# High Rate Telemetry Transfer – CCSDS SLE and Beyond

Martin Götzelmann[*], Martin Karch[†] and Dario Lucia[‡]
*VEGA Space GmbH, Darmstadt, 64393, Germany*

Ricard Abello[§] and Holger Dreihahn[**]
*European Space Agency, European Space Operations Centre, Darmstadt, 64393, Germany*

**This paper describes results of an ESA study on High Rate Telemetry Transfer Services which analyses technologies, protocols, and products with respect to their capability of supporting transfer of high volumes of telemetry data from a ground station to mission user facilities at high data rates. The objective of the study is to identify the throughput limits of the widely used CCSDS Space Link Extension Services, to analyze whether they can support mission requirements in the next 15 years, and to propose improvements or alternatives if applicable.**

## I. Introduction

The CCSDS Recommendations for Space Link Extension (SLE) transfer services have become the predominant internationally accepted standard for interoperability between TT&C networks and mission user facilities. The first systems supporting SLE services were developed around the year 2000 when the data rate requirements would typically not exceed a few hundred kilobits per second for telemetry. For missions with much higher downlink rates such as Earth Observation (EO) missions, the limited ground communication bandwidth enforced special solutions typically based on execution of the initial processing steps on the ground station and subsequent transfer of files. Since introduction of SLE services both the available ground communication capacity and the telemetry data rates have grown significantly and are expected to grow even faster in the coming years; science missions scheduled for launch 2015 to 2020 are preparing for downlink rates between 50 and 100 Mbps; EO missions currently in phase C/D (e.g. the GMES Sentinels) will downlink data at about 600 Mbps and for EO missions in earlier phases downlink rates of 1 Gbps and more are being discussed.

ESA has therefore started a study on high rate telemetry transfer services to evaluate the maximum rates that can be supported by the current SLE services and existing equipment, to analyze the factors that limit the achievable rates, and to study new technologies and protocols with respect to their capability of supporting high data rates over high latency Wide Area Network (WAN) links. Starting from the current SLE Provider performance of 160Mbits for SLE offline telemetry retrieval, the objective of the study is to investigate whether SLE services can support mission requirements for the coming 15 years. Depending on the outcome of the this investigation, the study shall elaborate recommendations for improvements (if applicable) or develop draft specifications for a new generation of high rate telemetry services including both online streaming and file transfer as input for the development of a new generation of TM and TC processing equipment and as input to CCSDS standardization of Cross Support Transfer Services (CSTS).

The study has concluded the first phase of investigations and tests, the results of which are reported in this paper focusing on the measurements related to data transfer via wide area networks. We describe the test configuration, identify and justify the selection of protocols and features evaluated, present the tests performed and report the results. The paper concludes with a summary of the lessons learned and initial conclusions drawn.

[*] Principal Consultant, Technology Division, VEGA Space GmbH, Europaplatz 5, D-64293 Darmstadt, Germany
[†] Project Manager, Technology Division, VEGA Space GmbH, Europaplatz 5, D-64293 Darmstadt, Germany
[‡] Software Engineer, Technology Division, VEGA Space GmbH, Europaplatz 5, D-64293 Darmstadt, Germany
[§] Signal Processing Engineer, Ground Systems Engineering Department, European Space Operations Centre (ESOC), Robert-Bosch-Str. 5, D-64293 Darmstadt, Germany
[**] Software Engineer, Ground Systems Engineering Department, European Space Operations Centre (ESOC), Robert-Bosch-Str. 5, D-64293 Darmstadt, Germany

## II. Test Set-up

The configuration used to measure the achievable throughput of different protocols is shown in Figure 1. Data were transferred on a Gigabit point to point connection from a server machine (Machine A) to a client machine (Machine B). The traffic was routed via a third machine running a Wide Area Network (WAN) emulator[††] to simulate delays and packet loss. No further network equipment such as routers or bridges has been included in the configuration. The machines used are standard PCs with Intel Xeon Quad-Core 2.8 GHz CPUs, 8 MB cache, and 4 GB RAM. Machine A is equipped with 4 disks (1 TB, 7200 rpm) in RAID 5 configuration. The operating system used on all machines is SUSE Linux Enterprise Server 11 (SLES-11).

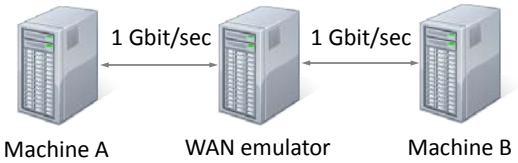


**Figure 1. Physical Test Configuration.**

The achievable throughput for pure transport layer protocols was measured using of-the-shelf tools[‡‡]. For the measurement of the achievable data rate using SLE Return Services, we developed a special performance test platform based on existing SLE User and SLE Provider components, which were stripped down to the minimum required functionality and carefully tuned for performance. The test components that can be configured within the platform are shown in Figure 2: The Frame generator serves as online telemetry source and can generate TM frames for any mixture of virtual channels at configurable rates. The test provider application is a minimalistic SLE provider application capable of transmitting telemetry on one or more SLE service instances and storing / reading telemetry to and from disk storage. Likewise the Test user Application is a minimalistic SLE User Application which can receive telemetry on one or more SLE service instances but does no further processing. The test platform includes the standard ESA SLE API package[5,6] which is in operational use since many years. In the tests we used the latest released version of this package as well as special test versions that were tuned for performance. All software components except user interfaces are implemented in C++.

