

Design Space for a Bulk Transport Tool

Stanislav Shalunov
Internet2

Lawrence D. Dunn
Cisco

Yunhong Gu
UIC

Steven Low
Caltech

Injong Rhee
NCSU

Steven Senger
UW-La Cross

Bartek Wydrowski
Caltech

Lisong Xu
UNL

May 2005

Contents

1	Tool Requirements	2	2.4.7 Request Compression	9
2	Design Space Dimensions	2	2.5 Window-based <i>vs</i> rate-based protocols	10
2.1	Explicit congestion feedback	2	2.6 TCP-compatible <i>vs</i> TCP-friendly . . .	10
2.1.1	Source Quench	3	2.7 State mostly kept at the sender <i>vs</i> the	
2.1.2	Explicit Congestion Notification	3	receiver	10
2.1.3	eXplicit Congestion Protocol	3	2.8 Single <i>vs</i> multiple streams	11
2.1.4	MaxNet	4	2.9 User and application programmer in-	
2.1.5	Explicit Queuing Delay Ac-	4	terfaces	11
2.1.6	counting	4	3 Conclusions	11
2.2	Implicit input for congestion control	5	4 Future Work	12
2.2.1	algorithm	5		
2.2.2	Loss-based congestion control	5	Status of this Document	
2.2.3	Delay-based congestion control	6	This document is the output of the Internet2 Bulk	
2.2.4	Congestion control based on	6	Transport working group.	
2.3	combined distributions	6		
2.3.1	Discussion	7	Introduction	
2.3.2	Kernel-space <i>vs</i> user-space implemen-	7	The purpose of this document is to describe the de-	
2.3.3	tation	7	sign space for a transport protocol aimed at bulk	
2.4	Protocol features and framing	8	transfer. Several largely independent decisions need	
2.4.1	In-band vs out-of-band signaling	8	to be made in the design process. An attempt is made	
2.4.2	Timestamps	8	to distinguish between separate issues and point out	
2.4.3	Security nonces	9	interdependencies.	
2.4.4	Path MTU discovery	9	The context of this description is building a bulk	
2.4.5	Explicit Congestion Notification	9	transfer tool for high-performance networks that is	
2.4.6	Selective Acknowledgments	9	easy to deploy and use. This limits the space given	

high-performance environment, using an advanced transport tool could also be beneficial on lower-speed links: mainly, the delay through a congested bottleneck could be decreased improving the experience of interactive applications such as telephony and games.

1 Tool Requirements

Bulk file transfer is one of the existing killer applications of Internet2. Many of the scientists using Internet2 (working in high-energy physics, astronomy, and other fields) are primarily interested in transferring large files. Often, we're asked to recommend a way to transfer large files. Presently, we can point to a large body of research, but we cannot recommend any specific tool for reasons specified later. A suitable tool must satisfy these requirements:

1. Easy deployment: anything more complicated than “download this tarball, unpack, make” would make a tool inaccessible to most users.
2. Portability (at least Linux, FreeBSD, and Solaris): operating system conversion (especially if it implies hardware changes) is too much of a burden.
3. Ability to use without administrator privileges on the end machines: the tool needs to be useful on PlanetLab machines, guest accounts, and deployable without the need to negotiate with organizational information technology departments that might have standardized on particular platforms and configurations.
4. Ability to transfer at speed that are close to 1 Gb/s in near term and 10 Gb/s in medium term with, preferably, no fundamental limitations on performance in longer term.
5. Ability to be used both for straightforward bulk file transfer and advanced interactive multimedia applications that can require unreliable (and partially reliable) datagram service.
6. Tolerance for minor non-congestive packet loss.
7. Lack of dependence on network upgrades or configuration changes, especially those that involve all routers along the path.
8. Usability over a wide range of scenarios without complex tuning.

9. Ability to be tuned for the most unusual of circumstances (such as lossy radio links).
10. Reasonable fairness among instantiations of itself and some measure of fairness towards conventional TCP.
11. The protocol needs to resist attacks where a third party attempts to disrupt a transfer.
12. The protocol needs to prevent third parties from directing traffic at unsuspecting receivers.
13. The protocol needs to be designed in such a way that the service provided by a server using the protocol to distribute static files can degrade gracefully when the request load increases (due to natural causes or a malicious attack).

2 Design Space Dimensions

In this section, the design space is mapped out. A major division is between explicit (section 2.1) and implicit (section 2.2) congestion feedback. *Explicit congestion feedback* is direct communication from routers to the communicating hosts of the state of the network or of the prescribed sending rate (or window size), accomplished by sending separate packets or by changing some fields in packets as they traverse the routers, whereas *implicit congestion feedback* transmits the same information incidentally to the delivery of the packets without embarking on making the state of the network known. The tool could be implemented in the kernel or in user-space (see section 2.3). Protocol features (section 2.4), window- and rate-based protocols (section 2.5), TCP-friendliness (section 2.6), state location (section 2.7), the use of multiple streams (section 2.8), and the interface (section 2.9) are also discussed.

2.1 Explicit congestion feedback

Explicit congestion feedback mechanisms include Source Quench [1], Explicit Congestion Notification (ECN) [2, 3], eXplicit Congestion Protocol (XCP) [4, 5], MaxNet [6, 7], and, e.g., a hypothetical scheme where a field is allocated in the IP headers for cumulative queuing delay and each router increments this field by the number of nanoseconds that the

packet spends in its transmission queue (the field is initially zero). The number of different explicit congestion feedback approaches is sufficiently small, so that many individual ones are mentioned.

