FTP Application Development of UDP based transport protocol for Satellite Based Grid Technology

By

Pravina Mehta
(04MCE006)

Department Of Computer Science And Engineering
Nirma University Of Science And Technology
Ahmedabad 382481
May 2006

FTP Application Development of UDP based transport protocol for Satellite Based Grid Technology

By

Pravina Mehta
(04MCE006)

Guide

Mr Krishan Kumar
Networks Division,
SAC-ISRO, Ahmedabad.

A Dissertation
Submitted to

Nirma University of Science and technology
in partial fulfillment of the requirement
for the degree of

Master of Technology

Department of Computer Science & Engineering
Ahmedabad 382481
May 2006

This is to certify that the Dissertation entitled

FTP Application Development of UDP based transport protocol for Satellite Based Grid Technology

Presented by

Pravina Mehta

has been accepted toward fulfillment of the requirement
for the degree of
Master of technology in Computer Science & Engineering

Professor In Charge Prof. S.N. Pradhan	Head of The Department Prof. D.J. Patel

Date: 29/04/06

ABSTRACT

FTP Application Development of UDP based transport protocol for Satellite Based Grid Technology

Satellite can be useful as communication resource in Grid technology, whereas Grid is useful in effective satellite resource management. Grid protocols are normally written over TCP/IP. Standard transport protocol TCP/IP cannot be used here, as throughput of TCP/IP is inversely proportional to RTT and also bandwidth being costly resource in satellite environment. So for this Satellite based Grid Technology Project, we planned to work out protocols to suite Grid and communication satellite environment. A lot of efforts are going on the modification of TCP (TCP-variants) and the introduction of new protocols (UDP-variants). For high bandwidth, higher window size (RFC 1323) is needed. This can be done by window enhancing mechanism, which requires special skills for tuning the OS. In UDP, there is no such constraint but disadvantage is there is no acknowledgement. But for grid data transfer we need acknowledgement. Both literature survey and simulation experiments studies demonstrate that Simple Available Bandwidth Utilization Library (SABUL) is suitable for grid data transfer under GSAT-4 Applications Project. SABUL, UDP based protocol which uses TCP for control information exchange and UDP for data information exchange. But SABUL does not support FTP application, which is required in our project to transfer files. There are three approaches to develop this FTP application for SABUL in our project : 1) to develop driver interface for SABUL in GridFTP XIO, 2) to change data connection of FTP from TCP to UDP and 3) to develop SABUL-FTP application.

TABLE OF CONTENTS

a. List of Figures

b. List of Tables

Acknowledgments

My sincere gratitude for the individuals, as even their smallest contribution turns out to be one of the biggest breakthroughs.

Starting off, I am indebted to my respected Sir **Prof S.N. Pradhan**, (Computer Engg. Department, Nirma University-Ahmedabad) because of whose sincere efforts I managed to get opportunity to work for an evolving project at a prestigious research institute of Department of Space (SAC, ISRO).

It was a privilege to work on a project in a government organization, which has been the founder of the modern technology in India. My sincere gratitude to **Dr. Haresh S. Bhatt** (SCI/ENGR-SF, Space Applications Centre, ISRO), **Mr. Krishan Kumar** (SCI/ENGR-SF, Space Applications Centre, ISRO), my project guides were altruistic in providing me their domain expertise whenever needed.

Above all, I am grateful to Network Division head **Dr. V. H. Patel** (SCI/ENGR-G) for providing me an opportunity to work on such a research oriented and technically advanced field.

I am also thankful to the entire Network Division staffs that have keenly helped me, particularly among them were Mr. Jagdish Pandya and Mr. Chirag Oza who have promoted my work willingly by making arrangements of needed facility during project.

Last, but not the least, many thanks to all my colleagues who gave valuable feedbacks, and numerous accurate and insightful comments.

Pravina Mehta

Chapter 1

ORGANISATION PROFILE

1.1 Indian Space Research Organization

The Indian Space Research Organization (ISRO) under the Department of Space (DOS) plays a key role in the planning and execution of National Space Activities. It is also responsible for the technical management in the area of space applications and space technology.

The Headquarters of ISRO functioning at Bangalore is responsible for the overall co-ordination and direction of the programs and projects.

