# Harnessing Multicore Processors for High Speed Secure Transfer

John Bresnahan[1,2,3], Rajkumar Kettimuthu[1,2], Mike Link[1,2], Ian Foster[1,2,3]

[1]Math and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
[2]Computation Institute, University of Chicago, Chicago, IL 60637
[3]Department of Computer Science, University of Chicago, Chicago, IL 60637

{bresnaha, kettimut, mlink, foster}@mcs.anl.gov

*Abstract*—A growing need for ultra high-speed data transfers has motivated continued improvements in physical network layer transmission speeds. However, as researchers develop protocols and software to operate over such networks, they often fail to account for security. The processing power required to encrypt or sign packets of data can significantly decrease transfer rates, and thus security is often sacrificed for throughput. Emerging multicore processors provide a higher ratio of CPUs to network interfaces, and can in principle be used to accelerate encrypted transfers by applying multiple processing and network resources to a single transfer. We discuss the attributes that network protocols and software must have to exploit such systems. In particular, we study how these attributes may be applied in the GridFTP code distributed with the Globus Toolkit. GridFTP is a well accepted and robust protocol for high speed data transfer. It has been shown to scale to near network speeds. While GridFTP can provide encrypted and protected data transfers, it historically suffers transfer performance penalties when these features are enabled. We present configurations to the Globus GridFTP server that can achieve fully encrypted high speed data transfers.

***Index Terms*—Secure data transfer, GridFTP, Encryption, Parallel streams**

## I. Extended Abstract

In order to achieve processing parallelism on protected or encrypted transfers we must ensure that secure processing is performed on different portions of the data stream at the same time and on different CPUs. While simple in principle, this concept presents interesting problems for network protocols and software implementations. Encryption protocols that use cipher block chaining, such as TLS / SSL [1], require that data be decrypted in the same order that it was encrypted. Further, the way that bytes in a stream are processed varies with their position in the stream. Thus, it matters not only what is the value of the byte being processed, but also when it was processed.

These issues introduce difficulties when breaking up the streams for parallelization. For the reasons described above, we cannot take portions of a single data stream and process them in parallel against the same security context. To properly follow secure protocols we cannot process any one byte until the previous byte has been processed, thus there can be no parallelism against a single security context.

We can solve this problem by creating a many distinct security contexts for a single data transfer. A simple way to realize this approach with existing network protocols is by using parallel streams. Parallel streams are common in data transfers as a means of network optimization [2,3]. To minimize penalties associated with TCP slow start and dropped packets, many TCP streams are used for the same logical transfer, thus reducing the penalties associated with any one packet loss. This technique can also be leveraged for use in parallel encryption. Each stream has its own security context and is independent with regard to security processing. Thus, we can achieve parallel security processing.

### A. Related Work

Hardware accelerators have been used to address the SSL performance problems. Accelerator is a card that plugs into PCI slot or SCSI port and contains a co-processor that performs part of SSL processing. Network Interface Cards with offloaded SSL and IPSec [4] have also been produced. We want to achieve high-speed secure transfers with general-purpose hardware so that it can be used more commonly. We expect that multi-core processors would become more common than SSL/IPSec offload engines. Further, we would like to utilize the parallel and higher processing power that the multi-core technology promises to achieve high-speed secure data transfers. Also, the offload techniques do not help in achieving processing parallelism for security processing on a single node.

### B. Asynchronous Event Model

Solving this problem in software requires some type of threaded IO model. In order to get many parallel data streams processing at once multiple threads of execution must be occurring on different CPUs. This can happen via threads in a single user process or by making use of multiple processes. The Globus toolkit achieves this type of parallelism via an asynchronous event model and thread pools [5]. We present here the advantages of the asynchronous thread pool model.

In an asynchronous event model, the software developer posts I/O requests to the system. When the request is fulfilled (or an error occurs) the user is notified via a *callback* function which the developer defines in their own process space.

Between the time of posting the request and the time the application receives the callback, the developer's code is free to do whatever it likes, including post more I/O requests. The I/O subsystem services the developer's request by creating background threads. These threads handle the framing of I/O and interact with the operating system to send and receive messages without interfering with the developer's code.

The Globus Toolkit's I/O library, Globus XIO, is a protocol abstraction layer that allows developers to ignore the specific protocol in use and just post requests for read and write. Part of servicing secure protocol requests involves signing or encrypting the posted data. In Globus XIO, this task is performed as part of the event processing and is therefore handled by one of the system's background threads. If many events are posted at once, as is the case for parallel TCP protocols, many threads can be processing the posted events in parallel. As part of this work we set the default number of background threads used by Globus XIO (which can be set by the system administrator to any value) to the CPU count plus one. Because of this asynchronous thread pool model, the developer at no point needs to be aware of the number of background threads. It is a strictly site optimization parameter.

