TCP Congestion Control in Fast Long-Distance Networks

Jean-Philippe Martin-Flatin
IT Division
CERN
1211 Geneva 23, Switzerland
E-mail: jp.martin-flatin@ieee.org

Sylvain Ravot
California Institute of Technology
c/o CERN
1211 Geneva 23, Switzerland
E-mail: sylvain@hep.caltech.edu

Technical Report DataTAG-2002-1, FP5/IST DataTAG Project
and Technical Report CALT-68-2398, California Institute of Technology, USA
July 10, 2002

***Abstract* -- TCP congestion control is currently based on Jacobson's algorithms (slow-start and congestion avoidance), devised back in 1988, with improvements such as SACK and NewReno. Although TCP has proved remarkably flexible thus far and managed to adapt to vastly different types of networks, we show in this paper that some of the assumptions behind these algorithms are no longer valid in today's fast long-distance networks. Our analysis is backed by empirical data collected over a 622 Mbit/s transoceanic link dedicated to Grid applications. We propose that the additive increase and multiplicative decrease algorithms be changed for this type of networks, and describe a new fairness principle.**

I. Introduction

Data-intensive Grids (called *Grids* for short in this paper) are distributed applications that span multiple organizations and management domains and operate over fast Wide-Area Networks (WANs). They are characterized by massive file transfers (up to several terabytes) between geographically dispersed machines spread over different countries, sometimes even different continents. They assume that the network is transparent to the application, just like standard distributed systems do over Local-Area Networks (LANs). Grids enable organizations to share computing resources such as PC farms on an *ad hoc* basis.

Grids are particularly relevant to Particle Physics, because the amount of data produced by a single experiment is so huge that all the computing power of a single Research Institute or National Center may not suffice to process it. In such cases, the data processing must be distributed worldwide among several organizations. CERN is interested in Grids for processing the data that will be generated by its future Large Hadron Collider.

Caltech and CERN are involved in a mesh of international research programs that endeavor to put together such Grids for the Particle Physics research community. One critical aspect under study is the tuning of network protocols and the development of new protocol implementations in order to maximize the bandwidth offered by fast long-distance networks to Grids. To this end, a fast transoceanic link between Geneva and Chicago was provisioned and tested jointly by the two partners (see Section II.A).

Because Grid tools and middleware are usually built on top of the Transmission Control Protocol (TCP [16]), we focused on the bandwidth offered by fast WANs to long-lived TCP connections, and did not investigate alternate transport protocols such as the User Datagram Protocol (UDP [16]) or the Stream Control Transmission Protocol (SCTP [17]).

While experimenting with massive file transfers, we soon realized that the performance we were getting out of our testbed was rather poor. After tracing back these problems for some time and testing several end-hosts with different Gigabit Ethernet cards, we convinced ourselves that the problems lay in TCP, notably its congestion control algorithms. In this paper, we describe these problems, analyze them, and explain some of their causes. We propose some solutions and raise questions that still need to be answered.

The remainder of this paper is organized as follows. In Section II, we describe our testbed and show evidence of the performance problems experienced on our WAN link. In Section III, we analyze these problems and justify them theoretically. We define some metrics that highlight some issues inherent in today's TCP congestion control algorithms. In Section IV, we present a new fairness principle that leads us to more efficient algorithms for increasing and decreasing the congestion window. Its strengths and weaknesses are assessed. Finally, we investigate related work and present perspectives for future work.

II. Experimental Evidence of Performance Problems

Fast (622 Mbit/s and above) long-distance networks are considerably more sensitive to packet loss than today's standard networks. In this section, we demonstrate this with measurements performed in real life.

A. *Description of the testbed*

CERN and Caltech worked jointly to provision and test a 622 Mbit/s link between CERN, Geneva, Switzerland and StarLight, an optical switching facility located in Chicago, IL, USA. StarLight interconnects a number of research networks in the USA, including Abilene (Internet2). CERN has fast connections to GEANT, the backbone used by most National Research and Education Networks in Europe, and several other research institutes.

The layout of our testbed is depicted in Figure 1. At both ends of our 622 Mbit/s WAN link, we have Cisco 7609 routers with Packet over SONET (PoS) interfaces. (The equipment owned by our WAN provider is not represented here.) The end-hosts are PCs equipped with Gigabit Ethernet cards. They are both connected directly to the routers in order to limit the potentially adverse effects (buffering delay, data corruption, etc.) of extra equipment.

The end-host at CERN is a PC with a Pentium IV processor clocked at 1.5 GHz. It runs Linux kernel 2.4.17, has a 32-bit bus clocked at 33 MHz, and a SysKonnect SK-9843 SX Gigabit Ethernet card. The end-host at StarLight is a dual-processor PC; both processors are Pentium III's clocked at 1 GHz. The rest is identical to CERN's end-host. Note that release 2.4.17 of the Linux kernel implements both TCP NewReno [4] and TCP SACK [10].

Although the theoretical limit of a 32-bit bus clocked at 33 MHz is 1056 Mbit/s, which is slightly above 1Gbit/s (the maximum throughput that Gigabit Ethernet cards can cope with), we discovered that, in practice, it "saturates" at about 400 Mbit/s (two PCs connected back to back). At the time of this writing, we are in the process of upgrading our PCs to 64-bit buses clocked at 133 MHz.

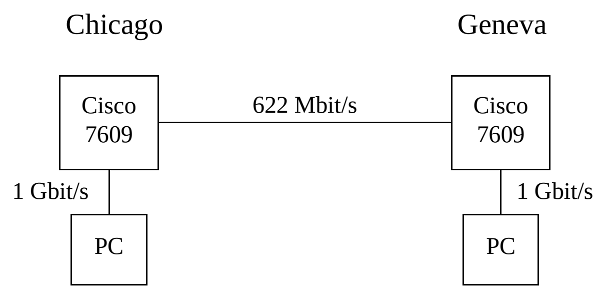

Figure 1: Setup of the testbed

B. *Requirements*

To date, it is difficult to characterize Grid-related traffic patterns over fast long-distance networks, because the research community still has little experience with this new technology. The possibility for researchers to use long-distance networks offering more than 100 Mbit/s to a single TCP connection only recently became a reality (e.g., see the Net100 project [11]).