- **Objectives of ISRO**

The Primary Objective of the Indian space Research Organization is to establish the Space service in a self-relevant manner. The main thrusts of programme are:

1. Satellite based communication for various applications, resource surveys, management and environmental monitoring, meteorological application, division and operation of satellites, launch vehicles and associated ground equipment for providing this space-based service.
2. Long distance telecommunication, diffusion of TV signals using the satellite.
3. Remote sensing of natural and renewable earth resources and meteorological parameters from satellites.
4. Satellite based resources survey, management and environment monitoring.
5. Development and operation of indigenous satellites.
6. Design and development of Rocket Launching vehicles to place these application satellites into the required orbits.
7. Establishment of ground stations and facilities for using these satellites and for launching.

To realize the above objectives, ISRO activities are oriented predominantly towards design and development of application satellites for communications, remote sensing, TV broadcasting and meteorology.

1.2 Space Application Center

The major tasks of the center are to conceptualize and conduct research and development, and execute projects in the field of space application. To this end, SAC has two board areas of activity viz., satellite based communication, including television, and remote sensing for natural resources survey and management, meteorology and geodesy.

The main activities of SAC are to develop satellites, launch vehicles, Sounding Rockets, associated ground systems and demonstrate applications to hardness space technology for national development in the areas of communication, broadcasting, remote sensing and meteorology, develop the satellite payloads operational for INSAT-II and series of satellites and products generation.

Chapter 2

OVERVIEW OF THE PROJECT

2.1 Grid Computing Applications at ISRO

Applications like disaster management and National Spatial Data Infrastructure (NSDI) are to process the large volume of data located on heterogeneous computing resources. Disaster Management application requires the processing in a near real time whereas NSDI Data are dispersed geographically. For such applications high–speed-computing environment is the essential requirement.

- **Disaster Management**

The satellite-based high speed-computing grid integrates Padma Param computers at Pune and Bangalore and Computing Grid at SAC, Ahmedabad to process the ASAR flight collected data for disaster-affected area. The ASAR flight will fly over the disaster-affected later the ASAR flight will land at nearby airport and dump the huge data in the server located at the airport. This huge data in turn will be sent to India Grid via satellite. India Grid will process this data along with the data sent by the ASAR flight using the computing resources in grid. The data communication will be done using India Grid. The processed Disaster related data would be transmitted to Disaster Support Center and then to concerned agencies.

Grid technology is the emerging technology for integrating geographically apart heterogeneous resources in a collaborative manner. The Figure-1 depicts this:

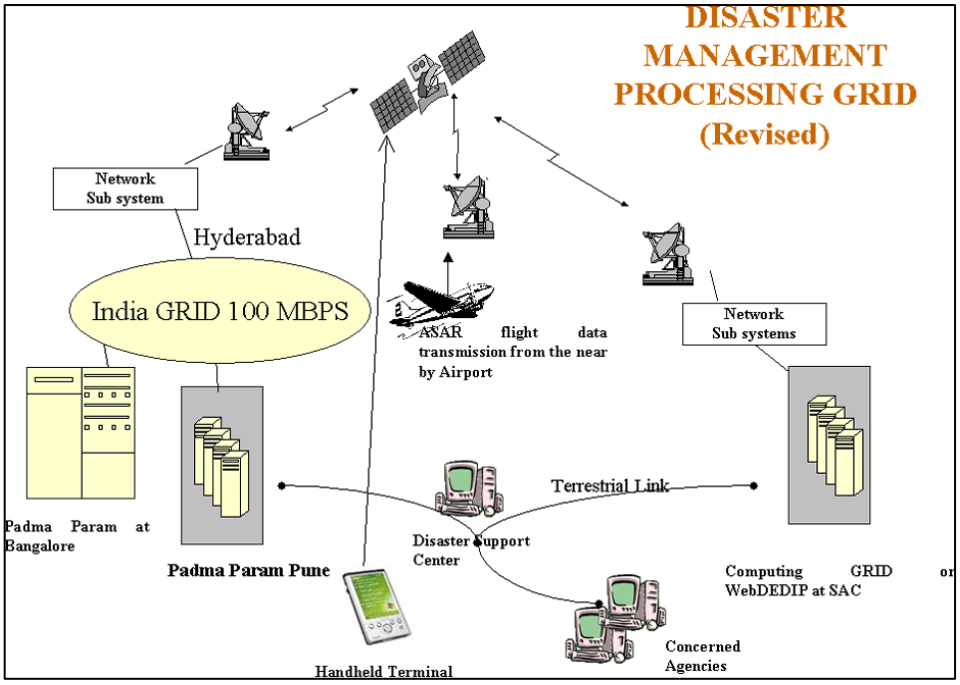

Figure 1: Disaster Management Processing GRID

Satellite link is used to ingest the ASAR flight collected data for disaster-affected area at the Hyderabad server. The data communication will be done using India Grid. The processed Disaster related data would be transmitted to Disaster Support Center and then to concerned agencies via terrestrial links available. Finally deacons will be communicated to concerned people having the hand-held devices.