2.1.1 Source Quench

Source Quench is discredited and discouraged for two main reasons: (i) congestion notification thus originated can be lost (indeed, the notifications are most likely to be lost during severe congestion, when notification delivery is especially critical to prevent congestion collapse of the network) and (ii) since additional packets are generated on a path that is already congested (albeit in the opposite direction), the mechanism itself might contribute to congestion collapse.¹ In addition to these fundamental problems, Source Quench, as specified, does not include any form of a nonce, which allows third parties to spoof Source Quench messages to slow connections down to a crawl, constituting a form of denial-of-service attack; this deficiency could be remedied by changing the Source Quench format to include a copy of the headers of the packet that caused the message to be generated; however, no such remedy is standardized. The current standard [8] says: “A router SHOULD NOT originate ICMP Source Quench messages.” Hereafter, Source Quench shall be considered no further.

2.1.2 Explicit Congestion Notification

ECN uses two bits in the IP header (the two last bits of the field formerly known as TOS) to transmit the following information: (i) whether the communicating hosts would be able to understand ECN signaling and (ii) whether the packet experienced congestion.

¹While the direction in which Source Quench packets are sent is the opposite of the direction that is congested, we believe that the conditional probability of a link being congested provided that the opposite direction is congested is higher than a simple probability of a link being congested. Further, one of the most important states of the network to examine is a state when it is close to congestion collapse. A good congestion control scheme would head away from congestion collapse; the extra traffic sent by the Source Quench mechanism could contribute to congestion collapse.

Unfortunately, ECN is almost exclusively a mechanism to replace some (possibly almost all) packet losses with congestion notifications, without a change in the nature of the congestion control algorithm. In fact, the specification appears to assume that congestion control is necessarily always loss-based or equivalent to being such in the sense that congestion notifications replace loss events as congestion indicators [2]: “We emphasize that a *single* packet with the CE codepoint set in an IP packet causes the transport layer to respond, in terms of congestion control, as it would to a packet drop.” Further, the specification prescribes very specific manner in which the notifications are to be employed: e.g., it prohibits the use of the instantaneous queue length as a basis for sending congestion notifications, while some of the current thinking [9, 10] seems to indicate that this would, indeed, be the better way to design an Active Queue Management (AQM) scheme. Regardless of these prescriptions, the goal of replacing losses with congestion notifications (and thus reducing the loss rate and improving network efficiency) is not relevant to the high-performance environment, as loss already must be very low in such environment. ECN does not in any way change the situation with non-congestive loss, which still limits performance.

2.1.3 eXplicit Congestion Protocol

In XCP, routers keep a small amount of state (the amount of state is independent of the number of connections that pass through the router) that allows them to instruct communicating hosts to use specific window sizes so that the system converges quickly to a state where the link is fully used and fairness is achieved.

There exists empirical evidence [11] that XCP is not effective if the bottleneck router does not support it. The issue of incremental XCP deployment is still open [12]. As the IETF work on the XCP specification is still in its early stages and most routers in the field currently do not support XCP, XCP, while an excellent congestion control scheme, does not appear to be immediately useful to our target community.

Further analysis of XCP [13] indicates that bottleneck link use might not always be optimal.

2.1.4 MaxNet

MaxNet is an approach to congestion control where the source receives only the maximum price on the path. (For comparison purposes, the implicit price fed back to FAST TCP is the sum of the link prices.) Like FAST TCP, the source chooses its sending rate to maximize its own utility function based on the price fed back, however as the price is the maximum link price, the rate allocation results in weighted max-min fairness. The particular way in which the congestion signal is communicated to the senders remains flexible and could include an AQM² scheme or explicit communication via altering packet header fields by the routers. If the latter approach is used, MaxNet becomes another example of a congestion control protocol based on explicit feedback. The deployment properties of such a protocol might be similar to those of XCP, but it might have advantages in resource allocation. In addition, router operation becomes simpler (a router would only need to replace a given field by its current link price—such as the queuing delay on the outgoing interface—if its value exceeds the value already in the field). With MaxNet, which communicates a congestion signal, host behavior might be modified later, should a revision of congestion control approach be deemed necessary, potentially reducing the required number router modifications.

2.1.5 Explicit Queuing Delay Accounting

The hypothetical scheme described above (where routers increment a field in the packet headers that records the total queuing delay experienced by a packet) shares some properties of XCP (e.g., all routers on a path need to support this mechanism before it becomes useful), but instead of instructing communicating hosts to use specific window sizes (or sending rates), routers merely communicate the state of the network explicitly thus enabling the hosts to make informed choices with respect to congestion control. The end hosts could use this information, e.g., in the same way FAST TCP [14, 15] uses queuing

²The term “AQM” is used narrowly here to mean a mechanism that communicates state of the router to network users with packet drops or equivalent actions, such as marking packets using ECN as having experienced congestion.

delay, but since the details of the algorithm would be implemented in software on the end hosts rather than in hardware on the routers, future improvements of the algorithm would be made easier. A disadvantage of such an approach, compared to XCP, would be slower convergence.

2.1.6 Discussion

The use of explicit congestion feedback is likely to result in superior congestion control protocols that converge faster and are less prone to disruption by spurious inputs (e.g., non-congestive packet loss or delay and jitter not related to queuing delay) than protocols that use implicit congestion feedback. However, given a lack of deployment of router features for explicit congestion feedback (and, indeed, lack of present consensus what these features might be), a tool that would need to be developed now would have to rely on implicit congestion feedback. A minor exception to this conclusion is ECN: while ECN will not significantly improve the performance of transport protocols in low-loss environments or improve their resistance to non-congestive packet loss, there is no harm in supporting ECN signaling.

