

AMS Internal Report

**Data Transfer
from Central Production Facility to Italian Ground Segment.
A prototype.**

M. Boschini , A. Favalli, M. Levtchenko
March 20, 2003

Abstract: in this note we present implementation and results of a prototype of a Data Transfer system, which is the mock-up of what will be set up to transfer AMS-02 data from the Central Production facility, located at CERN, to the Italian Ground Segment. The system is based on a client/server architecture and has been tested for more than 6 months continuously, with a low data loss. The system implements both a file transfer mechanism and a book-keeping system.

1. Introduction

As described in greater detail elsewhere ([1]), AMS-02 data handling will be composed of various parts, among which two main parts: the *Central Production Facility (CPF)*, which will be located at CERN, and the *Italian Ground Segment (IGS)*, which will contain the **master copy** of all AMS-02 data, both RAW and reconstructed.

One major component of the system will be the *Data Transfer (DT)*. Its goals will be moving data efficiently from CPF to IGS without interfering with CPF's activities. Efficiency is meant here as *no data loss at all, low network bandwidth consumption, negligible CPU load at CPF, efficient book-keeping*.

In order to reduce the interference with normal CPF's activities, a dedicated machine (*DTF*) will be located at the CPF, and all data transfer will happen from that machine.

Policies about data sharing between CPF and DTF have still to be decided upon.

Furthermore, DT will be monitored and surveyed by CPF, IGS and a third site, in order to allow maximum redundancy.

In order to start setting up a functional system, we developed a prototype and tested it against all above- mentioned requirements, for more than 6 months. The system is made out of a host at CERN and one at Milano-INFN. Greater details are given in the rest of this note.

2. Data Transfer System: hypothesis and architecture

The main goal of the Data Transfer System prototype is to move data from CERN to Milano, and keep track of what has been moved and if successfully or not. There are thus 2 components, which, for the sake of portability, we addressed separately and then merged together in the general architecture:

- data transfer protocol (*DTP*): part of DT in charge of actually moving the data.
- book-keeping mechanism (*BK*): part of DT in charge of keeping track of what has been moved and how (successfully or not).

The "natural" solution to integrate the two parts is a client/server architecture: move data from CERN to Milano, keep track at CERN of what/how was moved, notify Milano about moving, Milano receives data, Milano keeps track of what/how has been received, and so on...

This approach decouples the technical solutions adopted for the 2 components, making the whole system more portable and maintainable.

We will discuss the two components separately and then present the integration in the client/server system.

As for the DTP, we preferred to rely on wide-spread and commonly used protocols; before choosing we tested two different ones and finally studied how to optimize the chosen one.

BK has been taken care of by means of a Relational Data Base.

For the whole system we adopted OpenSource solutions, which are:

- Linux O.S.
- MySQL for the RDBMS
- Perl5, C for programming languages
- OpenSSL for encryption
- OpenSSH and bbftp for the DTP.

We used AMS-01 data, both raw HRDL files and n-tuples, as data to be moved. Obviously,

AMS-02 data files will be of different sizes, but, as we'll explain later, we also tested the system against various file sizes.

We also made the assumption that the expected data-rate will be the one expected for AMS02 [1b]

As for the requirements, we can summarize them as follows:

- low CPU load,
- sustain 3 times max data-rate, both in terms of bandwidth and of number of files,
- optimize bandwidth usage,
- low data loss,
- automatic recovery,
- acceptable security level,
- user-friendly GUI and Web tools.

3. Data Transfer Protocol implementation and optimization

As already mentioned, we decided to adopt an already existing Transfer Protocol.

We initially tested OpenSSH's `sftp` ([2]) utility, which implements a high level of security and data compression, but proved to be too heavy in terms of CPU load. In fact, `sftp` encrypts the whole transaction, which is not necessary for our needs, inducing a non negligible load on the sending and receiving side (even with low-level encryption algorithms such as `blowfish`) and data compression can be delegated to a pre-processing of the data. Since we had already tested with other members of the collaboration `bbftp` ([3],[4]), which had proven to be efficient and light in terms of CPU usage, we adopted `bbftp`. `bbftp` is a file transfer software, developed at IN2P3 computing Center in Lyon, France. It implements its own transfer protocol, which is optimized for large files, and has, among others, the following features which suit well out needs:

- Encoded username and password at connection
- Multi-stream transfer
- Big TCP windows as defined in RFC1323
- Automatic retry
- Customizable time-outs