- **National Spatial Data Infrastructure (NSDI)**

Spatial information has proven its importance in decision making in various areas like managing natural resources, disaster management, flood mitigation, environmental restoration, etc. With the availability of satellite-based remote sensing data and the organization of spatial databases around a Geographical Information Systems (GIS), combined with the Global Positioning System (GPS), the process of semantic spatial information systems has now became a reality. New challenges in standardization of different technology utilization, decision support system, and spatial information visualization and information access are emerged.

It requires the seamless integration of Main, State as district level NSDI data servers. Grid is best solution for this, which will act as data as well as access grid. For this application ISRO will design the NSDI data grid, which consists of:

- Main NSDI Server
- Central level data cluster
- State-level data cluster
- District-level data cluster
- SAC Application Grid

2.2 Grid Technology Development at ISRO

Grid Technology using terrestrial network has been the focus of attention for both technology and application developers. In its manifestation, Grid technology enables the utilization of distributed resources in terms of well-defined Grid services encapsulating the underlying complexity for the user. Satellite communication has proven cost effectiveness for broadcast applications and for the remote users. This is quite generic technology development and will be useful to all communication satellite. The development work is pioneering, challenging and complex.

In extending Grid technology over satellite communication[3], there are certain issues as compared to the terrestrial. The satellite has limited communication bandwidth compared to fiber medium. Propagation delay and channel noise of satellite medium further reduce the speed. So communication time cannot be overlooked while deriving the computing model. Further, the process of locating, acquiring, utilizing and managing communication resources is complex and requires lot of understanding at the user end.

The aim is to develop the framework for satellite-based Compute and Data Grid. Technology development envisages Grid service-oriented architecture for communication resources that encapsulate the communication service within the compute / data service and support Task, Resource and Service level agreement and negotiation.

Efficient Satellite Grid management could be achieved by paralleling Communication with the computation to reduce the effect of limited satellite bandwidth and delay. This way, satellite communication could significantly match the high performance computing requirements of many of the applications. To reduce the effect of satellite BER, data transfer approach on multiple socket connections of GridFTP will be explored. For faster data transfer GridFTP could be ported on UDP. To improve the efficiency of Bent pipe, Dynamic Bandwidth Negotiation Protocol is needed. Grid Command Communication Protocol and Network Establishment Protocol using OBP (On Board Processing) is proposed to encapsulate the complexity associated with the usage of communication resources. Further, to encapsulate the Grid Communication service within the Compute or Data Service so that user accessing the Compute Grid or Data Grid does not bother about the type of underlying communication. It requires co-allocation, co-reservation and co-scheduling of computing and communication resources. To produces the automatic and efficient grid resource management architecture for communication resources, the mechanism for resource registration, discovery, matchmaking, reservation, allocation, utilization, monitoring and load balancing are required.

The development will consists of (1) Communication/Interprocess Protocols, (2) User Environments, (3) Grid Scheduler, (4) Grid Resource Manager, (5) Satellite Resource Manager, (6) Co-allocation manager, (7) Broker, (8) N/W Establisher, (9) Network Management System, (10) Different types of Data Transfer Agents, (11) Grid Service Manager, etc .

2.3 Requirement of the Project

Satellite can be useful as communication resource in Grid technology, whereas Grid is useful in effective satellite resource management. Grid protocols are normally written over TCP/IP. Communication satellite has wrappers that provide compatibility to TCP/IP.

Standard TCP does not scale well in high bandwidth, large round time trip networks. TCP performance depends upon the product of the transfer rate and the round-trip delay. TCP survived the days of low bandwidth, high latency and high error rates. But for several reasons it is today not able to cope efficiently with the evolving new environment such as bulk data transfers in grid environment involving high latency, high BW and low loss paths. Development and testing of UDP based reliable protocols are being considered to suite Grid and communication satellite environment. FTP application will be developed for selected protocol.