### C. GridFTP Configuration

The Globus GridFTP server is built on the above described asynchronous event model. The GridFTP protocol uses parallel streams on the data channel. Because of these two facts we can achieve parallel secure processing. When the number of parallel streams is greater than or equal to the number of system CPUs, processing parallelism occurs.

### D. Striping

In some cases the number of CPUs on a single system inadequate to match the processing requirements of secure transfers. In these cases we can still achieve high throughput secure data transfers by tying many computers together in a striped transfer. Striped transfers allow portions of the data to come from different network endpoints. Thus, we can use CPUs in many machines for a single data transfer and therefore scale the processing power up to network speeds.

When we require many stripes to achieve network speeds we will unfortunately not max out the capabilities of the network cards in each stripe. This situation is not ideal, but it does provide a solution and gives us a test bed for finding the optimal processor to NIC ratio. Upcoming multicore technology promises to up the ratio of CPU to NIC in a single system and thus solve this problem.

### E. Results

Table 1 shows initial results of experiments in which we use the asynchronous event model to take advantage of multiple CPUs. The measurements were taken on UC TeraGrid Dual 1.5GHz Itanium machines. It shows how the use of one or two streams (P1 and P2, respectively) affects transfer rates in a single threaded system versus an asynchronous thread pool system. We see that when a thread pool system is used with two streams, we can basically double performance relative to a single stream or to two streams without a thread pool system.

TABLE I: TRANSFER RATES (IN MEGABYTE/SEC) WITH DIFFERENT THREADING MODELS

| Security Level | Single P1 | Single P2 | Pool P1 | Pool P2 |
|---|---|---|---|---|
| *Clear* | 903 | 905 | 903 | 906 |
| *Authenticated* | 899 | 899 | 899 | 899 |
| *Safe* | 488 | 517 | 488 | 770 |
| *Private* | 177 | 183 | 177 | 340 |

In the next set of results we show that it takes approximately eight Intel Xeon 2.4 GHz CPUs to achieve fully encrypted transfers at network speeds. Since eight core processors were not available to us, our experiments were performed with striped transfers of four dual CPU machines. Additional stripes increase the amount of memory and network and bus bandwidth resources available to a single transfer. However, with the exception of network bandwidth, those resource requirements are statically dependent on the bandwidth delay product of the transfer. An encrypted transfer may require more memory, but that requirement is still a function of the bandwidth delay product. We used a value of 128KB and each of our test bed nodes had 4GB of memory. Therefore a single node had a surplus of memory and our results were not tainted by the additional surplus of memory that comes with additional stripes. However, the additional network resources did allow us to exceed the 1 Gigabit network limit of a single machine. Because our goal is to show that with enough processing power we can meet network speeds exceeding it is acceptable in that it shows when given enough parallel processing power the bottleneck will shift from the CPU to the NIC. This is in line with our goals. The graph in figure 1 shows the results of this experiment.

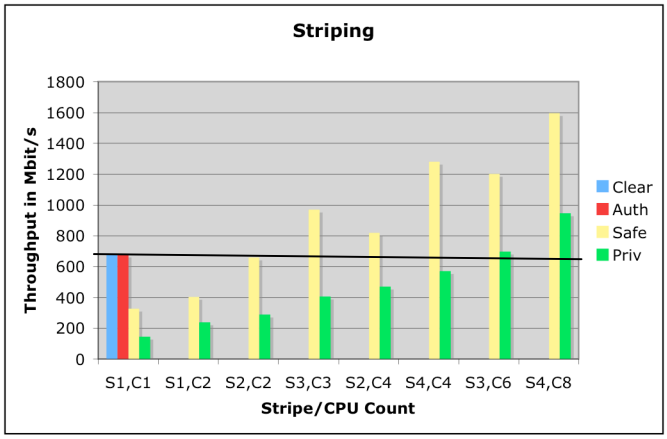


Figure 1: Extrapolation for more cores using striped transfers.

### References

[1] T. Dierks and E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.1, IETF RFC 4346, 2006.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The Globus striped GridFTP framework and server," in SC'05, ACM Press, 2005.

[3] T.J. Hacker et. al., Improving Throughput and Maintaining Fairness using Parallel TCP. IEEE InfoCom, 2004.

[4] S. Kent and R. Atkinson, Security Architecture for the Internet Protocol, IETF RFC 2401, 1998

[5] http://www.globus.org/toolkit/docs/3.2/developer/globus-async.html