Universiteit van Amsterdam

SNE
System and Network Engineering

# Security enhancement of GridFTP: Description and Measurements[1]

*Razvan Coca*
*razvan.coca@uva.nl*

*April 7, 2011*

**Abstract**
*In this paper is presented a modified version of gsiftp protocol which takes into account the client's host properties and enforces encryption on data connection. An implementation based on a modified globus toolkit is described and several types of throughput measurements are discussed. This document is intended as a multiple vehicle: software requirements document, software specification/design document, throughput measurements document.*

[1]This work has been performed in the framework of the Dutch program VL-e: Virtual Laboratory for e-Science

SNE technical report SNE-UVA-2011-01
`http://www.science.uva.nl/research/sne/reports/`

## Introduction

Several authors describe different security models for distributed storage within grid, which is common in the case of medical data [8].

Using several services, like CryptoSRM-a storage manager, Hydra Key Store, [8] describes a security protocol that enables safe data and meta-data storage. Paper [25] is concerned with scalability and fault tolerance of distributed filesystem and acknowledges the need for a security model in this environment. Paper [26] proposes the use of a XACML access scheme based on user properties using multiple services for access mapping.

Examining the legal requirements for medical data processing, [18] describes a security architecture based on a trusted storage broker that would allow, through specific protocol changes, to assign responsibility on data processing. We will discuss here the data transport part involved in [18].

**GridFtp** server/client pair is one of them and it extends the FTP protocol, in several steps, both with efficiency-oriented extensions parallel-striping mode transfer (relieving the load on the server) [4], and security-oriented extensions, mainly for control connection [17]. Another data-transport technique is the **global access to secondary storage** - type of data provider which, for instance, implements the HTTP family of protocols.

We chose Globus toolkit to implement the changes. In the following we will concern ourselves with the security requirements, changes in gridftp protocol, client and server design of software changes in the toolkit, and describe how the implementation fulfills the requirements. We then focus on performance measurements and describe in the end the final figures. Several other considerations become appearent from this work, and we sketch them, briefly.

## Requirements

The paper [18] describes several requirements for securing data access and usage. They span from binding job to proxy certificate and host certificate to Access Control List - based access, but not all of them have the same impact (or type of impact) on performance.

We will list them explicitly here and point out which ones we intend to satisfy and which not:

1. secure binding between jobs and proxy certificate [we will not change the job submission procedure in any way]
2. data owner enforced ACL/Host properties of computing element [we partially touch this, it will have a limited impact ]
3. microcontracts
   (a) for host-computing element: Host Property List/Remote Host Property List comparison and microcontracting [not adressed]
   (b) for all authorization decisions [some adressed here]
   (c) for accessing data [adressed here]
4. non-repudiation of transaction records [discussed here]

Requirement (2) is pointing to the host accessing the data has to provide a signed contract pertaining to the execution environment. Several other dynamical parameters of the job execution might be on interest (like memory constraints or time constraints), but we will limit ourselves to an *unspecified* contract for execution environment.

Requirement (3) is addressing the issue of data user being responsible for data access and is be accomplished in two co-operating ways: by protocol modifications and a supplementary logging service. [18] describes how this is solved by transitivity of trust from TSRB.

Requirement (4) is a strong one, since any sequential collection of access records can probably be repudiated unless the data user signs for the whole session. We will fulfill here only a minimal part of this requirement, since the main purpose of this text is to assess performance impact of added security.

## Functional and Design Specifications

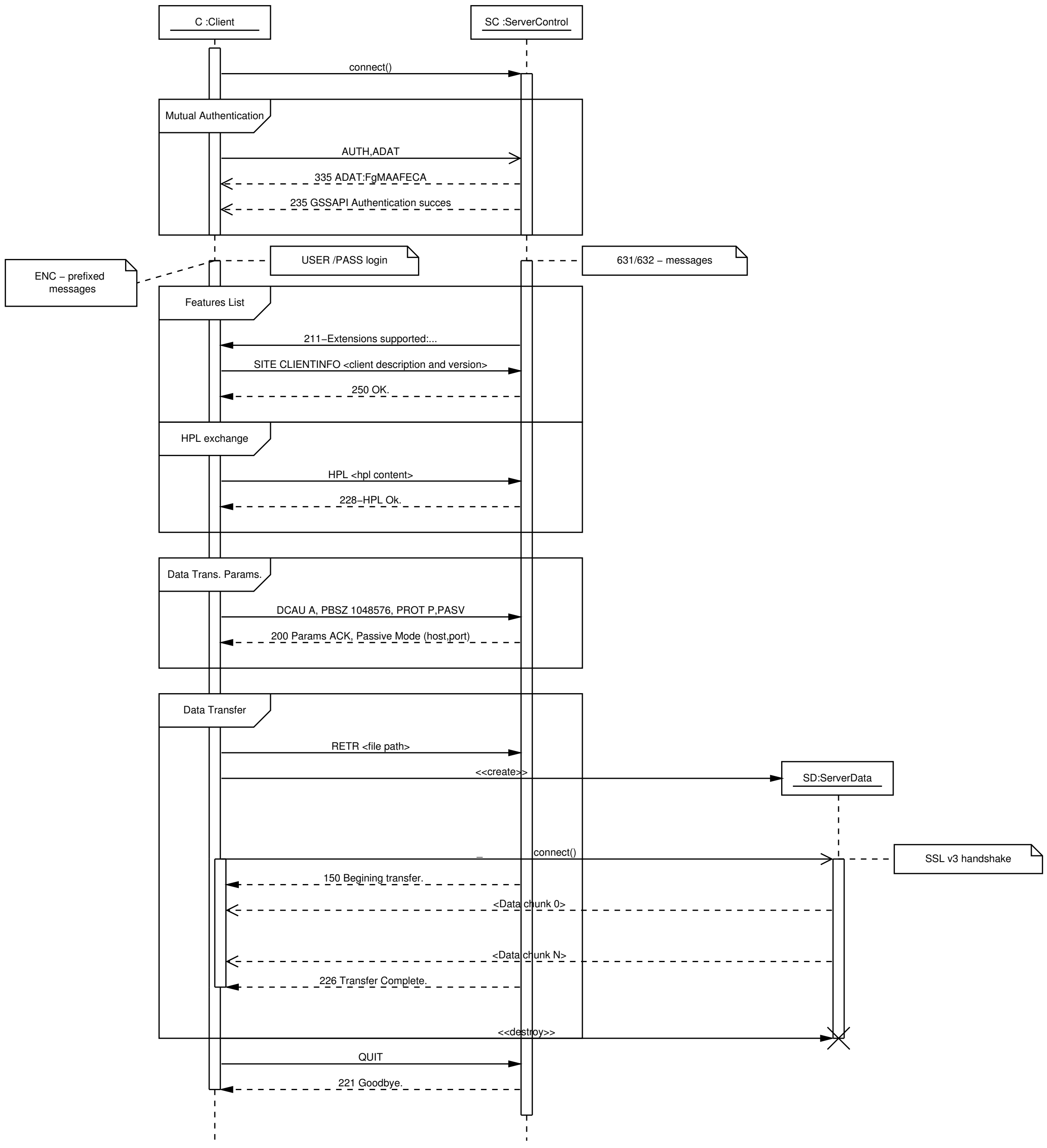


Figure 1: Modified gsiftp sequence diagram in a happy-case scenario. A particular flow of the original gsiftp behavior can be still observed if we ignore the HPL part.

GridFTP is based on several RFCs and extensions and provides data transfer service within grid [2, 3, 4, 5]. We will try to keep backward compatibility, as the common implementations of gridftp server do.

As part of the protocol change, we pre-pend an **s** from “secure” to URL string, transforming it into “sgsiftp://fqhn:port” syntax, which will allow client-side specification of the protocol.

The initial part of FTP protocol requires the client to authenticate (be it anonymous) to the server [2] through AUTH/ADAT message exchange. Providing additional information about the execution environment allows the data owner to assess the safety of data usage context, and thus, to reject the request on insecure machines. This requires an additional type of message, and has additional dependencies, in feature listing and access control. We call this message (as in [18]) **HPL**, standing for host property list. It adds an additional complementary layer of access control to data handling, thus fulfilling requirement (2).

As specified by [5], over the control channel, a key-exchange phase and mutual authentication takes place, leading to a sequence of messages exchanged, encapsulated as FTP Prefix+/SSL/Base64/TCP/IP, according to the standard(s), SSL.

The specifications above satisfy requirements (2), due to the HPL message, (3) due to mutual authentication during TLS handshake and partially (4) - see Figure 1

The sequence beginning with AUTH GSSAPI and ending with the reply “235 GSSAPI Auth success” outlines the credential exchange and mutual authentication phase and establishes a security association over which the following control messages are to be exchanged.

Depending on the authentication mode, a user-password authentication takes place next, which can be replaced by a grid identity.

Feature enumeration takes place at client request, listing among supported features HPL exchange.

As soon as connection parameters are “negotiated” and HPL accepted, an encrypted SSL/TCP data connection is established, over which data is transferred.

Deviation from this happy case scenario in the “HPL-exchange” box in Figure 1, produces 500-prefixed error message(s), which lets peer know the feature(s) are not supported. This satisfies an implicit requirement of backward compatibility. It could also control access to the filesystem based on the HPL.

**Logging:** On the control connection, logger service records the encrypted and plaintext of the messages exchanged. Besides the fact that message signatures are verified upon receiving, transcribing the transactions allows them to be verified later.

Base 64 encoding is used for encrypted messages. For the data connection, a similar mechanism is employed, except that only a HMAC of the data buffer will be stored, again in base64 encoded form.

**Authentication:** TLS handshake takes place with certificate request, so that both parties are authenticated and the identities of the parties are verified. Once the security association is established, the communication takes place over the negotiated encrypted channel. Client certificate in effect is the grid proxy certificate.

As far as the encrypted channel is authenticated, transcribing the messages will provide grounds for non-repudiation.