Based on the discussions we had with Grid experts, we expect to have three types of concurrent traffic on our WAN link when it goes into production. First, we should have a few massive file transfers (between one and 10) at any point in time. These resource-hungry transfers are based on tools such as GridFTP [5] and are likely to consume all the bandwidth that is offered to them, even if we later increase the link capacity. They rely on very-long lived TCP connections, which typically last several hours. The amount of data moved about by a single TCP connection (be it single- or multi-stream) is counted in terabytes. Second, we expect to have between dozens and hundreds of long-lived TCP connections that transfer gigabytes of data, usually last 5–10 minutes, and always last less than an hour. The third type of traffic corresponds to the standard use of a WAN link without Grids: Web, e-mail, interactive connections via `ssh`, videoconferences, small file transfers, etc. This background traffic is expected to account for at most 5% of the network capacity and can reasonably be considered negligible, except that it can temporarily cause congestion. This traffic includes many short-lived TCP connections and non-TCP traffic.

In this paper, we focus on the first type of traffic. We try to maximize the bandwidth offered by our fast long-distance network to a few very long-lived TCP connections, bearing in mind that the other two classes of traffic occasionally cause congestion and packet loss. Due to space constraints, our study of the effects of multi-streaming on TCP performance is not included here (see Hacker and Athey [8] for a good study on this issue).

C. *Results*

1) Time to recover from a single loss

The data plotted in Figure 2 was gathered with a modified version of `gensink` [15] generating TCP traffic. This home-grown tool offers functionality somewhat similar to the better-known `iperf` [12]. One point is plotted each time 10 MB of data have been transferred, as opposed to every N milliseconds, hence the slight smoothing of the curve.

Each drop of the throughput on Figure 2 is due to packet loss. Sometimes we have a single loss, sometimes several successive packets are lost. As expected, the throughput is divided by two whenever a packet is lost, due to Jacobson's multiplicative decrease algorithm [6].

What was less expected is that losses do not occur at the same value of the throughput; as a result, cycles all have the same slope but they do not have the same period. Although our PCs can cope with throughputs of up to 400 Mbit/s, our measurements show that packets are lost at lower throughput levels, between 140 and 280 Mbit/s. These lower values and their variation turned out to be due to the poor quality of the link offered by our WAN provider. We demonstrated this by generating UDP and TCP traffic at different throughputs and checking the error statistics reported by our SmartBits 2000 performance analysis test system and the POS interfaces of our Cisco routers. At 159 Mbit/s, the worse error rate that we measured was 2.3 10^{-4}, which is several orders of magnitude poorer than expected (see Section III.B). The problem may lie in a network device or an optical fiber of our WAN provider.

Independent of these error rates, Figure 2 shows evidence of the basic problem with TCP over fast long-distance networks: It takes too long (several minutes in our testbed) to recover from packet loss. Intuitively, this value should be in the order of milliseconds, or a few seconds at most. We will come back to this in Section III.A.

2) Estimation of β

Let us define β as the payload divided by the number of bits in the fiber. This ratio allows to go from a raw bit rate (bits in the fiber) to an application-level throughput (useful bits that make up the payload of TCP segments). It allows end-users to estimate the time it takes to transfer a certain amount of data over a WAN clocked at a given speed. β takes into account the overhead caused by application- and transport-layer retransmissions, the TCP or UDP headers, TCP ACKs, the IP headers, the Ethernet headers and footers, the Ethernet preambles, link-layer CRCs, SONET/SDH overhead (4.44%), etc.

β varies with the size of the payload, i.e. with the packet size. It also depends on the transport protocol, the error rate, and the proportion of TCP and UDP traffic. In the case of Grids, most of the traffic is due to massive file transfers over TCP. If we make sure that applications send data to the kernel in an efficient way, or that data is buffered by the kernel, we can assume that packets are completely filled. In other words, for Grids, we can reasonably assume that the average payload of a TCP packet (the average segment size) is approximately equal to the Maximum Segment Size (MSS): 1,460 bytes. (The standard MTU for Gigabit Ethernet networks is 1,500 bytes.)

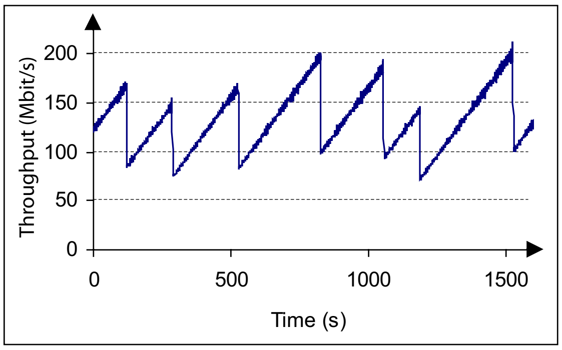

Figure 2: Long recovery time of TCP

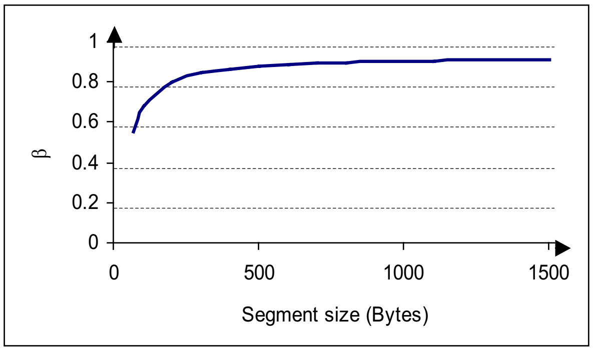

Figure 3: Estimation of β

Figure 3 shows our measurements of β for different packet sizes. Because we had no equipment to precisely measure the bit rate on our WAN link, we saturated our 622 Mbit/s WAN link with UDP traffic. To do so, we used a SmartBits box with two Gigabit Ethernet (GbE) interfaces. The traffic reported by the Cisco 7609 on the POS interface was 610 Mbit/s (with a 5% error window due to the way measurements were performed), which confirms that the link was indeed very close to saturation.

The traffic measured by the SmartBits box on the receiving GbE interface was 93.8% of the traffic generated by the sending GbE interface. This value indicates the payload of IP packets. For TCP traffic, we have a header of 20 bytes, so β_{TCP} is equal to:

$$\beta_{TCP} = 0.938 \cdot \frac{1480 - 20}{1480} = 0.925 \qquad (1)$$

provided that we have enough connections to properly use the available bandwidth (see Section III.C). For UDP traffic, we have a header of only 8 bytes, which yields:

$$\beta_{UDP} = 0.938 \cdot \frac{1480 - 8}{1480} = 0.933 \qquad (2)$$

For both β_{TCP} and β_{UDP} we have to also take into account retransmissions due to buffer overflows and line errors, application- or transport-level acknowledgments, etc. These parameters depend on the network state and vary dynamically. In this paper, we approximate β with a value of 0.9 for Grids.