2.4 Introduction of my Project

Satellite can be useful as communication resource in Grid technology, whereas Grid is useful in effective satellite resource management. Grid protocols are normally written over TCP/IP.

Standard TCP does not scale well in high bandwidth, large round time trip networks. TCP performance depends upon the product of the transfer rate and the round-trip delay. TCP survived the days of low bandwidth, high latency and high error rates. But for several reasons it is today not able to cope efficiently with the evolving new environment such as bulk data transfers in grid environment involving high latency, high BW and low loss paths. We planned to work out protocols to suite Grid and communication satellite environment.

Both literature survey and simulation experiments studies demonstrate that SABUL is suitable for grid data transfer under GSAT-4 Applications Project. But SABUL does not support FTP application, which is required in our project to transfer files. So we are on the move to develop FTP application for SABUL protocol – UDP based protocol.

SABUL protocol, Tsunami protocol, FTP, GridFTP and Globus-XIO are needed to be studied for this dissertation work. The issues with Simple Available Bandwidth Utilization Library (SABUL) protocol for its use in the existing application are needed to be clearly understood. SABUL, UDP based protocol which uses TCP for control information exchange and UDP for data information exchange.

There are three approaches to develop this FTP application for SABUL:

1) to develop driver interface for SABUL in GridFTP XIO,
2) to change data connection of FTP from TCP to UDP and
3) to develop SABUL-FTP application.

In third semester, third approach was adopted and few FTP commands were simulated. This approach was adapted to understand SABUL connection in detail. In fourth semester, FTP application using approach-1 is developed.

Chapter 3

PROBLEM DESCRIPTION

3.1 Issues

The two traditional transport protocols TCP and UDP both have limitations when it comes to the transport of sustained high bandwidth data flows across shared high speed networks. High performance transport protocols that perform efficiently at high BDP and high BER environment are required to be developed and tested.

A lot of efforts are going on globally for bulk data transfer in such networks. To solve these problems two main approaches[5] are being followed: one focuses on modification of TCP and specifically the AIMD algorithm, and other on totally new transport protocols. This is a very active research area in networks.

New approaches fall in two categories i.e. TCP variants and UDP variants. Some examples of the TCP variants include HSTCP, Scalable TCP, Fast TCP, XCP, CADPC/FTP, GridFTP. Some examples of UDP variants include RBUDP, Tsunami, SABUL, UDT etc. As per literature survey, all these protocols perform well at high BW and latency (around 100 m.sec.) , low BER (optical terrestrial networks), but none have been critically evaluated for efficient performance at high delay, high BW and high BER environments .

In phase one, our performance criteria will be to test and analyze some new protocols in lab. in a simulated environment using emulator tools like NISTNET. Protocols will be tested mainly for high BW utilization in wide band, high latency and high BER environment. In addition to high BW utilization, other criteria will be throughput consistency w.r to BER. In addition we will also look into aspects like TCP friendliness, FTP syntax compatibility etc.

3.2 Goal and Objective

For grid data transfer under GSAT-4 Applications Project we tested and compared the performance of different type of protocols. This is required, as these protocols have not been critically tested in satellite environment, which has delay as high as 500 m.sec and high BER. A satellite emulator environment is already configured in laboratory to test and compare the performance of different type of protocols, in addition to testing with satellite. Final protocols like GridFTP, GridTCP and SABUL/UDT are selected for implementation based on their meeting the sustained high bandwidth utilization. FTP application will be developed for SABUL/UDT.

Chapter 4

LITERATURE STUDY

4.1 Grid Computing

4.1.1 Overview

Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self-managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

4.1.2 Capabilities of Grid Computing

The most important capabilities of Grid Computing are:

- **Exploiting underutilized resources**

The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid. There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application.

Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage.

- **Parallel CPU capacity**

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others.

- **Virtual Resources and Virtual organizations for collaboration**

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. Distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources.

The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid.

- **Access to Additional Resources**

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity.

- **Resource Balancing**

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization.

- **Management**

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity

and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization. The grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resource hardware and the expenses associated with it.

4.1.3 Resources in Grid Computing

A grid is a collection of machines, sometimes referred to as “nodes,” “resources,” “members,” “donors,” “clients,” “hosts,” “engines,” and many other such terms. They all contribute any combination of resources to the grid as a whole. All users of the grid may use some resources while others may have specific restrictions.

