

Service Challenge 1 and 2

The Wide-Area-Network point of view

Andreas Hirstius

July 12, 2005

Abstract

This document will give a short introduction into wide area networking and some properties of TCP/IP. By using examples of real-life problems encountered during the Service Challenges the difficulties that are connected to finding the problems, to distinguishing "serious" problems from "hiccups" and to locating the root-cause in such a complex environment will be presented.

Based on the experience gained during the Service Challenges recommendations for a possible wide area network Monitoring and Debugging infrastructure for the Tier0 → Tier1 Data Transfer Service will be given.

Contents

1	Introduction	1
1.1	Wide Area Networking 101	1
1.2	Network Layout for Service Challenge 2	6
1.3	There's something about TCP...	6
1.4	... and the file size.	10
2	Network problems	13
2.1	In general...	13
2.2	The effect of network equipment at its limits	13
2.3	Throughput problem to FZK (and CNAF)	15
2.4	Throughput problem to RAL	18
3	Conclusions and Recommendations	19
3.1	Hosts – Conclusions and Recommendations	19
3.2	Network – Conclusions and Recommendations	20
3.2.1	Information to be collected – Ideas	23

Chapter 1

Introduction

The so called "Service Challenges" (SC) are meant to enable CERN and the LHC experiments to test the transfer the data coming from the experiments at CERN to the LCG Tier 1 sites around the world. The first Service Challenge was focused on general connectivity and basic functionality of the wide area links. Service Challenge 2 introduced the basic functionality of the gridftp service by transferring data from the local disks of the gridftp servers. The upcoming Service Challenges will introduce the access to the data on the CERN centralized storage system CASTOR and more and more of the Grid components necessary and eventually the software stacks used by the experiments.

The underlying (wide area) network infrastructure will become more and more complex until it reaches its final layout by about mid 2006.

While some of the observed problems that were related to some very special configurations will disappear in the final setup, other problems will inevitably appear. The additional software stacks on top of the networking layer will make it even harder to identify and debug problems.

The purpose of this note is to give a general introduction to wide area networking, to bring some of the inherent features of wide area networking to attention and to show the difficulties related to the identification and the debugging of problems in such an environment.

1.1 Wide Area Networking 101

This chapter will give a short introduction on the most relevant details of wide area networking and TCP/IP.

For an in depth view into these topics, the RFCs and other literature should be consulted.

TCP/IP is a set of communication protocols used to connect devices over a network infrastructure. This can be a Local Area Network (LAN) inside a computing center or a Wide Area Network (WAN) connecting computers across the planet.

The Internet Protocol (IP) defines a format for the packets (or *datagrams*) and a scheme for addressing the devices, much like a postal service.

The Transmission Control Protocol (TCP) establishes a connection between two hosts for exchanging data between them. It also guarantees the delivery of the packets to the higher level applications in the same order in which they were sent.

TCP/IP was developed in the 1970's as a reliable transport mechanism for the unreliable and slow networks of that time. All of its properties were designed for such an environment. It turns out that some of these properties are becoming a problem in modern very fast (and reliable) networks.

The most important feature of TCP is the ability to deliver the data reliably. To make sure that the data has arrived at the receiver, the receiver sends a so-called TCP ACK (TCP ACKnowledgement) back to the sender. This means that the sender has to keep the data available in memory for possible re-transmission in case the TCP ACK is not received within a reasonable amount of time.

The most important parameters to know about WAN connections and TCP are:

- The Capacity (\mathcal{C}) of the link (e.g. 622Mbit/s, 1Gbit/s, 2.5Gb/s, 10Gb/s)
- The distance in terms of the Round Trip Time (RTT) \equiv the minimum time for a TCP ACK to be received by the sender
- Maximum Transmission/Transfer Unit (MTU) is the largest size of an IP datagram for a particular connection
- The TCP congestion window (TCP window) is the maximum amount of data allowed to be in flight before the sender expects a TCP ACK from the receiving side. The size of the TCP window depends on the size of the send or receive sockets of the sending and receiving side. This means that the TCP window size can be controlled by the size of the TCP socket buffers.
- Packet Loss and TCP recovery algorithms. The standard TCP algorithm (TCP Reno) cuts the TCP window in half after a packet loss and only increases it by a Message Size (MSS = MTU - 40byte) per Round Trip Time.

From these "generic" parameters a number of important related variables can be calculated:

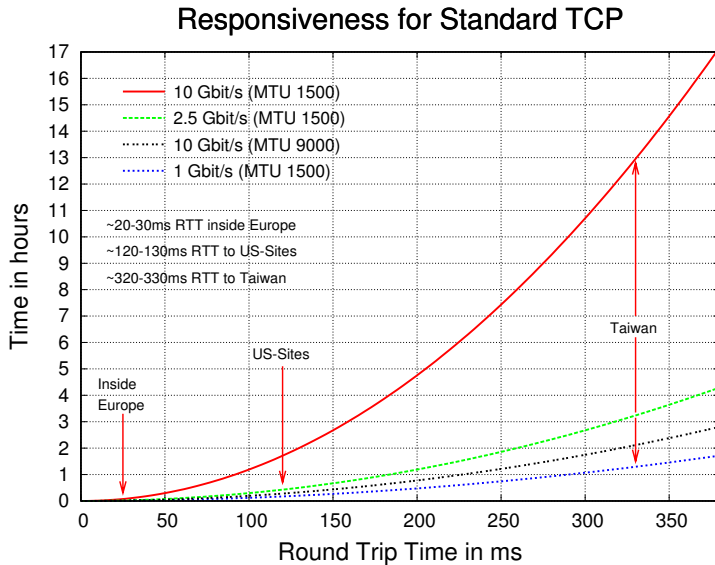


Figure 1.1: Responsiveness ρ of the TCP Reno algorithm. A single connection running at 1Gb/s from CERN to Taiwan with standard MTU would take about 90 minutes to recover from a packet loss. The same link at 10Gb/s would take 13 hours to recover from a packet loss.

- Bandwidth-Delay-Product (BDP) = $C * RTT$ is the maximum amount of data that can be in flight on a given link.
- Responsiveness $\rho = \frac{C * RTT^2}{2 * MSS}$ is the amount of time TCP Reno requires to recover from a packet loss (Fig. 1.1).

The BDP can become very large for a wide area link, e.g for a 1Gb/s connection to Taiwan (RTT ~ 330 ms), the BDP is ~ 40 MByte. While the size of the TCP window can be set to arbitrary values, the fundamental timing limits set by the Round Trip Time cannot be changed. Assuming a free link, a TCP window smaller than the BDP just means a waste of available bandwidth, because the sender stops sending before it can actually get the first TCP ACK back from the receiver. For example, a TCP window of 4MByte for a BDP of 40MByte limits the usable bandwidth for a single stream to (max.) 10% of the maximum available bandwidth.

