SCIENTIFIC PAPERS, UNIVERSITY OF LATVIA, 2011. Vol. 770
COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES 56–67 P.

File Transfer Protocol Performance Study for EUMETSAT Meteorological Data Distribution

Leo Truksans, Edgars Znots, Guntis Barzdins
Institute of Mathematics and Computer Science
University of Latvia
leo.truksans@lu.lv, edgars.znots@lumii.lv, guntis.barzdins@lumii.lv

The described study provides the experimental results and their analysis for selection of the file transfer protocol to be used in the upcoming Meteosat Third Generation Programme requiring sustained dissemination data rate in the range of 300-400Mbps over heterogeneous networks. This dissemination speed cannot be easily achieved with default TCP protocol settings and file transfer applications with significant round trip time (RTT) and packet loss typical to large heterogeneous networks. The designed test lab allowed finding the optimal TCP protocol and file transfer application settings reaching the target data rate at 70ms RTT and 10^{-6} packet loss, typical to terrestrial networks. Meanwhile, none of the surveyed applications were able to reach the target data rate at 700ms RTT, typical to satellite distribution networks.

Keywords: computer networks, data dissemination, satellite communications.

1 Introduction

In January 2010, the European meteorological union (EUMETSAT) commissioned IMCS to perform a detailed study on currently available open standards file transfer protocols for TCP/IP networks. The purpose of the study was to provide the background experimental material for the selection of a file transfer architecture in the upcoming next generation Meteorological weather satellite system to be launched in 2014 – EUMETSAT.

The results obtained in this study could be of interest to much wider audience, as there are much ungrounded myths about the performance of underlying TCP protocol and data transfer applications built on top of it. Under permission from EUMETSAT, in this paper we provide a condensed version of the original technical report.

The purpose of the study was to perform a multi-dimensional survey of four file transfer protocols (FTP, UFTP, bbFTP, GridFTP, RSYNC) under widely varying conditions characteristic to various network conditions. Namely, performance 70ms and 700ms RTT characteristic to intercontinental terrestrial Internet and geostationary

satellite communications were studied. Additionally, various packet loss patterns were examined.

The measurements were conducted in controlled laboratory environment, which was meticulously fine-tuned and validated to ensure that the lab setup itself could not be the cause of negative artifacts during measurements. The lab itself was built from open-source components rather than from closed commercial network emulators. This enabled full tunability of the network simulator performance characteristics and parameters (e.g. insertion of various packet loss patterns: random packet loss, packet loss in random bursts, etc.) – we were not limited by the constraints of the given test platform.

2 Test Lab Description

As depicted in Fig. 1, a single test bed (two identical test bed sets were used during this study) consisted of a file transfer server connected via LAN switch with one of the clients and a network simulator. Second client was placed behind a network simulator. Switch port connected to the server was mirrored and all traffic originated from or sent to the file transfer server was copied to the traffic monitoring server. In case of unicast file transfer scenarios, data was sent between the server and client behind the network simulator. For multicast scenarios, data was sent from server to both clients. All machines had at least dual 1GbE NICs, and each machine had a separate interface used for management purposes only.

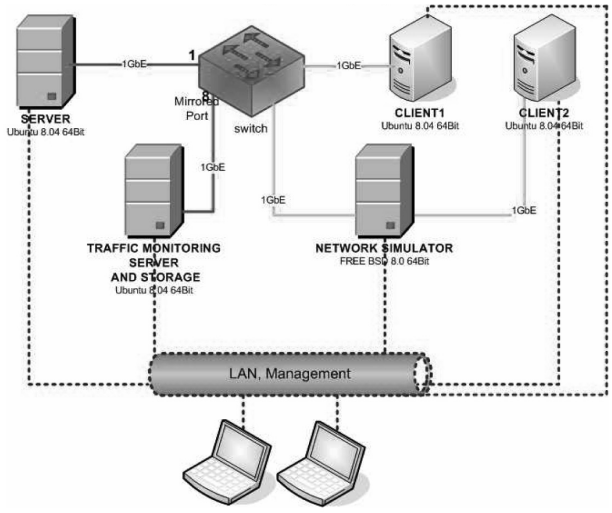

Fig. 1 Test lab topology

2.1 Hardware

All servers used within the test bed achieved or surpassed the necessary performance levels to ensure that results obtained in the study were not biased due to performance bottlenecks in equipment used.