The various resources of the Grid are:

- **Computation**

The most common resource is computing cycles provided by the processors of the machines on the grid. The processors can vary in speed, architecture, software platform, and other associated factors, such as memory, storage, and connectivity.

- **Storage**

The second most common resource used in a grid is data storage. A grid providing an integrated view of data storage is sometimes called a “data grid.” Each machine on the grid usually provides some quantity of storage for grid use, even if temporary. Storage can be memory attached to the processor or it can be “secondary storage” using hard disk drives or other permanent storage media. Memory attached to a processor usually has very fast access but is volatile. Secondary storage in a grid can be used in interesting ways to increase capacity, performance, sharing, and reliability of data.

- **Communications**

This includes communications within the grid and external to the grid. Communications within the grid are important for sending jobs and their required data to points within the grid. Some jobs require a large amount of

data to be processed and it may not always reside on the machine running the job. The bandwidth available for such communications can often be a critical resource that can limit utilization of the grid. External communication access to the Internet, for example, can be valuable when building search engines. Machines on the grid may have connections to the external Internet in addition to the connectivity among the grid machines.

- **Software and Licenses**

The grid may have software installed that may be too expensive to install on every grid machine. Using a grid, the jobs requiring this software are sent to the particular machines on which this software happens to be installed. When the licensing fees are significant, this approach can save significant expenses for an organization.

- **Special equipment, architecture, policies and capacities**

Platforms on the grid will often have different architectures, operating systems, devices, capacities, and equipment. Each of these items represents a different kind of resource that the grid can use as criteria for assigning jobs to machines. While some software may be available on several architectures, for example, PowerPC and x86, such software is often designed to run only on a particular type of hardware and operating system. Such attributes must be considered when assigning jobs to resources in the grid.

- **Jobs and Applications**

Although various kinds of resources on the grid may be shared and used, they are usually accessed via an executing “application” or “job.” Usually we use the term “application” as the highest level of a piece of work on the grid. However, sometimes the term “job” is used equivalently. Applications may be broken down into any number of individual jobs.

- **Scheduling, Reservation and Scavenging**

The grid system is responsible for sending a job to a given machine to be executed. In the simplest of grid systems, the user may select a machine suitable for running his job and then execute a grid command that sends the job to the selected machine. More advanced grid systems would include a job “scheduler” of some kind that automatically finds the most appropriate

machine on which to run any given job that is waiting to be executed. Schedulers react to current availability of resources on the grid.

In a “scavenging” grid system, any machine that becomes idle would typically report its idle status to the grid management node. This management node would assign to this idle machine the next job that is satisfied by the machine’s resources. Scavenging is usually implemented in a way that is unobtrusive to the normal machine user. If the machine becomes busy with local non-grid work, the grid job is usually suspended or delayed.

4.1.4 Grid Infrastructure Components

Next, we studied the Grid Infrastructure Components. The various Grid Infrastructure Components are:

- **Security**

Security is an important component in the grid-computing environment. If you are a user running jobs on a remote system, you care that the remote system is secure to ensure that others do not gain access to your data. If you are a resource provider that allows jobs to be executed on your systems, you must be confident that those jobs cannot corrupt, interfere with, or access other private data on your system.

- **Resource Management**

The grid resource manager is concerned with resource assignments as jobs are submitted. It acts as an abstract interface to the heterogeneous resources of the grid. The resource management component provides the facilities to allocate a job to a particular resource, provides a means to track the status of the job while it is running and its completion information, and provides the capability to cancel a job or otherwise manage it.

- **Information Services**

Information services are a vital component of the grid infrastructure. It maintains knowledge about resource availability, capacity, and current

utilization. Within any grid, both CPU and data resources will fluctuate; depending on their availability to process and share data. As resources become free within the grid, they can update their status within the grid information services.

- **Data Management**

When building a grid, the most important asset within your grid is your data. One has to determine his/her data requirements and how he/she will move data around the infrastructure or otherwise access the required data in a secure and efficient manner.

Above, are the various basic components required for the development of any Grid Computing Environment. The Globus Toolkit provides the set of basic services that map these components into Globus Toolkit.

4.2 UDP based Protocols

As network bandwidth and delays increase, TCP becomes inefficient. These problems are due to slow loss-recovery, a RTT bias inherent in its AIMD congestion-control algorithm, and the bursting data flow caused by its window control. Data-intensive applications over high bandwidth delay product (BDP) networks, such as computational grids, need new transport protocols to support them.