As can be easily understood from the above mentioned characteristics, `bbftp` implements a sufficient level of security, since username and password are encoded at connection (which is not the case for standard ftp implementations), without overloading the system encrypting data which, for our needs, do not need such a high level of security. Furthermore, TCP parameter customization is a feature which we wanted to test, in order to optimize the system. The first test was to check if `bbftp` is able to use the maximum available bandwidth. Since the host is connected to a 10 Mb/s LAN part of CERN (Bld 506), we "filled the pipe" as much as possible, finding out that `bbftp` was able to transfer at a rate of 8.1 Mb/s. This result is confirmed by previous measurements performed by the AMS Software Group ([4]). We want to state here that, since the chosen protocol is able to use all necessary bandwidth, the bandwidth requirement of 3 times maximum data-rate is actually a requirement on the network link between CPF and IGS. According to the current trend in the European Research Network, this would not be an issue for AMS-02.

One of the characteristics of `bbftp` is the high level of customization of TCP parameters: in

particular we were interested in studying the TCP window size versus Number of Streams parameters, in order to optimize the overall performances.

In particular, we wanted to find the optimal ratio Number of Streams / TCP Window size, since, although at a first glance a high number of streams may seem a good approach to increase data transmission rate, this is not always true.

In fact, as stated in various studies about this issue ([5],[6],[7],[8]), increasing the number of streams increases the number of collisions on the network link, and their initialization induces a non negligible load on the CPU and thus, at the end, may slow down the data transfer.

On the other hand, a typical approach to high performance TCP, indicates the TCP window size as one of the most important parameters¹.

According to theory ([9],[10]), the maximum achievable throughput on a RFC1323 compliant network is obtained when the window size is given by:

$$\begin{aligned} \text{optimal window_size} &\sim 3 \times \text{capacity} \\ \text{where} & \hspace{10em} \text{(A)} \\ \text{capacity} &= \text{bandwidth} \times \text{RTT} \end{aligned}$$

In principle thus, for any given network, setting the appropriate window size would ensure maximum throughput without requiring the use of parallel streams.

In reality, in the case of network links with a high capacity (a.k.a. LFNs²), the optimal window size may be in the order of hundreds of Kbytes: but often various devices in the path may have a maximum value for the window size which is much smaller³. This means that setting the optimal window size on the sender and the receiver may fail or be anyway useless because of windows resizing performed at one of the path hops.

In this case, using a 64 Kbyte⁴ window size and various parallel streams reliably implements maximum throughput.

This is particularly true for transcontinental links (USA --> EU⁵) which are characterized by high RTT's.

In general thus, one should first evaluate the network's capacity and the optimal window size, and then, if it is a "reasonable" value (i.e. a value which can be easily and safely implemented across the link) use that value and one single stream, otherwise use a "safe" window size (typically 64 Kbytes) and a high number of streams.

In our implementation we thus performed several tests on the Bld506 to Milano INFN network, initially using `iperf` ([9]), to characterize these network parameters, and then with `bbftp`. The two measurements show an optimal agreement. In our tests we thus adopted a window size of 64 KB and 1 stream.

One should nevertheless keep in mind that a realistic production environment will be characterized by higher bandwidths; we plan on performing TCP optimization again as soon as a high bandwidth link will be available for our test.

¹This is so true, that a dedicated datagram has been added to TCP to enable a fast calculation of the optimal window size.

²Large Fat Networks

³CISCO PIX Firewalls, old routers, Linux kernel 2.2, Linux kernel 2.4.x not correctly configured, etc.

⁴ Which is the default for most routers and for kernel 2.4.x Linux.

⁵ For example, a link FNAL --> Milano INFN has a capacity of approximately 200 Kbytes.

4. Book-Keeping implementation and integration with the DTP

One major issue of the Data Transfer System is book-keeping: one has to keep track of what was available to be copied, if and how it was copied, where it can be found in the master copy. We define this as a **book-keeping** problem, and one has to keep in mind that this has nothing to do with the data catalog, which keeps track of what kind of data is contained in a certain file.

The problem, which can easily and efficiently be approached using a relational DB system, can be described as follows:

- which file is ready to be moved to IGS
- how has it been moved (successfully or not)
- which file has been received at IGS
- where has it been stored.