III. Identification of what is wrong with TCP over fast long-distance networks

Faced with the problems described so far, we first tried to understand whether these problems were general problems or artifacts due to our experiments. During this exercise, we identified two causes of the high errors rates we were getting (PCs with 32-bit buses and poor quality of the WAN link).

Once we had convinced ourselves that the poor performance of our massive file transfers were due to TCP, we studied in detail how TCP SACK and TCP NewReno (simply referred to as "TCP" hereafter) work. Our goal was to work out whether this problem could be ascribed to the way TCP operates. Our results were enlightening. Rather simple calculations show that the characteristics of fast long-distance networks are so different from today's and past "regular" networks that they make some of TCP's fundamental assumptions incompatible with high throughput.

In this section, we give three reasons for TCP to perform poorly over fast long-distance networks.

A. Existing congestion control mechanisms are not responsive enough

TCP is considerably more sensitive to packet loss in fast WANs than in LANs or regular WANs. This is primarily due to its congestion avoidance algorithm, based on the Additive Increase Multiplicative Decrease (AIMD) principle [6]. The effect of a single loss is disastrous in fast long-distance networks: A TCP connection reduces its bandwidth use by half immediately after a loss is detected (multiplicative decrease), but instead of taking hundreds of milliseconds or at most a few seconds to use all the available bandwidth again, it takes minutes or hours (additive increase). Slow-start also has an effect on the poor performance of TCP over fast WANs, but its impact is lower than congestion avoidance: Packet loss is detected more often by triplicate ACKs than by timeouts, and TCP connections spend much more time in congestion avoidance than in slow-start.

To quantify this phenomenon, let us define new metrics that clearly show that there is a problem when we use fast long-distance networks.

1) Responsiveness

Let us formalize the concept of responsiveness, which is rather intuitive to end-users. How quickly do things go back to normal after a TCP connection experiences packet loss? We saw on Figure 2 that it takes a long time in real life, but let us analyze it theoretically as well.

The following two equations give rather intuitive definitions of responsiveness:

$$\rho = N_{inc} \cdot RTT \tag{3}$$

$$N_{inc} = \frac{C \cdot RTT}{2 \cdot inc} \tag{4}$$

The increment *inc* measures the additive increase of the TCP congestion avoidance algorithm. As per RFC 2581 [1], it is equal to one MSS (Maximum Segment Size), where MSS is equal to 1,460 bytes in general today (Ethernet MTU), 536 bytes in the old days (RFC 1122, p. 60), and 8,960 bytes if we use Gigabit Ethernet jumbo frames (which have not been standardized yet). *C* is the capacity of the network link. More precisely, it is the estimated link capacity at the network bottleneck. In Section IV.D, we will study how this capacity can be estimated. The responsiveness ρ measures how quickly we go back to using the network link at its full capacity after experiencing a packet loss (and thus halving our use of the network). Finally, N_{inc} is the number of increments, i.e., the number of times *cwnd* has to be increased before we use the link at full capacity again.

Note that we do not assume in (3) and (4) that the window size always remains equal to one. What we measure here is the time is takes to go back to normal, that is, to use the same bandwidth as what we were using prior to the packet loss. For a single TCP stream, this means using the full capacity of the network link.

If we combine (3) and (4), we get:

$$\rho = \frac{C \cdot RTT^2}{2 \cdot inc} \tag{5}$$

Equation (5) assumes that we use consistent units for all variables. In practice, however, *C* is expressed in raw bits per second on the fiber, while *inc* is expressed in useful bytes transferred in the payload of TCP packets. In Section II.C, we defined β, the ratio that allows us to go from a raw bit rate to an application-level throughput. If we call C_b the capacity expressed in bit/s and inc_B the increment expressed in bytes, (5) becomes:

$$\rho = \frac{\beta \cdot C_b \cdot RTT^2}{2 \cdot inc_B \cdot 8}$$

which yields:

$$\rho = \frac{\beta \cdot C_b \cdot RTT^2}{16 \cdot inc_B} \tag{6}$$

Similarly, the number of increments becomes:

$$N_{inc} = \frac{\beta \cdot C_b \cdot RTT}{16 \cdot inc_B} \tag{7}$$

Let us quantify this responsiveness for different types of networks. In the cases analyzed next, the first two correspond to typical LANs and WANs in 1988, when

Jacobson devised his congestion control algorithms for TCP [6]. The next two correspond to typical LANs and WANs today. Cases 5 to 8 correspond to the different types of WAN links that we envision between Geneva and Chicago. The last two cases correspond to different but equally interesting scenarios where RTT is very large and C is very small: Sun-Earth and Mars-Earth communications.

Since the number of increments must be an integer, the values of N_{inc} are rounded up to the nearest greater integer. For the sake of consistency, we assume that β is equal to 0.9 in all cases (see Section II.C). In practice, its value is smaller for spatial communications.

Case 1: typical LAN in 1988 (Ethernet)

C = 10 Mbit/s
RTT = [2 ms; 20 ms]
(20 ms is really the worst case; 2 ms is more typical)
increment = 1,460 bytes
then
number of increments = [1; 8]
responsiveness =~ [1.5 ms; 150 ms]
Responsiveness is good.

Case 2: typical WAN in 1988

C = 9.6 kbit/s
RTT = 40 ms (worst case)
increment = 1,460 bytes
then
number of increments = 1
responsiveness = 0.6 ms
Responsiveness is good.

Case 3: typical LAN today

C = 100 Mbit/s
RTT = 5 ms (worst case)
increment = 1,460 bytes
then
number of increments = 20
responsiveness = 96 ms
Responsiveness is good.

Case 4: typical WAN today

C = 2 Mbit/s
RTT = 40 ms (worst case)
increment = 1,460 bytes
then
number of increments = 4
responsiveness =~ 120 ms
Responsiveness is acceptable.

Case 5: old WAN link between CERN and StarTAP

C = 155 Mbit/s
RTT = 120 ms
increment = 1,460 bytes
then
number of increments =~ 720
responsiveness = 86 s
Responsiveness is poor.