Network simulator had AMD Opteron 148, 2.2 GHz single core CPU, 2GB RAM, dual Broadcom NetXtreme 1GbE network interfaces (BCM5704), 160GB SAMSUNG HD160JJ WU100-33 HDD.

File transfer server, clients and traffic monitoring server had two AMD Opteron 275, 2.2 GHz, dual core CPUs, 8GB RAM, dual Broadcom Tigon3 1GbE network interfaces (BCM95704A7), 80GB WDC WD800JD-00LSA0 HDD.

A small but capable HP ProCurve Switch 1800-8G was used for LAN connectivity.

2.2 Software

Network simulator operating system was FreeBSD 8.0-RELEASE. Network simulator software: ‚ipfw' and ‚dummynet' subsystems in the default system kernel.

File transfer server, clients and traffic monitoring server had Ubuntu Linux 8.04.4 LTS, 64-bit operating system. Usage of 64-bit kernel was essential for optimal memory addressing and necessary for large TCP buffers. File transfer applications: ProFTPd 1.3.1, vsftpd 2.0.6, bbFTP 3.2, GridFTP 4.2.1, UFTP 2.10.3, RSYNC 2.6.9. Traffic logging software: ‚tcpdump', default version provided with distribution.

2.3 Network tuning

After the default server installation, TCP parameters for all machines were tuned for better TCP throughput.

The following system configuration variables were tuned in all Ubuntu servers in accordance with best current practice [1]:

```
#Tuning Linux TCP settings
sysctl net.ipv4.tcp_window_scaling=1
sysctl net.ipv4.tcp_timestamps=1
sysctl net.ipv4.tcp_sack=1
sysctl net.ipv4.tcp_moderate_rcvbuf=1
sysctl net.ipv4.tcp_syncookies=0
sysctl net.ipv4.tcp_no_metrics_save=1
sysctl net.ipv4.tcp_ecn=1
sysctl net.ipv4.tcp_adv_win_scale=7

#Tuning Linux TCP buffers
sysctl net.core.rmem_max=16777216
sysctl net.core.wmem_max=16777216
sysctl net.ipv4.tcp_rmem=»4096 16000000 180000000»
sysctl net.ipv4.tcp_wmem=»4096 16000000 180000000»

#Tuning network interface parameters
ifconfig eth1 txqueuelen 10000
```

After initial server distribution installation on the network simulator, FreeBSD kernel was recompiled to enable Dummynet network simulator functionality, as well as to increase kernel time resolution to 40000Hz for more precise and consistent RTT

simulation. After kernel recompilation, both network interfaces were configured in single virtual bridge, so that traffic was transparently passed through FreeBSD network simulator between Ubuntu server and client machines on both interfaces.

2.4 Test bed validation

In order to validate test bed host capability to execute all test cases and produce correct measurements for scenarios specified in the study, several baseline performance measurements were performed. Initially, raw TCP and UDP throughput was measured for test bed hosts connected in a back-to-back configuration. After initial measurements, another set of test runs was performed with addition of switch between test bed hosts. Also, baseline RTT measurements for back-to-back and switched cases were performed for later comparison with test bed configuration accommodating network simulator.

Back-to-back, test bed hosts achieved raw TCP throughput of 941Mbps and raw UDP throughput of 957Mbps. RTT of 73µs – 7.8ms (avg. 76µs) in a back-to-back configuration was measured. The highest RTT was observed for the first packet, and was caused by necessity to perform the ARP request. The standard deviation of 26µs shows reasonably timed and predictable network interface operation.

Addition of a switch between test bed hosts did not have any significant effect on achievable TCP/UDP throughput or RTT. Switched performance was exactly the same as in case of back-to-back configuration – 941Mbps TCP and 957Mbps UDP. RTT was a little lower – 61µs-103µs (avg. 66µs) with a low standard deviation of 11µs.

