

# Extending Globus to support Multicast Transmission

Karl Jeacle and Jon Crowcroft  
University of Cambridge Computer Laboratory  
15 JJ Thomson Avenue, Cambridge CB3 0FD, UK  
{firstname.lastname}@cl.cam.ac.uk

## Abstract

The Globus Toolkit is an open source software toolkit that has become the de-facto standard for middleware used to build Grid services. Globus XIO is an eXtensible Input/Output library for Globus. It provides a POSIX-like API to swappable I/O implementations – essentially “I/O plugins” for Globus.

TCP-XM is a modified TCP engine that supports multicast transmission. We have taken this protocol and wrapped it using XIO to extend the Globus Toolkit to support multicast transmission.

This paper describes the implementation and operation of our Globus XIO multicast transport driver. It provides an overview of the TCP-XM protocol design, and describes some initial experimental results.

## 1 Introduction

The Globus Toolkit is an open source software toolkit primarily developed by the Globus Alliance. It has become the de-facto standard for middleware used to build Grid services.

The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications.

At present, almost all bulk data transfer is carried out using the GridFTP protocol [17, 1]. This is based on the conventional FTP protocol, but includes some extra features to optimize bulk data transfer e.g. parallel data streams. Software that makes use of the protocol must support the Grid Security Infrastructure (GSI) [11] so that user authentication can take place using Grid certificates.

## 2 Globus XIO

Globus XIO is an eXtensible Input/Output library for the Globus Toolkit. It provides a

POSIX-like API to swappable I/O implementations – essentially “I/O plugins” for Globus [3].

There are two main goals for Globus XIO:

1. Provide a single user API to all Grid I/O protocols. There are many different APIs for many different protocols. XIO should abstract this complexity for Grid developers.
2. Minimize the development time for creating new protocols. Writing with the XIO framework in mind allows the protocol designer to maximize the time spent on protocol code.

This approach is similar to the Streams [18] concept originally introduced in System V Unix. A stream is a full-duplex connection between a user process and a device. It consists of one or more connected processing modules, and is analogous to a Shell pipeline, except that data flows in both directions. A key advantage of the Streams module approach is the ability to develop new code within the protocol stack without requiring changes to the kernel source.

Figure 1 shows the Globus XIO Architecture. The User API provides a familiar and con-

venient POSIX-like open/close/read/write interface to programmers. The Framework facilitates the XIO system itself, while the Driver Stack comprises one or more transform drivers over a single transport driver.

Transform drivers manipulate data buffers passed to them via the user API and the XIO framework. Transport drivers are always at the bottom of the stack and are solely responsible for sending data over the wire.

Example transform driver functionality includes tasks such as compression, logging and security. Transport drivers typically implement protocols such as TCP or UDP.

The configuration and order of drivers in the stack can be selected at runtime.

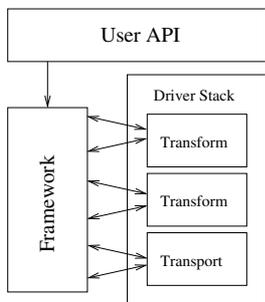


Figure 1: Globus XIO Architecture

Globus XIO provides an ideal mechanism for introducing new protocols to Grid users deploying Globus applications. We have built a modified version of TCP that supports multicast, and wrapped it using XIO to create a multicast transport driver for Globus.

### 3 TCP-XM

TCP-XM is a modified TCP engine that supports multicast, and runs in userspace over UDP. It forms the basis for our XIO transport driver.

This section provides some background rationale. A more detailed description of the protocol can be found in [13].

Today, applications use TCP for reliable unicast transfers. It is a mature and well-understood protocol. By modifying TCP to deliver data to more than one receiver at a time, and use multicast when available, an

application can transparently send data reliably to multiple recipients. Using existing TCP mechanisms, the modified protocol ensures that data is delivered reliably. Multicast transmission is attempted for performance reasons, but fallback to unicast preserves backwards compatibility and reliability guarantees, and capitalizes on more than a decade of experience that TCP implementations enjoy.

Network protocols are typically implemented in the kernels of hosts and network devices. Any proposals that require modifications to these protocols imply changes to kernel code. This immediately restricts deployment opportunities. By limiting changes to code that runs in userspace on end stations, new protocols can be developed and tested on live networks.

Our approach is to implement a modified multicast TCP over UDP. User-space applications can freely send and receive UDP packets, so a small shim layer can be introduced to encapsulate and decapsulate the TCP-like engine's packets into UDP.

While there are performance implications by running in userspace, the instant deployment potential of a userspace implementation, coupled with the scalability of multicast, mean that any such limitations are more than acceptable.