The deployment properties of an explicit congestion feedback scheme would depend on the upgrade requirements: do *all* routers on a path need to be upgraded before a flow traversing the path can start using the notifications, or does the support of the explicit congestion feedback feature by *some* routers already beneficial? ECN was specifically designed to be deployable incrementally. Incremental deployment of more explicit congestion feedback schemes, where the information transferred is not equivalent to a certain number of packet losses, can be difficult: congestion control schemes need to respond to the state of congestion on the bottleneck link; it is, therefore, important whether the router at the head end of the bottleneck link supports the explicit feedback scheme; while it is possible to find out whether all routers on a path support a given feature [16], it is far more difficult to determine whether the router at the head end of the bottleneck link (or the relevant tight link) supports the feature; the problem is further complicated by the fact that the device at the

head end of the relevant link might be a switch rather than a router.

2.2 Implicit input for congestion control algorithm

If explicit congestion feedback is used, there is no need for implicit inputs, except, perhaps, for auxiliary purposes (e.g., when the network is so severely congested that major packet loss causes the explicit congestion feedback signals to be lost). However, given the discussion of section 2.1, implicit congestion feedback inputs need to be evaluated.

The number of different implicit congestion feedback approaches (such as HighSpeed TCP [17], FAST TCP [14], BIC TCP [18], CUBIC TCP [19], Scalable TCP [20], H-TCP [21], HighSpeed TCP–Low Priority [22], Layered TCP [23, 24], and many more) is sufficiently large that describing most of them, or even the representative members, would take significant space. Here, an attempt is made to classify the implicit congestion feedback schemes based on the type of input.

When a stream of packets is injected into the network with no explicit congestion feedback, the only information that is conveyed by the network about the state of congestion is the set of sending times and delivery times of the packets in that stream (for the purposes of this statement, loss is considered to be a special case of extreme, or even infinite, delay). Various parameters can be extracted from this sample, including (i) rate of packet loss, (ii) some measure of overall delay, and (iii) various measures of empirical combined probability distribution of the sample (e.g., packet train timing changes fall into this category).

2.2.1 Loss-based congestion control

Purely loss-based congestion control is limited for two main reasons:

1. Lacking AQM (to which the practicality considerations of section 2.1 apply), to receive congestion feedback, senders need to drive the queue full. Given the widely-adopted rule of the need to have queuing capacity corresponding to the round-trip delay at the bottleneck router [25],

the user-observable delay would need to double before losses would occur.³

2. The ability of the network to communicate the desired sending rate to the sender using losses is limited; it is unlikely that the response function of sending rate to loss probability p can be made to grow faster than $1/p$ as $p \rightarrow 0$.

Non-interactive data transmission applications might well tolerate the increase of round-trip time by a factor of two. However, these applications need to coexist with interactive applications, such as voice over IP (VoIP). (VoIP is merely an example of a kind of interactive application. It is used here because the requirements for voice are better understood than requirements for networked games, remote instrument control, and other applications that require or benefit from low delay.) Acceptably low delay is important for such applications. Voice is presumed by telephony engineers to need at most 150 ms of user-to-user round-trip delay before quality starts to deteriorate; given serialization delay at both ends and the need for some jitter buffer, network delay budget is likely to be close to 100 ms. There is simply no space for delay doubling even for many voice connections that do not span an ocean.

Recent work [26] shows that it might be possible to run routers with buffer space smaller than $\text{RTT} \times \text{capacity}$; namely, $\text{RTT} \times \text{capacity} / \sqrt{\text{number_of_flows}}$ could be used instead, if the number of concurrent bulk TCP flows is sufficiently large (in the hundreds).⁴ While this promises to make future routers cheaper, more compact, and less power-

³The buffer will, of course, depend on the type of the network interface and the device in front of the bottleneck link. The claim of provisioning of buffer space sufficient to hold round-trip time worth of data applies primarily to backbone router interfaces; it also holds for most host operating systems (should the first link be the bottleneck). The claim does not necessarily hold for Ethernet switches, which traditionally come with smaller buffers. However, the presence of large buffers in front of at least some of the bottleneck links makes the claim relevant. (Naturally, if it makes sense for a router to have round-trip's worth of buffers, so it does for an Ethernet switch.)

⁴If the buffer space is to be reduced using this formula, the number of TCP flows that would pass through the router would need to be estimated beforehand. This could be challenging for versatile general-purpose routers.

hungry, the presently deployed routers already have large amounts of buffer memory (at least $\text{RTT} \times \text{capacity}$ and often more), so reducing the factor of 2, by which delay is increased for loss-based congestion control, while potentially possible, requires router configuration or software modification.

The $1/p$ barrier can be explained as follows: as link speed doubles, the tolerance to non-congestive packet loss is at least halved (since non-congestive loss, naturally, cannot exceed total loss). Further, since conventional TCP has a response function of $1/\sqrt{p}$, the tolerance actually decreases by a factor of four. The situation with HighSpeed TCP [17] is in-between ($1/p^{0.82}$). Thus, not only do loss-based congestion control protocols at least double the delay, they also impose increasingly stringent requirements on non-congestive packet loss.

2.2.2 Delay-based congestion control

As we see in section 2.2.1, most of the information about the state of the network conveyed by the bottleneck router is discarded. Delay-based congestion control protocols use a much larger fraction of information. Indeed, a loss-based protocol only extracts about 0.0000000347 bits of information per packet in a connection with a loss rate of 10^{-10} , regardless of method (Additive Increase with Multiplicative Decrease (AIMD) or any other) by which information is extracted from packets; a delay-based protocol might extract perhaps as much as a bit of information from the same packet. It is not surprising that, since delay-based protocols have so much more information about the state of the network than loss-based ones, they can work quite well [27, 28].