Well, this is the theory and in practice the situation is even worse. Experience says that a TCP congestion window of about $2 * BDP$ is required to fully utilize a given connection. A possible explanation for this rule of thumb is the way TCP handles packet losses. After a simple packet loss the TCP window is cut in half and if the window is twice the BDP, it is still large enough to provide full throughput. This means that the connection can survive rare packet losses without affecting the throughput. Table 1.1 summarizes the important Round Trip Times and TCP window sizes. Even though the Tier 1 site in Taiwan (ASCC)

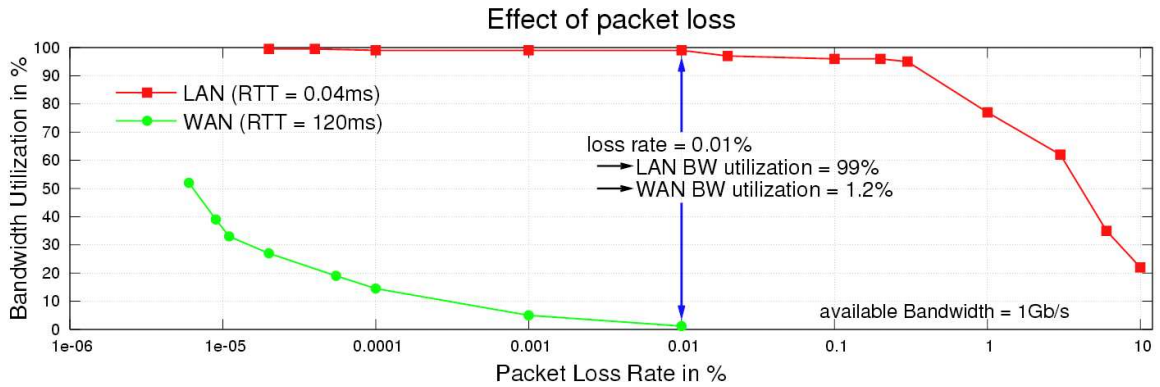


Figure 1.2: Maximum possible utilization of a network connection as a function of the packet loss rate, for 1Gb/s LAN (0.04ms RTT with proper equipment) and WAN (120ms RTT) connections. (Measurements done by the DataTag project).

was not involved in SC 2, it will be involved in the upcoming Service Challenges and naturally the final service. Due to the very large Round Trip Time it can be considered to be the most difficult case.

It also has to be mentioned that a standard 10Gb WAN connection using the current Synchronous Optical Network/Synchronous Digital Hierarchy (SONET/SDH) technologies does not have a capacity of 10Gb. The payload capacity is only $\sim 9621\text{Mb/s}$ due to the entirely different transfer mechanism compared to 10Gb Ethernet. For simplicity it will still be referred to as 10Gb WAN. This implies automatically buffering mechanisms when 10Gb LAN and 10Gb WAN links are being connected. The size of the required buffer is about 1MByte for every 10ms of Round Trip Time (see. [1],page 9).

The maximum possible utilization of a given network connection is, of course, closely related to the packet loss rate on this network connection. Figure 1.2 shows the relation between packet loss rate and possible utilization of the network connection. It can be seen that already a relatively small packet loss rate (0.01%) makes a WAN link completely unusable. In order to achieve a reasonable utilisation ($\gg 50\%$) of a WAN connection the packet loss rate has to be in the order of 10^{-8} . A faster link (i.e. 10Gb/s) requires an even smaller packet loss rate to achieve a high utilization (see Fig. 1.1).

All these problems of the current standard TCP implementation are well known and there is a lot of work ongoing to find a replacement for TCP Reno. One particular alternative TCP stack (TCP Westwood [2]) is already available in

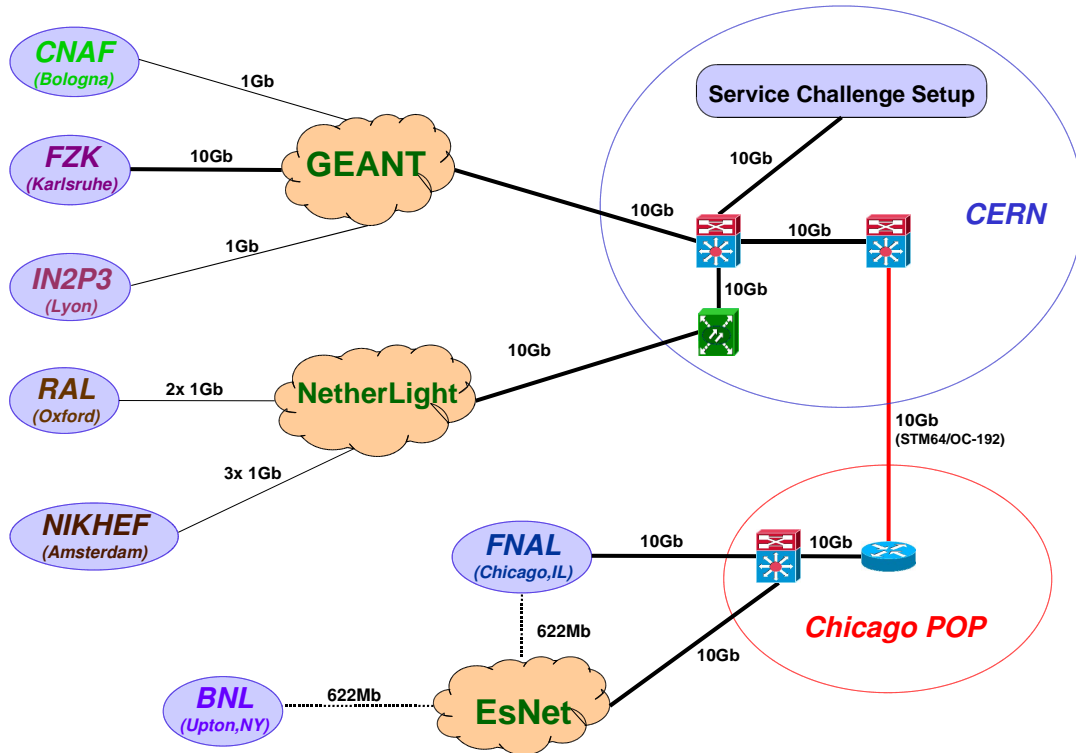


Figure 1.3: The network layout for Service Challenge 2.

the "vanilla" 2.6 Linux kernel series. The latest kernel dedicated to testing new features (the -mm series) contains now even more alternative stacks. The simple ones (highspeed TCP and scalable TCP) just provide a different method of calculating the TCP congestion window. The more advanced alternative stacks TCP Westwood+ and TCP Vegas use algorithms which are based on the Round Trip Time. BIC TCP [4] is an even more complex TCP algorithm which looks into many other events in order to optimize the TCP window. Other alternative TCP stacks such as FAST TCP [3] are only available as add-on patches to the kernel sources.