**Control channel:** On the control channel, the exchange of messages follows RFC 2228, which specifies AUTH / ADAT pair for credential exchange, with support for mutual authentication.

**Data channel:** On the data channel a TLS handshake takes place per data buffer sent, which makes each transaction signed and accounted for.

**Auditing:** No auditing of transactions records is implemented so far, we will sketch a possible approach. A clearer picture on how strong the repudiation/secrecy requirements are fulfilled can be obtained by examining potential attacher/defender scenarios and their strategies both for system security and for repudiation. We start by noticing that neither the attacker, nor the defender of a secure (private) service have infinite means of action, so their respective action space is limited [26].

Whenever possible, the whole authentication scheme together with logging and microcontracting might be examined within a game theory context, and if there is a equilibrium solution, the whole strategy leading there can be regarded as a defense strategy. Such analysis, together with stability analysis might underline the relevant events for the desired purpose - be it intrusion detection or non-repudiation- and allow an optimal logging service/log filter to be designed, focused on relevant events. Of course, different type of actions (claims) come into play for (non)repudiation case from server attacks case.

## Applications Description

Client operations rely on a callback loop synchronized on a condition variable signaled inside each callback function. Some callback functions are protocol statefull and advance the FSM to fulfilling the request state(s). This structure allows abstracting the details of the protocol through a couple of high level functions (GET, PUT ... ).

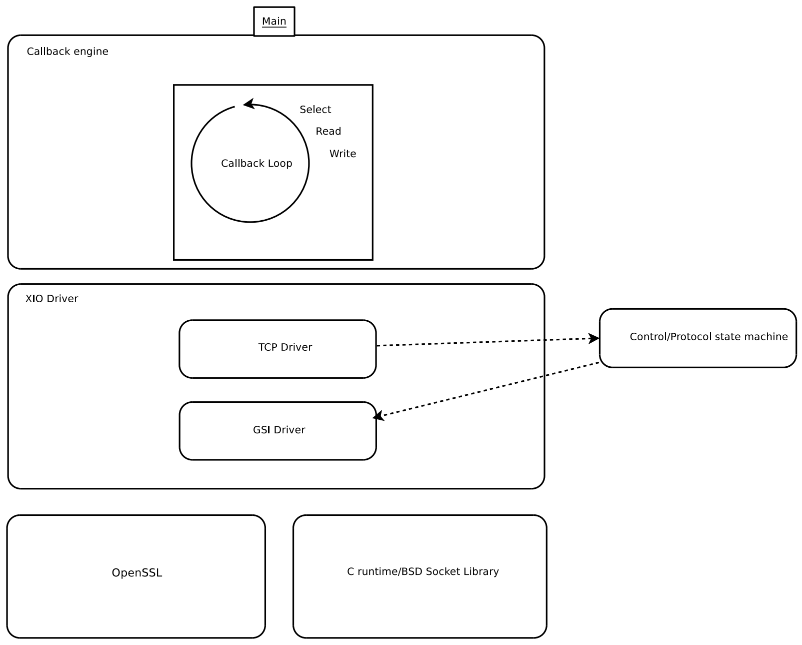


Figure 2: Client block diagram. IO operations are fulfilled by XIO abstraction layer. Client control library abstracts the protocol details.

Server structure is a little different, as it features two separate programming language “namespaces”, one of the “FTP server”, dealing with initial complicated phases of authentication, while the second, called “GFS” exposes a command-callback API, filesystem-like interface to the developer, and is only functional after the initial handshake phase ends successfully. The two namespaces have in common the connection and session handles with properties set in the first part of the protocol. As process model, server spawns a new server-process for each socket-connection accepted, all requests being satisfied within the boundaries of that process. As a structure, server tends to be simpler than the client, taking into account the FTP protocol state machine library.

**HPL exchange** On the client side, a trimmed version of the control connection state machine looks like in figure 2.

On the server side, two separate namespeces, namely the “server control” and “gfs” are present, handing over informations through connection/session handles.

As soon as client initiates a request, the GridFTP control library start advancing through the state machine, moving through authentication, data parameter setting and, finally data transfer.

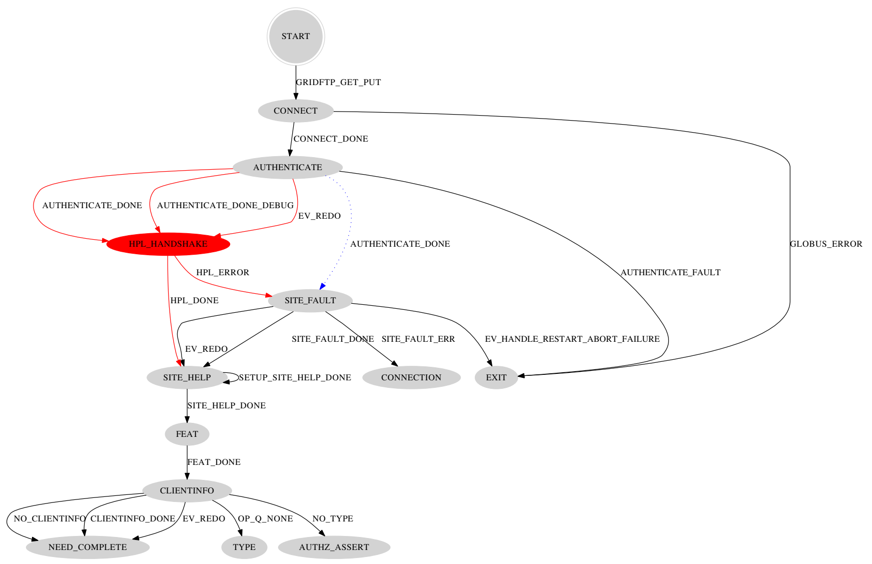


Figure 3: Client control connection state machine with changes. HPL - related changes are depicted in "red"

**Microcontracting** Since SSLv3 session in itself is to an extent non-repudiable, it is just a matter of transcribing enough data to have an implementation of microcontracts, and data is available in gss read/write callbacks for messages and through the get interface of GSI for keys.

We call the non-repudiable records of data transfers and actions micro-contracts. Micro-contracts on all exchanged messages are defined as:

$(message, encrypted\,signed\,message)$ -tuple for control connection messages and

$(data\,checksum, signature)$ - tuple for data connection,

where we call $message$ the plaintext ftp control message (stored for convenience) and $data\,checksum$ the cryptographic hash of the data portion of the message, which the receiver verifies when receiving.

The data connection micro-contracting tuples are base64 encoded too, leading to a plaintext log. This part is not implemented yet.

Since little or no additional operations are added for micro-contracting, the overhead produced is probably indiscernible on the final performance of the system.

**Logging** Logging is done by hooking a logging API within the context where both the encrypted and plaintext components of the exchanged messages are present. **gss_wrap** and **gss_unwrap** routines and their callers are the ones that provide the place.

**globus_gss_assist_init_sec_context** and **globus_gss_assist_accept_sec_context** provide the place where key data is accessible.

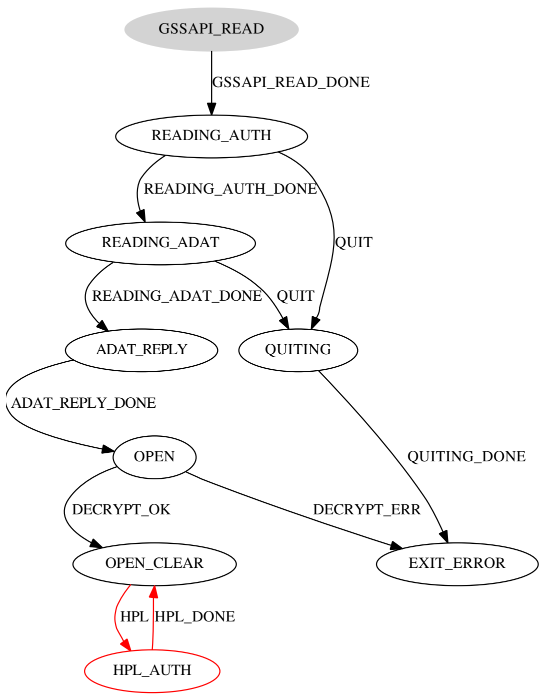


Figure 4: Server state machine. HPL is a FTP command here, just like any other transfer parameter setting. OPEN_CLEAR corresponds to the state of an authenticated client session, ready to engage in data exchange - one of GET/PUT/DEL/LIST commands - with server. In this context, HPL lies within "GFS" namespace.

## Profiling data

Profiling data can come into the picture to complete the design specifications and the measurements.

Client software was run through valgrind "virtual machine" using the tool callgrind. We attempt to include hints on the application structure together with the profiling data. The profiling data is in "Instructions read" unit, which we acknowledge to be an inadequate measure of timing: Figures 5,6 7 and 8.

Profiling data is sorted by total "percent" spent and is inclusive (root-close calls contain the callee's statistics) and thus, the display technique allows understanding both calltree structure and its typical use. On small portions, the profiling data might resemble a weighted call tree/stack dump, due to the inclusive counting, which allows display of both high level and low level call usage statistics. System calls instructions read (IR) usage is bar-plotted over the totals, allowing understanding the operations taking place.

In all graphs, main tends to be equivalent to callback_space_poll, and this is the typical structure of the application. SSL handshake plays a small role, something below 10% of the total execution time. Profiling data show a balanced call graph, with little difference between branches, suggesting good modularity and previous optimizations of the toolkit. System "times" are normed to their own maxima. Logarithmic plot avoids saturation around large numbers and a better resolution of middle-range values.