Some difficulties of implementing a delay-based congestion control algorithm are:

1. **Fallback to conventional TCP:** Delay-based congestion control works well when it congests the link. When the link is already congested, delay-based protocols will have difficulty distinguishing between full queues and empty queues (the delay, in both cases, is close to being a constant). However, the requisite reaction to a persistently full queue is the exact opposite of that to a persistently empty queue. Misidentification of a full

queue as an empty queue could result in the delay-based flow increasing its sending rate in an attempt to increase the delay (which is already at its maximum) and potentially driving out loss-based flows and perhaps contributing to congestion collapse as the flow continues to respond to congestion with increases in its sending rate. Thus, it is critical that full queues are correctly identified. Once a flow controller knows that the queue is full, fallback to conventional loss-based TCP appears to be the only reasonable behavior.

2. **Filtering out the noise:** Individually measured packet delays will contain significant noise. The measurements will require a filter to obtain a robust measure of delay on the network.
3. **Ascertaining base delay:** Propagation, serialization, and forwarding delay need not affect the sending rate, as only queuing delay characterizes congestion state. Thus, base delay would need to be subtracted from the overall (filtered) measure of delay. The measurement of base delay in the face of transient and persistent queuing, route changes, and noise in the measurements is a formidable task.

2.2.3 Congestion control based on combined distributions

This is potentially an even more promising approach to congestion control than delay-based, as even less information is discarded. In fact, delay-based protocols might use some information from packet trains to distinguish between full and empty queues in the case of near-constant delay.

A whole family of congestion control algorithms could be built based on bandwidth estimation techniques, which attempt to find out, by sending probes into the network, the available spare capacity (the probes could be part of the data stream): namely, a flow might use one of the techniques to estimate capacity and then send at the estimated rate. While, for an ideal algorithm that always flawlessly estimates unused capacity, such an approach would work well for a single sender, it is much less clear what would happen if multiple senders were introduced.

Two protocols that fall into this category are SABUL [29] and SOBAS [30]. In addition, TCP Westwood [31] is a sender-side modification of TCP for networks with large bandwidth-delay products, and with potential packet loss due to both network congestion and transmission errors. Instead of halving the congestion window after detecting a loss, TCP Westwood sets it according to the current *eligible rate*, which is estimated by monitoring the rate of returning acknowledgment packets. In addition, upon persistent lack of congestion, as detected by the *persistent non-congestion detection algorithm*, TCP Westwood increases the congestion window more aggressively.

Methods that measure spare capacity often depend on very precise time measurements (the parameter that is ultimately being estimated is often on the same order of magnitude as the serialization delay at the bottleneck link). This requires microsecond and sub-microsecond precision. Delay-based congestion control, of course, requires time measurements as well, but here the precision requirements are less stringent: millisecond and sub-millisecond. The difficulties involved in such precise measurements become more formidable, given that even a simple invocation of the `gettimeofday()` system call on a Unix-like operating system can take a few microseconds (4.3 μ s on FreeBSD 5.1-RELEASE running on a 2-GHz Mobile Intel Pentium 4 CPU). The use of the TSC register for timestamping can help solve that problem, but the delays and jitter associated with the delivery of packets to the application, as well as factors such as interrupt coalescence [32] still leave the problem more difficult than it is for delay-based protocols that require more lax measurements.

2.2.4 Discussion

Purely loss-based congestion control algorithms appear limited (i) by the need to double the delay before losses occur and the protocols work as designed and (ii) by the slow congestion information transfer rates due to discarding most of the information that the network implicitly provides. Delay-based algorithms have a sound foundation, but might be difficult to implement so that they work robustly in all scenar-

ios. Packet-train-based algorithms, while promising to deliver the best performance and convergence due to high information transfer rates, require measurements even more difficult to obtain than those necessary for delay-based algorithms.

2.3 Kernel-space *vs* user-space implementation

The transport tool could be implemented in the kernel or in user space.

2.3.1 Kernel-space implementation

Kernel-space implementation has traditionally been thought of as more efficient and better performing. An advantage of changing the kernel is that if TCP is replaced, all user applications benefit immediately and transparently.

2.3.2 User-space implementation

User-space applications (such as SABUL [29]) can deliver performance that is as good as that of kernel implementations [33]. Kernel-space implementations tend to encourage event-driven approach, which is more conducive to window-based control than to rate-based.

2.3.3 Discussion

A kernel-space implementation has an inherent efficiency advantage over user-space implementations: with a user-space approach, each packet⁵ requires a pair of context switches (in this case, transitions between user and kernel mode and *vice versa*); with a kernel-space approach, the number of context switches per byte can be reduced, since the I/O block size can be made larger than the packet size. The increase of CPU use with user-space tools might or might not matter depending on its size. It can be difficult to compare CPU cycles use by different programs, as implementation efficiency and degree of optimization might not be identical. For the purposes of

⁵This is only true for packets that are not fragmented. See section 2.4.4 for discussion of fragmentation.

this design exploration, it, thus, appears reasonable to try to estimate the loss of efficiency with user-space tools. Assume that a pair of context switches takes $4\ \mu\text{s}$; this is a reasonable number for a CPU on a computer than might be equipped with a 1-Gb/s network interface. Assume that the MTU is 1500 B. For the link to become fully saturated, about 86,000 packets/second are needed, so the overhead associated with context switches of a user-space tool might account for approximately 35% of the available CPU capacity. While not, by any means, insignificant, this amount of CPU use might be appropriate for some applications, especially if the CPU would have been partially idle otherwise.