Rapid sending of 100,000 ICMP ECHO requests (further – ‚ping flood‘) was used for measuring consistency of introduced RTT at network simulator. Iperf TCP and UDP tests were carried out to measure raw TCP and UDP throughput. These tests were run for one hour. RTT measurements were performed for first 100,000 ICMP ECHO (ping) packets.

At no packet loss and no introduced delay the raw TCP and UDP performance was again 941Mbps and 957Mbps, respectively. The RTT slightly increased to the range of 175-411µs (avg. 330µs) with a standard deviation of 44µs. These numbers demonstrated network simulator performance as stable and low impact on packet flow [2].

Results from measurements of RTT consistency at pre-configured RTT of 70ms and 700ms showed that maximum observed deviation from specified RTT was 1.9ms, average deviation from specified RTT was 0.58ms at 70ms RTT and 1.2ms at 700ms RTT. Since this falls well below 1% error margin relative to specified value, network simulator RTT simulation could be considered as consistent.

3 Testing Methodology

To understand and demonstrate the practical limitations of selected applications and protocols, a plan of test scenarios was created. The scenarios fall into six categories as detailed in the Table 1: small unicast, medium unicast, large unicast, mixed unicast, mixed multicast, large multicast. All the tests were run for one hour, except test 19 which was run for 5 hours. The 5th category used a mix of all file sizes with all RTT variants but with only two packet loss rates (10^{-6}, 10^{-3}). The 6th category used just 2GB large files and the worst packet loss (10^{-3}).

Table 1

All test scenarios

Scenarios	Category: file sizes	RTT	Packet loss rate
1,2,3,4,5,6	Cat1: 10kB	1,2,3: 70ms 4,5,6: 700ms	1,4: 0 2,5: 10^{-6} 3,6: 10^{-3}
7,8,9,10,11,12	Cat2: 5MB	7,8,9: 70ms 10,11,12: 700ms	7,10: 0 8,11: 10^{-6} 9,12: 10^{-3}
13,14,15,16,17,18	Cat3: 2GB	13,14,15: 70ms 16,17,18: 700ms	13,16: 0 14,17: 10^{-6} 15,18: 10^{-3}
19,20	Cat4: mix of 10kB, 500kB, 5MB, 50MB, 2GB	19: 70ms 20: 700ms	0
21,22,23,24	Cat5: mix of 10kB, 500kB, 5MB, 50MB, 2GB	21,22: 70ms 23,24: 700ms	21,23: 10^{-6} 22,24: 10^{-3}
25,26	Cat6: 2GB	25: 70ms; 26: 700ms	10^{-3}

During each test, the tcpdump utility on traffic monitoring server was used to capture first 68 bytes of each packet. After each test, raw captured data was uploaded to a separate system with large storage for off-line analysis. The analyses allowed to account data packets separately from protocol control messages and was protocol specific. All performance indicators in this study are for actual data without protocol overhead.

To better observe behavior of the protocols and applications, two histograms were generated for each scenario:

- 1st histogram: 5 second period for the 70ms RTT scenarios; 50 second period for the 700ms RTT scenarios;
- 2nd histogram: 15 minute period for all scenarios.

The reason why the first histogram represents the longer period of 700ms RTT scenarios is that in most such scenarios little data throughput was observed during the first 5 seconds. These seconds were mostly used for session initiation and negotiation. It can be seen in the actual histograms that traffic pattern of 5 second period at 70ms RTT was very similar to that of 50 second period at 700ms RTT, given other parameters equal. This confirms earlier observations that TCP throughput is inversely proportional to RTT [3].

Although all the tests were run for at least 1 hour, the 15 minute interval was determined to demonstrate at least one full session, yet be short enough to distinguish session cycles.

Graphs are drawn in logarithmic throughput scale to better illustrate performance patterns in presence of highly disproportionate absolute values.

4 File Transfer Applications and Test Results

The following applications performance results were obtained based on data gathered during this project. For every application, the impact of different file sizes and packet drop rates at fixed RTT was demonstrated. We chose to base observations on fixed RTT because from all the condition variables particularly RTT seemed outstanding for two reasons:

- Both RTT values (70ms and 700ms) represent different real life usage scenarios. The first is a representative RTT for a global terrestrial network. The second is extremely high RTT appropriate for geostationary satellite communication;
- All the protocols and applications tested in this project demonstrated substantial performance decrease in 700ms RTT link. The best case at this latency was GridFTP in scenario 16 reaching throughput of about 38MBps (304Mbps).

