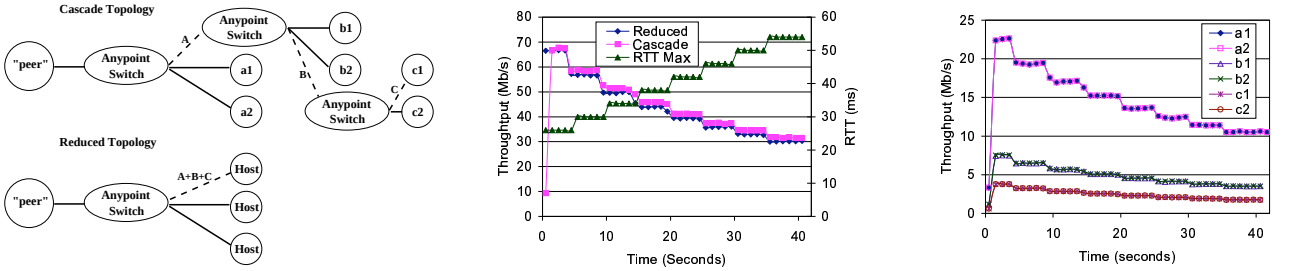


Fig. 10. A cascaded topology of Anypoint switches should behave identically to a server on a link with identical delay or bandwidth. Here an inbound flow behaves identically with either one or three switches as the maximum RTT increases ("RTT max"), and the constituent flows (rightmost graph) (a1,a2,b1,b2,c1,c2) achieve their respective allocations.

combine to form a powerful and useful transport for Anypoint communication.

## References


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

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

[3] 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 ACM Eigth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 1998.

[4] 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: s elected papers from the Fourth Symposium on Operating System Design and Implementation (OSDI), October 200 0*, vol. 20, no. 1, February 2002.

[5] J. Satran, D. Smith, K. Meth, C. Sapuntzakis, M. Wakeley, P. V. Stamwitz, R. Haagens, E. Zeidner, and L. D. Ore, "Internet Engineering Task Force, Internet draft: iSCSI," January 2000.

[6] D. Ott, T. Sparks, and K. Mayer-Patel, "Aggregate congestion control for distributed multimedia applications," in *Proceedings of IEEE Infocom*, 2004.

[7] R. Wu and A. Chien, "GTP: Group transport protocol for lambda-grids," in *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, April 2004.

[8] K. Yocum, D. Anderson, J. Chase, and A. Vahdat, "Anypoint: Extensible transport switching on the edge," in *Proceedings of the 4th USENIX symposium on Internet technologies and systems (USITS)*, March 2003.

[9] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," in *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, August 2002.

[10] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator," in *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI)*, December 2002.

[11] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proceedings of the ACM Conference on Communications Architectures and Data Communication (SIGCOMM)*, 2000, pp. 43–56.

[12] S. Fahmy, R. Jain, R. Goyal, and B. Vandalore, "Fair flow control for ATM-ABR multipoint connections," *Computer Communications*, vol. 25, no. 8, pp. 741–755, May 2002.

[13] U. Nguyen, "Evaluation of flow control algorithms for ABR multipoint services," *Ninth International Symposium on Computer Communications*, vol. 2, pp. 1077–1084, July 2004.

[14] 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*, vol. 5, no. 6, pp. 784–803, 1997.

[15] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like congestion control for layered multicast data transfer," in *Proceedings of IEEE INFOCOM*, 1998, pp. 996–1003.

[16] 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.

[17] L. Eggert, J. Heidemann, and J. Touch, "Effects of ensemble-TCP," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 1, pp. 15–29, January 2000. [Online]. Available: http://www.isi.edu/~johnh/PAPERS/Eggert00a.html

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