To summarize, kernel-space approach

1. is consistent with the traditional Unix approach of implementing transport within a monolithic kernel;
2. can use less CPU;
3. transparently provides transport modifications to all applications.

User-space approach

1. can perform as well as kernel-space;
2. is easier to deploy, especially on machines administered by others;
3. is easier to support in a production environment that requires a stable kernel;
4. is easier to experiment with and develop due to ease of modification and deployment; and
5. leaves somewhat more leeway in choosing between rate-based and window-based control.

This suggests that, at least in the short term, the advantages of a user-space approach outweigh those of a kernel-space one. In any case, a tool prototype should use the user-space approach.

2.4 Protocol features and framing

The features and framing depend on the choice between implementing the tool in the kernel or as a user application or library. If the implementation is a kernel patch that provides a drop-in replacement for TCP, no questions of protocol and framing arise. Therefore, for the purposes of this section it is assumed that the user-space approach is taken. Presently, the choice between TCP and UDP is the

same as the choice between user space and kernel space.⁶

2.4.1 In-band vs out-of-band signaling

Desired rate signaling could be done in-band or out-of-band. In-band signaling has the following advantages (and no obvious disadvantages):

1. it is less likely to be filtered out by firewalls;
2. it uses a little fewer packets;
3. it appears more susceptible to being made more robust in the sense of failing closed (it is a desirable property for the protocol to back off if both data and information about the state of congestion stop coming);
4. it is likely to work somewhat better with network address translators;
5. it uses fewer file descriptors and other logical local resources;
6. it appears to be more elegant and clean (this is, of course, entirely subjective).

While these advantages are relatively small, the lack of disadvantages points strongly towards the use of in-band signaling.

2.4.2 Timestamps

Delay-based congestion control will benefit from having timestamps in the packets: the reverse traffic problem would become easier to solve by using measurements of one-way delay. Regardless of the short-term choice of congestion control algorithm, the framing should contain timestamps with resolution sufficient for the purposes of both delay- and packet-train-based congestion control, so that packet format does not preclude future congestion control choices.

Note that with the use of the TSC register for timestamping, the performance should not suffer, as long as the time representation is not overly complex.

⁶It is possible to consider hypothetical kernel TCP implementations that, as part of the API, allow the user to specify the functions to use in congestion control instead of the standard ones. Implementations of UDP that allow the user specify congestion control algorithms to run in kernel space are also potentially possible. We do not know of such implementations.

2.4.3 Security nonces

Transporting large amounts of data opens an avenue for attackers to direct traffic to unsuspecting third parties. This could be done with relatively small amounts of data. Conventional TCP is protected against this attack by two mechanisms: (i) hard-to-guess initial sequence numbers and (ii) the ability of the party under attack to close a connection by sending a reset packet. The former measure is more important in preventing the attack in practice, but, should an attack succeed, the latter mechanism would help mitigate it. Similar, but stronger, mechanisms, such as explicit security nonces, should be included in future transport mechanisms.

2.4.4 Path MTU discovery

The ability to use larger MTU than some *de facto* Internet minimum, such as 1500 B, is important in reducing the number of interrupts, context switches, memory fragmentation, and, thus, in achieving good performance. IP fragmentation—and, especially, en-route fragmentation—should, of course, be avoided.⁷ User-space applications cannot easily receive ICMP Fragmentation Needed messages traditionally used for path MTU discovery [34]. However, these messages (along with other ICMP messages) are increasingly becoming blocked by firewalls; this in any case necessitates a solution that does not rely on out-of-band MTU signaling. Work is under way [35] to up-

⁷In the past, fragmentation was sometimes caused intentionally for what were considered to be performance reasons; e.g., the Network File System (NFS) block size, which translated directly to UDP `send()` size, was traditionally chosen to be 4 kB or 8 kB even on Ethernet networks with an MTU of 1500 B. To some extent, this practice might reflect differently optimized code paths where fragmentation and reassembly of IP packets happens to work faster than corresponding fragmentation and reassembly in the application. However, more importantly, this technique of sending UDP packets larger than the MTU also allows to reduce the number of context switches from two per about 1460 B of data to two per 8 kB of data. This way of reducing the number of computationally expensive context switches might be inevitable in some circumstances, but clearly is a kludge. And, of course, fragmentation *en route*, where the CPU load is shifted from the data sender (an end host) to the fragmenting router, is not a practice that network operators would find most considerate.

date the path MTU discovery algorithm so that it does not rely on ICMP. This new algorithm should be used.

2.4.5 Explicit Congestion Notification

As discussed in section 2.1, ECN should be eventually supported. Depending on where state is kept, this might or might not require special header fields to carry information about congestion notifications (if the state is kept at the receiver, there will be no need for such fields).

2.4.6 Selective Acknowledgments

Originally, TCP used cumulative acknowledgments only. The need to infer which segments were actually lost and require retransmission hampers performance and creates wasteful unnecessary retransmissions. Modern TCP uses Selective Acknowledgments (SACK) [36, 37, 38] to avoid the problem. The protocol framing design should allow retransmitting only specific lost packets. Preferably, the mechanism for retransmission should be the same as for regular transmission.

2.4.7 Request Compression

Consider a unidirectional data transfer. Let us call the ratio of the number of packets sent in the direction opposite to the direction of the data flow to the number of packets sent in the direction of the data flow the *request compression ratio*. For example, the request compression ratio of the standard TCP is 1, whereas the same ratio for TCP with delayed acknowledgments is 0.5. To use the network resources efficiently, it is desirable to make this ratio sufficiently small and, perhaps, to be able to control it; in particular, values considerably smaller than 1 are desirable. Naturally, reducing the backflow of packets (and bytes) is a goal subservient to conveying sufficient information about the state of the network.