4.1 70ms RTT

Fig. 2 shows seven FTP scenarios: 1, 2, 7, 8, 13, 14, 19. They represent all three file sizes for 70ms RTT and 0 or 10^{-6} loss. The graphs have been grouped in 3 pairs – each for a different file size. The performance differs dramatically: the large files (2GB) are transferred at about 100MBps while small files (10kB) are transferred at about 20kBps. It can also be concluded that low or no packet loss does not impact the average performance much due to Fast Retransmit [4]. Transfer of mixed file set in scenario 19 shows varying average performance as it increased during the transfer of larger files and decreased during the transfer of small files.

Fig. 3 shows the same set of scenarios for GridFTP. Scenarios 13 and 14 show that the full benefits of GridFTP protocol were achieved when transferring large files over parallel streams.

Usage of multiple streams and TCP window scaling proved to be most advantageous for medium and large files on links with no packet loss, since GridFTP has the opportunity to accelerate and reuse each data connection. But, this same approach seems to be less effective in presence of packet loss. The larger is the file transferred over a TCP connection, the more likely is that some packets will be dropped for that file. Thus, the longer a TCP connection is used on a lossy link, the more likely this connection will experience degrading throughput over time due to several close packet drops that cause decrease of TCP window. Hence, re-use of existing data connections in case of high packet loss is undesirable. It is much more effective to use a connection only for a lifetime of singe file transfer, and then reopen new connection with the highest possible initial TCP window.

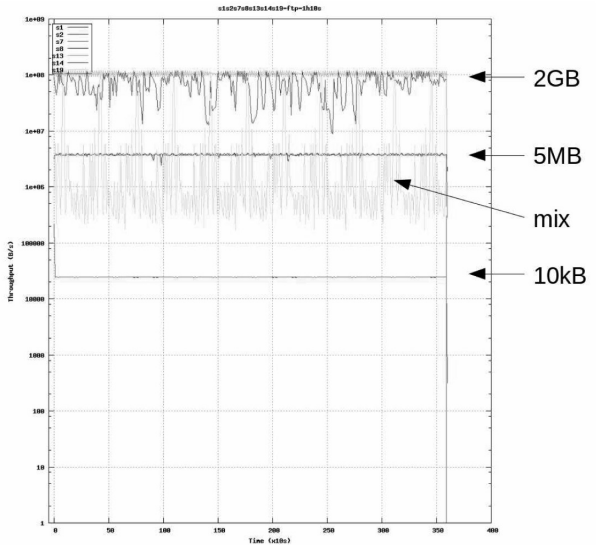

Fig. 2 FTP protocol, little or no packet loss, 70ms RTT

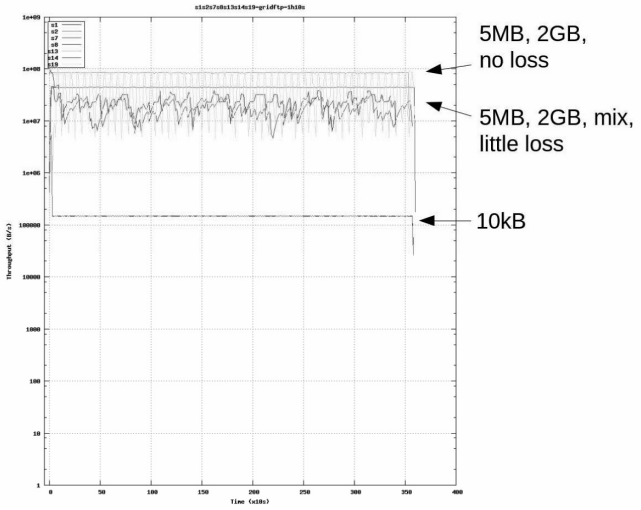

Fig. 3 GridFTP protocol, little or no packet loss, 70ms RTT

This explains why throughput curves are almost identical for scenarios 8 and 14 – because if the same data connection(s) are reused for the whole duration of file transfer session, the TCP window in both cases will converge to the same size depending on

packet loss rate, and irrespective of whether medium or large files are being transferred. As it can be seen, in case of high-throughput of large files packet loss rate is much more important than RTT.