To meet this requirement, different application-level protocols built above UDP (ie UDP variant) like UDT, SABUL, Tsunami, RBUDP etc[8]. UDP-based Data Transfer protocol, or UDT has a congestion control mechanism that maintains efficiency, fairness and stability, and its application-level nature enables it to be deployed at the lowest cost, without any changes in the network infrastructure or operating systems.

4.2.1 SABUL (Simple Available Bandwidth Utilization Library) Protocol

Both literature survey and simulation experiments studies demonstrate that SABUL is suitable for grid data transfer under GSAT-4 Applications Project.

SABUL[6], an application-level data transfer protocol for data-intensive applications over high bandwidth-delay product networks. SABUL is designed for reliability, high performance, fairness and stability. It uses UDP to transfer data and TCP to return control messages. A rate-based congestion control that tunes the inter-packet transmission time helps achieve both efficiency and fairness. In order to remove the fairness bias between flows with different network delays, SABUL adjusts its sending rate at uniform intervals, instead of at intervals determined by round trip time. This protocol has demonstrated its efficiency and fairness in both experimental and practical applications. SABUL has been implemented as an open source C++ library, which has been successfully used in several grid computing applications.

Rate-Control Algorithm[7] Rate control is seen as a way to control the burstiness so often observed in TCP flows. This burstiness may cause losses as router queues suddenly fill up and packets are dropped or network interface cards cannot keep up. Also, rate control allows a flow to more quickly fill a pipe without going through the initial ramping-up process characteristic of TCP. Rate control or inter-packet delay, which adjusts to packet loss and/or network congestion as reported by the receiver, has been added in some form to all the UDP protocols to counter the charges of unfair use of capacity and potential to create network problems.

SABUL begins with a preset IPD (inter-packet delay) of 10 usec which it converts to CPU cycles. The receiver generates a SYN packet based on a timed interval (200ms) which signals the sender to use both the number of lost packets and the number of packets--including retransmits--sent since

the last SYN time to calculate a current loss rate. This loss rate is then input to a weighted moving average formula to give a history-rate. If the history-rate is greater then a preset limit (.001), the IPD is increased; if less than the limit, the IPD is decreased; if equal, .1 is added. In a former release, SABUL attempted to keep the loss rate between an upper and lower limit. The latest implementation is similar in concept to TSUNAMI's in that both keep the delay between blocks/packets between an upper and lower limit rate implementation. SABUL is the one to implement the IPD (inter-packet delay) between individual packets as opposed to groups of packets. The delay is implemented by repeated calls to rtdsc( ) until the requisite number of clock cycles have passed. FOBS checks the sending rate after a burst of packets(25) and implements the delay with a gettimeofday calculation until time to send the next burst. TSUNAMI uses a timed select() to implement the delay between blocks of data.

4.2.1.1 SABUL General Architecture

As shown in Figure-2, SABUL uses two connections: the control connection over TCP and the data connection over UDP. A SABUL connection is uni-directional: data can be only sent from one side to the other side. According to the relationship between the two sides involved in SABUL, we call them the sender (side) and the receiver (side), respectively. The sender initializes the connection and waits for the receiver to connect to it and then constructs the control connection. The data connection is built up following a successful control connection.

Data flow is from the sender to the receiver only. The control information is from the receiver to the sender only, so we also call the control information as feedback.

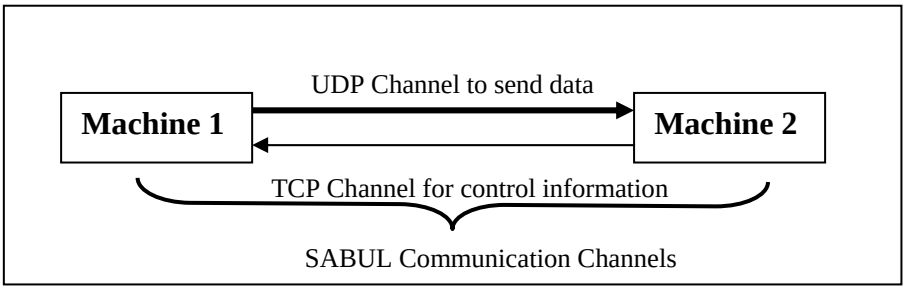

Figure 2: Overview of SABUL Architecture

As shown in Figure-3, the sender manages application buffer and is responsible for its transmission, retransmission, and release according to the feedback from the receiver. It manages a queue to record the lost packets. The receiver reorders the packets according to the sequence number and puts them into its own buffer or the application buffer. A flag array is kept for packet reordering and loss detection. The sequence numbers of the lost packets number are kept in a lost list.