The principal approach that the different stacks take is quite similar. They try to optimize a given connection by using additional information about this particular connection. For example, it is possible to analyze the Round Trip Time or the TCP acknowledgement. The way they change the behavior of TCP is typically by changing the multiplicative parameters in the congestion avoidance and congestion control protocols. The actual implementation is, of course, very different among the alternative TCP stacks. They all have advantages and disadvantages compared to TCP Reno and compared to the other alternative TCP stacks.

Because of the impacts that a new TCP stack will have, the process of selecting the new "official" TCP stack is still ongoing.

1.2 Network Layout for Service Challenge 2

Figure 1.3 show the generic layout and the complexity of the network connectivity that was available for SC 2. Already at this stage there were nine carriers and NRENs (National Research and Education Networks) involved. Some of the links were dedicated (e.g. NetherLight) while other links were shared and some were even running without any guarantee for available transfer rate or support (e.g. GEANT to FZK). All computers on the CERN side were connected with 1Gb to a 10Gb switch. The data transfer was running from 15 machines, so the 10Gb switch was slightly over-subscribed (3 : 2).

The main results of SC 2 was an average transfer rate of ~ 600 MByte/s for the duration of 10 days with a peak transfer rate of ~ 800 MByte/s. There are a number of notes and presentations available which describe the immediate results and outcomes of Service Challenges 1 and 2 in more detail. (At the Service Challenge web site: [5] or at external sites: e.g. CNAF [6])

1.3 There's something about TCP...

For 1Gb end-systems at FNAL, the TCP window size required for maximum performance is ~ 30 MByte, while European sites would require 8 MByte or even only 4 MByte, and a 1Gb link to Taiwan would require a TCP window size of a stunning ~ 80 MByte (Tab. 1.1).

This shows that it is impossible to set the TCP window size on the gridftp servers to a value that suits all external sites due to the wide range of Round Trip Times.

During SC 2 the TCP window size was initially set to 2 MByte, but later increased to 4 MByte, which improved stability and throughput without any negative side effects. A 4 MByte TCP window means that the maximum possible single stream throughput to the US sites was much less than 20 MByte/s.

The main reason for using such a small window size is the use of parallel streams in gridftp to transfer a single file resulting in a very large total number of streams. The average number of (established) TCP connections per gridftp

Location	TCP window @1Gb	TCP window @10Gb
LAN (RTT: 0.04ms)	~ 10kByte	~ 100kByte
Europe (RTT: ~25ms)	~6MByte	~60MByte
US (RTT: ~125ms)	~32MByte	~320MByte
East-Asia (RTT: ~325ms)	~ 80MByte	~ 800MByte

Table 1.1: The recommended TCP window size for maximum throughput for the "standard" cases of the Service Challenges.

server during SC 2 was about 200, with a maximum number of about 600. For "historical" reasons people use 10 or even 20 parallel streams for a single file transfer using gridftp and each stream requires its own TCP buffer. With the current network infrastructure a single stream would be perfectly sufficient. Taking the downsides of TCP into account, like the time it takes to recover from a packet loss, 2 – 4 streams per gridftp session would be a more reasonable choice.

A large number of streams also wastes memory in the gridftp servers. TCP buffers have to be in physical memory and cannot be swapped. This means that even a relatively small number of gridftp sessions can cause the machine to run out of memory. With the 200 connections seen on average during SC 2, a full 800MByte of physical memory is required to accommodate the TCP buffers. At the peak of 600 connections a stunning 2.4GByte of memory were needed. This has serious implications on the underlying operating system. If the system runs out of memory the Linux kernel starts to kill processes in an attempt to keep the system alive. In principle this should not affect the system stability. Unfortunately it has been observed that the memory management subsystem in the Linux kernel is less robust on some architectures. Tests were done with x86 (SLC3 default kernel, 1-2GB RAM), amd64 (Fedora Core 2/3 default kernels + latest 2.6 kernels, 2-16GB RAM) and ia64 (SLC3 default kernel + latest 2.6 kernels 2-16GB RAM) based systems. The result was that the kernel on the x86 and amd64 based systems showed instabilities within a relatively short period of time (a few minutes to a few hours) after running into the out-of-memory condition. The kernel on the ia64 based system behaved in a much more stable manner. It survived several days of running under out-of-memory conditions.

In addition, a large number of streams per gridftp actually hinders the writing of the incoming data to disk or the data storage system. The transferred data in the separate streams contains information about the size and the offset of the chunk of the file that is being transferred. It will be written to disk immedi-

ately. For a gridftp session with 10 parallel streams this means, that the arriving data has basically a random order if there is even only a tiny difference in the transfer rates per stream, which is basically always the case. The effect of this is particularly bad for transfers to a local harddisk. It is possible to write between 40MByte/s and 80MByte/s to a modern harddisk with a single stream of data with sequential access (reading is similar). With more streams the transfer rates drop dramatically. The aggregate transfer rate for 10 parallel streams can be as small as a few MByte/s. When using RAID systems or network based storage this effect is usually smaller, but a random (write) access to a single file will always be much slower than a single sequential access. It also has to be considered that the data storage systems (CASTOR, dCache) have to provide blocks in a random order to the application. The effect of such a behavior on a larger scale is not known yet and has to be investigated.

Up to now it made sense to have a large number of parallel streams, because the total available bandwidth was very limited and additional streams meant a few kByte/s higher transfer rate. With the network infrastructure in place for the Service Challenges and for the final data transfer service itself this is no longer the case. To show the capabilities of such a setup, single stream disk-to-disk transfer rates of 475MByte/s between CERN and CalTech have been demonstrated (details in ANNEX A), using the same link to Chicago as the Service Challenges as well as a shared link between Chicago and CalTech.

During the upcoming Service Challenges and eventually in the production data transfer service the number of active TCP connections per gridftp server will be larger than during the Service Challenges 1 and 2 because access to the respective local mass-storage systems (CASTOR in the case of CERN) will be required. Fortunately, gridftp reads only a single stream of data off the data storage system and splits the traffic internally into multiple streams.

The TCP window size is related to the size of the TCP send and receive socket buffers, e.g. the TCP window size for a particular connection is set to the size of the TCP buffer the software chooses. Currently, the TCP send and receive socket buffer sizes in Linux are global system parameters allowing a minimum, a default and a maximum size of the respective buffer. Most software uses default buffer sizes when opening a socket. Since the TCP buffer sizes are global parameters, the same TCP buffer sizes, e.g. TCP window sizes, will be used both for the LAN and the WAN connections.

