Directions in Internet Transport Evolution

By Bryan Ford

At the Transport Area Open Meeting at IETF 70, area directors Magnus Westerlund and Lars Eggert noted that their queues are empty and that several Transport Area working groups are nearing completion. It may therefore be an opportune time to step back and consider possible directions for new projects in the Transport Area. This article summarises some of the ideas presented and discussed at that meeting, and it attempts to synthesise those ideas into an outline that describes a number of potentially promising directions for future Transport Area work. Identifying a complete shopping list of issues or work areas was not the goal of the meeting, nor is it the goal of this article. Instead, what follows can be best described as a cross section of possibilities.

The topics discussed at the meeting and summarised here fall into three main categories: *transport semantics, traffic management,* and *end/middle interaction*.

- *Transport semantics* is concerned with what transport abstraction the application writer sees or would like to build on. Described here is a new experimental transport protocol I presented at the meeting and how it compares with several existing transports.
- *Traffic management* is concerned with mechanisms that allow transport endpoints to take advantage of available network bandwidth while being fair to other applications and users competing for network resources. This section summarises some of the key issues in defining fairness, as identified by Bob Briscoe in his presentation.
- *End/middle interaction* is concerned with the interaction between the transport protocols at the endpoints, which traditionally assume that they have a clear end-to-end path provided by the IP layer, with middleboxes (such as NATs) and firewalls that intentionally obstruct or otherwise complicate this end-to-end path in various ways.

Transport Semantics

TCP introduced the ordered byte stream abstraction on which most Internet applications have been built. The conceptual simplicity, elegance, and flexibility of this minimalistic stream abstraction continue to be among TCP's greatest strengths. Unfortunately, one of the basic assumptions embodied in this abstraction—that all bytes communicated in one direction as part of a given stream must always be delivered in order—creates practical performance problems for modern Internet applications that did not exist when TCP was designed. Because one lost packet in a TCP stream holds up all data queued behind it until the lost packet has been successfully retransmitted, TCP is almost unusable for real-time audio or video, in which it is much better just to drop and interpolate over isolated lost frames than to delay delivery of a whole series of frames. Modern Web browsers and other transaction-oriented applications similarly challenge TCP's simple, totally ordered stream abstraction, with their need to submit many logically independent or parallel requests to one or more servers efficiently (for instance, to load all of the images and other embedded objects on a complex Web page). TCP forces applications to choose between (1) using one stream per transaction, as in HTTP 1.0, which can be inefficient due to the costs of creating and destroying short-lived streams, and (2) multiplexing many logical transactions onto a smaller pool of streams, as in HTTP 1.1, which creates the same head-of-line blocking problem, as in real-time media applications, where one lost packet delays delivery of all of the (potentially unrelated) transactions queued behind it on the same stream.

TCP's limitations have been recognised for years, and they constitute the motivating force behind the development of several alternative transport protocols, such as RDP, SCTP, and DCCP. While each one of the alternative transports tends to address heavily overlapping needs and problem areas, each one also takes a different approach to solving them. A common feature of all existing alternative transports is that they move away from TCP's conceptually simple *byte stream* semantics and toward transport abstractions. DCCP provides unreliable, unordered delivery equivalent to UDP but with congestion control. RDP provides reliable, optionally sequenced message delivery with congestion control. SCTP provides reliable, optionally sequenced message delivery similar to RDP, but it also allows the application to associate each message with one of several *logical streams* within the application's transport connection, thereby permitting messages on different streams to be delivered out of order. A common issue with all of these message-oriented transports is that their message abstraction does not scale arbitrarily the way that TCP's byte stream abstraction does; instead, the application must break up large transactions into reasonably sized messages to avoid subtle performance problems or outright transport failures. This reasonable-size threshold is not generally well-defined and often varies with network conditions.