Therefore, it is possible to build a new end-to-end protocol, and implement as a userspace library. Grid applications can avail of this via a Globus XIO driver.

The key advantage of this approach is that any Globus application can make use of this new protocol by simply pushing its XIO driver onto the stack, while non-Globus Grid applications can link against an independent userspace library.

No changes are required in the network (other than enabling IP multicast).

Because the protocol is not tightly coupled to the application, should it become adopted for widespread use, a native implementation can be built in the kernel to boost performance.

Our implementation of TCP-XM has been built as an extension to the lwIP TCP/IP stack [7].

## 4 Previous work

Reliable multicast has proved to be a difficult problem to solve, and over the last decade, much research has been carried out into how best to approach this problem [10, 14, 15, 19, 5].

Moving TCP out of the kernel into userspace is not a new idea. A number of projects have done this in the past, either as a by-product of a larger project, or as an end in itself [9, 16, 6, 8, 2, 7].

A number of transform and transport drivers have been built as part of the Globus XIO system. These include GSI and HTTP transform drivers, and TCP, UDP, File and SABUL transport drivers. An XIO implementation of GridFTP is also underway [4].

There are no known existing XIO multicast transport drivers.

## 5 XIO driver operation

Two important XIO data structures must be considered when implementing and making use of a transport driver:

1. **Attribute** – in order to set driver-specific parameters, a custom attribute structure can be used. For TCP-XM, the list of  $n$  destination addresses is supplied in this way.
2. **Handle** – this is returned to the user once the XIO framework has all the information needed to open a new connection. It is then used to reference the connection on all future I/O calls. An lwIP `netconn` structure will constitute the handle for multicast.

With these data structures in place, the transport driver is built by mapping XIO's POSIX open/close/read/write calls to the appropriate TCP-XM API calls.

Figure 2 shows how a developer can take advantage of XIO. First, the XIO stack is initialized. Next, drivers are pushed onto the stack (in this example, a TCP-XM transport driver). Any necessary driver attributes are created and with a stack in place, a handle for all subsequent I/O operations is created. The

example finishes with six bytes of data being written to the network.

```
// init stack
globus_xio_stack_init(&stk, NULL);

// load drivers onto stack
globus_xio_driver_load("tcpxm", &tdrv);
globus_xio_stack_push_driver(stk,tdrv);

// init attributes
globus_xio_attr_init(&attr);
globus_xio_attr_cntl(attr, tdrv,
    GLOBUS_XIO_TCPXM_SET_REMOTE_HOSTS,
    hosts, numhosts);

// create handle
globus_xio_handle_create(&handle, stk);

// send data
globus_xio_open(handle, NULL, attr);
globus_xio_write(handle, "hello\n", 6,
    1, &nbytes, NULL);
globus_xio_close(handle, NULL);
```

Figure 2: Sample XIO User Code

## 6 Internals

### 6.1 Data structures

The `globus_l_server_t` structure is used for XIO servers when TCP-XM is receiving data. `listen_conn` points to the initial `netconn` structure used on server listen, while `conn` points to the post-accept structure.

The `globus_l_attr_t` structure contains TCP-XM specific protocol information. If the driver is acting as a server, the `server` variable points to the relevant `globus_l_server_t` structure. It acts as a bridge for the driver between the creation of a server object and the assignment of a handle. This is because the `globus_l_xio_tcpxm_open()` call will use the value of the attribute server pointer to determine if the driver is operating as a client or a server. If the pointer is non-NULL, the user must have called `globus_xio_server_create`. Otherwise, the driver is a client, so a call to `netconn_connect()` will yield a new handle.

Two client specific variables in `globus_l_attr_t`

are `hosts` and `numhosts`. These are passed into the driver when acting as a client. `hosts` is an array of destination hostnames, while `numhosts` is the number of hosts. The `GLOBUS_XIO_TCPXM_SET_REMOTE_HOSTS` command is used to set these attribute values.

`globus_l_attr_t` also contains `srcport` and `dstport` variables. These are used by both clients and servers to set the underlying UDP ports used by TCP-XM. The `GLOBUS_XIO_TCPXM_SET_UDP_PORTS` command is used to set these attribute values.

Finally, the `globus_l_handle_t` structure is very simple, containing a single pointer to the `netconn` structure that is ultimately used as the user handle for all I/O calls.

## 6.2 Function calls

[Note: the `glxt` prefix is used in this section as to abbreviate `globus_l_xio_tcpxm`.]