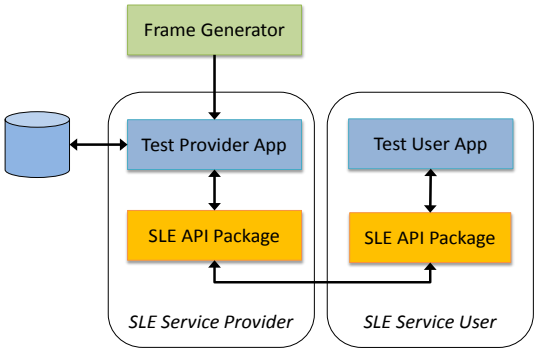


**Figure 2. SLE Performance Test Platform.**

## III. Transport Layer Protocols

SLE Services are in principle independent of any specific transport but in practice all known implementations make use of the default transport layer interface defined by CCSDS (Internet SLE Protocol One, ISP1[7]), which in turn is based on TCP. Obviously TCP was therefore the prime candidate to analyze. The User Datagram Protocol (UDP) has been analyzed as a potential candidate for telemetry transfer across local interfaces. Although pure UDP based transfer across a wide area network is not an option, the results are of general interest and therefore reported in this paper.

At the beginning of the last decade much research was performed into communication on "long fat pipes", i.e. connections with high latency and high bandwidth, and various improvements to TCP as well as new transport layer protocols have been proposed. These proposals differ from TCP primarily in the approach to flow control which is typically rate base rather than window based and the algorithms applied for congestion avoidance. Examples for such proposals include FAST[§§] (Caltec 2005), Scalable TCP[***] (CERN 2002), and High Speed TCP[†††] (ICSI 2002). Other researchers have chosen to implement the transport protocol above UDP instead of IP to avoid the need for

[††] We used netem (http://www.linuxfoundation.org/collaborate/workgroups/networking/netem), a tool included in many Linux distributions.

[‡‡] Throughput of TCP and UDP was measured with iperf (http://code.google.com/p/iperf/); the throughput of UDT was measured with a tool provided together with the product.

[§§] See http://netlab.caltech.edu/FAST

[***] See http://datatag.web.cern.ch/datatag/papers/pfldnet2003-ctk.pdf

[†††] See http://www.icir.org/floyd/hstcp.html; this proposal has also been published by the IETF as RFC 3649.

kernel development and enable provision of a product that is operation system independent and can be easily deployed. An interesting product in this category is UDP based Data Transfer[‡‡‡] (UDT). As the performance figures published on the project website are impressive and the product provides an API that is very similar to the TCP Socket API and therefore supports easy exchange with TCP, we selected this product for further evaluation.[§§§]

We have measured the performance of the three protocols TCP, UDP, and UDT in various scenarios and analyzed the impact of configuration parameters, including send and receive buffers for varying network delays, Maximum Transmission Unit (MTU) sizes, and performance of congestion control algorithms.

### A. Send and Receive Buffers

It is well known that the theoretically achievable throughput using TCP is bounded by the Bandwidth-Delay Product (BDP), i.e. the product of a data link's capacity (in bits per second) and its end-to-end delay (in seconds). The result, an amount of data measured in bits (or bytes), is equivalent to the maximum amount of data on the network circuit at any given time, i.e. data that has been transmitted but not yet received. When the end-to-end delay is replaced by the Round Trip Time (RTT) the BDP is equivalent to the data that have been transmitted but not yet acknowledged. The TCP window size (configured as the TCP receive buffer) must be as least as large as the BDP to fully exploit the available bandwidth. Table 1 displays the BDP values for a number of data rates and RTT values. As can be seen, the TCP receive buffer must be at least 75 MB to fully utilize the 1 GBps bandwidth. The dependency of the achievable throughput on the configuration of the window size was very clearly seen in the measurement results. As the BDP describes a fundamental characteristic of window based protocols it is not surprising that this dependency can be seen for similar protocols such as UDT as well.

**Table 1. Bandwidth-Delay Products**

| | 10 Mbps | 100 Mbps | 1000 Mbps |
|---|---|---|---|
| 0.1 ms | 0.125 KB | 1.25 KB | 125 KB |
| 100 ms | 125 KB | 1,250 KB | 12,500 KB |
| 400 ms | 500 KB | 5,000 KB | 50,000 KB |
| 600 ms | 750 KB | 7,500 KB | 75,000 KB |

As the TCP has to store all transmitted data until they have been acknowledged the send buffer has significant impact on the throughput at high data rates as well. Figure 3 shows the maximum data rates measured for TCP for various RTT and send buffer sizes with the TCP window size (receive buffer) set to 75 MB. Figure 4 shows the same measurements for UDT. However for UDT there are two buffers – the UDP buffer and the UDT buffer: for the measurements the UDP buffer was set to the best value determined experimentally, which was 256 KB. As can be seen from these figures, TCP provides significantly better results than UDT at high RTT if sufficient buffers are configured. It must be stressed, however, that these measurements have been taken in a lab environment with zero packet loss. The impact of packet loss is briefly addressed later in this paper.