4.2 700ms RTT

Fig. 4 shows seven other scenarios for FTP: 4, 5, 10, 11, 16, 17, 20. They represent all three file sizes for 700ms RTT and 0 or 10^{-6} loss. It can be easily seen that the graphs are grouped in 3 pairs again – each for a different file size. The performance also differs dramatically but the throughput is significantly lower. Rare packet loss impacted average performance of the large file transfer in scenario 17. Occasional packet drops close one to another made TCP to reduce window and drop performance in few of the transfers. This is indicated by the "ladders" in the graph. Comparing Fig. 4 to the Fig. 2 (FTP, 70ms) an observation can be made that traffic patterns are "stretched" about 10 times as RTT becomes 10 times longer. At the same time, performance decrease was about 10 times as the RTT was increased 10 times. As mentioned previously, it complies with findings in [3].

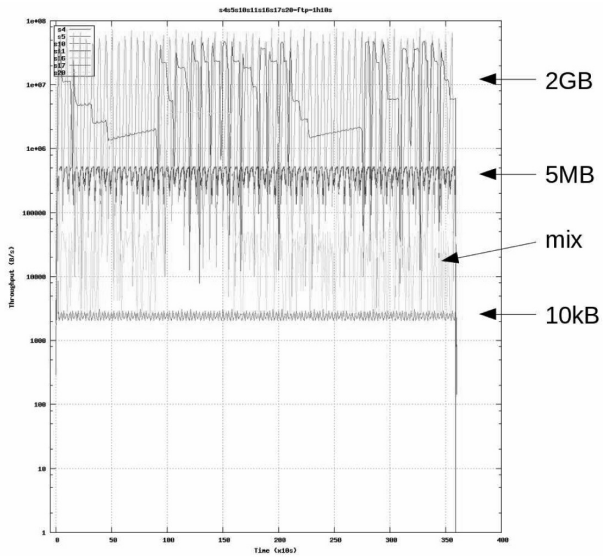

Fig. 4 FTP protocol, little or no packet loss, 700ms RTT

Fig. 5 shows all three file sizes for 700ms RTT and 10^{-3} packet loss. At this packet drop rate, large and medium file sizes show degraded performance while small files show no difference. Larger RTT decreased performance even more. As a result, 2GB file transfer at worst conditions was possible only at average throughput of 50kBps.

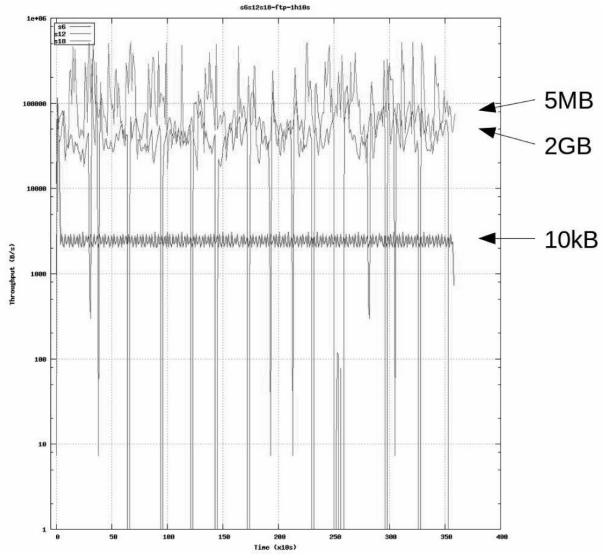

Fig. 5 FTP protocol, high packet loss, 700ms RTT

As in case with 70ms RTT, also here performance with large files was slightly lower than performance with medium files. The reason is the same – fast starting TCP could transfer most 5MB files with few or no lost packets and did not need to decelerate [5].

5 Conclusions and Recommendations

5.1 Conclusions on application suitability

The criterion of application suitability in this study was the earlier defined *target* performance. The target performance level was specified by the EUMETSAT as the minimum throughput of 350Mbps (43,7MBps) for the next generation real time content delivery.