2.5 Window-based *vs* rate-based protocols

If the tool is implemented as a kernel TCP patch, this issue is less relevant, as TCP uses window size to control the sending rate. While it might be possible to use a sending rate computed from the window size and an estimate of the round-trip time to reduce bursts of traffic, this approach seems to not have been studied.

Window-based protocols are often thought to be easier to implement, and thus are widespread. Rate-based protocols are thought to be more gentle on the network, since they can have fewer bursts or the bursts can be smaller. This last difference needs to be qualified: self-clocking window-based protocols might not be very bursty, while rate-based protocols running on systems with 10-ms time slices (such as most Unix-like operating systems) could generate noticeable bursts. The choice between window- and rate-based control is probably best left to experimentation once a prototype is written.

Regardless of the choice between rate-based and window-based congestion control, minimizing bursts injected into the network is advantageous (and could, in the case of window-based protocols be accomplished by explicit pacing or inherent self-clocking). If the choice ends up being dictated by the availability of non-busy-wait fine-grained sleep (which is more important for rate-based protocols), the balance might shift in the future, if the ratio of CPU and network capacity changes significantly.⁸

2.6 TCP-compatible *vs* TCP-friendly

Best current practice [39] is often understood to recommend making flows TCP-compatible. TCP-compatible flows behave under congestion like conventional TCP flows (the usual understanding of this definition is that the response function is the same as that of conventional TCP). Making a protocol only produce TCP-compatible flows implies, in particular, that the congestion control algorithm is loss-based.

⁸Some believe that multi-core CPUs might make CPU cycles more abundant, relative to network capacity, than they are now.

Given the poor TCP performance (the median bulk TCP flow on Abilene only receives about 2.5 Mb/s [40]) and the disadvantages of loss-based control outlined in section 2.2.1, requiring strict adherence to TCP-compatibility and, thus, obtaining the same performance as conventional TCP seems counterproductive, at least in the high-performance networking environment.

If one defines a TCP-friendly flow as one that does not affect the throughput of standard TCP flows detrimentally to an extent greater than a standard TCP flow would affect it, the property of TCP-friendliness becomes not only possible, but desirable. (Note that this definition of TCP-friendliness is not necessarily standard; the literature uses the term in various ways, often synonymously with TCP compatibility.) Since delay-based protocols, at least in certain use scenarios, do not appear to affect the throughput of standard TCP flows to any appreciable extent [28], it might be possible to design a tool that both performs better than standard TCP and is TCP-friendly (either using delay-based congestion control or another technique).

2.7 State mostly kept at the sender *vs* the receiver

Many transport protocols implementations keep most of the state on the sender. Since senders are often servers that distribute files all across the network, a trade-off emerges: keeping information on the sender helps provide an environment where it is more difficult for clients to receive an unfair share of the network capacity; yet keeping most state on the receiver helps design the server in such a way that it is more resistant to state exhaustion denial-of-service attacks [41]. It should be pointed out that, even with conventional TCP, a misbehaving receiver can still obtain an unfair share of network capacity with ACK-flooding [42] despite standards work [43] that alleviates the problem.

With TCP, each connection requires a certain amount of memory (which depends on the buffer sizes). The specification does not allow to easily reduce this amount if memory becomes tight; the deterioration of a TCP-based service under increasing

load can thus be quite sharp, once the memory limits are reached. With an approach that keeps state at the receiver, sender memory is only used for optimization of performance; therefore, the deterioration could be more graceful and gradual. Note that even an approach that, like TCP, keeps state on the sender, but, unlike TCP, shifts the implementation to user-space, could help alleviate the sharp degradation problem, because the memory allocated by the sender in user space would be virtual; hence, it will be possible to page some of the sender state out (present-day implementations of TCP are in kernels that do not allow kernel memory to be paged out to disk).

Note that the sender needs to be able to retransmit unacknowledged packets in any case. With static file transmission, the memory required to keep the data available for retransmission could be made the same as operating system's disk cache; under high load, the graceful degradation would manifest itself in an increasing number of disk cache misses.

2.8 Single *vs* multiple streams

The present practice of using TCP includes opening several concurrent connections in parallel. With standard TCP, using n streams instead of 1 (i) increases the slope of the effective congestion window from 1 MSS per RTT to n , (ii) increases the reduction factor of the effective congestion window from $\frac{1}{2}$ to $1 - \frac{1}{2n}$. The resulting increase in aggressiveness partially masks lack of tolerance for minor non-congestive packet loss. Since, with multiple streams, the throughput of any given stream is lower than the target data transfer rate, TCP can operate closer to the range of throughputs for which it was designed. New protocols should not need the crutch of multiple streams—at least not until they are used with data transfers order of magnitude faster than what the protocols are designed for.

2.9 User and application programmer interfaces

The following decisions need to be made with respect to User Interface (UI) and Application Programmer Interface (API):

1. To provide or not to provide a UI; if provided:
 - (a) Is UI graphical? A good portable GUI would require a non-trivial effort to write and is not the focus of any transport work. It might be reasonable to add later if the tool is turned into a finished product.
 - (b) Does it emulate any existing application?
 - (c) What authentication database is used?
2. To provide or not to provide an API; if provided:
 - (a) Use an existing API? If so, using what mechanism would the existing functionality be replaced?
 - (b) Follow an existing API with slightly different function names?
 - (c) Congestion control behavior: block, fail for application retry, or fail with notification of desired rate?
 - (d) How is the number of memory-to-memory copies minimized? The choices made in API selection can affect this, if special preparation is required for the data.
 - (e) Can zero-copy be achieved?
 - (f) Can the same memory be used for disk and network buffers, perhaps with `mmap()`?