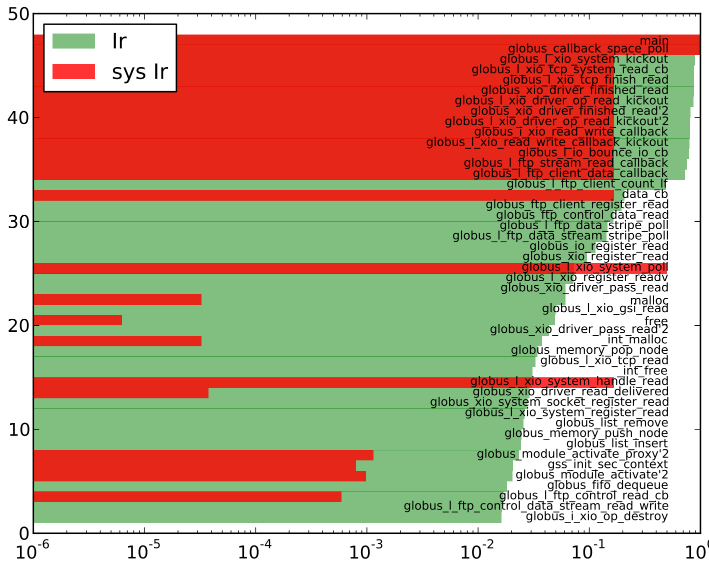


Figure 5: gsiftp profile statistics.Measuring unit is the percent of total instructions read. With green percent of instructions read by application, with red, percent of instructions read while executing in kernel space.

## Protocols

**Gsiftp:** blocked in the callback execution engine loop ("callback_space_poll"), spends most of its system time waiting in select while user time is approximately equally divided between different flavors of read callback, as shown in figure 5.

System read callback (globus_l_xio_tcp_system_read_cb) is concerned with recv operations in this case, which explains the large "time" spent in it and its descendants, so that on small sets, the order of the profile figures is also the oder of API calls.

Callbacks are one-time callbacks, so it is natural to have them re-register themselves and thus, appear in profiling figures.

globus_ftp_data_stream_stripe_poll is used for a single stripe, as striping mode is not enabled in these experiments.

globus_l_ftp_client_data_callback ends the call sub(graph) tree that begins with globus_l_xio_tcp_system_read_cb, eating most of its execution time.

An interesting amount of time (although still below 1%) is spent in memory operations, meaning that the toolkit might benefit from a memory pool.

gss_init_sec_context, the SSL handshake barely make it within the 0.1% of the time globus_xio_system_poll is basically a call to select and data_cb is a call to write

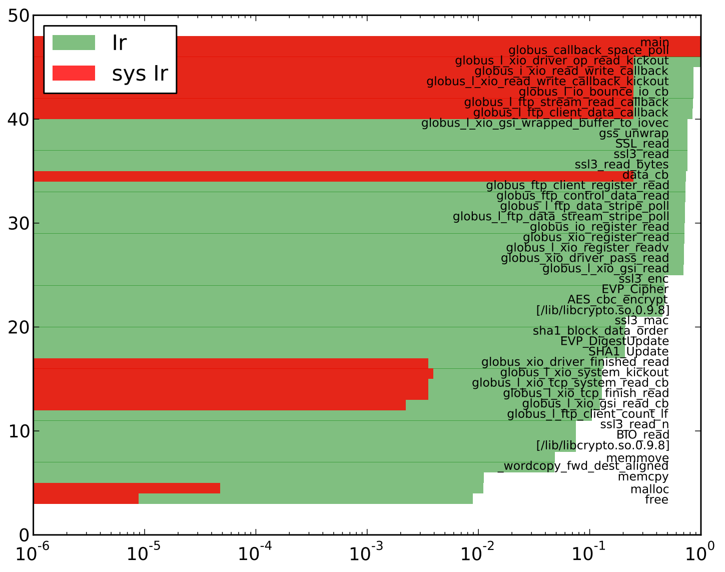


Figure 6: sgsiftp profile statistics. Measuring unit is the percent of total instructions read. With green percent of instructions read by application, with red, percent of instructions read while executing in kernel space.

**Sgsiftp:** callback space poll also plays a big role, but several other operations make it into the top as it can be observerd from Figure 6

globus_l_ftp_client_data_callback does the same as in the original gsiftp and takes the most time of the application and application system calls.

just below it, a new set of functions: gss_unwrap, SSL_read, ssl3_read_bytes, are the decryption part that raise to higher places, followed by data write data_cb

All openssl encrypt-type functions have flags indicating direction of operation, since are similar, regardless of direction.

Figures show top 50% contributors to usage statistics, both user and system space, as reported by valgrind runs, in instructions read, as a percentage.

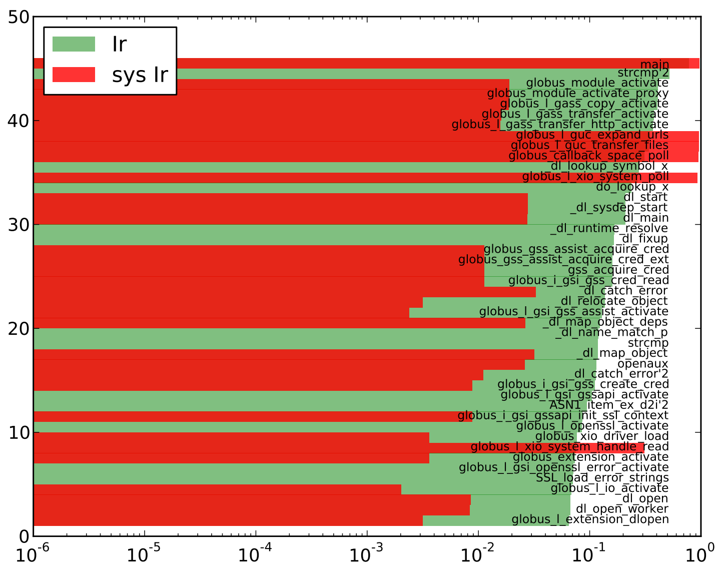


Figure 7: Http profiling data

**HTTP:** Almost the same application structure can be found in the secondary storage transport mechanism, with a big time (long select call time included) in globus_callback_ space_poll. Figures 7 and 8.

The rest of time spent in function calls is evenly distributed between a larger number of functions, such that their individual amount is small and probably there's very little to say. This might be a case where the best explanation of the code behavior is the code itself.

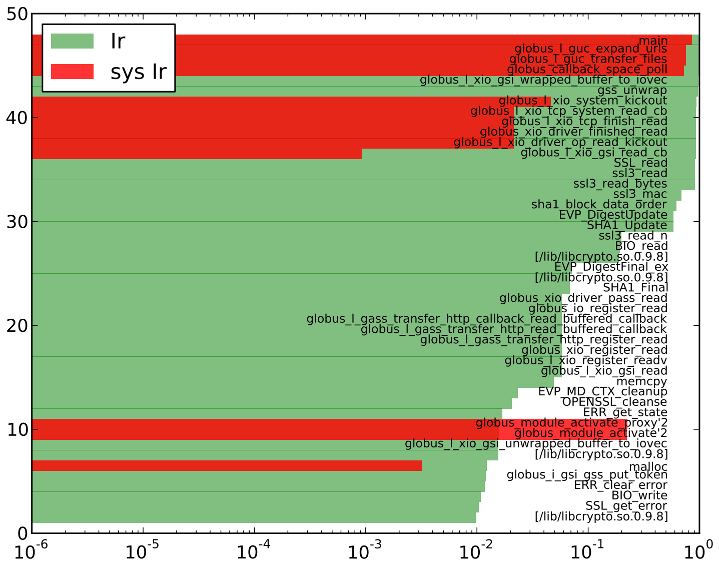


Figure 8: https profiling data

**HTTPS:** The common SSL-type of operations "emulating" globus_l_xio_gsi_wrapped_buffer_to_iovec, seem to take a lot of time, together with decryption and verifications in gss_unwrap.

## Measurements

Several transfer times have been measured in order to provide an estimate of the transfer rate and its variation under microcontracting and encryption:

- "Wall clock" transfer time
- Resource usage statistics as provided by kernel

Although "wall clock" time provides an estimates close to the user experience, the competing processes on the system produce noisy plots as seen from figure [9] and a "maximum transfer rate" could be inferred from that type of measurements, as data points tend to accumulate eventually at the small time(s) part of the graph.

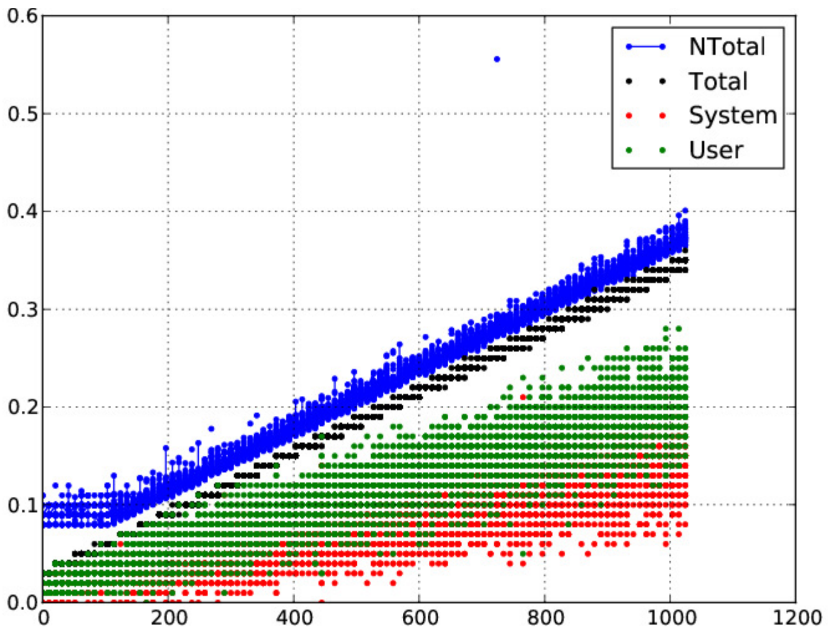


Figure 9: Illustrative plot of different measured transfer times for gsiftp protocol. On x-axis: is the file size in MB and on y-axis: transfer time in seconds. User and system times overlap and obscure each other, they are about the same, their sum is smooth: Total. Wall-clock data: NTotal- noisy total- tend to have a large spread, increasing with the size of the transferred file.