For reasons that are not yet fully understood, TCP shows unexpected erratic

behavior when the TCP buffers are "too" large for a given connection, e.g. the TCP window is much larger than the BDP. This is basically always the case for LAN connections (BDP @ 1Gb is only ~5kByte). Unfortunately this behavior limits the useful TCP buffer (e.g. window) size to 4MByte or 8MByte.

Table 1.2 shows example output for a 10Gb/s LAN connection (BDP ~100kByte) with 2MByte TCP window (left) and 200kByte TCP window (right). The connection was tested back-to-back and via a switch in order to check on the possible influence of too small buffers in the switch. There are no differences in the behavior of the transfer rates.

----- TCP window size: 2.00 MByte -----				----- TCP window size: 200 KByte -----			
[3]	0.0- 5.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	0.0- 5.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	5.0-10.0 sec	3.69 GBytes	6.33 Gbits/sec	[3]	5.0-10.0 sec	4.11 GBytes	7.05 Gbits/sec
[3]	10.0-15.0 sec	122 MBytes	205 Mbits/sec	[3]	10.0-15.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	15.0-20.0 sec	1.46 GBytes	2.51 Gbits/sec	[3]	15.0-20.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	20.0-25.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	20.0-25.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	25.0-30.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	25.0-30.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	30.0-35.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	30.0-35.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	35.0-40.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	35.0-40.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	40.0-45.0 sec	1.30 GBytes	2.23 Gbits/sec	[3]	40.0-45.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	45.0-50.0 sec	127 MBytes	213 Mbits/sec	[3]	45.0-50.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	50.0-55.0 sec	99.8 MBytes	167 Mbits/sec	[3]	50.0-55.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	55.0-60.0 sec	124 MBytes	208 Mbits/sec	[3]	55.0-60.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	60.0-65.0 sec	156 MBytes	262 Mbits/sec	[3]	60.0-65.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	65.0-70.0 sec	80.6 MBytes	135 Mbits/sec	[3]	65.0-70.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	70.0-75.0 sec	171 MBytes	288 Mbits/sec	[3]	70.0-75.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	75.0-80.0 sec	142 MBytes	239 Mbits/sec	[3]	75.0-80.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	80.0-85.0 sec	388 MBytes	651 Mbits/sec	[3]	80.0-85.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	85.0-90.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	85.0-90.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	90.0-95.0 sec	4.11 GBytes	7.06 Gbits/sec	[3]	90.0-95.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	95.0-100.0 sec	4.11 GBytes	7.05 Gbits/sec	[3]	95.0-100.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	100.0-105.0 sec	2.20 GBytes	3.78 Gbits/sec	[3]	100.0-105.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	105.0-110.0 sec	96.6 MBytes	162 Mbits/sec	[3]	105.0-110.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	110.0-115.0 sec	206 MBytes	345 Mbits/sec	[3]	110.0-115.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	115.0-120.0 sec	163 MBytes	274 Mbits/sec	[3]	115.0-120.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	120.0-125.0 sec	87.5 MBytes	147 Mbits/sec	[3]	120.0-125.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	125.0-130.0 sec	111 MBytes	187 Mbits/sec	[3]	125.0-130.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	130.0-135.0 sec	161 MBytes	271 Mbits/sec	[3]	130.0-135.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	135.0-140.0 sec	86.3 MBytes	145 Mbits/sec	[3]	135.0-140.0 sec	4.11 GBytes	7.06 Gbits/sec
[3]	140.0-145.0 sec	184 MBytes	309 Mbits/sec	[3]	140.0-145.0 sec	4.11 GBytes	7.06 Gbits/sec

Table 1.2: The erratic behavior of TCP on a LAN link (BDP ~100kByte) with "too;; large TCP windows.

There are a few solutions or workarounds possible to enable (much) larger TCP buffer sizes and to have a stable LAN connectivity at the same time. One possible workaround would be to "hardwire" the TCP buffer size in the application accessing the LAN. However, "hardwired" TCP buffer sizes are, in general, never a good idea. This seems to be the only practical "solution" at the moment, because it requires only small changes to the software. In the case of CERN this would be mainly CASTOR. However, for the external sites this workaround might not be possible, especially if commercial software is being used.

Another possible solution requires changes to the Linux kernel. At the moment the TCP parameters are system-wide parameters. It might be possible to

change the kernel to be able to assign each network interface its own TCP parameters. This opens the possibility of using two network interfaces: one with small TCP buffer sizes for LAN access and one with large TCP buffer sizes for WAN access. Although it is a relatively nice solution and definitely preferred compared to the previous workaround, it solves the problem by adding complexity to the network setup.

The only real solution for this problem and, at the same time, also improving overall efficiency is a kind of adaptive assigning of TCP buffers inside the application or even the TCP stack in the Linux kernel itself. If the application "knows" the RTT and the available bandwidth to the receiving side it would adjust the TCP buffer accordingly (within certain boundaries). This is the cleanest solution, but requires a lot of work inside the application (gridftp). If the default TCP buffers are set to standard LAN value the data storage software (CASTOR etc.) does not have to be changed. The "adaptive" algorithm would have to work inside the boundaries set by the minimum (or default) and the maximum TCP buffers.

There is already research ongoing in the field of adaptive TCP buffers for applications and also for alternative TCP stacks. The latter is naturally the most versatile solution. Unfortunately the progress in these fields of research is very slow.

1.4 ... and the file size.

The previous section was dedicated to the problems related to the TCP congestion window from a kernel and transport-software point of view. This section will describe the implications this has on the organization (i.e. file size) of the transported data and what kind of repercussions a particular file size of data has on wide area transfers. Only trans-oceanic connections are considered here, because connections to European sites can almost be seen as LAN (or MAN) (see Tab. 1.1) in terms of Round-Trip-Time.

During SC 2 the "standard" file size was 1GByte. This was reasonably large with respect to the TCP window size used (4MByte) and the BDP of the different connections. When the end-system will have a TCP window size close to the BDP, this file size might be too small, especially for transfers to East-Asia.

For efficient wide area transfers it is crucial to have large amounts of data to transfer at once. It is also important to have a transfer program which can handle canceling and reestablishing of connections (usually all pro-

grams can do that today).

In case of serious network problems all connections are canceled and if the duration of the problem is relatively short, it makes sense to restart the transfers where they stopped instead of retransmitting everything. Only if the files are tiny a complete re-transfer would be less problematic, but with tiny files the transfer itself is highly inefficient anyway.