As already mentioned, we used MySQL as RDBMS, after performing some stress-tests that showed that it was robust and reliable enough to suit our needs ([10]). In particular these tests show that the DB design adopted is able to populate the database at a rate of 3500 rows/sec⁶. Thus, if we assume that we insert 1 row per transferred file, from the DB point of view we could be able to transfer at least 1000 files/sec, which is obviously much more than needed.

According to the above described requirements, the book-keeping and the data-transfer have been integrated in the Data Transfer System, always keeping in mind that the whole system has to be totally decoupled from the technical solutions adopted for each component (in fact, just for the sake of exercise, the system can also use Oracle RDBMS for book-keeping, just changing a command line parameter...)

The whole system can be briefly summarized as follows:

there is a daemon in infinite loop at CERN which

- looks for new file(s) in a "spool" directory at CERN.
- If file(s) found, fork-itself and go back to daemon mode, while the forked process performs
 - DB at CERN update
 - bbftp file(s) to IGS (with a bbftp slightly modified by us to avoid clear passwords in files)
 - update DB at CERN with bbftp status
 - connects to a network daemon at IGS
 - updates DB and log system with final status

At IGS, as just mentioned, there's a daemon listening on a dedicated TCP port, which, when connected to from CERN,

- forks a DB update with info about file,
- checks for file,
- updates DB and log system with last info and Status.

⁶In the worst case, with 3 concurrent writes to the DB, insert rates drops to 2500 rows/sec. Selection rate is 20 rows/sec

Network connection between CERN and IGS is performed by means of the *Secure Socket Layer* protocol, since sensible data is transmitted (DB table, DB info).

If for any reason (bbftpd daemon failure, network failure) the transfer is unsuccessful⁷, the file is made available for a later transfer. If transfers keep failing, a dedicated daemon moves files from standard spool directory to a special directory, in order to avoid filling up the spool space.

4.1 Data Integrity

Data integrity is checked by means of the MD5 algorithm. An MD5 digest is calculated⁸ at the sending site (CERN) and sent to the receiving site (Milano) together with the DB request for update. The receiving site calculates the MD5 digest of the received file and compares it with the one received from the sender. If the two differ, DB is updated accordingly (INTEGRITY = 'FAIL') and the DB consistency mechanism, described in the following paragraph, takes care of re-transmitting the file.

5. Data Transfer Management and Survey

Obviously, in the described scenario, although there are several checks, an overall DT Survey is necessary. We approached the problem with 2 solutions that, implemented together, should provide the system with a general and reliable survey.

The first component is pretty simple, and just because of this, pretty reliable: a check between DB entries at CERN corresponding to files which seem to have been transferred ok and DB entries at the IGS, which should contain only those files, is performed every hour.

This check is based on pure SQL selections and "textual" comparisons. In case of any difference, a mail is sent to people in charge (as for now, only the authors) and a web page is updated. Furthermore, all files which, according to CERN's DB seem not to have been transferred correctly or are in an undefined state (e.g., to-be-transmitted since a long time), are anyway re-transmitted. This, as demonstrated during the tests, recovered automatically approximately all data that was not sent correctly, which was anyway ~ 0.03% of a whole sample of 380K files.

The second component of the Survey system is based on a redundancy approach, in which a third host, located in Milano, will keep a "copy" of both CERN and IGS DB's.

The copy is performed in 2 different ways: DB dump/restore and a on-line update.

The first approach is very simple: DB at CERN and IGS are dumped periodically and sent to the third host, where they are restored in a RDBMS.

The latter approach is based again on the client/server architecture: when DB is updated at CERN, a parallel DB is updated⁹ at the third site. Similarly, whenever the DB is updated at IGS, a request for update is sent also to the third site. Currently, the "dump" solution is fully operational, while the client/server one is still under development

6. Database WEB-Browser

We developed and tested an Internet Database Browser. The main goal of this project is enabling a fast and easy to use access to the DT system and its monitoring.

The system, initially written in Perl5, has been moved partially to PHP, because of the better performances in accessing the Data Base and building dynamic web pages.

⁷Failure is defined ONLY from bbftp return status different than 0. Because of minor bugs in bbftp, we rely only on this value.

⁸See `md5sum(1M)`

⁹Sending a request for update to the DT server.

The Internet Database Browser includes two modules: the first is a server monitoring module, that provides status information about Data-Transfer services and database status; the second one is a DB browser.

The part responsible for the system status monitoring is written using Perl5, because it has a greater and more flexible interface to the Linux O.S.. This is necessary in order to obtain information about running processes.