Resource usage statistics provided by kernel and the sum of kernel space-spent time and user space spent time is suitable for estimating a transfer rate. Measurements have been performed on Linux machine[s], running kernel Linux 2.6.32-25-generic SMP x86_64 flavor. The standard round-robin time-sharing policy was used. CPU used was - as reported by cpuid- "Intel(R) Core(TM) 2 Duo CPU E8500 @ 3.16GHz"

In order to exclude other noise sources and underline better the encryption contribution to the transfer time, localhost transfer was used. TCP stack is still used, but as far as networking is concerned, is equivalent to memory copy, thus any I/O on networking is absent. Moreover, influence of HDD I/O is limited, since operating system yields timeslice on long I/O wait operations. All these extra contributions omitted here could be added explicitly from separate measurements and using the notes formulated in the next section (see relations 1,2).

Cipher suite in all encrypted communications for these measurements is DHE-RSA-AES256-SHA1: Diffie–Hellman ephemeral, RSA authentication, AES 256 symmetric encryption, and SHA1 HMAC. Only GET operation was benchmarked throughout this paper and there's little reason to believe that other type of data transfer behaves differently.

## Resource Usage Data

Resource usage time measurement is basically execution time "as if" measured on the execution thread of the application. As discussed in previous section, resource usage data was taken as a measure of throughput, providing an accurate estimation of actual application throughput. It also allows idealizing application run and can provide a best-case scenario.

The sum user time+system time spent for transfer was regarded as an dependent variable, while the file size as independent variable. The dependent variable was considered linear in the independent variable, and the values fitted with a line by least squares method and the covariance matrix used to provide errors estimations in fitted parameters.

In order to have adequate pointers do discuss transfer behavior and performance, we will state explicitly the "underlying model" through the equation:

$$T = T_0 + \frac{Size}{Rate} \tag{1}$$

where:

$T$ - is the time it takes the application to transfer the file from the server to the client

$T_0$- is a constant component of this transfer time

$Size$ - is the size of the file to be transfered

$Rate$ - is the observable to be determined, the transfer rate.

Within these notations, $T_0$ will account for the time to spawn the client process, load dynamically linked modules, handshake the initial conversation with server, and any of the operations with little dependence on file size. The size-dependent $\frac{Size}{Rate}$ part would account for transfer and encryption and, in a less degree, hashing and signature verification.

### Scaling properties

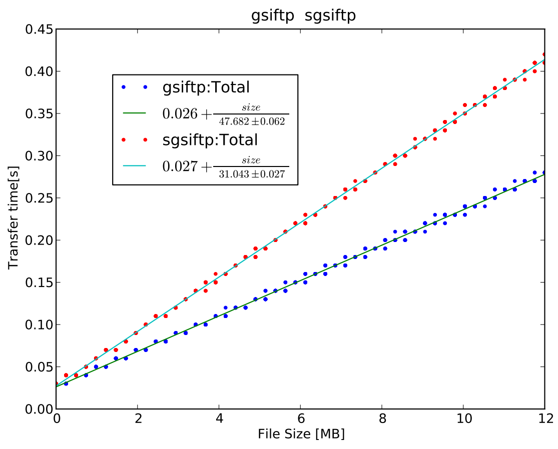
We note that the inverse of transfer rate is an additive parameter, since transfer time is an additive parameter. Percentages transcribe for transfer rates:

$$P = R_{encrypted} \left[ \frac{1}{R_{encrypted}} - \frac{1}{R_{plaintext}} \right] \tag{2}$$

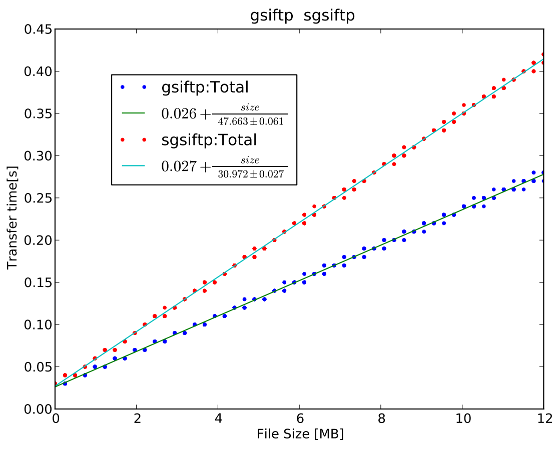
where $P$ is the percentage of transfer rate degradation (due to encryption), $R_{encrypted}$ is the transfer rate for encrypted connection, $R_{plaintext}$is the transfer rate for plaintext connection. We notice from relation 2 that changes in encrypted/plaintext rates due to the same cause will cause the "degradation of performance due to encryption" percentage to decrease.

Frequency scaling measurements as well as CPU core affinity setting for server and client were performed to investigate the scaling properties of data transfer.

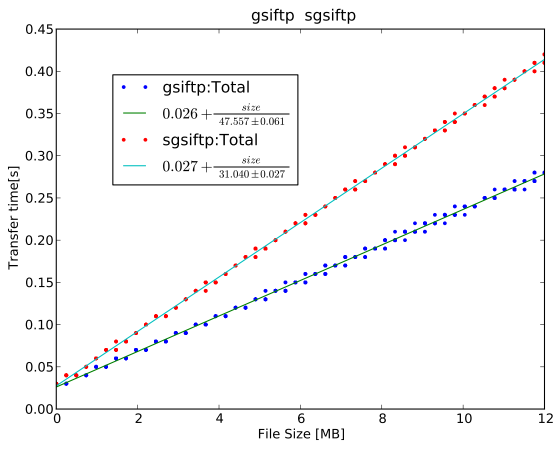
**Affinity setting** Since transfer is measured locally, an affinity study was employed to outline the effect of running on same/different CPU (core).



(a) 0x1,2.33GHz

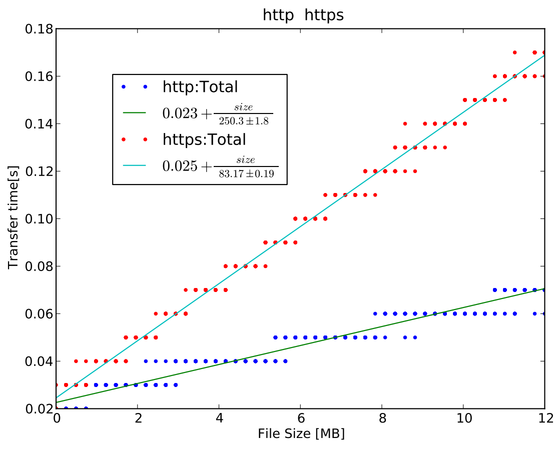


(b) 0x2,2.33GHz

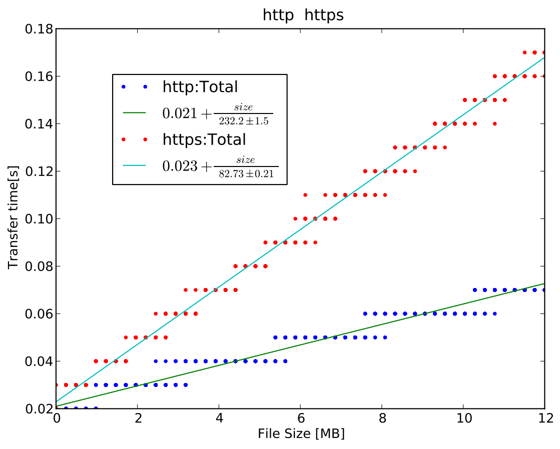


(c) 0x3,2.33GHz

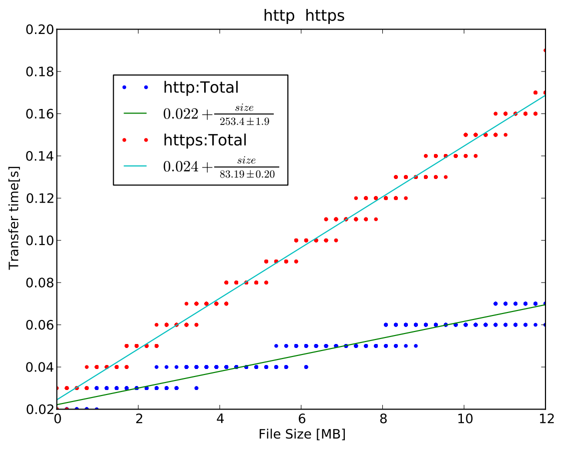
Figure 10: Affinity effect: gsiftp protocol. For illustration purposes only 2.33 GHz results were presented here.



(a) 0x1,2.33GHz



(b) 0x2,2.33GHz



(c) 0x3,2.33GHz

Figure 11: Affinity effect: http[s] protocol

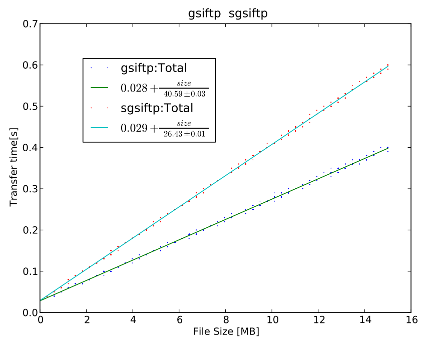
As it can be seen from figure 10, gsiftp family of protocols show the same performances, differing by less than 1% on different affinity masks, so affinity setting plays no role in this measurements. Even more, there's a slight improvement of transfer performance on no-affinity setting, which could be attributed to the OS scheduler.

Http family of protocols have been considered as a comparison, and, aside for their boosted performance in this benchmarking scenario, have the same type of behavior as gsiftp-based ones.