The XIO framework uses `glxt_activate()` to activate the driver and `glxt_deactivate()` to later deactivate it. These are followed respectively by `glxt_init()` to tell XIO what functions are present in the driver, and later by `glxt_destroy()` to deallocate the driver.

Handles are created with `glxt_handle_init()` and destroyed with `glxt_handle_destroy()`.

`glxt_attr_init()` initialises attributes; copies are made with `glxt_attr_copy()` and then, when no longer of use to the driver, destroyed with `glxt_attr_destroy()`. User-specified commands such as `GLXT_SET_REMOTE_HOSTS` used to set the destination hosts and number of host, and `GLXT_SET_UDP_PORTS` to set the UDP ports used are made via `glxt_attr_ctl()`

If the driver is a server, `glxt_server_init()` initialises the `globus_l_server_t` structure, starts the lwIP/TCP-XM threads, and then binds and listens for new connections. Calling `glxt_server_destroy()` cleans up.

If a client, `glxt_connect()` is used internally to open a connection and create a new handle. `glxt_server_accept()` is the corresponding internal function on the server side; it blocks waiting for an incoming connection.

When a handle has been created, `glxt_open()`

will open a new connection if a client, or block waiting if a server. Reads are then made via `glxt_read()`; writes are via `glxt_write()`, and `glxt_close()` is used to clean up.

## 7 One-to-many caveats

Two significant caveats with the current XIO approach have become apparent during the implementation of the multicast driver.

1. The XIO architecture assumes one-to-one connections. The XIO User API therefore requires modifications to better support one-to-many protocols. While minimal changes are required at the API, there may be more significant changes required within the XIO framework.
2. GSI is one-to-one. Most Globus application make use of GSI to authenticate with peers on connection setup. However, as it stands, GSI cannot be expected to authenticate  $n$  peers. Some form of “GSI-M” that supports one-to-many authentication is required.

The first of the above caveats is a relatively minor difficulty. Workarounds are possible due to the flexible nature of the XIO attribute data structure.

The second caveat, however, is more serious. From a practical perspective, the multicast transport driver provides Globus applications with multicast data transfer capability to multiple destinations. But as it is not possible to push a one-to-one transform driver on top of a one-to-many transport driver, multicast support currently comes at the expense of security.

It is worth noting that security for many-to-many is often a problem because of late joiners and early leavers. But unlike many multicast protocols, TCP-XM has per-receiver state in the sender. And for bulk transfer from one to  $n$  hosts, it is assumed that session and transport lifetimes are aligned. Because of this, while building a “GSI-M” transform driver may require changes to XIO, it is a far less onerous task than addressing many-to-many security [12].

## 8 Initial test results

Note that these experiments were carried out using a standalone test program linking against the TCP-XM userspace library, and not with a Globus application using the XIO driver. This is primarily due to the currently limited number of available machines running a sufficiently recent version of Globus that supports XIO (initial support was introduced in 3.2 alpha).

Accounts on machines at eScience Centres around the UK are available provided suitable application processes and procedures are followed. Table 1 lists the hosts used. Sites in bold have readily available multicast connectivity (i.e. multicast extends to more than a single AccessGrid node).

Site	Hostname
Belfast	gridmon.cc.qub.ac.uk
<b>Cambridge</b>	mimiru.escience.cam.ac.uk
<b>Cardiff</b>	agents-comsc.grid.cf.ac.uk
Daresbury	gridmon.dl.ac.uk
Glasgow	cordelia.nesc.gla.ac.uk
<b>Imperial</b>	mariner.lesc.doc.ic.ac.uk
<b>Manchester</b>	vermont.mvc.mcc.ac.uk
<b>Newcastle</b>	ramshope.ncl.ac.uk
<b>Oxford</b>	esci1.oucs.ox.ac.uk
Rutherford	gridmon.rl.ac.uk
<b>Southampton</b>	beacon1.net.soton.ac.uk
<b>UCL</b>	sonic.cs.ucl.ac.uk

Table 1: UK eScience Testbed Hosts

Table 2 shows the unicast transfer speeds from Cambridge to eScience hosts. Not all hosts from table 1 are listed due to intermittent network and firewall issues.

The speeds vary widely because of both the network infrastructure in place and the speed of the local network and CPU at the remote sites. It is important to note the speed of the slowest receivers when initiating bulk group transmissions.

Figure 3 illustrates the performance of the protocol when transmitting data to the eScience centres. The unicast plot represents  $n$  individual connections i.e.  $n$  application-level unicast connections, while the multicast plot shows the effects of a single application-level TCP-XM connection with  $n$  TCP PCBs underneath.