On the other hand, memory management of PHP's MySQL API's is more efficient than Perl5 DBI single pointer array management. This provides the end-user with a more flexible and secure way to process all necessary requests. This tool, which has undergone a stress-test of 5 months, also enabled the Data Transfer Group to debug and test the Data Transfer System. The second part is thus build using PHP and PHP's MySQL API's and provides dynamically generated web pages. Those pages are not only giving the list of database entries with date, time of transfer, location of data, status of transfer, but include a powerful search engine.

7. DataTransfer GUI

In order to make the data transfer semiautomatic and user friendly, a DataTransfer GUI was developed.

When monitoring the data transfer with the Database Browser, users may experience situations in which (due to power cuts or system stability problems), data transfer went partially down or in other ways some of the services are not working properly.

For these cases we provide, both at CERN and IGS, a GUI which allows an easy monitoring of each and every part of the DT system.

The GUI also allows to perform operations on bringing up and shutting down the whole system and every desired part just by clicking one button.

This tool proved to be very useful when we were reinstalling Data transfer on a new computer at CERN.

The GUI is entirely written in Perl5 and GTK.

8. Tests and Results

The first release of the system ([12]) has been run successfully for more than 6 months, until a data loss of 1% was detected, thanks to the web browser system just mentioned.

The system has then undergone major bug fixes and recoding. In particular we modified the TCP window size approach and the DB consistency checks. The new version, which we described in this note, has run successfully (except for power outages at CERN...) for more 6 months, with no data loss.

The system has then undergone a stress test period of 3 months, during which we randomly stopped one or more of the services (network, bbftpd daemon, MySQL, etc) at one or both of the sides and studied the ability of the system to recover after those failures.

While the system was pretty robust in discovering daemon failures and alerting personnel in charge (as for now, the authors) and recover automatically after the services had been restarted, network outages were more problematic.

As long as the network outages are short enough not to fill the available spool space, and as long as enough bandwidth is available, the system successfully transmits all data. Problems arise with bbftp if it is transmitting data while the network goes down: a small fraction of the times (~ 0.03%), bbftp returns a successful state even if the data are not correctly transmitted. This problem is, as for now, addressed by means of the DB consistency and recovery system.

9. To do

Here we briefly summarize what will be done in the next future:

- Implement tests on a 100 Mb/s CERN LAN in order to study and optimize TCP window size and NUM_STREAMS in a scenario more similar to the AMS-02 one.
- Implement on-line redundancy with third host.
- Integrate with High Speed Network hosts at CERN (pcgiga.cern.ch¹⁰) as suggested and discussed with IT Communication Services.

10. Conclusions

We presented the actual setup, performances and test results of the prototype of the Data Transfer System, which will in be in charge of copying data from CERN's CPF to the Italian Ground Segment, where a master copy of all AMS-02 data will be kept.

The current release of the system has been tested for an overall period of 9 months.

More than 380000 files have been transferred correctly, of which ~ 5% needed to be retransmitted because of network outages. A minor (~0.03%) data loss, due to protocol inconsistencies was discovered and now is automatically recovered.

TCP parameters have been studied and tweaked in order to optimize the bandwidth usage and reduce the overall load on the hosting CPUs.

The whole data set is organized in a DB, which acts as a data transfer book-keeping system.

The DB proved to be robust and fast enough to suit our needs.

The system can be handled through a GUI and monitored via web.

References

[1] A.Klimentov, V. Choutko, AMS-note-2001-11-02

[1b] P.Fisher, A.Klimentov, AMS- note-2001-05-01

[2] <http://www.openssh.org>

[3] <http://doc.in2p3.fr/bbftp/>

[4] A.Klimentov, A. Eline, AMS-note-2001-11-02

[5] RFC 2914

[6] <http://www.csm.ornl.gov>

[7] Matthews, Cottrell, "Achieving High Data Throughput in Research Networks", CHEP2001.

[8] <http://www-iepm.slac.stanford.edu/monitoring/bulk/>

[9] R. Stevens, "TCP/IP Illustrated, Vol.1: The Protocols", Addison-Wesley, 1994",

[10] Downey, Dijkstra, SC2002

[11] <http://dast.nlanr.net/Projects/Iperf/>

[12] M. Boschini, A. Favalli, AMSnote-2002_01_04d

¹⁰As suggested by M. Gerard of IT Networking Group.

Appendix A

Short description of technical details of HW and SW of current setup of the prototype.