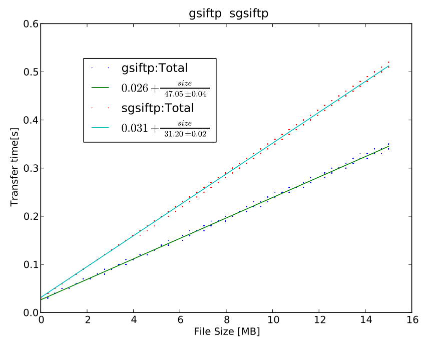
They differ within $0.5-8\%$ with clear degradation on collocation of client and server on the same CPU. Peak value is for same-core scheduling and is simultaneous with high CPU load. Slight improvement occur on no-affinity setting, which means the operating system scheduler does better than just separating the client and server cores.

The apparently boosted performance for http(s) might be related to localhost transfer, and we expect it to degrade in real-world conditions (Relation 1).

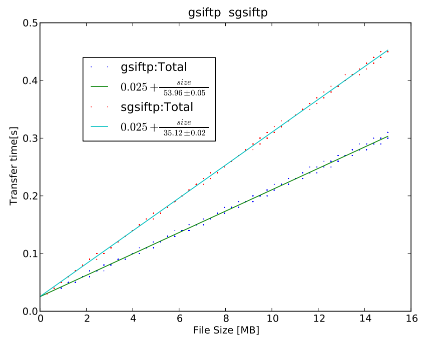
**Frequency scaling** One other setting that might provide insight into scaling properties is the cpu frequency scaling. Even though it provides only a small set for parameter variation, linear scaling with small deviation could be regarded as interesting.



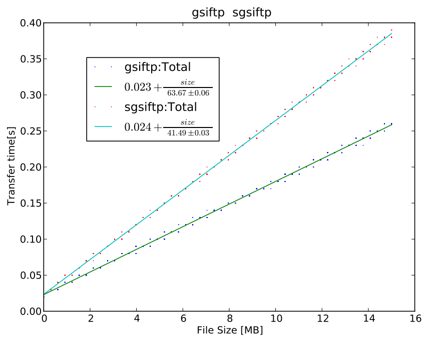
(a) 2.0 GHz



(b) 2.33 GHz

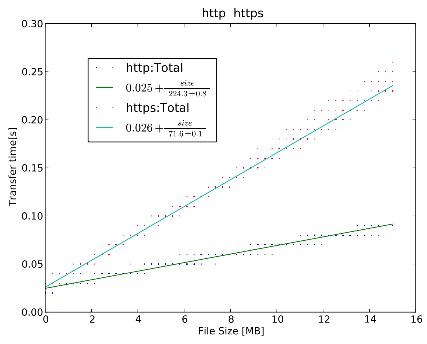


(c) 2.67 GHz

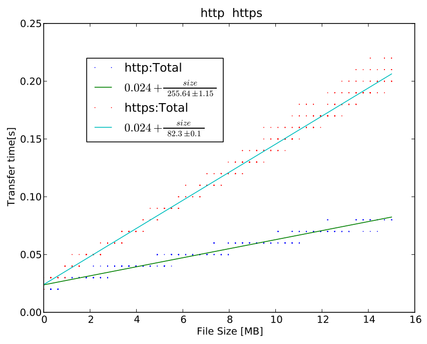


(d) 3.17 GHz

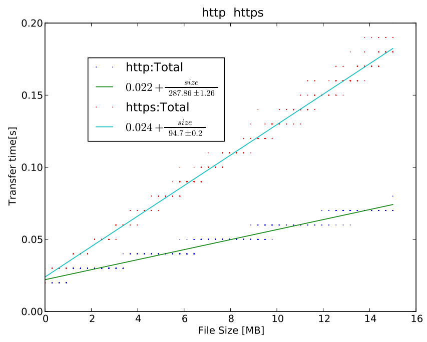
Figure 12: Cpu Frequency scaling effect gsiftp protocol. The computed transfer rate show the same ratio as the cpu frequencies ratio. In all cases performance degradation due to encryption/signing is around 35% in transfer rate.



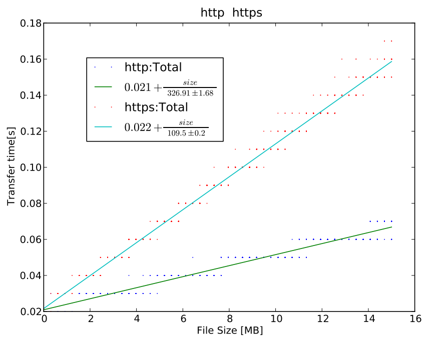
(a) 2.0GHz



(b) 2.33GHz



(c) 2.67GHz



(d) 3.17GHz

Figure 13: Cpu Frequency scaling effect on http protocol. Performance degradation is around 67% for encryption versus plaintext version. The performance enhancement tend to be a little lower for faster cpu freq. scaling 1.18 yields 1.14 factor of performance enhancement.

**Error treatment** Besides the least squares optimization procedure mentioned at the beginning of the section, other independent error estimation techniques are presented here.

Round-up errors play a significant role in resource usage statistics and "time slice" is accounted as user-space or kernel-space depending where the process spends most time.

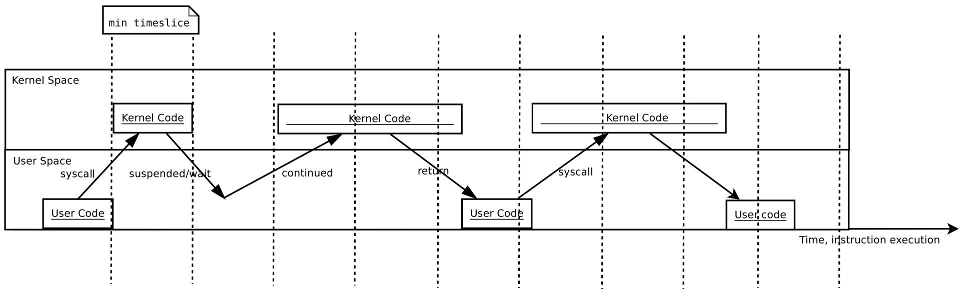
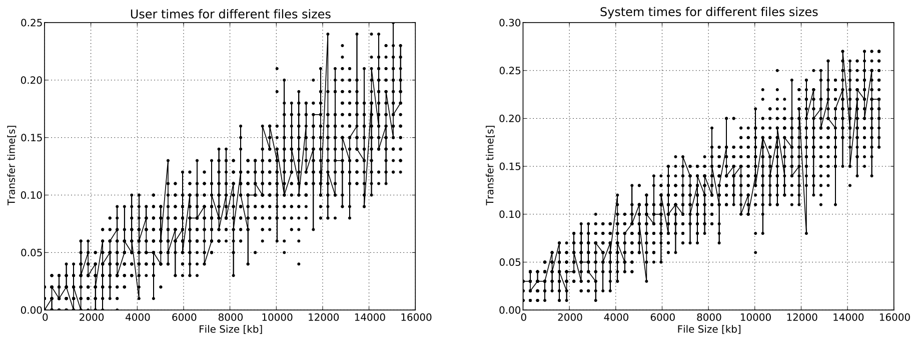


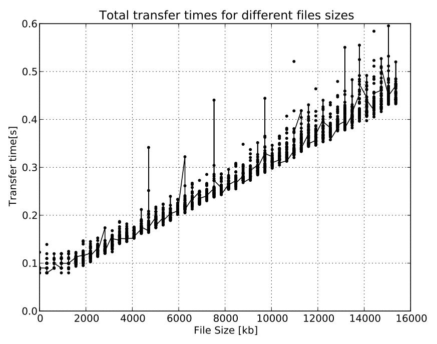
Figure 14: Effect of roundup errors on resource usage statistics. Task switching before timeslice end discards the statistics for the current space and increments statistics on the other “side”. The same effect can be observed from figure 18 and algorithm 1.



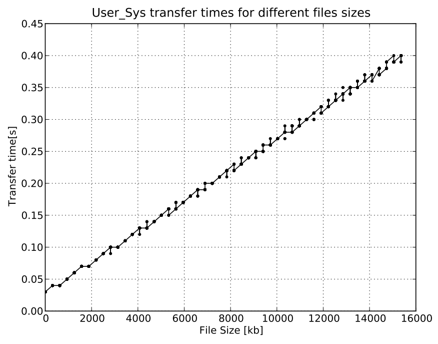
(a) Contribution of user time to transfer rate. gsiftp protocol

(b) Contribution of kernel-space time to transfer time. gsiftp protocol

Figure 15: Components of resource usage statistics. Data shows a large granularity of time-slice, around 10 ms

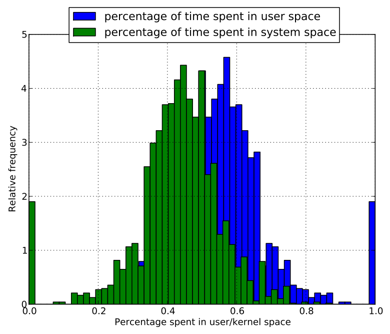


(a) Wall clock time. gsiftp protocol

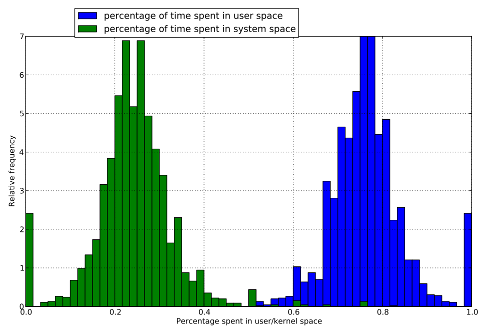


(b) Resource usage time. gsiftp protocol

Figure 16: Wall clock time and resource usage time.



(a) “plaintext”



(b) encrypted

Figure 17: Cumulative resource usage statistics histograms. Data from entire file range is split into percents spent on behalf of process in user and system spaces. Histograms are normed in the sense of probability density function. Peaks near 0% and 100% account for small file range spurious data. gsiftp protocol.