Continued on next page

Directions in Internet Transport Evolution, continued from page 29

At the Transport Area Open Meeting, I presented a new experimental transport called *Structured Stream Transport* (SST). Instead of moving away from TCP's familiar, conceptually simple, and scalable byte-oriented stream abstraction like other alternative transports do, SST enhances TCP's byte stream abstraction to permit applications to use streams in larger numbers easily and efficiently. With SST, for example, transaction-oriented applications like Web browsers need not either multiplex many transactions onto one TCP stream, as in HTTP 1.1, or retool to run on a message-oriented transport; instead, the application simply opens one new stream per transaction and relies on SST to implement those streams efficiently enough, whether it needs a few large streams, or a large number of short-lived streams, or some of each. An audio-streaming or video-streaming application on SST can preserve the transmission independence of separate frames simply by (1) opening a new (rather short-lived) stream for each frame it wishes to transmit and (2) using SST to take on the challenge of transmitting those ephemeral streams efficiently. Thus, SST's philosophy is not to discard TCP's serial byte stream abstraction but to adapt it to the demands of modern applications that demand nonserialised communication.

SST's main application-visible enhancement to TCP's stream abstraction is what amounts to a fork operation, meaning that given any existing SST stream, either endpoint can initiate a new stream as a child of that existing stream. The other endpoint accepts this child stream by performing a listen and accept on its corresponding end of the parent stream rather than on a traditional listen socket. For example, a Web browser using SST might open a top-level stream to communicate with a particular Web server, and then open a child of that stream to fetch the HTML for a given page on that server, and then further fork a Web page's HTML stream to load each of the embedded objects on that page. SST thus organises streams into a heredity structure—hence the term *structured streams*. Because SST preserves and communicates this heredity structure between the participating hosts, applications do not have to bind port numbers or authenticate each new stream: a child stream always starts with a clear communication context defined by the parent stream it is derived from.

One of the basic assumptions embodied in the ordered byte stream abstraction—that all bytes communicated in one direction as part of a given stream must always be delivered in order—creates practical performance problems for modern Internet applications that did not exist when TCP was designed.

Once created, each SST stream is independent and provides semantics essentially identical to TCP streams, including reliable delivery, ordering, and flow control independent of all other streams. The SST protocol contains optimisations that allow the application to create and start sending data on new streams, with no three-way handshake delay as in TCP, and SST can destroy streams without maintaining their state for a four-minute TIME-WAIT period as in TCP. All SST streams between given pairs of endpoints automatically share congestion control state, thereby avoiding the performance costs of a separate slow start for each new stream. The application can limit the length of time a stream's data is buffered for retransmission, which permits SST to be used for unreliable delivery when needed, such as when streaming media. For further details about the protocol, see my SIGCOMM paper.[1]

A natural question, then, is, How does SST differ in practice from SCTP, which also supports the sharing of congestion control and some other transport state among multiple logical streams? In addition to the basic semantic difference between SST's TCP-like *streams of bytes* and SCTP's RDP-like *streams of messages*, there are two, key, pragmatic differences. First, an SCTP application cannot dynamically open or close individual streams; instead, it opens a *connection* representing all the streams it will need, and SCTP negotiates the number of streams to be multiplexed onto that connection only once during connection setup. Second, SCTP provides receiver-directed flow control only for the entire connection and not independently for individual streams. The application can receive only the next message available on *any* stream and cannot pick a particular stream on which to receive. This means that the receiver cannot hold off the sender's transmission on one stream—such as in the case of a video file being downloaded for playing at a constant frame rate—while continuing to accept data on another stream, such as in the case of a file being downloaded to disk as quickly as the disk will accept it. SST streams, in contrast, work like fully independent TCP streams, only implemented more efficiently: the application can open and close them at any time and can read from some streams while holding off the sender on others.

Could SST be implemented as a layer on top of SCTP? Yes and no. It may be relatively straightforward to imple-

1. "Structured Streams: A New Transport Abstraction," Bryan Ford. ACM SIGCOMM, August 2007. http://www.bford.info/pub/net/sst-abs.html.

ment SST's hierarchical stream abstraction with dynamic open/close on top of SCTP, making use of SCTP's fixed set of streams negotiated at connection time as a pool of low-level message streams on which to multiplex SST's TCP-like byte streams. Making up for SCTP's lack of independent per-stream flow control may be more difficult to do this way, however, because it would require the adaptation layer to maintain an additional set of send and receive buffers between SCTP's and the application's, thereby subjecting data to additional copying on the critical path.