3 Conclusions

Based on the discussion in section 2, we draw the following conclusions for the transport tool (the requirements mentioned below are listed in section 1):

1. The congestion control scheme needs to be TCP-friendly without emulating TCP (see requirement 10 and section 2.6).
2. A security nonce shall be used in the protocol (see requirements 11 and 12; section 2.4.3).
3. The tool is to be based on transport with implicit congestion signaling (see requirement 7 and section 2.1.6).
4. The implicit congestion signaling scheme (to be used according to conclusion 3) is to be primarily delay-based with fail-over to a standard loss-based algorithm when persistent congestion is detected (see requirements 6, 4, and 10).
5. The tool is to be written as a user-space application using UDP for transport (see require-

- ments 1, 2, 3, and 5).
- When used to distribute static files (rather than for sending streams of data coming from other applications or for exchanging datagrams for coordination or other purposes), the protocol shall be able to operate so that a minimum of state is kept at the sender and the recipient is responsible for congestion control (see requirement 13 and section 2.7).
 - Provide an API (see requirement 5); in addition, provide a command-line bulk file transfer tool (see requirement 1).

4 Future Work

The next steps in the development of the transport tool are:

- Develop a protocol specification;
- Develop an API specification;
- Experiment with code in parallel with 1 and 2 and use the implementation experience in the design.

References

- Robert Braden and Joe Postel. Requirements for Internet gateways. *RFC 1009, Historic*, June 1987.
- K. K. Ramakrishnan, Sally Floyd, and David L. Black. The addition of explicit congestion notification (ECN) to IP. *RFC 3168, Proposed Standard*, September 2001.
- Neil Spring, David Wetherall, and David Ely. Robust explicit congestion notification (ECN) signaling with nonces. *RFC 3540, Experimental*, June 2003.
- Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *ACM SIGCOMM*, 2002. URL <http://www.ana.lcs.mit.edu/dina/XCP/p301-katabi.ps>.
- Dina Katabi and Charles Blake. A note on the stability requirement of adaptive virtual queue. *MIT Memo*, 2002. URL <http://www.ana.lcs.mit.edu/dina/XCP/avq-memo.ps>.
- Bartek Wydrowski, Lachlan L. H. Andrew, and Moshe Zukerman. MaxNet: A congestion control architecture for scalable networks. *IEEE Communications Letters*, to appear. URL http://netlab.caltech.edu/~bartek/Data/MaxNet_Stability.pdf.
- Bartek Wydrowski, Lachlan L. H. Andrew, and Iven M. Y. Mareels. MaxNet: Faster flow control convergence. to appear. URL http://netlab.caltech.edu/~bartek/Data/MaxNet_Speed.pdf.
- Fred Baker. Requirements for IP version 4 routers. *RFC 1812, Proposed Standard*, June 1995.
- C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. A control theoretic analysis of RED. In *IEEE INFOCOM*, pages 1510–1519, April 2001. URL <http://www.ieee-infocom.org/2001/paper/791.pdf>.
- C. V. Hollot, Vishal Misra, Donald F. Towsley, and Weibo Gong. Analysis and design of controllers for AQM routers supporting TCP flows. *IEEE Transactions on Automatic Control*, 47(6):945–959, June 2002.
- Yongguang Zhang and Tom Henderson. An implementation and experimental study of the eXplicit Control Protocol (XCP). to appear, 2004. URL <http://www.isi.edu/isi-xcp/share/extended.pdf>.
- Aaron Falk and Dina Katabi. Specification for the explicit control protocol (XCP). *draft-falk-xcp-spec-00.txt, work in progress*, October 2004. URL <http://www.ietf.org/internet-drafts/draft-falk-xcp-spec-00.txt>.
- Steven H. Low, Lachlan L. H. Andrew, and Bartek P. Wydrowski. Understanding XCP: equilibrium and fairness. *IEEE Infocom*, March 2005. URL <http://netlab.caltech.edu/pub/papers/XCP-infocom05.pdf>.
- Cheng Jin, David Xiaoliang Wei, and Steven H. Low. FAST TCP: motivation, architecture, algorithms, performance. *IEEE Infocom*, March 2004. URL <http://netlab.caltech.edu/pub/papers/FAST-infocom2004.pdf>.
- Fernando Paganini, Zhikui Wang, John C. Doyle, and Steven H. Low. Congestion control for high performance, stability and fairness in general networks. *IEEE/ACM Transactions on Networking*, to appear 2004. URL <http://www.ee.ucla.edu/~paganini/PDF/Papers/fast-jour.pdf>.
- Amit Jain, Sally Floyd, and Pasi Sarolahti. Quick-start for TCP and IP. *draft-amit-quick-start-03.txt, work in progress*, September 2004.