The worst possible scenario is the one where files are smaller than the TCP window or much smaller than the BDP. Taking, for simplicity, 1MByte files, the average number of files in flight to the US-sites (BDP \sim 16MB) would be about 16 and to Taiwan/East-Asia (BDP \sim 30MB) about 30. This makes efficient transfers absolutely impossible for a variety of reasons

Firstly, the TCP connections have to be kept open/established for an extremely long time compared to the actual time the transfer takes. The 1MByte file can be transferred in 10ms on a 1Gb link but for connections to the US-sites it would take another 110ms before the sender can receive the TCP ACK, not to mention the 300ms delay for a connection to Taiwan. In other words, during the time the connection is waiting for the TCP ACK, another 10MByte or even 30MB could have been transferred to the US or Taiwan, respectively. This means that gridftp is doomed to remain idle (for \sim 90% or \sim 97% of the time), because the TCP session for this 1MByte file that it has just transferred cannot be closed until TCP knows that the file has been received correctly.

In order to get a somewhat reasonable utilization of the connection of a given gridftp server, there would have to be a huge number of active gridftp sessions per server. Even with only a single stream per gridftp session and a TCP buffer as big as the file, the amount of used TCP buffer memory would be enormous and mostly "idle".

With a file size of 1GByte, the average number of connections per server during SC 2 was about 200. With tiny 1MByte files, the number of connections would easily grow into the thousands.

Secondly, the transfer itself is inherently inefficient. A paper by the ATLAS TDAQ group [7] describes the problems that arise from the use of small files (\sim 1.5MByte) using their application as an example. Their result is basically that the performance of the application did not depend on the (possible) TCP throughput. Only the Round-Trip-Time determines the performance of their application. Even though the ATLAS TDAQ use case is not a pure data transfer service, the basic results can be applied to a data transfer service because it is

also based on TCP.

The last worry concerns the data storage and the data handling systems such as CASTOR or GRID catalogs, etc. In order to achieve an overall transfer rate of 1GByte/s those systems have to handle a minimum of 1000 files per second. With file sizes in the 1GByte range the number of files to handle would be significantly smaller.

Chapter 2

Network problems

2.1 In general...

There is a wide range of failure scenarios possible of course. This includes some hard-to-believe and somewhat "funny" problems. For example, during SC 2 a network cable was ruptured by an anchor of a trawler in Amsterdam. Then there are always the thoughtless construction workers cutting underground cables. This unfortunately happened and happens relatively often. The best story is, of course, the cable that was "eaten" by sharks.

The more serious problems range from obvious or "easy to debug" problems to well hidden and "really tough" to debug problems. Although such categories are anything but well defined. A simple symptom can have a really complicated cause and what seemed to be a really tough problem can have a trivial cause. While this true for basically every environment, the complexity of WANs, with its large number of technical/political/business/legal/administrative entities involved adds a whole new level to the issue of debugging problems.

The following sections will describe a few real problems encountered during the Service Challenges 1 and 2.

2.2 The effect of network equipment at its limits

Service Challenge 1 used the transatlantic 10Gb connection which was previously used by the DataTAG project. From measurements done prior to SC 1 it was known that data rates of more than 8Gb/s could be achieved with very small fluctuations of the transfer rate, e.g. a single stream transfer between CalTech and CERN could be run at about 7.4Gb/s for long period of time.

During SC 1, a stable transfer to FNAL was achieved over this link at about 5Gb/s. Because it was known that the link runs stable at much higher data rates, it seemed possible to take advantage of another ~ 1.5 Gb/s transfer rate to CalTech without affecting the transfer to FNAL. (The idea was to use 10Gb NICs at both ends in order to allow debugging of some serious issues with the drivers for 10Gb NICs and SATA controllers.)

Unexpectedly, it turned out that the transfer rate to FNAL already dropped when injecting only a few 100Mbit/s of additional traffic between CERN and CalTech. In order to not affect the transfers to FNAL, the additional traffic was kept to the absolute minimum necessary to debug this behavior.

The most plausible explanation is that some piece of network equipment was very close to its limits with the FNAL traffic alone and the end-systems at FNAL were already at their limits. Standard TCP (Reno) sends all data in the TCP buffer within a round trip time as fast as possible, because it can not make intelligent assumptions about the best way of sending the data. With much more than 10 machines connected via 1Gb links, this puts a lot of pressure on the buffering in the network equipment on the route, bringing one (or more) pieces of equipment very close to its limits. The additional TCP traffic (with TCP Reno) from nodes connected at 10Gb (but, of course, not running at 10Gb/s) caused this network equipment to behave in an unexpected way. Most likely it sent out the packets to the end-systems at FNAL in an extremely "bursty" manner. From previous measurements it was known that the end-systems at FNAL were already at their limits before, so the additional "burstiness" of the incoming traffic was too much for these machines and the result were less accepted packets. An illustration of the possible effects is shown in Figure 2.1.

In order to test this theory, a different TCP stack (FAST TCP) was used on the 10Gb end-systems. This algorithm distributes the transferred data more evenly over the Round Trip Time (Fig. 2.1). By using FAST TCP it was possible to run with a data rate of 2.5Gb/s between CERN and CalTech, bringing the aggregate rate close to 7.5Gb/s, without any effect on the 5Gb/s transfer rate to FNAL.

This is, of course, no absolute proof, but it is a possible hint into the right direction and it shows that there can occur quite unexpected things on a link that was thought to be fully understood.

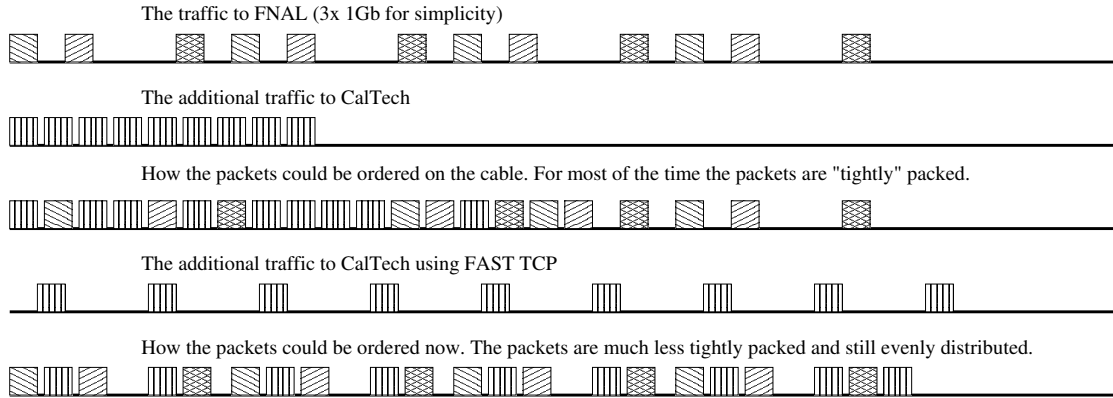


Figure 2.1: A highly simplified presentation of the effects that the additional traffic to CalTech has on the traffic to FNAL. With TCP Reno the additional traffic leads to significant changes in the traffic "pattern" of the traffic to FNAL. FAST TCP has a much smaller effect.