Case 6: current WAN link between CERN and StarLight

C = 622 Mbit/s
RTT = 120 ms
increment = 1,460 bytes
then
number of increments =~ 2,900
responsiveness =~ 6 min

Responsiveness is poor. Note that the responsiveness measured in our testbed is twice as large, due to a side effect of delayed ACKs. Release 2.4.17 of the Linux kernel implements delayed ACKs, that is, the destination sends an ACK every second packet when it receives two packets within 500 ms (as recommended by RFC 2581, Section 4.2). When (i) the congestion window is large, (ii) the mean inter-packet arrival time is largely below 500 ms, and (iii) many packets are sent in bursts, then all packets are acknowledged with delayed ACKs (except the last packet of the congestion window if the total number of packets in this window is odd). As a result, on average, the number of ACKs sent by the destination is only half the number of packets sent by the source. Meanwhile, every time the kernel of the source receives an ACK, it increases *cwnd* as follows (as per RFC 2581, Section 3.1):

$$cwnd_{i+1} = cwnd_i + \frac{SMSS \cdot SMSS}{cwnd_i} \qquad (8)$$

where SMSS is the MSS of the sender. Unfortunately, this formula assumes that the source receives one ACK per packet sent; in our case, we receive one ACK every second packet sent. Because the source receives only half of the ACKs expected by this formula, its congestion window increases at only half the rate it is supposed to. This explains why the experimental responsiveness derived from Figure 2 is 12 minutes while the theoretical responsiveness is only six minutes.

To avoid this side effect, we recommend that (8) (which is called equation (2) in RFC 2581, Section 3.1) be replaced with the following:

$$cwnd_{i+1} = cwnd_i + \frac{SMSS \cdot NAB}{cwnd_i} \qquad (9)$$

where *NAB* is the number of acknowledged bytes in the ACK we just received. Without delayed ACKs, NAB is always equal to SMSS and (9) is equivalent to (8). If we experience delayed ACKs from time to time, NAB oscillates between SMSS and 2*SMSS. If we always have delayed ACKs, NAB is always equal to 2*SMSS.

Case 7: future WAN link between CERN and StarLight (2.5 Gbit/s scenario)

C = 2.5 Gbit/s
RTT = 120 ms
increment = [8,960 bytes; 1,460 bytes]
(the left value is for Gigabit Ethernet jumbo frames, the right value is for standard frames)
then
number of increments =~ [1,900; 11,600]
responsiveness =~ [4 min; 23 min]

Responsiveness is poor with an MSS of 8,960 bytes, and very poor with an MSS of 1,460 bytes.

Case 8: future WAN link between CERN and StarLight (10 Gbit/s scenario)

C = 10 Gbit/s
RTT = 120 ms
increment = [8,960 bytes; 1,460 bytes]
then
number of increments =~ [7,500; 46,200]
responsiveness =~ [15 min; 1 h 30 min]

Responsiveness is very poor with an MSS of 8,960 bytes, and dreadful with an MSS of 1,460 bytes.

Case 9: Sun-Earth satellite communications

C = 10 bit/s
RTT = 1000 s
(The average Sun-Earth distance is 15 10^{10} m.)
increment = 1500 bytes
then
number of increments = 1
responsiveness =~ 6 min

Responsiveness is poor, but not as bad as for fast long-distance networks.

Case 10: Mars-Earth interplanetary communications

C = 10 bit/s
RTT = [20 min; 50 min]
(The Mars-Earth distance varies more than the Earth-Sun distance.)
increment = 1500 bytes
then
number of increments = [1; 2]
responsiveness =~ [9 min; 1 h]

Responsiveness is as bad as for fast long-distance networks.

In short, the responsiveness defined in (6) allows us to immediately show and quantify that there is a problem specific to fast long-distance networks. Neither the LANs and WANs of 1988, nor those that are typical today, exhibit the poor responsiveness that affects fast long-distance networks. Even Earth-Sun satellite communications are not as bad (they have other problems, however, due to high loss rates).

Jacobson's congestion avoidance algorithm, and all its variants implemented in TCP Tahoe, Reno, New Reno, SACK, and Vegas, make the implicit assumption that the responsiveness should remain low, below a few seconds. Thus, the number of increments is also assumed to remain low. This implicit assumption, which has been valid for all the networks used so far, is no longer valid for today's fast long-distance networks, that is, for the networks envisioned for Grids in the near future. Jacobson's algorithm needs to be changed for the type of networks of interest to us.

2) Number of packets in transit

If we blast data at full rate into the pipe (i.e., if *cwnd* is large enough that we can we use the network link at full capacity), the maximum number of bytes in transit is given by:

$$NBT_{max} = \frac{\beta \cdot C \cdot RTT}{8} \qquad (10)$$

where *C* is the network link capacity and β is the ratio determined in Section II.C.

NBT_{max} is the maximum amount of data already sent into the pipe, but for which the sender has not yet received an acknowledgment from the receiver. It corresponds to the flight size defined in RFC 2581.

NBT_{max} is often referred to as the *bandwidth delay product*, assuming that β is equal to 1. In our view, the term *bandwidth* is a bit of a misnomer here, because it usually refers to the proportion of the capacity that is actually used, and is thus dynamic. The capacity of a link, conversely, is the upper bound of the bandwidth and is static. The two quantities are equal if, and only if, the network link is traversed by a single stream of data that takes up all the network resources—a very rare event indeed.

Because routers and switches have buffer limits expressed in packets rather than bytes, we define a more practical metric, the maximum number of packets in transit:

$$NPT_{max} = \frac{\beta \cdot C \cdot RTT}{8 \cdot ASS} \qquad (11)$$

where ASS is the average segment size for TCP traffic. As mentioned already, in the case of Grids, packets are assumed to be fully filled TCP segments, so ASS = MSS = 1,460 bytes.

Case 1: typical LAN in 1988 (Ethernet)

C = 10 Mbit/s
RTT = 2 ms
then
NBT_{max} = 2250
NPT_{max} = 2

At most, we have two packets in transit in such a network. Buffers are unlikely to fill up frequently on the path between the source and the destination. The maximum size of *cwnd* is very small.

Case 2: typical WAN in 1988

C = 9.6 kbit/s
RTT = 40 ms (worst-case scenario)
then
NBT_{max} = 44
NPT_{max} = 1

Such networks are so slow that packet N has already been acknowledged by the destination and the ACK has already reached the source by the time the source is ready to send packet (N+1). Once again, the maximum value of *cwnd* remains small.