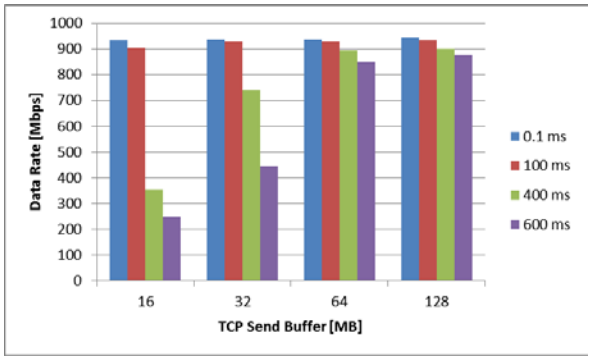


**Figure 4. TCP throughput depending on RTT and send buffer size.** *(TCP window size =75 MB)*

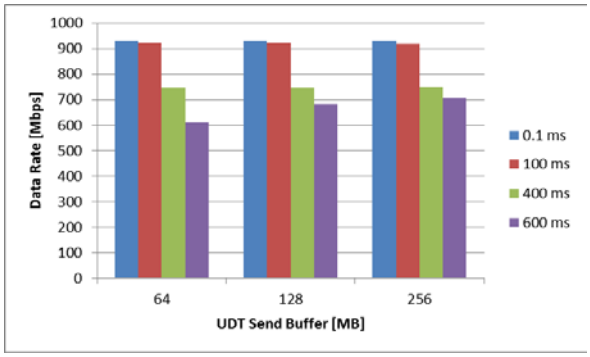


**Figure 3. UDT throughput depending on RTT and send buffer size.**

For TCP we have experimented with the "buffer auto-tuning" feature available on more recent Linux systems with good results. With this feature it is possible to define the maximum size for the buffers and let TCP determine

[‡‡‡] See http://udt.sourceforge.net

[§§§] Evaluations of transport protocols for high bandwidth / high latency networks have been performed at Stanford University in 2003 (http://www-iepm.slac.stanford.edu/monitoring/bulk/fast/) and by the Oak Ridge National Laboratory in 2004 (www.ornl.gov/~webworks/cppr/y2001/rpt/121150.pdf)

the size needed to achieve the best possible data rates on a given connection. The results with programmatic configuration of the buffers for each individual connection have not been better.

### B. Transmission Unit Sizes

With the introduction of Jumbo Frames on the Ethernet it is now possible to increase the size of the Maximum Transmission Unit (MTU) which was previously fixed to 1500 bytes up to 9000 bytes. For the transfer of large data volumes increasing the frame size had a significant effect for all analyzed protocols as can be seen in Figure 5.

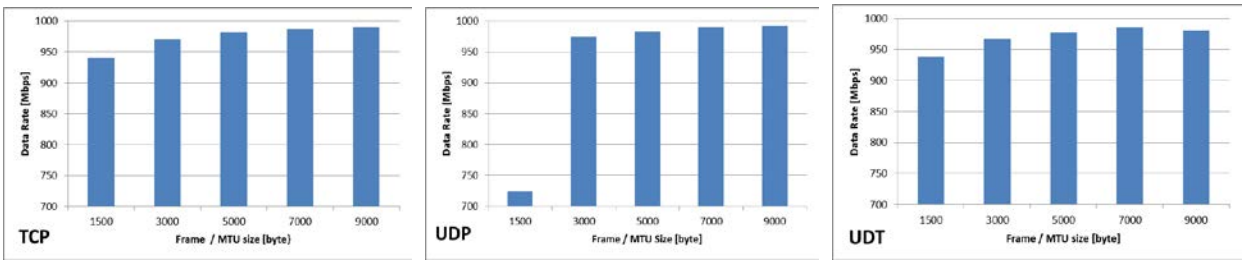


**Figure 5. Maximum data rates for various Ethernet frame sizes for TCP, UDP, and UDT (RTT < 0.1 ms)**

The reason for the poor throughput of UDP with the standard 1500 byte frame (724 Mbps) is that the iperf tool generates and transmits data at the maximum possible speed which causes a very high packet loss for UDP. When generating data at a rate of approximately 1 Gbps using the frame generator of the test platform, the data rate increased to 950 Mbps for this configuration. Nevertheless the packet loss rate was significant also in that case; in our tests we have always observed packet loss for UDP above rates of 700 Mbps even for local point-to-point connections without additional activities on the sending and receiving machines.

In the nearer future increased MTU sizes can probably only be exploited for local interfaces as the effective MTU size is determined by the smallest MTU size on the complete transmission path across a network and use of jumbo frames is not widely spread today.

### C. Congestion Control

If the TCP receives several acknowledgements for the same data or the retransmission timer expires it assumes that packets have been lost due to network congestion. Many algorithms have been developed to deal with this situation all aiming at the best balance between transmission rate and congestion avoidance. We have measured the maximum average data rates for random data loss rates or 0.1% and 1% induced by the network emulator. Table 2 reports the results for:

- RENO – the algorithm used by the Linux kernel until version 2.6.8;
- CUBIC – the default algorithm used by the Linux kernel as of version 2.6.19;
- The UDT protocol.