5.2 FTP suitability

FTP protocol mandates a separate TCP connection for control session and a new TCP connection for every data stream. A data stream is either a data file or a directory listing. FTP commands get sent over single permanent control connection. Inefficiency of the multiple file sending process can be clearly seen in the analyzed data and histograms given earlier in this document. It took at least two round-trips to initiate a new file download in the presence of an already open session. In case of 70ms RTT, it meant at least 140ms lost in protocol "chat" for every file regardless of its size.

The analyses show that FTP reached *target* only in scenarios 13 and 14, which were sending 2GB files at 70ms RTT and no or rare packet loss. None of the other FTP scenarios reached *target* as a consequence of either smaller files, higher RTT, more frequent packet loss, or a combination of these.

5.3 UFTP suitability

UFTP application spent at least 4 seconds for every session initiation in our tests. The sender was always explicitly provided with IP addresses of both receivers. Session initiation took even longer time in case of open client participation. UFTP had to start a new session for every file it sends. As a result, UFTP application is unsuitable for sending files of small or medium size. Those are even out of scope in a UFTP design and performance study [6] that focuses only on large files.

UFTP reached *target* only in scenario 25, which was sending only large (2GB) files. The other parameters were: 70ms RTT and high packet loss. One of the reasons why this scenario was added to the test plan was to show at least one multicast scenario when UFTP reaches *target*. None of the other UFTP scenarios (mixed file sizes) reached *target* as a consequence of either smaller files, higher RTT, or a combination of both.

Scenario 25 shows another interesting point. UFTP does not suffer much from packet drops. Its delivery process does not retransmit a lost packet immediately following a NACK. Instead it continues to transmit file at the given rate (900Mbps in all tests) and collects NACKs for the next phase. During a subsequent phase, it transmits only the lost fragments at the same given rate. It repeats phases until every receiver has received a complete file. Another welcome feature of UFTP phased delivery process is that any receiver that has received a complete file finishes the session with the sender while other receivers may continue with more phases in case of NACKs.

Scenario 25 was run in presence of worst packet loss. UFTP was the only application that reached *target* at worst packet loss rate (10^{-3}).

5.4 bbFTP suitability

Although bbFTP protocol allows sending large files over multiple parallel streams, it has the same protocol limitations as standard FTP. bbFTP reopens data connections for transfer of each subsequent file, thus it was not possible to send files of small or medium size at a high throughput. Moreover, bbFTP uses fixed TCP window size, and has several implementation restrictions on stable settings for the number of parallel streams and TCP window sizes used. The usage of fixed TCP window size may have advantages only in high packet loss scenarios. Even then, bbFTP was unable to achieve high enough throughput.

bbFTP did not reach *target* throughput in any scenario, it performed worse than even standard FTP in all scenarios except ones with packet loss ratio of 10^{-3}. Best case throughput for bbFTP – scenario 13: 36MBps (288Mbps).

Also, bbFTP was poorly implemented and crashed periodically during deployment, configuration and execution of test scenarios. If properly implemented, ensuring more stable and reliable operation, as well as allowing usage of more than ten parallel streams and optimal performance with non-default TCP buffer parameters, bbFTP could possibly be considered for use in large file transfer in high packet loss cases. But, considering the state of bbFTP at the time of this study, it was more perspective to research on the possibilities of achieving the same benefits of using fixed TCP windows during transfers with high packet loss by tuning GridFTP operation specifically for high packet loss scenarios.

bbFTP cannot be suggested based on the data gathered in this study.

5.5 GridFTP suitability

GridFTP protocol opens permanent data connection(s) that can be reused to transfer multiple files. This feature resembles protocols like RSYNC and clearly allows achieving higher throughput with small files. For files large enough (bigger than what can be sent within one TCP window), GridFTP was able to utilize parallel transfer of single file over several streams – a feature common with bbFTP. However, we could not confirm how scalable GridFTP was in capability to send multiple files simultaneously over the open parallel connections, as it was outside the scope of this project.