Case 3: current WAN link between CERN and StarLight

C = 622 Mbit/s
RTT = 120 ms
then
NBT_{max} =~ 8,400,000
NPT_{max} =~ 5,750

We have already sent 5,750 packets when we receive a duplicate ACK, and 11,500 packets when we receive at last a triplicate ACK indicating that a packet was lost. As today's routers and level-3 switches rarely have queues larger than 200 packets, and as the number of hops between any two end-hosts rarely exceeds 40 hops in the Internet today, the network devices cannot buffer more than 8,000 packets. With 11,500 packets in transit, we can thus fill up the queues of all level-3 equipment (routers and switches) along the path. When the destination asks the source to reduce its *cwnd*, or when it reports that a packet was lost, it is already too late for the source to adjust to the new situation. Responsiveness is dreadful!

Case 4: future WAN link between CERN and StarLight (2.5 Gbit/s scenario)

C = 2.5 Gbit/s
RTT = 120 ms
then
NBT_{max} =~ 34,000,000
NPT_{max} =~ 23,000

This time, we have already sent 23,000 packets when we receive a duplicate ACK, and 46,000 packets when we receive at last a triplicate ACK indicating that a packet was lost. The situation is even worse than in case 3.

B. *At fast bit rates, line errors are no longer negligible*

The second reason for TCP to perform poorly over fast long-distance networks is that packet loss is always interpreted as congestion by TCP. The implicit assumption here is that packet loss due to line errors is negligible compared to buffer overflows. Yet again, this assumption falls apart in the case of fast long-distance networks—and even when RTT is small.

To date, standard bit error rates for optical fibers range from 10^{-12} to 10^{-14}. If we take into account the errors due to network equipment (which does not include buffer overflows), these values become 10^{-9} to 10^{-11}. These rates may seem extremely low at first sight. Still, with fast networks, these bit error rates correspond to:

10^{-9}. 622 Mbit/s = 0.6 errors/s
10^{-11}. 622 Mbit/s = 0.006 errors/s

10^{-9}. 2.5 Gbit/s = 2.5 errors/s
10^{-11}. 2.5 Gbit/s = 0.025 errors/s = 1.5 errors/min

10^{-9} errors/bit . 10 Gbit/s = 10 errors/s
10^{-11} errors/bit . 10 Gbit/s = 0.1 errors/s = 6 errors/min

With SONET and SDH, error correcting codes are able to mask some of these errors (e.g., by using Forward Error Correction). Supposing that only 10% of line errors are not corrected, which is very optimistic, we still have:

10^{-9}. 622 Mbit/s . 10% =~ 1 error every 17 s
10^{-11}. 622 Mbit/s . 10% =~ 1 error every 28 min

10^{-9}. 2.5 Gbit/s . 10% =~ 1 error every 4 s
10^{-11}. 2.5 Gbit/s . 10% =~ 1 error every 7 min

10^{-9} errors/bit . 10 Gbit/s . 10% = 1 error/s
10^{-11} errors/bit . 10 Gbit/s . 10% =~ 1 error every 2 min

Compared to the time it takes congestion avoidance to make the throughput go back to its value when congestion was detected (responsiveness of six minutes for a 622 Mbit/s link, 23 minutes for a 2.5 Gbit/s link, and 1 hour 32 minutes for a 10 Gbit/s link), these error rates are so high that we *never* go back to this value, except when we use a 622 Mbit/s with a 10^{-11} bit error rate. At high bit rates, line errors are no longer negligible.

The first consequence of this is that the loss of a single, isolated TCP packet should *not* be interpreted as congestion by TCP. Congestion requires the loss of at least two successive packets. An acceptable approximation of this is the following rule:

TCP congestion requires the loss of at least two packets in the same congestion window. The loss of a single packet should be interpreted as a line error and should not trigger the multiplicative decrease of the used bandwidth.

Note that very lossy communications (e.g., over wireless or interplanetary networks) may require even more drastic definitions [14]. This one assumes that the bit error rate remains reasonably small.

Another consequence is that TCP should behave differently with networks operating at different speeds. The current versions of TCP (New Reno and SACK) are well suited to today's standard WANs (up to 155 Mbit/s), but are not to tomorrow's fast WANs (2.5 Gbit/s and higher).

C. *With TCP, we cannot use the link at full capacity*

The people who try to break the world record of mega file transfers over very long distances all come across the same problem: A single TCP stream cannot use the full capacity of a network link because of the time it takes TCP to recover from a single loss. This limitation is inherent in the design of all flavors of TCP to date: Tahoe, Reno, NewReno, SACK, Vegas, etc.

Let us suppose that we have a single TCP stream going through a network link. Whether we increase one MSS at a time or several MSSes at a time, the overall used bandwidth remains unchanged: 75% of link capacity. This is due to the AIMD algorithms. Fast long-distance links are expensive, and we should not lose 25% of the capacity just for the sake of using a standard transport protocol.

This problem can affect Grids. As mentioned earlier, we expect to have between one and 10 massive transfers at any time. Outside office hours, Grid-dedicated links may not be traversed by any other kinds of traffic (see the three types defined in Section II.B). When this is the case, the total bandwidth (TBW) used by N concurrent TCP connections is given by:

$$TBW = \left(1 - \frac{1}{2^{N+1}}\right) \cdot C \qquad (12)$$

With N=2, we use 87.5% of the capacity. With N=3, we use 93.75%. With N=5, we use 96.9%. So, if we have only a few concurrent mega file transfers, this effect is important. Otherwise, it is negligible. In particular, this issue becomes irrelevant in backbones, where we usually have between 10^3 and 10^6 concurrent TCP connections.

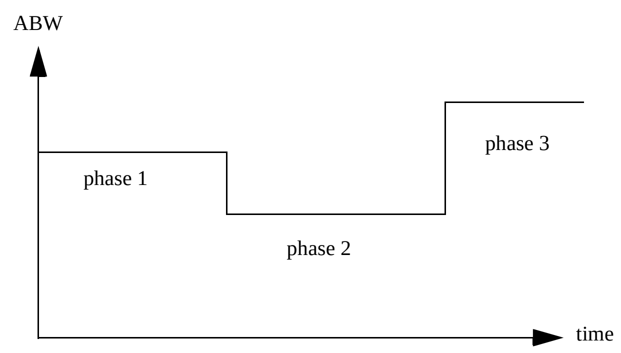

Figure 4: The available bandwidth is constant by slices

IV. Toward a new fairness principle for long-lived connections

In the previous section, we have demonstrated that losses cause serious problems to TCP when RTT is large and the link capacity is high. In this section, we propose a new fairness principle and new algorithms for increasing and decreasing the bandwidth used by TCP connections, thereby increasing the overall use of the network link capacity.