CERN:

*AMD Athlon 1700+, 256 MB
Disk 50 GB IBM-DTLA ATA
RedHat 7.1 (tested also on RedHat 6.2 and 7.0)
MySQL 3.23.46
OpenSSL 0.96h
bbftp 2.1.2
Perl 5.6.0, DBI, DBD:1.33*

Milano:

*AMD Athlon 1900+, 512 MB
Disk ADATPEC/BROWNIE IDE-SCSI Raid5 355 GB
RedHat 7.3 (tested also on RedHat 6.1, 7.0, 7.1 and 7.2)
MySQL 3.23.53
OpenSSL 0.96h
bbftp 2.1.2
Perl 5.6.1, DBI, DBD:1.34*

Appendix B

Here following bandwidth usage of 3 protocols on various network links.

**CERN-->MIB on a 10 Mbit/s line (10/16/100)
capacity¹¹ @ 65 KBYTE (calculated on average RTT)**

	<i>IPERF¹² (Mb/s)</i>	<i>BBFTP¹³ (Mb/s)</i>	<i>NCFTP (Mb/s)</i>
1 stream 2K	1	0.6	1.1
2 stream 2K	2	1.3	
4 stream 2K	4	2.2	
1 stream 8K	3.1	3	3.2
2 stream 8K	6.2	6.1	
4 stream 8K	8.4	8.1	
1 stream 16K	6.8	6.5	6.6
2 stream 16K	8.5	8	
4 stream 16K	8.6	8.1	
1 stream 32K	8.5	8.1	8.1
2 stream 32K	8.6	8.2	
4 stream 32K	8.6	8.2	
1 stream 64K	8.7	8.2	8.1
2 stream 64K	8.7	8.2	
4 stream 64K	8.7	8.2	

**MIB INFN -->MIB INFN on a 100 Mbit/s line (LOCAL)
capacity @ 1.5 KBYTE**

	<i>IPERF (Mb/s)</i>	<i>BBFTP (Mb/s)</i>	<i>NCFTP (Mb/s)</i>
1 stream 2K	47.3	28.1	60
2 stream 2K	88.8	59.6	
4 stream 2K	91	62.2	
1 stream 8K	88.5	82.3	86.5
2 stream 8K	91.6	87.7	
4 stream 8K	92.6	88.1	
1 stream 16K	91.7	87.6	86.8
2 stream 16K	93.4	88.6	
4 stream 16K	93.7	88.6	
1 stream 32K	93.8	88.7	87.4
2 stream 32K	93.8	88.7	
4 stream 32K	93.8	88.7	
1 stream 64K	93.8	88.7	87.4
2 stream 64K	93.8	88.7	
4 stream 64K	93.8	88.7	

¹¹As defined in [9].

¹²Average on 30 measurements.

¹³Average on 10 measurements with variable file size (8MByte < file_size < 100 Mbyte).

**MIB-->CILEA on a 34 Mbit/s line (100/34/34/100)
capacity @ 27 KBYTE**

	<i>IPERF (Mb/s)</i>	<i>BBFTP (Mb/s)</i>	<i>NCFTP (Mb/s)</i>
1 stream 2K	2.6	1.6	2.8
2 stream 2K	4.9	2.9	
4 stream 2K	9.2	5.6	
1 stream 8K	6.9	6.9	6.8
2 stream 8K	11.3	11.1	
4 stream 8K	13.2	12.8	
1 stream 16K	12.2	12	11.5
2 stream 16K	13.6	12.9	
4 stream 16K	13.6	13.1	
1 stream 32K	12.6	12	2.8
2 stream 32K	13.1	12.9	
4 stream 32K	13.6	13.1	
1 stream 64K	13.8	13.6	13.4
2 stream 64K	13.8	13.6	
4 stream 64K	13.8	13.6	

**FNAL¹⁴-->MIB on a 10 Mbit/s line
capacity @ 200KBYTE**

	<i>IPERF (Mb/s)</i>
1 stream 2K	0.13
2 stream 2K	0.26
4 stream 2K	0.53
1 stream 8K	0.23
2 stream 8K	0.48
4 stream 8K	0.94
1 stream 16K	0.50
2 stream 16K	0.96
4 stream 16K	1.89
1 stream 32K	0.81
2 stream 32K	1.82
4 stream 32K	3.79
1 stream 64K	1.56
2 stream 64K	2.82
4 stream 64K	6.14

¹⁴ Fermi National Accelerator Laboratory, Chicago, U.S.A.