Due to GridFTP capability to reuse open data connections, utilize TCP window scaling provided by operating system, as well as parallel transfer of large files through several streams, GridFTP is highly suited for medium and large file transfer on WAN networks with no packet loss. But in case of packet loss on the network, GridFTP will experience dramatically decreased throughput depending on packet loss rate, amount of parallel streams used, and duration of file transfer session.
GridFTP reached *target* only in scenarios 7 and 13, which were sending 5MB and 2GB files at best conditions (70ms RTT and no packet loss). None of the other GridFTP scenarios reached *target* as a consequence of either smaller files, higher RTT, more frequent packet loss, or a combination of these.

5.6 RSYNC suitability

RSYNC protocol opens single TCP connection for the duration of whole session. This connection carries all the control commands and file data, including multiple file transfer. Upon starting the session, RSYNC application compares the given local directory with the given remote directory, calculates differences and only then starts to send actual files. This initial comparison makes the average throughput of the tests lower than the performance that can be observed during actual file transfer.

Still, single TCP connection process gives good results with all tested file sizes in case of no severe packet loss. RSYNC reached *target* in scenarios 1 and 13, when sending the smallest and the largest files (10kB and 2GB, respectively) at best conditions (70ms RTT and no packet loss). RSYNC was the only application that reached *target* with 10kB files. The other RSYNC tested scenarios (3, 15, 18) had worse conditions and did not reach *target*. This limitation on achievable throughput at high RTT or packet loss is common for all tested applications that rely on TCP window scaling provided by the operating system (FTP, GridFTP, RSYNC).

5.7 Conclusions on Applications and Protocols

Exceptional results produced within the study:

- Highest performance – FTP scenario 13: 105MBps (840Mbps);
- For small files (10kB) only RSYNC reached *target* (scenario 1: 55MBps (440Mbps));
- At worst packet loss (10^{-3}) only UFTP reached *target* (scenario 25: 45MBps (360Mbps));
- At 700ms RTT only GridFTP came close to *target* (best case – GridFTP in scenario 16: 38MBps (304Mbps)).

Only FTP application could surpass 2x *target* mark (700Mbps, 87,5MBps) in scenario 13. GridFTP and RSYNC were close to that mark in that scenario reaching 84MBps and 87MBps, respectively.

It can be concluded that four of the five applications tested have shown their best performance at some specific scenarios. Any of them may be considered for use depending on the anticipated file sizes and infrastructure or dissemination process constraints. Only bbFTP was unable to reach *target* and can be excluded from further consideration.

Several recommendations were given considering various possible assumptions about the infrastructure. Summarizing all recommendations at different assumptions the following general but not strict recommendation was made: use UFTP for multicast or if packet loss is high, otherwise use RSYNC or *tar+nc*.

The tar+nc as a very simple recommendation emerged as an afterthought after the detailed tests contracted by Eumetsat and that was described here. Tar+nc is a combination of archiving tool ‚tar' and network session tool ‚nc'. It packs together a given set of files in a single network session. We believe the performance patterns of such solution to be similar to rsync, but without the need to compare directories at session beginning. These tools are present in any mature network operating system and have evolved to be very powerful, yet simple and achieving same top throughput rates.

References

1. K. Sataki, B. Kaskina, G. Barzdins, E. Znots, M. Libins: BalticGrid-II project final report on Network Resource Provisioning, 2010. URL: http://www.balticgrid.org/Deliverables/pdfs/BGII-DSA2-9-v1-2-FinalReport-IMCSUL.pdf
2. M. Carbone, L. Rizzo: Dummynet Revisited, SIGCOMM CCR, Vol. 40, No. 2, April 2010.
3. Lee J., Cha H., Ha R.: A Two-Phase TCP Congestion Control for Reducing Bias over Heterogenous Networks, In: Proceeding of Information networking: convergence in broadband and mobile networking : international conference, ICOIN 2005, Jeju Island, Korea, January 31-February 2, 2005, LNCS Vol.3391, Springer, 2005.
4. M. Allman, V. Paxson, W. Stevens, RFC 2581: TCP Congestion Control, April 1999.
5. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. RFC 2018: TCP Selective Acknowledgment Options, October 1996.
6. J. Zhang, R. D. McLeod: A UDP-Based File Transfer Protocol (UFTP) with Flow Control using a Rough Set Approach, submitted to IEEE Transactions on Networking, 2002.