A. *New assumption*

Based on our experience with many kinds of networks (LANs or WANs, slow or fast, short or long distance), we believe that we can make the following assumption: On WAN links, the available bandwidth remains almost constant over extended periods of time, roughly between 10 minutes and one hour. There are exceptions to this empirical rule, but we have verified it in many different environments. It is particularly true in networks underlying Grids, where we expect to have between one and ten massive file transfers that take up most of the available bandwidth. In other networks, it is usually true during slack hours, and sometimes also during peak hours.

This hypothesis allows us to make the following approximation: For long-lived TCP connections, the available bandwidth (ABW) is constant by slices (see Figure 4). ABW_i denotes the available bandwidth during phase i. Building on this simplification, we can now change the algorithms used for increasing and decreasing the bandwidth used by a TCP connection. Instead of decreasing very fast and increasing very slowly (AIMD) with no idea of the level at which we will encounter congestion, we can try to reach the stationary level for phase i fairly quickly and to keep the oscillations of the used bandwidth in the vicinity of ABW_i. The challenges now are to work out how we detect a phase change and to find out the new ABW_i when we enter phase i.

B. New fairness principle

TCP connections are greedy by nature: They always try to use more bandwidth until a packet is lost. But when a loss does occur, TCP connections become very conservative: They drastically reduce their throughput and then start increasing it again very slowly. The problem with this behavior is that losses have a very adverse effect on the performance of data transfers and on the overall use of bandwidth in fast long-distance networks. How can we alleviate these problems for Grids?

Losses due to congestion are rarely one-off events: they usually come in bursts. When a new TCP connection begins sending data into an end-to-end virtual pipe (*socket*), most of the TCP connections generating cross-traffic through the bottleneck of this pipe will soon lose packets due to buffer overflows. As a result, most of these TCP connections will be aware that there is a newcomer. Based on this remark, we can improve the increase and decrease algorithms used by TCP during congestion avoidance.

Assume I am an existing TCP connection that detects congestion. Instead of dividing my throughput by two (that is, I give up all that is needed by the newcomer while the other TCP connections make no effort to reduce their share), I assume that (i) all existing TCP connections use the same bandwidth as me and (ii) everyone will make the same effort to reduce its bandwidth use. Both assumptions are fair, insofar as no single TCP connection is favored over others. So, instead of reducing my own use of the network by 50%, I should reduce it by ABW_i - ABW_{i+1} where ABW_{i+1} is defined by:

$$\frac{C}{ABW_i} = \frac{C}{ABW_{i+1}} - 1 \tag{13}$$

C is the estimated link capacity and ABW_i is the available bandwidth during phase i (see Figure 4). C remains constant “forever” (at the timescale of interest to us) while ABW_i remains constant over long periods of time (typically, for 10 minutes to one hour).

Since ABW_i is a fraction of the network link capacity, we have:

$$\forall i, \exists \alpha_i, (0 \le \alpha_i \le 1) \wedge (ABW_i = \alpha_i \cdot C) \tag{14}$$

Combining (13) and (14) yields the following when we decrease the bandwidth:

$$\alpha_{i+1} = \frac{\alpha_i}{1 + \alpha_i} \tag{15}$$

The difference between ABW_i and ABW_{i+1} is then:

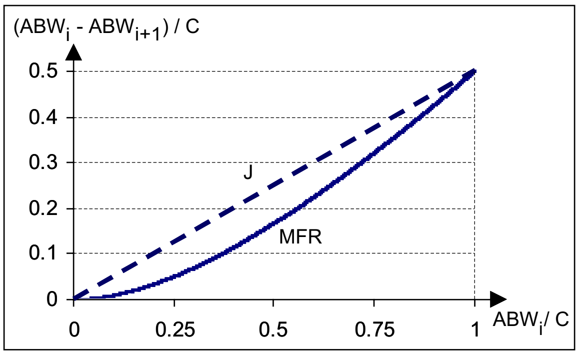

Figure 5: Two multiplicative decrease algorithms

$$ABW_i - ABW_{i+1} = \frac{\alpha_i^2 \cdot C}{1 + \alpha_i} \tag{16}$$

This defines our new decrease algorithm, referred to as *MFR* on Figure 5. Equation (16) improves on Jacobson’s decrease algorithm (called *J* on Figure 5), which is given by:

$$ABW_i^J - ABW_{i+1}^J = \frac{ABW_i^J}{2} = \frac{\alpha_i \cdot C}{2} \tag{17}$$

If we normalize the bandwidth, (16) and (17) become:

$$\frac{ABW_i - ABW_{i+1}}{C} = \frac{\alpha_i^2}{1 + \alpha_i} \tag{18}$$

$$\frac{ABW_i^J - ABW_{i+1}^J}{C} = \frac{\alpha_i}{2} \tag{19}$$

The two curves given by (18) and (19) are depicted in Figure 5. For instance, if $\alpha_i = 5\%$, then $\alpha_{i+1} = 4.76\%$ after a single loss with our decrease algorithm, instead of 2.5% with Jacobson’s. With a capacity of 622 Mbit/s, we save 14 Mbit/s. If $\alpha_i = 20\%$, then $\alpha_{i+1} = 16.7\%$ with our decrease algorithm instead of 10% with Jacobson’s. With a 622 Mbit/s link, we save 41 Mbit/s, which is significant! The only cases when our decrease algorithm behaves like Jacobson’s is when we have a single TCP stream or when we have infinitely many. If $\alpha_i = 100\%$, then $\alpha_{i+1} = 50\%$ after we experience a packet loss.

Figure 5 clearly shows that our decrease algorithm outperforms Jacobson’s for fast long-distance networks. We decrease the throughput less than Jacobson when a loss occurs, so it takes less time to go back to ABW_i when we increase again.

Note that α_i is not expected to be very small in the case of Grids. As mentioned earlier, we do not expect to have millions of concurrent, bandwidth-hungry Grid applications on a single link.