2.3 Throughput problem to FZK (and CNAF)

This problem was seen during the setup and test phase of Service Challenge 2, when only a small number of sites were running at the same time.

Initially a small asymmetry in the transfer rates between CERN and CNAF was observed, which was eventually detected and solved by GEANT. Starting from this relatively small problem other, much more serious, problems were discovered in the attempt to debug this first problem.

CNAF and FZK shared a part of the route to their sites via GEANT (see Fig.1.3). In order to check if this part was causing the problem, some tests between CERN and FZK were started under the assumption of course that the non-shared part of the route to FZK was without any problems. It turned out that the asymmetry in the transfer rates between CERN and FZK was extreme.

- CERN \rightarrow FZK: 900Mbit/s
- FZK \rightarrow CERN: a few 10Mbit/s (more than 90% of any traffic is lost)

Traffic in one direction (CERN \rightarrow FZK) was very stable and at a high rate, but traffic in the opposite direction (FZK \rightarrow CERN) was running with a very low and unstable rate.

One could, unfortunately, argue that the important link (CERN \rightarrow FZK) was working fine, so why bother...

The reason for bothering is that the TCP acknowledgements from FZK have to go via the problematic route and are therefore very likely to be lost. This results in retransmitted packets even though the original packet arrived without any problem. These retransmitted packets will, of course, reduce the usable bandwidth. A simple calculation: if just half of the packets have to be retransmitted at any given time, only half of the bandwidth carries useful data.

More than a week of very intensive tests of the respective local setups did not show any problems with the end-systems or the network equipment. For the local test at CERN, for example, the network group provided an additional 1Gb link to the external router, which made it possible to have a "virtual" external connection for these tests (see. Fig.1.3). All these local tests indicated that the problem is somewhere on the WAN link.

The problem was eventually escalated to GEANT/DANTE. Eventually they identified the root-cause as a faulty patch-panel connection somewhere in the GEANT backbone in France. Since information travels as light over optical fibers, any connection between two fibers has to be appropriate in order to avoid loss of light (= information). If, for any reason, there is no proper connection between the two fibers, like a large enough gap between the two end-faces, improper alignment or dirty end-faces, it is basically random which and how much of the light can travel between the two cables.

A very simple, even "trivial", cause with severe consequences which lasted several weeks. Furthermore, this type of problem is very difficult, if not impossible, to spot with standard monitoring tools, because the faulty connection was still capable of carrying data and the carrier (GEANT) can not know how much data is supposed to flow in either direction.

Unfortunately the problems did not disappear completely. There was still an unexplained and very regular packet loss pattern on the connection FZK → CERN.

After several weeks of intensive testing the cause was eventually identified while working locally on something unrelated: the improvement of the monitoring system for the gridftp servers. The packet loss on the receiving machine coincided with a call of the program *netstat* or *cat /proc/net/tcp*. The transmitting side did not experience a packet loss under the same conditions. At first this seemed unbelievable, but further tests with a large number of different kernel versions (2.4.xx and 2.6.xx) and on two different hardware platforms (Itanium2

and Xeon) proved that this actually was the cause of the packet losses. With 2.4.xx kernels a `cat /proc/net/dev` also causes packet loss, although much less severe.

A more detailed description of the packet loss problem can be found here [8].

kernel version	Itanium2	Xeon
2.4.xx	20%	60-70%
2.6.xx	10-15%	50-60%

Table 2.1: The packet loss rate in the second after a call of `netstat`. Measured with `iperf` running at the maximum possible UDP data rate and intermediate output every second.

This problem went unnoticed for so long, because CERN is usually the transmitting site and there is no problem on the transmitting side. In addition, a much more detailed monitoring has to be in place for the Service Challenges compared to the minimal monitoring necessary during pure R&D phases.

Although a lot of effort went into the debugging of this problem, involving also a number of Linux kernel networking experts in the Open Source/Linux community, it still isn't solved due to a lack of time.

The severity of this problem becomes clear when realizing that all streams lose packets at the same time, which is the worst case scenario for the TCP recovery algorithm(s) (see Fig. 1.1 and Fig. 1.2), because all TCP connections have to recover simultaneously from this packet loss. So at one point, all connections want to increase their TCP window to where the additional bandwidth will cause another "normal" packet loss. In the worst case more than one connection is affected by this.

Besides the technical aspects, an important lesson can be learned here: What started as a small problem with slightly asymmetric transfer rates eventually "ended up" finding physical problems thousands of kilometers away and even pointing to a serious problem inside the Linux kernel.

During normal operations the problem with the highly asymmetric transfer rates to and from FZK would be spotted immediately, but without an appropriate monitoring and, especially, debugging infrastructure in place it took several weeks to debug the problem, something that cannot happen during normal operations. The irony with this problem is that the monitoring itself causes the problem it should help to investigate (Heisenberg is everywhere...).

2.4 Throughput problem to RAL

The connection to RAL consisted of two 1Gb lines which were bundled into a "virtual" 2Gb link ("trunk"). Actually the two 1Gb lines were bundled and unbundled several times on the path to RAL. At a certain point some strange patterns in the overall transfer rate appeared. The debugging was a little complicated, because all other Service Challenge traffic was not to be disturbed. It turned out that TCP connections to machines at RAL with odd numbered IP addresses (last number) had very low and very unstable data rates. The TCP connections to machines with even numbered IP addresses were perfectly normal. To make matters more interesting, UDP connections to any of the IP addresses did not show problems. The obvious explanation was a load sharing/bundling problem. Due to a lack of access (even read-only) to the actual network equipment on the path this theory could not unfortunately be validated further. This problem was eventually "solved" by changing the network setup.

Chapter 3

Conclusions and Recommendations

This chapter will summarize the conclusions drawn from the information that was presented in detail in the previous chapters. Also a few recommendations will be given for a more effective monitoring/debugging system.

3.1 Hosts – Conclusions and Recommendations

The current setup of the hosts and the service is not optimal at all. There are several optimizations possible on the end hosts. They concern the kernel, the TCP stack and the applications itself.

- move to 2.6 kernels and alternative TCP stacks (like FAST TCP) as soon as possible
- reduce the number of parallel streams in gridftp
 - with standard TCP stack: 2 streams for links to EU/US; 4 streams to East-Asia (tests required)
 - with alternative TCP stacks: single stream to EU/US; 1-2 streams to East-Asia
- increase TCP buffer size (e.g. TCP window)
 - only possible if number of parallel streams is being reduced
 - because of LAN connections, maximum reasonable TCP buffer size most likely between 8MByte and 16MByte

- Try to modify software and/or kernel to be able to use different TCP buffers for LAN and WAN.
- Ideally introduce automatic/dynamic TCP buffer allocation in gridftp. This would make it possible to use the best TCP buffer size for a particular connection.