SST is still in an early experimental stage, and all transports other than TCP and UDP face serious deployment challenges due in part to the end/middle interaction issues discussed later. Nevertheless, the amount of effort expended over the past decade on alternative transports with relaxed ordering and delivery semantics suggests that there is clearly a widely perceived need for such alternatives to TCP, even if the best approach is not yet clear.

Traffic Management

A large portion of ongoing transport-related work both in the IETF and elsewhere is concerned with mechanisms for controlling the flow rate of network traffic. The goal is to permit applications to take full advantage of whatever bandwidth is available over a given path while ensuring that different applications and users share available bandwidth fairly and avoiding congestive collapse. The Internet's traditional approach to traffic management has been via end-to-end congestion control implemented in such transports as TCP, by which the transport dynamically senses the amount of bandwidth available and adjusts its transmission rate to match. A large portion of Transport Area work both within and outside of the IETF is devoted to development of new congestion control algorithms or to refinement of existing ones.

Unfortunately, the traditional end-to-end congestion control approach suffers from a serious flaw: since each transport endpoint has a limited view of the network, in which it sees only the results of its transmission attempts on particular end-to-end paths, the whole notion of fairness can be seen by a transport protocol only in terms of fairness between end-to-end *flows*. As Bob discussed in his presentation at the Transport Area Open Meeting, network operators tend to think of fairness not in terms of flow rate equality but in terms of volume accounting—in other words, in terms of how much traffic load a particular application or user is placing on the network regardless of whether that load consists of one end-to-end flow or many and regardless of whether the user causes the flow to be active continuously or intermittently. The problem with the traditional per-flow definition of fairness is exemplified by BitTorrent, an application that routinely uses dozens of concurrent TCP connections to different remote hosts. Each of those connections uses standard TCP congestion control and thus is entirely fair—in the per-flow sense—to other TCP applications. However, because BitTorrent's dozens of flows in aggregate consume dozens of times the bandwidth of a competing application using only one flow and because all of its flows run continuously, it is perceived by users and administrators as extremely unfair to other applications.

On the other hand, the volume-accounting view of fairness fails to take load variability into account. In other words, a given volume of traffic that causes considerable congestion during a time of peak load might cause little or no congestion at other times. Enforcing a simplistic view of fairness in terms of volume accounting can thus prevent applications from opportunistically taking advantage of available network capacity.

ISPs are increasingly deploying traffic-rate control devices in the middle of the network. Those devices sometimes attempt to enforce a sense of fairness among different applications and/or users. Unfortunately, the rate-control policies the ISP can enforce effectively are limited by what an ISP's routers can heuristically discover through deep-packet inspection; even when the ISP's intentions in setting the rate-control policies are honorable, the results often do not really correspond to what either the ISP or its users expect, thereby causing confusion and anger.

Network operators tend to think of fairness not in terms of flow rate equality but in terms of volume accounting—in other words, in terms of how much traffic load a particular application or user is placing on the network regardless of whether that load consists of one end-to-end flow or many and regardless of whether the user causes the flow to be active continuously or intermittently.

There are many difficulties in trying to come up with a truly workable notion of fairness for traffic management on the Internet. Bob proposes that deciding on such a notion is not the IETF's job; instead, he says, the IETF should focus on developing design-time accounting metrics and management mechanisms that enable sensible resource-sharing

Continued on next page

2. For details on the proposal, see draft-briscoe-tsvwg-relax-fairness-00.txt and Bob Briscoe's presentation slides at http://www3.ietf.org/proceedings/07dec/slides/tsvarea-3/sld1.htm.