**Table 2. Throughput for RENO, CUBIC, and UDT versus packet loss rate.** *The figures present the average data rate in Mbps sampled over extended periods*

| | 0% | 0.1% | 1% |
|---|---|---|---|
| **TCP RENO** | 826.250 | 0.641 | 0.164 |
| **TCP CUBIC** | 925.212 | 3.065 | 0.528 |
| **UDT** | 950.762 | 107.931 | 25.485 |

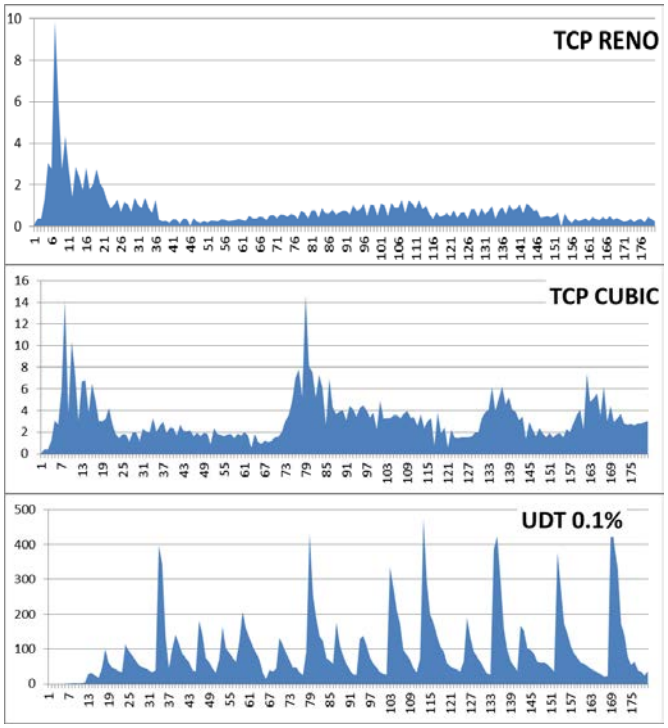


**Figure 6. Congestion control performance at 01% packet loss for TCP RENO, TCP CUBIC, and UDT.** *The data rate in Mbps was sampled every second over 3 minutes, the RTT was < 0.1 ms. Note that the scale of the vertical axis differs for the three charts.*

Figure 6 shows the data rates sampled every second over a period of three minutes, illustrating the effects of the specific algorithms.

While CUBIC provides significantly better performance than RENO, UDT outperformed all TCP congestion control algorithms we analyzed. It must be stressed however that we have taken the measurements with randomly distributed packet loss which is probably not very realistic scenario for a real network. In addition, an average packet loss rate of 0.1% may be too pessimistic for connections that are not based on the open internet. The figures presented should therefore be considered indicative only. Ref 1 and Ref 2 report on field tests on wide area networks, which resulted in much better figures both for UDT and TCP, e.g. 940 Mbps for UDT and 127 Mbps for TCP on a 1 Gbps connection between Chicago and Amsterdam.

## IV. SLE Return Transfer Services

We have further analyzed the achievable throughput of the SLE Return Services Return All Frames[3] (RAF) and Return Channel Frames[4] (RCF) using the test setup described in section II. We have first evaluated the latest release of the ESA SLE API package currently in operational use, which we refer to as the "Reference SLE API" in this paper. We have then performed an initial round of performance tuning and repeated the measurements done for the reference version. Finally we measured the cost for data encoding by the Abstract Syntax Notation One[8] (ASN.1) and Basic Encoding Rules[9] (BER) used by SLE services and analyzed the impact of CPU power on the achievable data rates. Unless noted differently in the following, all tests were performed with frames of 1115 octets and with online complete mode[****] as specified by the SLE Standards.

### A. Reference SLE API Version

*1. TCP Buffers*

As a first step we measured the dependency of the throughput on TCP buffers, using a single RAF service instance and 20 frames per transfer buffer[††††]. For this test, both the receive and send buffers on the service provider side and the service user side were set to the indicated values and the achievable throughput was measured for various network delays as shown in Figure 7. Measurements were run over three minutes and the average data rate determined. The fact that the average over the last minute shown on the right side of the figure is significantly better is due to the slow start algorithm which reduced the data rate for longer periods for higher RTT values.