Since resource usage statistics show time-steps in plot, a time-resolution evaluation of resource usage was developed.

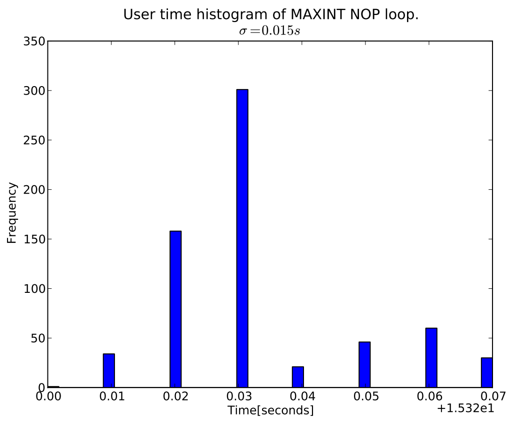


Figure 18: Histogram for timer resolution

**Algorithm 1** Time resolution estimation for resource usage

```
    0x08048424 <+0>:     push   %ebp
    0x08048425 <+1>:     mov    %esp,%ebp
End Call sequence ....

    0x08048427 <+3>:     and    $0xfffffff0,%esp
    0x0804842a <+6>:     sub    $0x60,%esp
=>  0x0804842d <+9>:     movl   $0x1,0x5c(%esp)
    0x08048435 <+17>:    jmp    0x804843d <main+25>
Begin For loop
    0x08048437 <+19>:    nop
: loop body
    0x08048438 <+20>:    addl   $0x1,0x5c(%esp): i++
    0x0804843d <+25>:    cmpl   $0x0,0x5c(%esp): loop condition
    0x08048442 <+30>:    jne    0x8048437 <main+19>: repeat
End For loop....

    0x08048444 <+32>:    lea    0x14(%esp),%eax
    0x08048448 <+36>:    mov    %eax,0x4(%esp)
    0x0804844c <+40>:    movl   $0x0,(%esp)
    0x08048453 <+47>:    call   0x8048344 <getrusage@plt>
```

A NOP loop run for MAXINT and printed usage statistics. Since no system call is done during NOP, we expect (almost) all of system times to be 0. Out of a total of 2600 runs, only 73 were non-zero. Resource usage time dispersion is $\sigma = 0.015s$, 15 miliseconds, natural, since counters would only account for timeslices during scheduled run of the process.

## Results

The main results obtained are condensed here for convenience. A degradation of performance in the case of gsiftp profocol of around 35% is noted with main figures of transfer rates being $30\,MB/s$ in the case of sgsiftp to $40\,MB/s$ for gsiftp. These numbers are close to the real case, since the resource usage data tend to accumulate close to the lower anvelope of the wall clock time.

Http family of protocol show a bigger performance degradation of around 70% wit huge transfer rates $70MB/s$ for https to $200MB/s$ for http. In the case of http these are ideal case numbers, since wall clock data and resource usage data show a large gap.

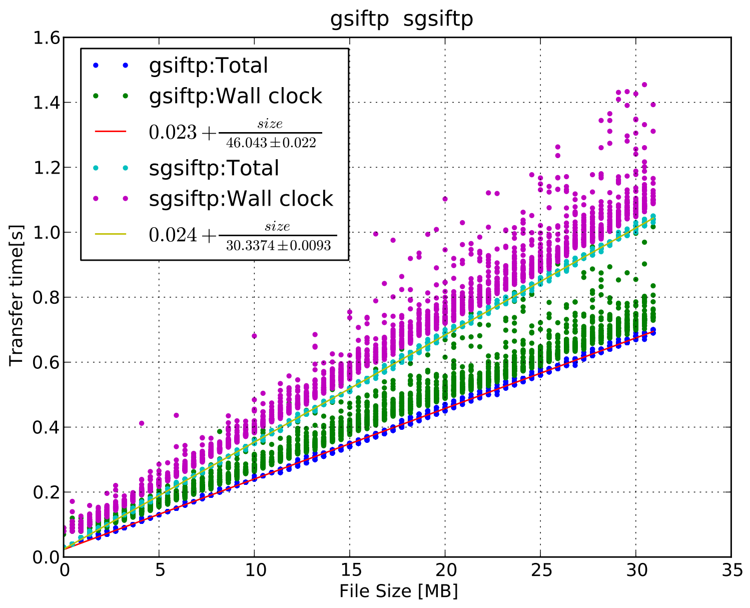


Figure 19: gsiftp family of protocols. Wall clock data together with resource usage data.

## Wall Clock Timing Statistics

Wall clock data deserves special attention, since they point to the final user experience and, in the same time, have a noisy behavior, unfit to define a throughput rate.

### Raw Data

Raw wall clock data account for access to the system and "network" resources of the application, summing all delays / execution times.

Optimized servers place a considerable load on client side, making resource usage data a lower limit for the transfer time, as is the case of "gsiftp family" paired with gridftp server, while the http family paired with gass secondary storage transport share the load between client and server which makes the wall clock time far apart from resource usage time. However resource usage time still gives an indication of the ideal case where a data reservoir answers data requests from ftp client.

### Statistical Properties

The nature of wall clock data is stochastic due to variable execution times of API calls, or rather due to occurrence of random racing events, suggesting

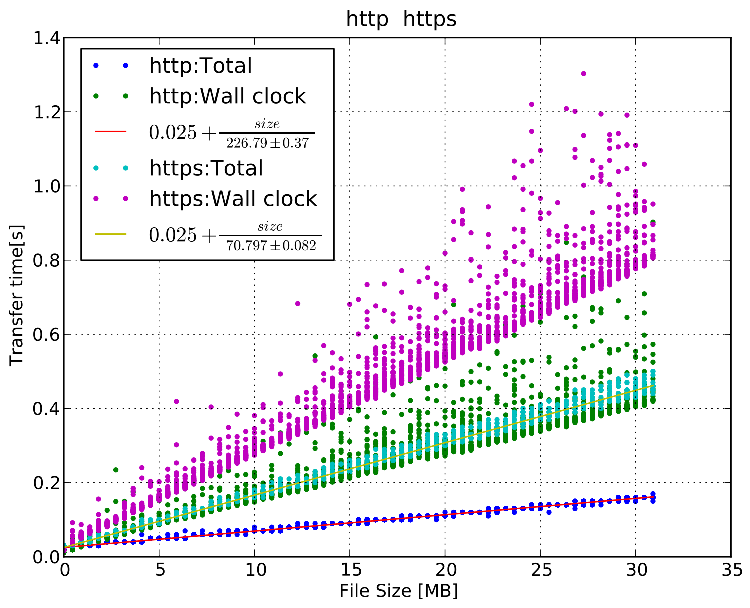


Figure 20: http family of protocols. wall clock data and resource usage data

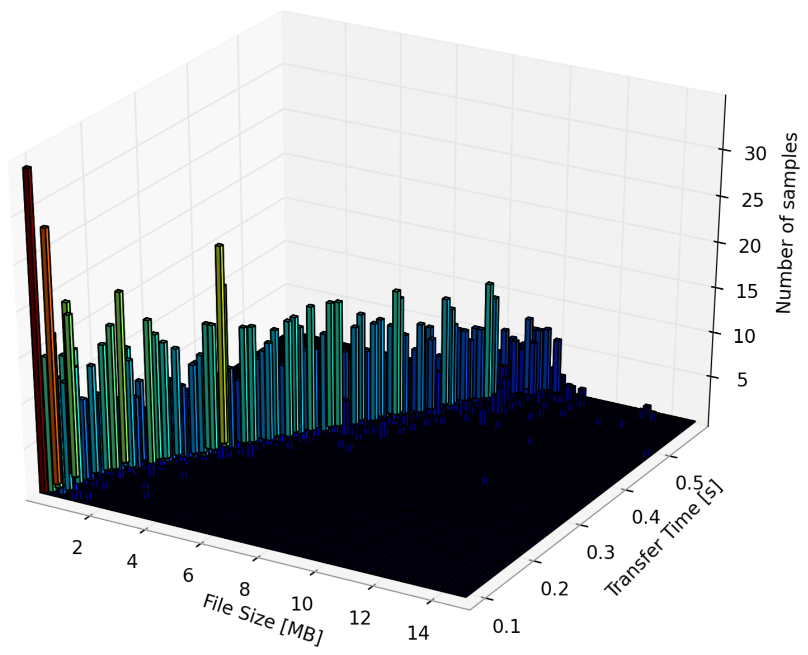
a cascaded series of service queues - possible with message drops, with the application being the central cascader for enqueueing. Dependent services get enqueued by higher level calls and might not be be separable anymore as distinct figures on the probability density graphs. The whole process might be affected by constant delays which would only be determined by independent measurements.

Several types of queues can be conceptually identified and they are serviced in variable time-frames:

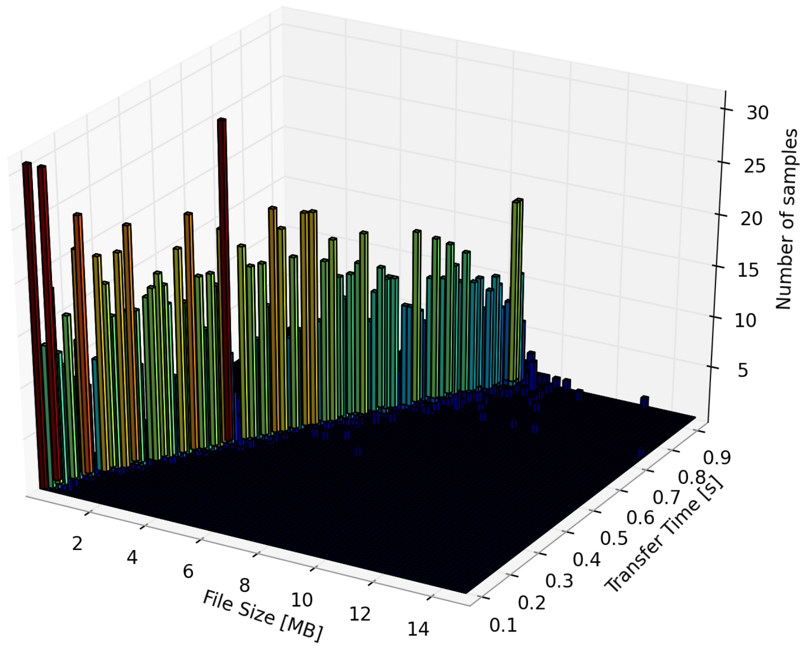
1. Operating system service - task scheduling, "local" syscalls
2. Network queues - transport queues
3. Protocol queues - encryption / decryption queues, TCP FSM, encapsulation / decapsulation

some of which might play an insignificant role in the whole process and, thus, become indiscernible in final time transfer data.