Site	Kilobits/sec
Imperial	25806
Daresbury	14545
Cardiff	13793
Manchester	11428
Southampton	10810
Belfast	5031
Oxford	4159
UCL	2305

Table 2: Speeds from Cambridge

As can be seen, the maximum transmission speed obtained when using either unicast or multicast is limited by the speeds of the slowest receivers in table 2.

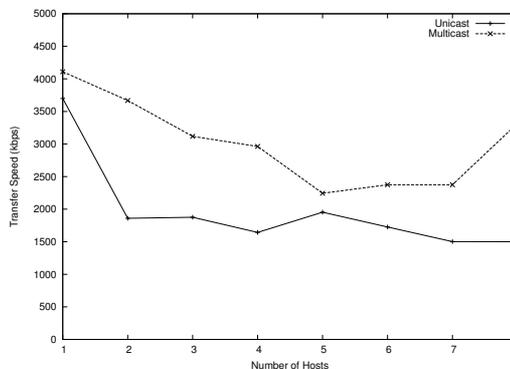


Figure 3: Speed: Unicast vs Multicast

While the use of multicast via TCP-XM cannot offer higher transmission speeds across the network, the benefit of using the protocol manifests itself in its efficient use of the network. While not all remote sites will have multicast capability, those that do will receive a significant proportion of their data via single network-wide multicasts, and not replicated unicast packets.

## 9 Conclusion

We have described the work to date on the TCP-XM protocol, and its implementation as a Globus XIO transport driver. By implementing this protocol in userspace above UDP, we are in a position to test the operation of the protocol in live networks, while delivering a reliable multicast transport mechanism to the UK eScience community.

## References

- [1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. GridFTP Protocol Specification. GGF GridFTP Working Group Document, September 2002.
- [2] Torsten Braun, Christophe Diot, Anna Hoglander, and Vincent Roca. An Experimental User Level Implementation of TCP. Technical report, INRIA RR-2650, September 1995.
- [3] John Bresnahan. Globus XIO. [www-unix.globus.org/developer/xio/](http://www-unix.globus.org/developer/xio/), December 2003.
- [4] John Bresnahan and Bill Allcock. Globus XIO and GridFTP for Developers. In *Proceedings of GlobusWorld 2004*, San Francisco, January 2004.
- [5] John W. Byers, Michael Luby, Michael Mitzenmacher, and Ashutosh Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *SIGCOMM*, pages 56–67, 1998.
- [6] Tom Dunigan and Florence Fowler. A TCP-over-UDP Test Harness. Technical report, Oak Ridge National Laboratory, ORNL/TM-2002/76, May 2002.
- [7] Adam Dunkels. Minimal TCP/IP implementation with proxy support. Technical report, Swedish Institute of Computer Science, SICS-T-2001/20-SE, February 2001.
- [8] Aled Edwards and Steve Muir. Experiences Implementing a High-Performance TCP in User-Space. Technical report, HP Laboratories Bristol, HPL-95-110, September 1995.
- [9] David Ely, Stefan Savage, and David Wetherall. Alpine: A User-Level Infrastructure for Network Protocol Development. In *Proceedings of USENIX USITS*, 2001.
- [10] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.
- [11] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. In *5th ACM Conference on Computer and Communications Security*, pages 83–92, 1998.
- [12] Thomas Hardjono and Gene Tsudik. IP Multicast Security: Issues and Directions. *Annales de Telecom*, 2000.
- [13] Karl Jeacle and Jon Crowcroft. Reliable high-speed Grid data delivery using IP multicast. In *Proceedings of All Hands Meeting 2003*, Nottingham, UK, September 2003.
- [14] John C. Lin and Sanjoy Paul. RMTP: A Reliable Multicast Transport Protocol. In *INFOCOM*, pages 1414–1424, San Francisco, CA, March 1996.
- [15] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven Layered Multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, August 1996. ACM Press.
- [16] Ian Pratt and Keir Fraser. Arsenic: A User-Accessible Gigabit Ethernet Interface. In *Proceedings of IEEE INFOCOM*, 2001.
- [17] The Globus Project. GridFTP: Universal Data Transfer for the Grid. Globus Project White Paper, September 2000.
- [18] Dennis M. Ritchie. A Stream Input-Output System. *AT&T Bell Laboratories Technical Journal*, 63(8):1897–1910, 1984.
- [19] Luigi Rizzo and Lorenzo Vicisano. A Reliable Multicast data Distribution Protocol based on software FEC techniques. In *The Fourth IEEE Workshop on the Architecture and Implementation of High Performance Communication Systems (HPCS'97)*, Sani Beach, Chalkidiki, Greece, June 1997.