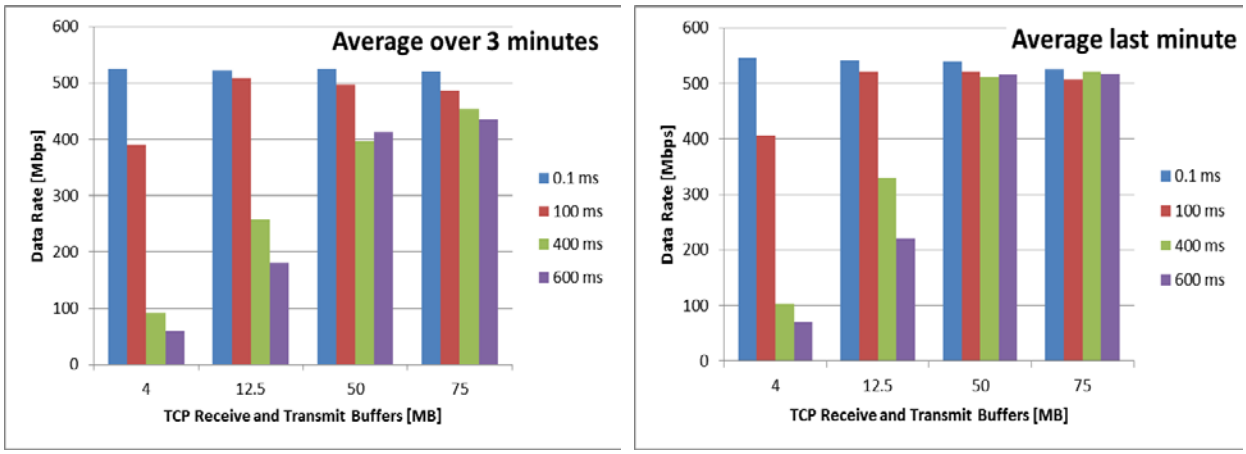


**Figure 7. SLE Throughput versus TCP send and receive buffers for various RTT values** *(RAF Transfer Buffer*

With buffers of 75 MB, the measured average data rate is between 507 and 526 Mbps in the last minute and thus around 400 Mbps lower than the one measured for pure TCP. In reality the difference is lower because the data rate recorded is the pure frame data rate i.e. excluding SLE annotations and protocol control information.

[****] In "complete online mode" frames are transmitted completely. If the SLE Service User is not able to receive the data at the rate transmitted or in case of network congestion, backpressure is built up and the SLE Service Provider is required to buffer the frames until they can be transmitted. In the test system the backpressure is extended to the frame generator which will only generate as many frames as can be transmitted.

[††††] The transfer buffer is a feature of the SLE Services by which several frames can be combined into a sinlge protocol data unit for the purpose of transfer.

*2. Transfer Buffer Size*

Figure 8 shows the impact of the RAF Transfer Buffer Size on the throughput at different values of RTT. Although the rates measured with 80 frames per buffer are mostly better than those for smaller buffers, there are measurements where smaller buffers give the same or even better values.

The very poor performance for transfer buffer sizes of 1 and 10 frames per buffer (0.2 and 2.2 Mbps respectively) is a good example of the large effect that local design choices can have for communication at high data rates. In the ESA SLE API package data transfer is split between the actual SLE service instance and a central communications server process, which handles the network interface. The interface between the service instance and the communications server uses a local TCP connection. The application protocol is based on a one-to-one handshake, i.e. a message will only be sent when reception of the previous message has been confirmed by a return message. In order to handle small data junks efficiently TCP uses the Nagle and the "delayed Ack" algorithms. The Nagle algorithm implies that transmission of a data segment that is smaller than the Maximum Segment Size (MSS) will be delayed some time expecting that it can be combined with further data; the "delayed Ack" algorithm implies that TCP ACK segments will be delayed for some time expecting they can be piggy-baked onto data messages. On an Ethernet interface MSS is typically 1460 byte but for local connections Linux sets MSS to 16 KB by default. Hence a transfer buffer of 10 frames is smaller than MSS and therefore the Nagle and "delayed Ack" algorithms are applied for the buffer and for the return message, slowing down the transfer rate.

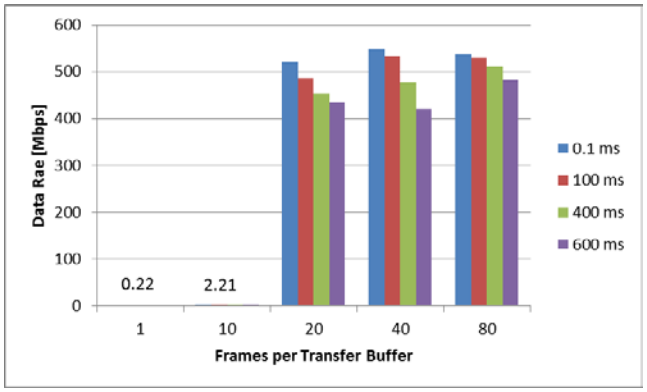


**Figure 8. Data rates depending on RAF Transfer Buffer sizes and RTT.***(TCP buffers all set to 75 MB)*

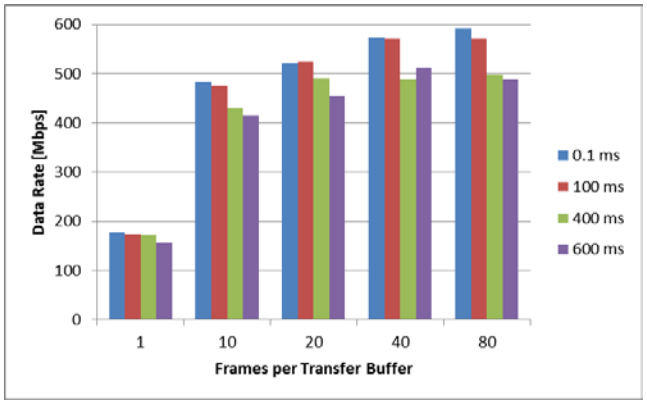


**Figure 9. Data rates depending on RAF Transfer Buffer sizes and RTT after tuning**

If the local MSS is reduced to 8 KB, the results measured for a transfer buffer of 10 frames are comparable for those measured for a transfer buffer of 20 frames. The same effect can be achieved by disabling the Nagle and "delayed Ack" algorithms.