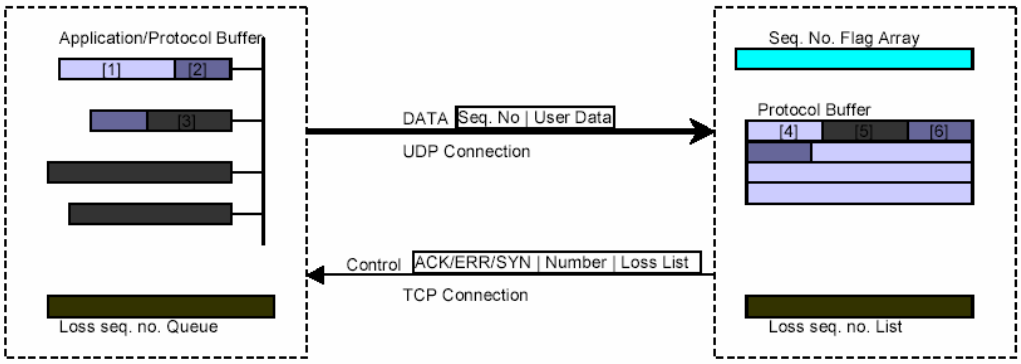

Figure 3: SABUL Communication

4.2.1.2 Packet Formats

There are four kinds of packets (Figure-3) in SABUL. The user data is packed in DATA packet and sent through UDP connection from the sender to the receiver. The other three are feedback control information from the receiver to the sender through TCP connection. ACK packets are positive acknowledgement telling the sender that the receiver has received all the packets till the sequence number carried in the ACK packet. ERR packets are

negative acknowledgement that carry the sequence numbers of the lost packet. In addition, an SYN packet is a synchronization packet with the packet-receiving rate and triggers a rate control events in the sender side.

4.2.1.3 Data Sending/Receiving Algorithm

Data Sending Algorithm

The sender has two basic functionalities: to send out packet periodically according to the rate control mechanism and to process the feedback from the receiver. The retransmission of a lost packet has higher priority than the first-time sent packet, otherwise the unacknowledged packets will be kept in the buffers of both sides and prevent further packet transmission because the buffers have limited lengths. The processing of feedback should be synchronized with data sending. In addition, the sender is responsible for timer expiration detection.

- **Data Structure**:
    1. The protocol buffer is linked list whose nodes are the head pointers of application buffers.
    2. The loss queue is a FIFO queue that stores the sequence numbers of the lost packets from ERR feedbacks.

- **Algorithm:**

STEP 1. Initialization:
STEP 2. Get current time;
STEP 3. Poll Control Channel. Receive control packet and process it if there is any. Otherwise, calculate the time passed since the last ACK or ERR feedback packet was received, and if the interval is greater than the time expiration threshold, generate an EXP, event and process it;
STEP 4. If the loss queue is empty, GOTO STEP 5. Otherwise, read (and remove) a sequence number and retransmit the proper packet, then GOTO STEP 6;
STEP 5. If the protocol buffer is not empty and the difference between

the largest sent sequence number and the last acknowledgement number is less than a valve, send a new packet and update the most recent sequence number;

STEP 6. Block the sending for an inter-packet time interval and GOTO STEP 2.

Data Receiving Algorithm

The receiver is responsible to reorder the received packets, detect the loss, and feedback error, acknowledgement and synchronization messages.

- **Data Structure:**
  1. The protocol buffer is a logically circular memory block for temporally storing and reordering the received packets.
  2. The offset array is a flag array that records the packet with which offset (difference between the sequence number and the last acknowledgement number) is received or not.
  3. The loss sequence number list is a linked list whose node is a structure of a lost sequence number and the last error report time of the sequence number.