Although a sequence of cascaded queues suggest a Poisson process, multiple maxima in probability density function undermine this assumption. The following figures attempt to illustrate the description above: Figures 21, 22, 23 and 24.

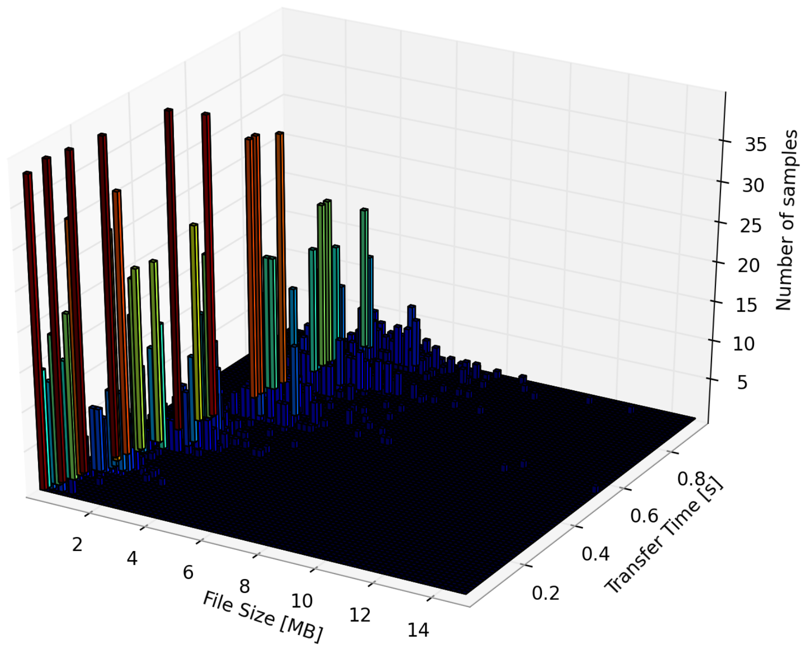


(a) gsiftp histogram data

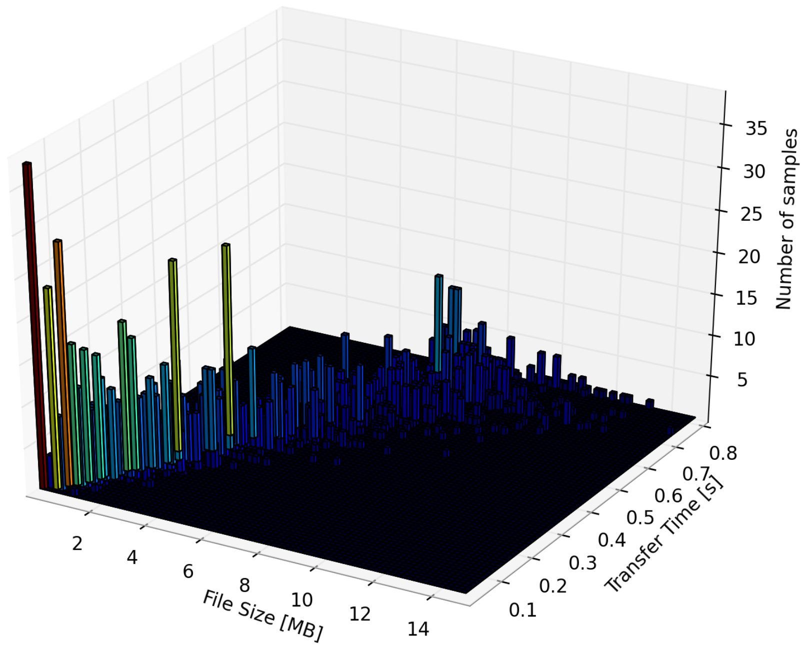


(b) sgsiftp histogram data

Figure 21: gsiftp family of protocols. Histograms are normalized in the PDF sense, in time and file size. Multiple peaks are visible in accumulation of transfer times.

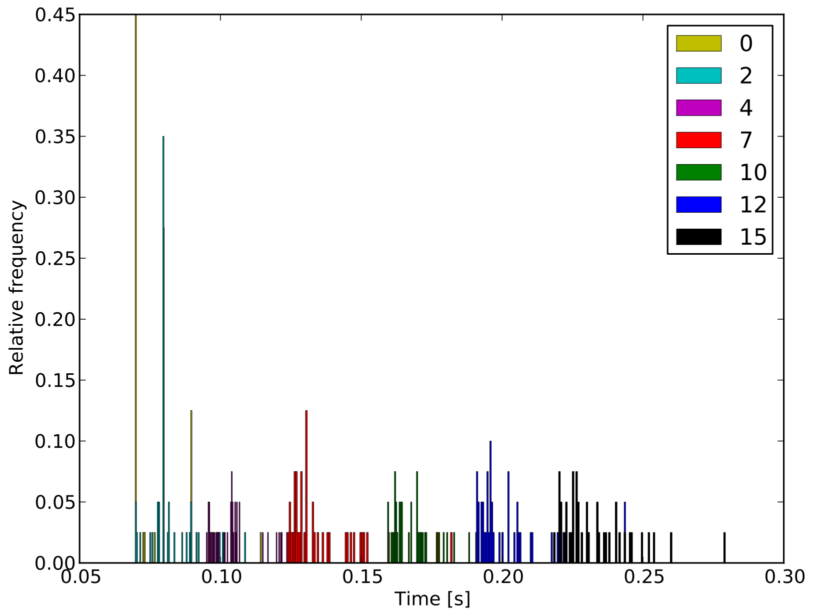


(a) http

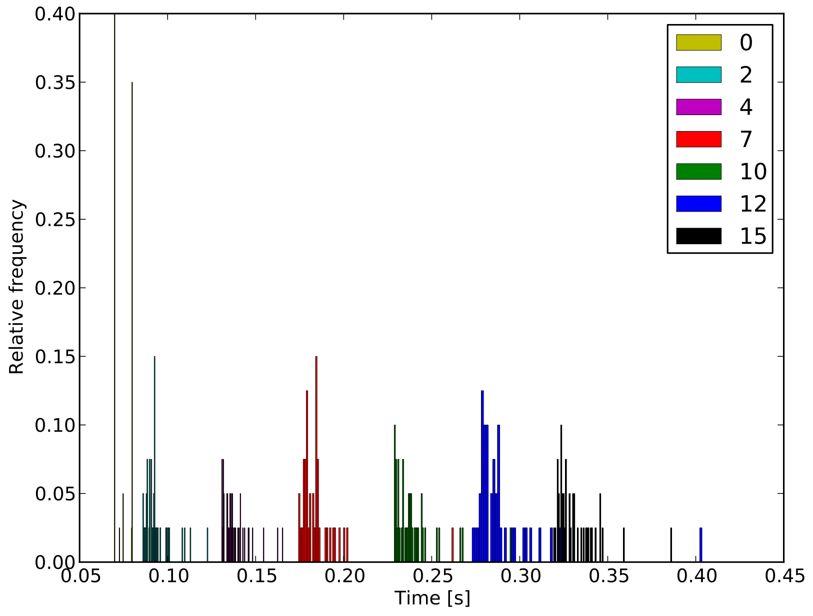


(b) https

Figure 22: http family of protocols. Histograms are normalized in the PDF sense, in time and file size. A spread even larger of statistically significant values can be noticed in the large file range.