Directions in Internet Transport Evolution, continued from page 31

policies to be enforced at run time.[2] Whatever the case, finding a reasonable way to escape both the per-flow fairness mind-set of traditional congestion control and the congestion-insensitive volume-accounting mind-set of administrative traffic control mechanisms and synthesising them into a scheme that enables end-to-end transports to work with middle-of-the-network devices so as to create truly useful fairness policies represent the next big challenges to be addressed in the Transport Area.[3]

End/Middle Interaction

The IETF already has three working groups—MIDCOM, NSIS, and BEHAVE—that are concerned at least in part with improving the way traditionally end-to-end transports traverse and interact with middleboxes, such as NATs and firewalls. Several non-IETF projects, such as UPnP and NAT-PMP, take similar-but-different approaches to the middlebox interaction/traversal problem. Still another approach—extending STUN into a middlebox control protocol—was discussed at the SAFE BOF. Why did we end up with so many different approaches to solving the same problem? Because, as Lars pointed out in the open meeting, any of us can design a middlebox control protocol, but "nobody has designed one that anyone really wants to deploy."

We cannot simply hope that this problem will go away during the transition to IPv6, because even if NATs eventually disappear, firewalls are clearly here to stay. Many feel that NATs are here to stay, too, because of perceived benefits unrelated to IPv4 address space limitations such as administrative isolation/modularisation of address space and obfuscation of internal addresses from the viewpoint of external hosts. Furthermore, there appears to be renewed interest in using NAT to provide interoperability between the IPv4 and IPv6 universes. Because IPv6 is only now starting to see widespread deployment, it is still somewhat malleable, so now would be a great time to make any changes required to enable IPv6 transport protocols to work well over NATs and firewalls.

Because IPv6 is only now starting to see widespread deployment, it is still somewhat malleable, so now would be a great time to make any changes required to enable IPv6 transport protocols to work well over NATs and firewalls.

One possible explanation for the lack of deployment of current middlebox control/interaction protocols is simply that the need for them is not yet great enough, but perhaps it will be soon. As IPv4 address space pressure increases, the cost of static IP addresses will increase, and higher-profile players will start seeking cost-effective solutions to give their hosts full functionality even from behind NATs. As multilevel NAT scenarios become more prevalent and adjacent network domains increasingly end up using overlapping private address spaces, current traversal solutions make legitimate traffic arrive at unintended destinations, creating both efficiency concerns and security concerns that may increase the pressure for explicit middlebox control.

Mobility may also create pressure for the deployment of control protocols in order to reduce the power-draining keepalive traffic that current ad hoc traversal solutions require to hold their UDP bindings open. On the other hand, simply moving middleboxes and applications away from fixed-rate binding timers and keepalives and toward using binding timers with exponentially increasing periods might address the keepalive problem without explicit middlebox interaction.[4]

Therefore, on one hand, maintaining a wait-and-see attitude toward the current crop of middlebox control mechanisms and avoiding new work in this area until the waters clear up a bit might be an appropriate strategy. On the other hand, there are inherent risks with this strategy, mainly because most of the current mechanisms have limitations that may further increase the Internet's brittleness if those protocols become widely deployed without undergoing more-careful analysis and standardisation. For example, some control protocols, such as UPnP and NAT-PMP, do not address multilevel scenarios at all, whereas others can, but only when adjacent private address spaces do not overlap. This suggests that one potentially worthwhile near-term project in this area is to perform a careful, mechanism-neutral, side-by-side analysis of the currently available middlebox interaction mechanisms, clearly identifying the limitations of each and the potential risks to the Internet's future evolution if a given mechanism were to become the de facto standard for end/middle interaction. If we can't identify the right middlebox control mechanism, at least we can try to consolidate what wisdom we do have on the alternatives. This should provide useful guidance for vendors and customers that may now or in the future be considering deployment of middlebox control mechanisms.

3. For more-complete background and motivation, see "Flow Rate Fairness: Dismantling a Religion," Bob Briscoe. ACM CCR 37(2) 63-74, April 2007. http://www.cs.ucl.ac.uk/staff/B.Briscoe/projects/refb/#rateFairDis.
4. See "A Simpler Way to Reduce Keepalive Traffic." http://www1.ietf.org/mail-archive/web/safe/current/msg00073.html.