Some of those changes can be implemented in a very short timeframe. Other changes, like the automatic/dynamic TCP buffer allocation, are much more difficult to implement and might actually never be available.

3.2 Network – Conclusions and Recommendations

As presented in the major parts of this document, there is a wide variety of possible network problems.

Some of them are likely to re-appear in some shape or form while others will happen never again. It is crucial for the functioning of the data transfer service to find the cause of such problems as soon as possible and solve them.

This section summarizes the lessons learned during the debugging of network problems during the Service Challenges and during other projects, e.g. DataTag. It will also give some recommendations concerning the structure of a centralized monitoring system in this environment.

None of the recommendation are crucial to the functioning of the data transfer service itself, but they would significantly reduce the time to debug a network problem.

The main purpose of such a monitoring system is not just the pure monitoring. There is already a large number of monitoring tools out there. Such a centralized system will be able to provide powerful debugging capabilities through the database back-end and the data-mining capabilities such a back-end provides. The actual intervention at the remote sites or the carriers has to be done by the respective responsible persons.

Point 1. "Simple" network problems need to be solved only once.

Point 2. A central point of data collection is essential

- Point 3.** Easy access to the Service Challenge/production data transfer service machines at the external sites via a network connection that is not the dedicated data connection is crucial
- Point 4.** Immediate information from the network equipment of the carriers would help a lot
- Point 5.** Short reaction time by the carriers and/or direct (read-only) access to the equipment would also help a lot
- Point 6.** The most important one: Communication with other experts/sites (and accepting even the weirdest ideas - even if it's only for a moment)

While **Point 1.** is pretty much obvious, just look at Section 2.3, the points 2 - 5 naturally leave a lot of room for discussion.

Implementing all or most of these points in an actual monitoring system would significantly reduce the time necessary to debug a network problem.

Where would these things help??

- **Point 2:** A large number of sites will share at least some of the WAN infrastructure (especially inside Europe). Collecting all available information about the network equipment and the link status at a central site makes it possible to spot problems efficiently and finding correlations. Having a central point of data collection with access to all information provided by the carriers also helps preventing "surprises" during scheduled interruptions. In a de-centralized system it is quite likely that information about scheduled events does not reach everybody concerned, including also end-users. Furthermore, archiving and efficient data-mining is much more complicated, if not impossible, in a de-centralized system.
- **Point 3:** In case of a serious network problem on the dedicated link for the data it will not be possible to log into external sites via this link. Since there is always at least one additional link to the Tier 1 sites which is routed via different network equipment, it would be extremely helpful for efficient debugging to have a simple, easy and fast way of logging into machines at the external site via this link. If it is not possible to get access to the external sites, people at the external sites have to be present and help during the debugging. For most debugging purposes it is not necessary to have access to the entire external Computing Center. It seems to be sufficient to have access to 2 –

4 machines per subnet. The problem described in section 2.4 shows that access to only a single machine is not sufficient. Such access might need a somewhat special setup concerning security.

- **Point 4:** Having information about the status of all the network equipment on the path is also extremely helpful, if not crucial, for efficient debugging of certain types of problems (see Sec. 2.2 and 2.3). The network between the centers is a very complex system and it can involve a number of different carriers. With sufficient information from all equipment on the respective paths, it is much easier to spot a problem correctly and avoid "accusation-loops" (while the problems exists and afterwards).
- **Point 5:** Having a high priority for the carriers or even (limited) access to the network equipment will naturally help to debug problems much more efficiently.

Setting up a centralized monitoring system can be done with a reasonable effort by using already existing tools for distributed monitoring, data collection and data analysis like MonALISA [9], Ganglia [10], Cricket [11], MRTG [12], graphical traceroutes, etc. and maybe commercial tools like the one being used for the CERN internal network. Network operation centers like the Abilene NOC [13] can be seen as an example, although the environment in this case is much more homogeneous. The collection of information from the remote sites and the carriers seems to be also relatively simple (via SNMP). Unfortunately there are security issues related to SNMP access via WAN. Those issues can be overcome by using a machine at the remote site or a carrier to collect the information and send it to the monitoring system via http(s) or a specialized protocol for example. The part that requires most work is setting up powerful and easy to use archiving and data mining facilities.

With these facilities in place it will be possible to develop expert systems which automatically analyze the incoming flow of monitoring data. Although such expert systems have to learn over time, just like the "real" experts, for the most common (and recurring) problems they will be able to provide helpful information within a relatively short period of time. The system will be capable to do certain actions starting at day one. For example, it will be able to detect "trivial" things like scheduled interventions which otherwise would alarm the operators. Those would be presented as "known" and not trigger unnecessary action. Even though an expert system helps to debug and solve recurring problems, hands-on debugging will still make up the main part of the debugging efforts.

With a monitoring system in place that implements (most of) the recommendations, the time to debug a problem similar to the one with CNAF and FZK

(Sec. 2.3) would be as little as one or two days and not several weeks.

3.2.1 Information to be collected – Ideas

The local monitoring systems at the respective computing centers are assumed to be operational, so network problems inside the computing centers are taken care of. Although there has to be some information flow into the WAN monitoring in case of an internal problem that could effect WAN transfers.

Because the main purpose of such a monitoring system is to support any debugging effort that will be necessary, a much larger set of information should be and has to be collected from the external network devices.

There is the standard set of information from the devices like transfer rates and error rates. The additional information that would be very helpful for debugging purposes is much more detailed though.

The type of network device and the operating system it uses should be known, because there might be known or newly discovered issues with this type of equipment. An example for a piece of equipment that behave unexpectedly is a class of routers from a certain vendor which introduces packet reordering. It could also help spotting interoperability issues between equipment from different vendors.

The layout of the actual connectivity to and from the network device on the backbone is also interesting: how many connections per blade, utilization of the different links per blade and utilization of the backplane between the blades. The typical example here are routers with oversubscribed blades (i.e. 4x 10Gb/s connections per blade but only a 20Gb/s backplane). This point is, at the moment, only interesting for relatively old equipment, but in the future issues might arise that would require this type of knowledge. Network equipment in the LAN is usually oversubscribed because here the number of ports is more important. On any backbone infrastructure oversubscription is not wanted, because here the throughput is more important.

How much of the network equipment at the carriers is used by more than one link to a Tier 1 site? It might be possible that a number of links to external sites shares network equipment on their path. This is especially true for traffic via GEANT, where many links might actually use the same routers on their paths. A simple traceroute might not discover if the traffic to different external sites shares network equipment at the carriers (different IP-addresses, etc.).

The information necessary to cover last three points is more of a descriptive nature than actual "monitoring" data. A relatively simple metric provided by the carrier is sufficient, also because this type of information doesn't change often (hopefully). The actual problem is to correlate all this information correctly, for analysis and already at the visualization level. Take "trunks" as an example: a number of interfaces somehow bundled in order to provide better connectivity to a site. The typical problem here is an inefficient load balancing between the links.