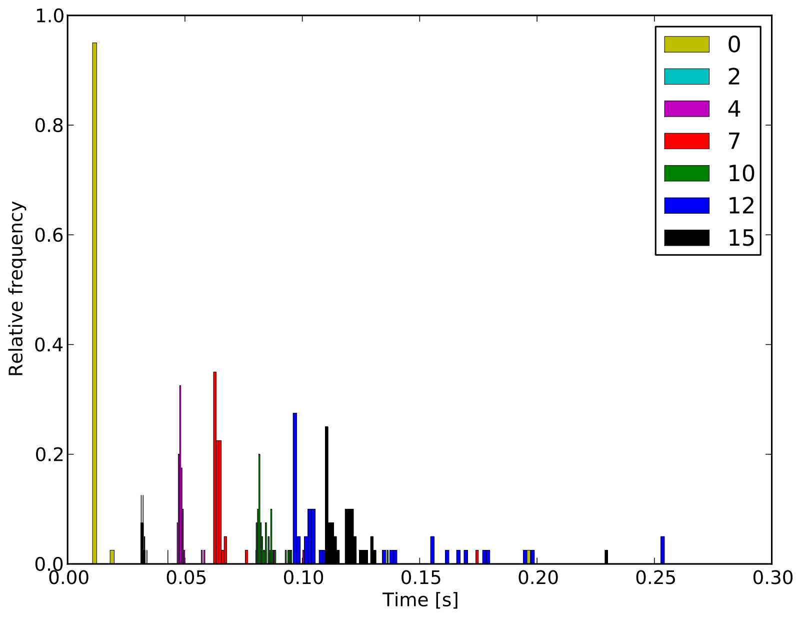


(a) original gsiftp

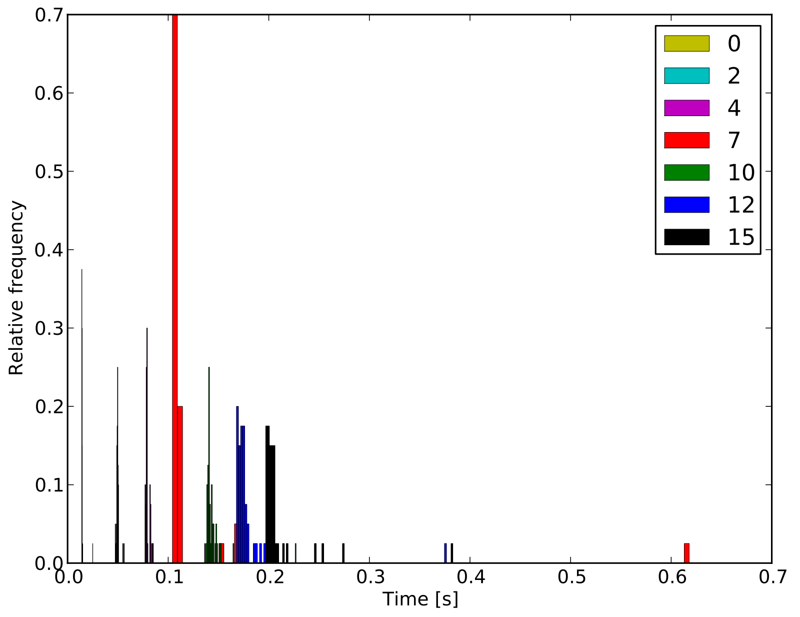


(b) security enhanced gsiftp

Figure 23: Time of transfer histograms for gsiftp family of protocols. Each set is normed to the largest value to prevent large ranges to obscure small ones. Numerical colored labels account for file size ranges in MB, several histograms are superimposed on the same plot.



(a) http, normed



(b) https, normed

Figure 24: Time of transfer histograms. http family wall clock data, normed. Numerical colored labels account for file size ranges in MB (0-2 MB, 2-4 MB ...), several histograms are superimposed on the same plot.

## Conclusions

In this paper, a partial implementation of the security improvement described in [18] is presented. This particular implementation takes advantage of the security association established within TLS handshake to implement the micro-contracts paradigm. This minimizes the overhead of cryptographic operations for data transfer and allows one step further in assigning responsability for data handling. It also addresses the non-repudiation aspect of data processing and ownership.

A section about profiling data in terms of "instructions read" is presented as an aid for design specification and complementary to measurements. We take an "inclusive" approach, where the top level contains the child call-tree statistics and that underlines both the high level operations involved and their "time usage". The "non-inclusive" approach of listing the top 10% where application spends 90% of the time would leave us with a list of low level routines that hardly allow any comment and tend to make the whole description as big as the code.

The investigation indicates that encryption operations climb to the top together with memcopy operations for secure versions of protocols and that the large gap between wall clock data and resource usage data is due to blocking select, waiting for server response. This is simultaneous with high CPU load on server side, as an additional note.

Gsiftp operation and its secure counterpart, show the same type of behavior, with crypto operations, mainly HMAC-ing being emphasized. But contrary to http-type, blocking select plays little if any role and resource usage data is close to the lower envelope of the wall-clock data. Also, sever-side operations show a lower CPU load in this case.Extensive benchmarking of GET operations is examined for gsiftp and http protocols and their secure counterparts, sgsiftp and https and several angles of examination are described. The one that yields reproducible results is the resource-usage measurement and, in that case, besides transfer rate changes, scaling relations of the processes involved are examined. The transfer degradation in the ideal case is estimated to 35%, expected to decrease in the real-world case, mostly noticeable on file sizes around/over 5 MB. The limitations of the measurements and their meaning are stated too. A simple analysis of the linear transfer model (1),(2) enables us to extrapolate the results to more complicated experiments.

Further, the common, wall-clock data measurements are described and briefly analyzed. It turns out that deeper statistics is involved in understanding this type of data, therefore a definite conclusion is not reached within this paper, but directions for future research are outlined.

## Acknowlegdements

The authors wish to thank Guido van 't Noordende for the review of this paper and many useful discussions and Cosmin Dumitru for pointing out affinity setting on multicore CPUs as a measurement variable.

## References

[1] M. Horowitz, S. Lunt, Request for Comments: 2228, FTP Security Extensions, http://www.ietf.org/rfc/rfc2228.txt

[2] J. Postel, J. Reynolds: Request for Comments: 959, http://www.ietf.org/rfc/rfc959.txt

[3] P. Hethmon Request for Comments: 2389, R. Elz:Feature negotiation mechanism for the File Transfer Protocol,http://www.ietf.org/rfc/rfc2389.txt

[4] R. Elz , P. Hethmon: Extensions to FTP, http://tools.ietf.org/html/draft-ietf-ftpext-mlst-16

[5] http://www.ggf.org/documents/GFD.129.pdf, Grid Storage Resource Management, Copyright OpenGridForum (2007, 2008), The Storage Resource Manager Interface Specification Version 2.2,

[6] H. Krawczyk, M. Bellare, R. Canetti, February 1997, RFC 2104, HMAC: Keyed-Hashing for Message Authentication, http://www.ietf.org/rfc/rfc2104.txt

[7] T. Dierks,C. Allen: Request for Comments: 2246 , January 1999,The TLS Protocol

[8] Using gLite to Implement a Secure ICGrid Jesus Luna, Marios D Dikaiakos, Harald Gjermundrod, Michail Flouris, Manolis Marazakis and Angelos Bilas, DOI 10.1007/978-0-387-85966-8_7, Grid and Services Evolution

[9] Enhancing Privacy and Authorization Control Scalability in the Grid Through Ontologies, IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 13, NO. 1, JANUARY 2009, I. Blanquer, V. Hernandez, D. Segrelles, and E. Torres

[10] Rajkumar Kettimuthu, Liu Wantao, Joseph Link, and John Bresnahan, A GridFTP Transport Driver for Globus XIO

[11] Johan Montagnat , A Secure Grid Medical Data Manager Interfaced to the gLite Middleware.

[12] Ravi Prasad and Constantinos Dovrolis, Georgia Institute of Technology Margaret Murray and Kc Claffy, Cooperative Association for Internet Data Analysis (CAIDA), Bandwidth Estimation: Metrics, Measurement Techniques, and Tools,IEEE Network, November/December 2003

[13] The Globus Striped GridFTP Framework and Server, William Allcock, John Bresnahan Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, Ian Foster,Proceedings of the 2005 ACM/IEEE SC|05 Conference (SC'05)

[14] Unpublished Papers,Increasing Timing Resolution for Processes and Threads in Linux, Stephen Hines, Bartow Wyatt, and J. Morris Chang , http://ww2.cs.fsu.edu/~hines/present/timing_linux.pdf

[15] Johan Montagnat, Ákos Frohner, Daniel Jouvenot, Christophe Pera, Peter Kunszt, Birger Koblitz, Nuno Santos, Charles Loomis, Romain Texier and Diane Lingrand, et al., A Secure Grid Medical Data Manager Interfaced to the gLite Middleware, JOURNAL OF GRID COMPUTING Volume 6, Number 1, 45-59, DOI: 10.1007/s10723-007-9088-2

[16] Key management for encrypted data storage in distributed systems,Ludwig Seitz Jean-Marc Pierson Lionel Brunie LIRIS, CNRS FRC 2672, INSA de Lyon , Proceedings of the Second IEEE International Security in Storage Workshop (SISW'03) 0-7695-2059-6/04

[17] Heinz Stockinger, Flavia Donno, Erwin Laure, Shahzad Muzaffar, Peter Kunszt, Giuseppe Andronico, Paul Millar, Grid Data Management in Action: Experience in Running and Supporting Data Management Services in the EU DataGrid Project, Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003

[18] van 't Noordende, G.J.; Olabarriaga, S.D.; Koot, M.R.; de Laat, C.T.A.; , "A Trusted Data Storage Infrastructure for Grid-Based Medical Applications," Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on , vol., no., pp.627-632, 19-22 May 2008 doi: 10.1109/CCGRID.2008.84

[19] Allcock, B.; Bester, J.; Bresnahan, J.; Chervenak, A.L.; Kesselman, C.; Meder, S.; Nefedova, V.; Quesnel, D.; Tuecke, S.; Foster, I.; , "Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," Mass Storage Systems and Technologies, 2001. MSS '01. Eighteenth IEEE Symposium on , vol., no., pp.13-13, 17-20 April 2001

[20] GRID COMPUTING Lecture Notes in Computer Science, 2004, Volume 3165/2004, 131-140,Pegasus: Mapping Scientific Workflows onto the Grid Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi and Miron Livny

[21] Concurrency and Computation: Practice and Experience Special Issue: Grid Computing Environments Volume 14, Issue 13-15, pages

1507–1542, November - December 2002, Economic models for resource management and scheduling in Grid computing,Rajkumar Buyya, David Abramson, Jonathan Giddy, Heinz Stockinger

[22] Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers,Steven L. Pratt Dominique A. Heger IBM,Linux Symposium 2004

[23] http://docs.sun.com/app/docs/doc/806-6546

[24] Globus Toolkit source code: http://www.globus.org/toolkit/

[25] Journal of Physics: Conference Series 180 (2009) 012047 IOP Publishing doi:10.1088/1742-6596/180/1/012047,Brian Bockelman ,Using Hadoop as a Grid Storage Element

[26] Tansu Alpcan @Deutsche Telekom Laboratories, Technical University of Berlin, Germany and Tamer Basar @ University of Illinois at Urbana-Champaign, Illinois, USA : Network Security: A Decision and Game Theoretic Approach, Preprint Version (v 1.0.3) , Cambridge University Press