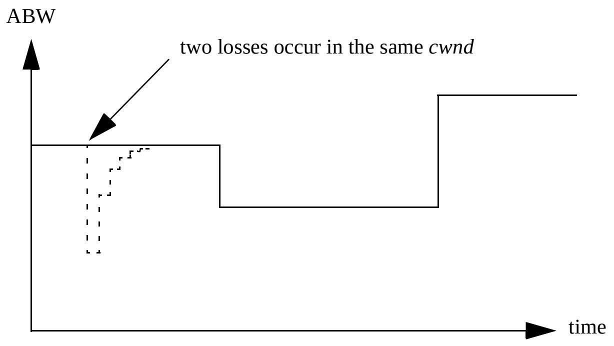

Figure 6: Binary search, same phase

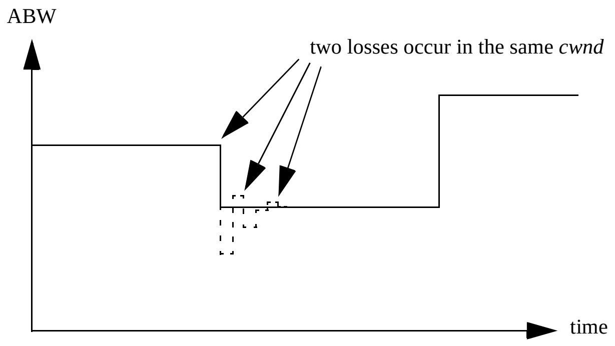

Figure 7: Binary search, new phase

C. New algorithm for increasing cwnd

Congestion is a way for a new TCP connection to signal its arrival to many (sometimes all) of the active TCP connections traversing the bottleneck of its path. Unfortunately, there is no signaling mechanism for a TCP connection to inform the others that it is about to be torn down. Thus we cannot devise a simple scheme such as:

$$\frac{C}{ABW_i} = \frac{C}{ABW_{i+1}} + 1 \qquad (20)$$

Instead, we propose to replace Jacobson's additive increase algorithm with a slightly more complex, yet more efficient algorithm. The bottom line is to use binary search until the increment size goes down to that of Jacobson's additive increase algorithm (i.e., one MSS). Afterward, we switch to the standard additive increase algorithm.

Let us present the details of our algorithm, which can be decomposed into five states.

State1: packet loss, same phase

The bottleneck of the link has just experienced congestion. We need to determine whether this phenomenon is transient or permanent. First, we suppose it is transient. We reduce the bandwidth used by the connection as defined in (18). Next, through binary search, we increase the bandwidth up to its previous stable level: ABW_i (see Figure 6.). If we successfully reach ABW_i without experiencing new packet loss, we are still in phase *i* and should move to state 3. Otherwise, if we experience at least two losses within the same congestion window, we should conclude that we have just gone from phase *i* to phase *i+1* and move on to state 2.

State2: packet loss, new phase

A new TCP connection traverses the network bottleneck, which causes congestion. We need to find a new equilibrium at a value ABW_{i+1} that is less than ABW_i. The new ABW_{i+1} is worked out via a *learning process* depicted in Figure 7. During this process, we use binary search both to increase and decrease the bandwidth use of the TCP connection. We decrease it whenever we detect a loss, and we increase it otherwise. This technique allows us to reach the new ABW_{i+1} very quickly.

At this stage, we have reached a new stationary mode and move on to state 3.

State3: no packet loss, no timeout

Once we enter this mode, we start a new timer called the *TCP greedy timer.* The value of this timer remains to be determined empirically, but we expect it to be comprised between 10 minutes and one hour. Our TCP connection keeps sending data at the rate ABW_i.

If we experience an isolated packet loss, we should not do anything. As explained in Section III.B, this loss may equally be due to congestion or line errors, and there is no reason to alter *cwnd* or to reduce the bandwidth use of the TCP connection in the presence of line errors.

If we detect N packet losses in the same congestion window (N remains to be determined empirically, but we expect to have $2 < N < 10$), we interpret these repeated losses as indicative of real congestion. We then move on to state 1.

If the TCP greedy timer goes off, we move on to state 4.

State4: no packet loss, timeout

We have been sending data at ABW_i for a long time, without experiencing any congestion. It is now time to investigate whether we could enter a new phase where ABW_{i+1} is larger than the current ABW_i. Once the TCP greedy timer goes off, we start increasing again the bandwidth used by our TCP connection. This time, we could adopt the increase algorithm defined in (20). Alternatively, we could use Jacobson's linear additive increase (one MSS per RTT), which is less aggressive. We will soon experiment with these different algorithms.

As in state 3, we do nothing special if we face an isolated loss. In case of real congestion, we move to state 1.

State5: bootstrap

When we create a new TCP connection, the seed value for ABW_0 should be the link capacity: *C*. We then move on to state 2.

D. Estimation of the link capacity

Estimating the link capacity of a WAN link—or, to be precise, the capacity of the bottleneck of a multi-hop WAN link—is renowned to be difficult.

One way is to use tools that temporarily flood the link (usually with UDP traffic) and derive the capacity by making a statistical analysis of the inter-packet arrival times at the sender (ACKs) or receiver (data). Examples of such tools include `udpmon` [9] and `pathload` [7].

Another way is to define a new ICMP message that all level-3 network devices (routers and switches) along the path must answer. Each ICMP reply reports the capacity of the inbound interface from which the ICMP message was received. By taking the lowest of these capacities, the end-host knows precisely the capacity of the bottleneck of the WAN link. In the presence of asymmetric routes, this discovery process must take place in both directions. The advantage of this method is that it is accurate. The main disadvantage is that it requires a new ICMP message to be supported by all routers and level-3 switches, which poses obvious deployment problems. Whether the market would adopt such a scheme remains to be seen. Another disadvantage is that it enables a new kind of Denial of Service (DoS) attack. A third disadvantage is that, if some of the level-3 devices do not support this new ICMP message, the lowest value among the advertised capacities may not be that of the bottleneck.

A third way of estimating the link capacity is to probe the network with a series of packets carrying real payload at the beginning of the lifetime of the TCP or UDP socket. As in the first solution, a statistical analysis of the inter-packet arrival times can indicate the value of the link capacity. The main advantage of this method is that it is performed while transferring useful data. The main disadvantage is that the precision of the estimation of the capacity is usually very low, which explains why people usually resort to the first solution.