We redesigned the interface between a service instance and the communication server: we removed the handshake between the two processes relying on TCP backpressure and introduced queues with high and low watermarks between local threads to optimize performance. The results presented in Figure 9 demonstrate that the measures applied have obviously removed the "small transfer buffer problem".

*3. Multi-Treading*

Figure 10 shows the effect of multi-thread processing for one to three RCF service instances. In one case all service instances have been processed in one thread and in the other case one thread has been used for each service instance. In case of a single service instance multiple treads give better results because processing of the API can be parallelized with processing in the application. The fact that multiple threads do not give better results in case of three service instances is due to limitations in the design of the test application, which we believe can be removed.

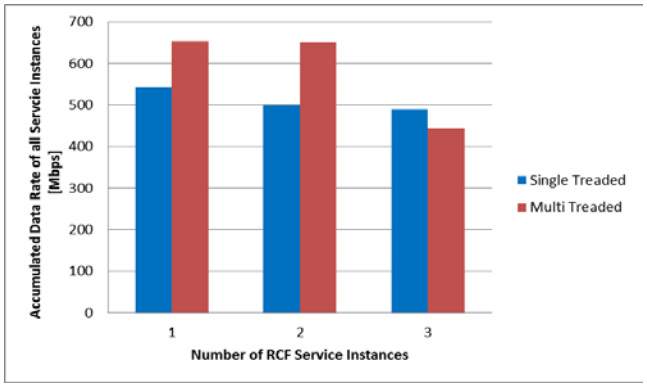


**Figure 10. Impact of multi-thread processing**

## B. SLE API Tuning

With 500 to 600 Mbps depending on the scenario, the throughput that can be achieved with the current release of the ESA SLE API package turned out to be better than we had expected at the beginning of the study. Nevertheless, as the example of the bottleneck between service instances and communication server discussed earlier shows, it is worth investigating whether the software can be further tuned for performance. We have therefore profiled the software and detected a number of options for improvements, including

- Cleanup of code carelessly wasting CPU cycles;
- Better decoupling of threads through queues of sufficient size with low and high watermarks;
- Partial redesign of the software to better exploit multi-core architectures[‡‡‡‡];
- Improving memory management to reduce the rate of memory allocation and de-allocation and copying of data units.

We performed an initial round of tuning removing the most obvious limitations and implementing improvements that could be easily done without significant re-design. The results are shown in Figure 9 and Figure 11 – the overall increase of the achievable data rate is in the order of 10%.

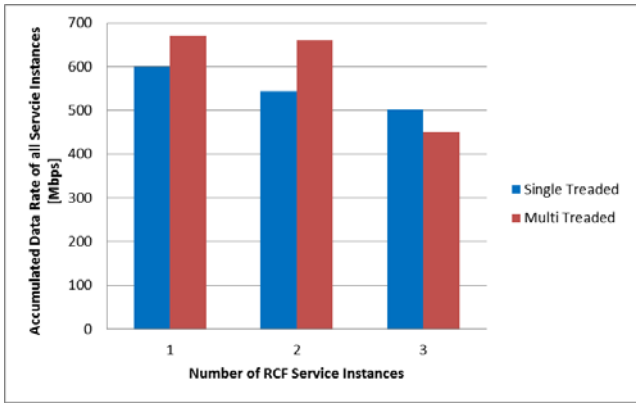


**Figure 11. Impact of multi-thread processing: optimized version**

## C. The Cost of Data Encoding

Because computers have architectures that present data in different ways, programming languages use different data formats and even compilers for the same language sometimes store data in incompatible ways it is necessary to a agree on a transfer syntax in which data are transmitted from system to another system. For this purpose, the SLE standards specify data formats using ASN.1[8] and the default transport mapping defined by CCSDS specifies the ASN.1 Basic Encoding Rules[9] for encoding of the data "on the wire". Obviously, encoding and decoding of data comes with a cost in terms of performance.

We have determined the percentage of time spent in the methods for encoding of one TRANSFER-DATA unit (i.e. a frame plus SLE annotations) by profiling, once before any application layer streamlining had been performed and once after completion of the performance test platform. The first measurement was 6.4% and the second was 26.3%, obviously because other parts of the software had been made faster. We have then removed the ASN.1 encoding and decoding and just transferred serialized C-structures taking advantage of the fact that both sides were running on the same hardware, and were implemented using the same language and complier. The resulting maximum data rates for selected configurations are presented in Figure 12 (all with RTT < 0.1 ms, TCP buffers set to 75 MB, single RAF service instance, 80 frames per transfer buffer). The gain in throughput obtained with the single thread is around 57 Mbps (9.5%) for the optimized API and 110 Mbps (20.3%) for the reference version. The corresponding increase for the multi-threaded version is 132 Mbps (19.7%) for the optimized API and 148 Mbps (22.6%) for the reference API.