- URL <http://www.ietf.org/internet-drafts/draft-amit-quick-start-03.txt>.
- [17] Sally Floyd. HighSpeed TCP for large congestion windows. *RFC 3649, Experimental*, December 2003.
- [18] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control for fast, long distance networks. *IEEE INFOCOM*, 2004. URL <http://www.csc.ncsu.edu/faculty/rhee/export/bitcp.pdf>.
- [19] Injong Rhee and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *PFLDnet*, 2005. URL <http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/cubic-paper.pdf>.
- [20] Tom Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *Submitted for publication*, December 2002. URL http://www-lce.eng.cam.ac.uk/~ctk21/papers/scalable_improve_hswan.pdf.
- [21] R. N. Shorten and D. J. Leith. H-TCP: TCP for high-speed and long-distance networks. *PFLDnet*, 2004. URL <http://www.hamilton.ie/net/htcp3.pdf>.
- [22] Aleksandar Kuzmanovic and Edward W. Knightly. TCP-LP: A distributed algorithm for low priority data transfer. *IEEE INFOCOM*, 2003. URL <http://www.ece.rice.edu/~akuzma/Doc/akuzma/TCP-LP.pdf>.
- [23] Sumitha Bhandarkar, Saurabh Jain, and A. L. Narasimha Reddy. LTCP: A layering technique for improving the performance of TCP in highspeed networks. 2003. URL <http://ee.tamu.edu/~reddy/papers/jogc2003.pdf>.
- [24] Sumitha Bhandarkar, Saurabh Jain, and A. L. Narasimha Reddy. Improving TCP performance in high bandwidth high RTT links using layered congestion control. *PFLDnet*, 2005. URL <http://students.cs.tamu.edu/sumitha/research/pfldnet2005.pdf>.
- [25] Curtis Villamizar and Cheng Song. High performance TCP in ANSNET. *ACM SIGCOMM Computer Communication Review*, 24(5):45–60, October 1994. ISSN 0146-4833. URL <http://portal.acm.org/citation.cfm?id=205511.205520>.
- [26] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. *ACM SIGCOMM*, September 2004. URL <http://yuba.stanford.edu/~appenz/pubs/sigcomm-extended.pdf>.
- [27] Sanjay Hegde, David Lapsley, Bartek Wydrowski, Jan Lindheim, David Wei, Cheng Jin, Steven Low, and Harvey Newman. FAST TCP in high-speed networks: An experimental study. *GridNets*, October 2004. URL <http://netlab.caltech.edu/pub/papers/gridnets04.pdf>.
- [28] Stanislav Shalunov. Testing FAST TCP over Abilene. *2nd PFLDnet Workshop*, February 2004. URL <http://dsd.lbl.gov/DIDC/PFLDnet2004/papers/Shalunov.pdf>.
- [29] Yunhong Gu and Robert L. Grossman. SABUL: A transport protocol for Grid computing. *Journal of Grid Computing*, 1(4):377–386, 2003.
- [30] Ravi S. Prasad, Manish Jain, and Constantinos Dovrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of Grid Computing*, 1(4):361–376, 2004. URL http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/Papers/sobas_grid.p%df.
- [31] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sansadidi, and Ren Wang. TCP Westwood: End-to-end congestion control for wired/wireless networks. *Wireless Networks Journal*, 8:467–479, 2002. URL http://www.cs.ucla.edu/NRL/hpi/tcpw/tcpw_papers/tcpw-wirelessnet02.pdf%.
- [32] Ravi S. Prasad, Manish Jain, and Constantinos Dovrolis. Effects of interrupt coalescence on network measurements. *Passive and Active Measurements (PAM) conference*, April 2004. URL http://www.cc.gatech.edu/fac/Constantinos.Dovrolis/Papers/intcoal_pam0%4.pdf.
- [33] Robert L. Grossman, Yunhong Gu, Dave Hanley, Xinwei Hong, Dave Lillethun, Jorge Levera, Joe Mambretti, Marco Mazzucco, and Jeremy Weinberger. Experimental studies using photonic data services at IGrid 2002. *FGCS*, 2003. URL <http://www.ncdm.uic.edu/papers/journal-20.pdf>.
- [34] Jeffrey C. Mogul and Steven E. Deering. Path MTU discovery. *RFC 1191, Draft Standard*, November 1990. URL <ftp://ftp.rfc-editor.org/in-notes/rfc1191.txt>.
- [35] Matt Mathis, John Heffner, and Kevin Lahey. Path MTU discovery. *draft-ietf-pmtud-method-03.txt, work in progress*, October 2004.
- [36] Matt Mathis, Jamshid Mahdavi, Sally Floyd, and Allyn Romanow. TCP selective acknowledgment options. *RFC 2018, Standards Track*, October

1996. URL <ftp://ftp.rfc-editor.org/in-notes/rfc2018.txt>.
- [37] Sally Floyd, Jamshid Mahdavi, Matt Mathis, and Matthew Podolsky. An extension to the selective acknowledgement (SACK) option for TCP. *RFC 2883, Standards Track*, July 2000. URL <ftp://ftp.rfc-editor.org/in-notes/rfc2883.txt>.
- [38] Ethan Blanton, Mark Allman, Kevin Fall, and Lili Wang. A conservative selective acknowledgment (SACK)-based loss recovery algorithm for TCP. *RFC 3517, Standards Track*, April 2003. URL <ftp://ftp.rfc-editor.org/in-notes/rfc.txt>.
- [39] Sally Floyd. Congestion control principles. *RFC 2914, Best Current Practice*, September 2000.
- [40] Stanislav Shalunov, Benjamin Teitelbaum, and Anatoly Karp. Internet2 NetFlow weekly reports. *netflow.internet2.edu/weekly/*, 2002–2004. URL <http://netflow.internet2.edu/weekly/>.
- [41] Stanislav Shalunov. Netkill—generic remote DoS attack. *www.internet2.edu/~shalunov/netkill/*, 2000.
- [42] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. TCP congestion control with a misbehaving receiver. *Computer Communication Review*, 29(5):71–78, 1999. URL <http://citeseer.ist.psu.edu/savage99tcp.html>.
- [43] Mark Allman. TCP congestion control with appropriate byte counting (ABC). *RFC 3465, Experimental*, February 2003.