In the ideal case it will be possible to inject test traffic between the routers on the path as part of an active monitoring system to support an actual ongoing debugging effort. Active monitoring of this kind should not be necessary during normal operation, actually it might even disturb the production traffic.

Acknowledgements

The author would like to thank Tiziana Ferrari (CNAF), Bruno Hoefft (FZK), Paolo Moroni, Edoardo Martelli (CERN), Sylvain Ravot and Dan Nae (CalTech), Catalin Meirosu (CERN, ATLAS) and all the other people involved in the Service Challenges.

Glossary

Institutions

BNL	Brookhaven National Laboratory http://www.bnl.gov
CalTech	California Institute of Technology http://www.caltech.edu
CERN	European Organization for Nuclear Research http://www.cern.ch
CNAF	INFN National Center for Telematics and Informatics, Italy http://www.cnaf.infn.it/
DFN	National Research and Education Network, Germany http://www.dfn.de/
FNAL	Fermi National Laboratory http://www.fnal.gov
FZK	Forschungszentrum Karlsruhe http://www.fzk.de
GARR	National Research and Education Network, Italy http://www.garr.it/
GEANT	pan-European data communications network connecting 26 National Research and Education Networks http://www.geant.net/
IN2P3	Institut National de Physique Nucléaire et de Physique des Particules http://www.in2p3.fr/
INFN	Istituto Nazionale di Fisica Nucleare, Italy http://www.infn.it
RAL	Rutherford Appleton Laboratory http://www.cclrc.ac.uk/Activity/RAL
Renater	National Research and Education Network, France http://www.renater.fr/
SURFnet	National Research and Education Network, Netherland http://www.surfnet.nl
UKLight	National Research Facility focused on optical networks http://www.uklight.ac.uk/

Technical Terminology

BDP	Bandwidth Delay Product
LAN	Local Area Network
MAN	Metropolitan Area Network
MSS	Message Size
MTU	Maximum Transfer Unit see RFC793 (http://www.faqs.org/rfcs/rfc793.html) and others
RAID	Redundant Array of Independent (or Inexpensive) Disks
RTT	Round Trip Time see RFC793 (http://www.faqs.org/rfcs/rfc793.html) and others
TCP	Transmission Control Protocol see RFC793 (http://www.faqs.org/rfcs/rfc793.html) and others
UDP	User Datagram Protocol see RFC768 (http://www.faqs.org/rfcs/rfc768.html) and others
WAN	Wide Area Network

LAN, MAN and WAN are usually defined by the Area they cover in terms of physical distance.

By this definition a LAN covers a "Computing Center", a MAN covers a "City" and a WAN covers "the World".

These definitions lack a sense for the "virtual" distance the signals have to travel which is measured in Round Trip Time. By using the RTT a LAN would be "defined" by $RTT \lesssim 1\text{ms}$, a MAN would have RTTs up to $\sim 10\text{ms}$ and WAN would have RTTs above that. Some low-end network equipment introduces an unnecessary latency into the link. This can turn a "physical" LAN easily into a "virtual" MAN or even WAN.

ANNEX A

The system configuration for the disk-to-disk transfer between CERN and CalTech.

The configuration at CERN:

- HP Integrity rx4640 (4x 1.5GHz Itanium2)
- 3x 3ware 9500S SATA controller
- 24x Western Digital HDD (WD740GD, 10k rpm)
- S2IO NIC
- 2.6.9 kernel

The configuration at CalTech:

- Newisys 4300 (4x 2.2GHz Opteron)
- 3x SuperMicro SATA controller
- 24x Western Digital HDD (WD740GD, 10k rpm)
- S2IO NIC
- 2.6.9 kernel

Achieved data rates:

- Local read/write CERN: 1100MByte/s and 550MByte/s
- Local read/write CalTech: 600MByte/s and 500MByte/s
- Disk-to-memory transfer CERN → CalTech: ~700MByte/s
- Disk-to-memory transfer CalTech → CERN: ~490MByte/s
- Disk-to-disk transfer CERN → CalTech: ~330MByte/s
- Disk-to-disk transfer CalTech → CERN: ~475MByte/s

ANNEX B

The TCP/kernel parameters used during Service Challenge 2.

```
### IPV4 specific settings
net.ipv4.tcp_timestamps = 1 #turns TCP timestamp support off
net.ipv4.tcp_sack = 1 #turn SACK support off, default on

# on systems with a VERY fast bus->memory interface this is the big gainer
net.ipv4.tcp_rmem = 262144 4194304 8388608 #sets min/default/max TCP read buffer
net.ipv4.tcp_wmem = 262144 4194304 8388608 #sets min/def./max TCP write buffer
net.ipv4.tcp_mem = 32768 65536 131072 #sets min/def./max TCP buffer space

### CORE settings (mostly for socket and UDP effect)
net.core.rmem_max = 4194303 #maximum receive socket buffer size
net.core.wmem_max = 4194303 #maximum send socket buffer size
net.core.rmem_default = 1048575 #default receive socket buffer size
net.core.wmem_default = 1048575 #default send socket buffer size
net.core.optmem_max = 1048575 #maximum amount of option memory buffers
net.core.netdev_max_backlog = 100000 #number of unprocessed input packets
                                #before kernel starts dropping them
```

The usage is very simple:
Copy those lines into a file and call "sysctl -p <file>".

Bibliography

- [1] Presentation: "Native Ethernet Transmission beyond the LAN" at TERENA Networking Conference Poznan June 2005 (http://www.terena.nl/conferences/tnc2005/programme/presentations/show.php?pres_id=56)
- [2] <http://www.cs.ucla.edu/NRL/hpi/tcpw/>
- [3] <http://netlab.caltech.edu/FAST/>
- [4] <http://www.csc.ncsu.edu/faculty/rhee/export/bitcp/>
- [5] <http://www.cern.ch/service-radiant>
- [6] Tiziana Ferrari et al., "Service Challenge at INFN: setup, activity plan and results", <http://www.cnaf.infn.it/~ferrari/sc>
- [7] R. Hughes-Jones et al. "Investigation of the Networking Performance of Remote Real-Time Computing Farms for ATLAS Trigger DAQ" in Proceedings of the 14th IEEE-NPSS RealTime Conference, Stockholm, Sweden, June 4-10, 2005
- [8] <http://www.cern.ch/openlab-debugging>
- [9] <http://monalisa.cacr.caltech.edu/>
- [10] <http://ganglia.sourceforge.net/>
- [11] <http://www.munitions.com/jra/cricket>
- [12] <http://people.ee.ethz.ch/oetiker/webtools/mrtg/>
- [13] <http://www.abilene.iu.edu/noc.html>