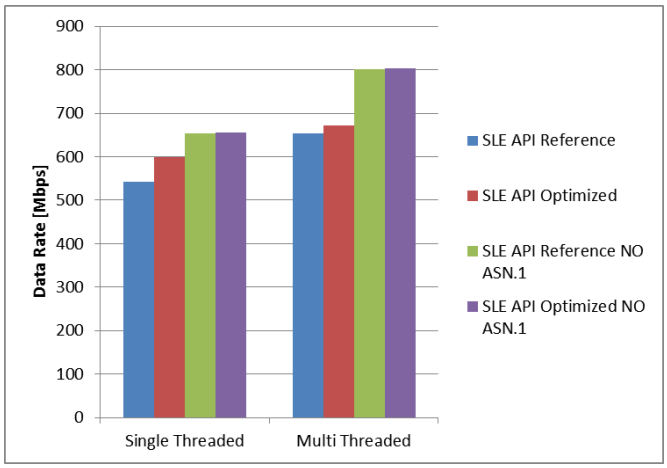


**Figure 12. Throughput without ASN.1 encoding**

[‡‡‡‡] This applies mainly to the application layer software; for the SLE API package we have not identified possible improvements in this area.

### D. Does more CPU Power Help?

In the tests with the single threaded version of the provider test application we observed that one of the cores was generally loaded to nearly 100% and therefore checked whether a more powerful CPU could increase throughput. We repeated the measurements presented in Figure 9 for the case of 0.1 ms RTT with an Intel Xeon Quad-Core 3.3 GHz CPU and 8 MB cache. As can be seen in Figure 13, the increase in throughput for the optimized version of the API was between 22% and 33% reaching a maximum rate of 749 Mbps with a transfer buffer of 80 frames. This clearly shows that the limiting factor is the local processing power or the local processing efficiency and not the network interface or the interactions on the wire.

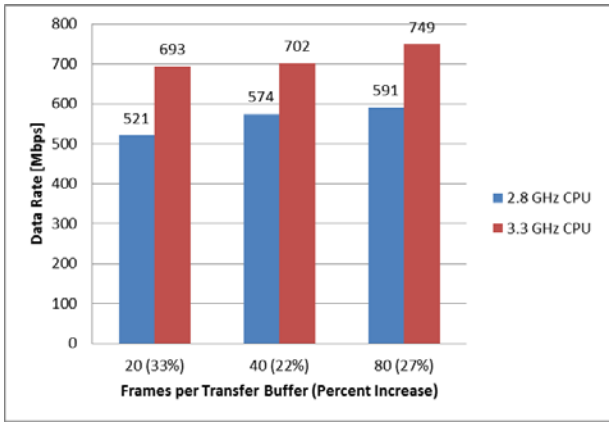


**Figure 14. Running the optimized SLE API on a more powerful CPU with the single threaded application**

## V. File Transfer

Beside SLE online return services we have analyzed the performance of file transfer protocols in the same test environment. The protocols and implementations used were:

- Pure FTPd[§§§§] included in the SLES11 distribution; and
- Tsunami UDP[*****].

Tsunami UDP is a file transfer protocol based on UDP and designed for connections with high bandwidth and high latency. It performs a file transfer by sectioning the file into numbered blocks of usually 32 KB size. Control information is exchanged over a low bandwidth TCP connection. The bulk data is transferred over UDP. Most of the protocol intelligence is worked into the client code - the server simply sends out all blocks, and resends blocks that the client requests. The client specifies nearly all parameters of the transfer, such as the requested file name, target data rate, block size, congestion behavior, etc., and controls which blocks are requested from the server and when these requests are sent. The file transfer is completed when all chunks have been acknowledged by the client.

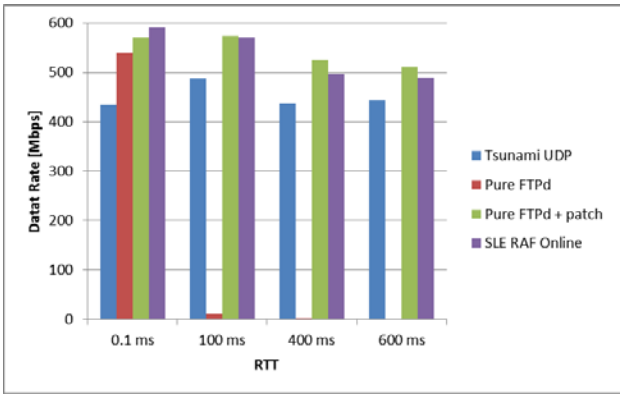


**Figure 13. Throughput of Tsunami UDP and FTP compared with SLE RAF online (optimized version)**

Achievable data rates were determined by transmitting a 10 GB file and measuring the time needed. Figure 14 shows the results and compares them with data rates measured for one RAF service instance with the optimized SLE API and the single threaded application using a Transfer Buffer of 80 frames (see Figure 9). The poor performance of Pure FTPd for RTT ≥ 100 ms was tracked down to the fact that the TCP buffer sizes were hard programmed and set to a far too small value. After removing this problem the throughput was good reaching 570 Mbps for RTT ≤ 0.1 ms and 510 Mbps for RTT = 600 ms (see Pure FTPd + patch in the figure). Tsunami-UDP maintained almost the same throughput of 430 to 490 Mbps independent of the RTT value; however in out measurements (without packet loss) the net throughput of Tsunami UDP was systematically lower than that of FTP. The data rate achieved for the SLE RAF service instance is more or less the same as the one for FTP.