In all cases, the value of the estimated capacity can be cached—e.g., it can be stored in a configuration file and read in next time we initiate a long-lived TCP connection to the same destination. This caching is particularly relevant to Grids, as the topology of the underlying network often remains fairly static over time.

E. Weaknesses of our new algorithms

The reliance of our new increase and decrease algorithms on the knowledge (or estimation) of the link capacity is both a strength and a weakness. It is a strength because it enables us to use less conservative algorithms than AIMD, and thus to better use the available bandwidth of expensive fast long-distance links.

It is also a weakness. First, the estimation of the capacity of a WAN link is always approximate, and the accuracy with which we estimate C is usually unknown. Second, estimating the capacity takes time. In theory, we could consider using our new increase and decrease algorithms for all sorts of TCP connections, not simply for long-lived ones. But for short-lived TCP connections, the time it takes to estimate the capacity may exceed the lifetime of the connection, which is not acceptable—unless we work out the capacity once and reuse this knowledge many times for different TCP connections. Third, routes change in real-life networks. Thus, the capacity calculated by the sender may not correspond to the capacity actually experimented by the TCP connection during its entire lifetime. If we cache the results of previous capacity estimations, it is not clear what the expiry time of the cache entries should be.

In short, there are several ways to estimate the capacity of a WAN link, but they all present some form of bias. This issue is still very much open to research. Before it is solved, the reliance on the knowledge of the link capacity is a weakness of our increase and decrease algorithms.

V. Related Work

Our work is not the first to question the versatilely and suitability of TCP for all kinds of networks. Several research teams have already described the performance problems experienced by satellite or interplanetary communications (e.g., the former IETF TCP over Satellite Working Group). In these cases, RTT is very large, but unlike the networks of interest to us, error rates are high and the network link capacity is small.

But fast long-distance networks have only recently become available to network researchers, and few papers specific to this type of networks have been published so far. Most of them are purely based on simulations, and few compare theoretical results with real measurements. Floyd recently issued an Internet Draft that gives a lot of insight for future research [3]. Low *et al.* proposed a new TCP congestion control mechanism with scalable stability [13]. Balakrishnan *et al.* investigated alternatives to TCP that do not react as drastically to a single packet loss [2].

VI. Conclusion

In fast long-distance networks, TCP is too sensitive to packet loss and takes too much time to recover from such losses. In addition, line errors occurring with a reasonable probability are mistakenly interpreted as congestion by TCP. In this paper, we have shown experimental evidence of these problems, analyzed them, and defined metrics that highlight these problems in such networks. We specified a new fairness principle and described new algorithms for increasing and decreasing the bandwidth used by long-lived TCP connections in Grids. We also justified why these algorithms should outperform Jacobson's. We are currently implementing them in Linux. Our modified version of TCP is known as *TCP Grid*.

In the future, we want to study the effects of buffering and shaping. It would also be useful to systematically investigate whether a single ubiquitous TCP for all kinds of networks is still possible for future networks, or whether it is a vision of the past. In our view, different networks with vastly different characteristics (e.g., local-area, fast long-distance, and interplanetary networks) should rely on different algorithms for congestion control. This standpoint poses a number of interesting challenges. For instance, how can we determine automatically the kind of networks that a given application needs to traverse? And how is fairness affected when we alter the TCP congestion avoidance algorithms?

Acknowledgments

The transatlantic link used for conducting this research is funded by the US Line Consortium (USLIC), which is financed by DoE via Caltech (grant DE-FG03-92-ER40701), NSF (grant ANI 9730202), CERN, Canadian HEP, IN2P3 in France, and WHO in Switzerland. J.P. Martin-Flatin is funded by the IST Program of the European Union (grant IST-2001-32459) via the DataTAG project. The authors would like to thank Harvey Newman and Olivier Martin for their feedback on this paper, and the network operations staff at StarLight and CERN for their help while the 622 Mbit/s link was provisioned and tested.

References

[1] M. Allman, V. Paxson, and W. Stevens (Eds.), "RFC 2581: TCP Congestion Control", IETF, April 1999.

[2] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms". In *Proc. SIGCOMM 2001, ACM Computer Communication Review*, Vol. 31, No. 4, August 2001.

[3] S. Floyd. "HighSpeed TCP for Large Congestion Windows". Internet Draft <draft-floyd-tcp-highspeed-00.txt>, work in progress, June 2002.

[4] S. Floyd and T. Henderson, "RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm", IETF, April 1999.

[5] Globus Project. "The GridFTP Protocol and Software". Available at <http://www.globus.org/datagrid/gridftp.html>.

[6] V. Jacobson and M.J. Karels. "Congestion Avoidance and Control", November 1988. This technical report is a slightly revised version of: V. Jacobson, "Congestion Avoidance and Control". In *Proc. SIGCOMM 1998, ACM Computer Communication Review*, Vol. 18, No. 4, August 1988.

[7] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput". In *Proc. SIGCOMM 2002, ACM Computer Communication Review*, Vol. 32, No. 4, August 2002.

[8] T.J. Hacker and B.D. Athey, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network". In *Proc. 16th IEEE-CS/ACM International Parallel and Distributed Processing Symposium (IPDPS) 2001*.

[9] R. Hugues-Jones. *udpmon: UDP Monitoring Tool*. Home page available at <http://datagrid.in2p3.fr/cgi-bin/cvsweb.cgi/network/udpmon/>.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "RFC 2018: TCP Selective Acknowledgment Options", IETF, October 1996.

[11] Net100 Project. Home page available at <www.net100.org>.

[12] NLANR. *Iperf version 1.6*. Available at <http://dast.nlanr.net/Projects/Iperf/>.

[13] F. Paganini, S.H. Low, Z. Wang, S. Athuraliya, and J.C. Doyle, "A new TCP congestion control with empty queues and scalable stability", submitted for publication, 2002.

[14] K. Pentikousis, "Can TCP be the transport protocol of the 21st century?", *ACM Crossroads*, Vol. 7, No. 2, Winter 2000.

[15] J. Sørensen. *gensink*. Available at: <http://jes.home.cern.ch/jes/gensink/>.

[16] W.R. Stevens. "TCP/IP Illustrated, Volume 1: The Protocols", Addison-Wesley, 1996.

[17] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson (Eds.). "RFC 2960: Stream Control Transmission Protocol", IETF, October 2000.