[§§§§] See http://www.pureftpd.org/
[*****] See http://tsunami-udp.sourceforge.net/

## VI. Future Work

Performance tuning performed so far has focused on the obvious bottlenecks and on improvements that were easy to implement without much impact on the design. We have identified further options for which we feel it will be worth investigating the degree of improvement that can be achieved. These are in particular in the area of memory management and related design changes that increase the decoupling of threads and improve exploitation of the multi-core architecture of the hardware.

Further work will be dedicated to the evaluation configurations and strategies for efficient storage and retrieval of telemetry to and from persistent storage, as preliminary tests have indicated that this could be a bottleneck at the data rates being considered. Aspects to be considered include storage technology, disk configurations and strategies for concurrent storage and retrieval as well as catalogue construction and usage.

## VII. Conclusions

The measurements performed in the study on High Rate Telemetry Transfer Services have provided unexpected results in several cases and these have reassured us that one should not trust common belief when considering data communication performance. Often applicable constraints are not accurately considered or the capabilities of modern hardware or tuned operating systems are not taken into consideration. There is considerable work being performed especially on the mainstream protocols such as TCP and therefore what has been true 10 year ago does not need to be true today. There is no way around testing and measuring the real performance.

Based on the results obtained so far, we can draw the following conclusions.

- Modern TCP implementations can support high data rates also with long delays (RTT $\geq$ 600 ms) provided sufficiently large buffers are provided, the protocol is carefully configured, and the bandwidth is sustained with no or only a minimum of packet loss.
- For virtually loss free connections UDT does not provide advantages above TCP – the buffer requirements and the achievable throughput are similar. However, in connections with random packet loss rates of 0.1% or higher UDT clearly outperforms all TCP congestion control algorithms we analyzed.
- UDP can achieve high throughput but because it is unreliable by design, direct use as transport protocol without loss detection and recovery and sequence preservation at application level is not an option at least not for transfer via a wide area network.
- All protocols might benefit from increased MTU sizes, but use of this feature would require that larger frame sizes are supported throughout the complete transmission path.
- At the data rates considered, the design and implementation of the applications exchanging data is critical for performance. Care must be taken that the software design exploits modern multi-core architectures; repetitive profiling and performance tuning pays off.
- Once the software is well-tuned, use of higher CPU power can bring additional performance.

For the basic questions to be answered by the study we can so far conclude the following:

- SLE Return Transfer Services based on TCP can meet all known requirements of scientific missions for the foreseeable future provided sufficient sustained bandwidth with low congestion (packet loss). For network connections with lower quality use of UDT instead of TCP as SLE transport might be an option.
- As far as the throughput is concerned the data rates of Earth Observation (EO) missions currently in phase C/D could be supported on virtually loss-free connections with 1 Gbps bandwidth but with little margin. However, with the current operations concepts for EO payload data, use of SLE services would probably only make sense for ground station internal interfaces.

## Appendix A
## Acronym List

| | |
|---|---|
| ASN.1 | Abstract Syntax Notation One |
| BDP | Bandwidth Delay Product |
| BER | Basic Encoding Rules |
| CCSDS | Consultative Committee for Space Data Systems |
| CSTS | Cross Support Transfer Services |
| GB | Giga Byte |
| Gbps | Giga bit per second |
| IP | Internet Protocol |
| ISP1 | Internet SLE Protocol One |
| MB | Mega Byte |
| Mbps | Mega bit per second |
| MSS | Maximum Segment Size |
| MTU | Maximum Transmission Unit |
| RAF | Return All Frames (SLE Service) |
| RAM | Random Access Memory |
| RCF | Return Channel Frames (SLE Service) |
| RTT | Round Trip Time |
| SLE | Space Link Extension |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| UDT | UDP based Data Transfer |
| WAN | Wide Area Network |

## References


[1]Yunhong Gu, Xinwei Hong, and Robert L. Grossman, "Experiences in Design and Implementation of a High Performance Transport Protocol", SC 2004, Nov 6 - 12, Pittsburgh, PA, USA.

[2]Yunhong Gu, Xinwei Hong and Robert L. Grossman, "An Analysis of AIMD Algorithms with Decreasing Increases", First Workshop on Networks for Grid Applications (Gridnets 2004), Oct. 29, San Jose, CA, USA.

[3]"Space Link Extension – Return All Frames Service Specification", Recommended Standard, CCSDS 911.1-B-3, Blue Book, Washington DC, January 2010

[4]"Space Link Extension – Return Channel Frames Service Specification", Recommended Standard, CCSDS 911.2-B-2, Blue Book, Washington DC, January 2010

[5]"Space Link Extension—Application Program Interface for Transfer Services—Summary of Concept and Rationale", Informational Report, CCSDS 914.1-G-1, Green Book, Washington DC, January 2006

[6]"ESA SLE API Package – Reference Manual", SL ANS RF 0001, Issue 4.8, October 2009

[7]"Space Link Extension—Internet Protocol for Transfer Services", Recommended Standard, CCSDS 913.1-B-1, Blue Book, Washington DC, December 2008

[8]"Information Technology—Abstract Syntax Notation One (ASN.1): Specification of Basic Notation", International Standard, ISO/IEC 8824-1:2002. 3rd ed. Geneva: ISO, 2002.

[9]"Information Technology—ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", International Standard, ISO/IEC 8825-1:2002. 3rd ed. Geneva: ISO, 2002.