- **Algorithm:**

  STEP 1. Initialization:

  STEP 2.

  a. Get current time;

  b. If the time passed since the last acknowledgment event exceeds the ACK interval, or the user registered buffer has been fulfilled, generate an ACK packet;

  c. If the time passed since the last error feedback event exceeds the ERR interval, and the loss list is not empty, generate an ERR packet;

  d. If the time passed since the last synchronization event exceeds the SYN interval, generate an SYN packet;

STEP 3. Start a timed receive; if the timer is expired, GOTO STEP 2;

STEP 4. Read the sequence number in the data packet, and calculate the offset since the last acknowledged sequence number;

a. If the offset is greater than or equal to the sequence number array size, generate an ACK packet, and GOTO STEP 2;

b. If the offset is less than 0, GOTO STEP 2;

c. If the offset is greater than expected offset, insert sequence number from current largest received sequence number to the current received sequence number to the loss list, generate an ERR feedback.

STEP 5. Update the next expected offset number, which is the smallest number since the most recent offset (current) in offset flag that hasn't been set;

STEP 6. Update the largest received sequence number;

STEP 7. Set the proper flag in the offset array, and GOTO STEP 2

4.2.2 TSunami Protocol

The goal of Tsunami[9] is to develop high performance file transfer protocol (running as a user space application) to transfer files faster in high-speed networks than that appears possible with standard implementations of TCP. Tsunami uses UDP for data transfer and TCP for transferring control information. The UDP datagram size is negotiated during the connection setup. The length of the file that is to be transferred is also exchanged during the negotiation phase. Single thread handles both network and disk activity at the sender side whereas the receiver employs separate threads for disk and network activities. Receiver periodically makes retransmission request and retransmissions have higher priority than normal sends. Receiver periodically updates the error rate (calculated based on the number of retransmissions in an interval and previous error rate) to the sender and the sender adjusts its inter-packet delay based on this value. Receiver sends a complete signal after receiving all the datagrams. Tsunami allows the user to configure parameters such as size of the datagram, the threshold error rate

(used to adjust sending rate), size of retransmission queue and acknowledgement interval.

4.2.3 UDT Protocol

UDT (UDP–based Data Transfer)[11,12] brings reliability and congestion control to UDP as shown in Figure 4. It uses packet-based sequencing (i.e., the sequence number is increased by 1 in MTU size for each packet, including all headers). Selective positive acknowledgement (ACK) is sent at every constant interval, whereas negative acknowledgement (NAK) is generated as soon as packet loss is detected. Congestion control combines rate-based and window-based control mechanisms to ensure efficiency and fairness, as well as TCP friendliness and delay independence.

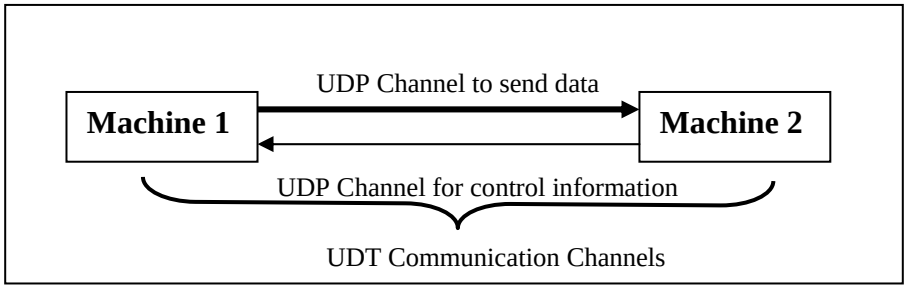

Figure 4: Overview of UDT Architecture

Rate control tunes the inter-packet time at every constant interval, which is called SYN. The value of SYN is 0.01 seconds, an empirical value reflecting a tradeoff among efficiency, fairness and stability. For every SYN time, when the packet loss rate during the last SYN time is less than a threshold, the maximum possible link Bit Error Rate (BER), the number of packets that will be sent in the next SYN time is increased by:

$$inc = \max(10^{\lceil \log_{10}((B-C)\times MTU\times 8)\rceil}\times\beta/MTU,\ 1/MTU)$$

where B is the estimated bandwidth and C is the current sending rate, both in number of packets per second. β is a constant value of 0.0000015. MTU is the maximum transmission unit in bytes, which is the same as the UDT packet size. The inter-packet time is then recalculated using the total

estimated number of sent packets during the next SYN time. The estimated bandwidth B is probed by sampling UDT data packet pairs.

The inter-packet time is increased by 1/8 (or equivalently, the sending rate is decreased by 1/9) when the sender receives a NAK packet whose largest lost sequence number is greater than the largest sent sequence number when the last decrease occurs, or the number of NAKs since the last rate decrease has exceeded an increasing threshold. No data is sent out during the next SYN time after a rate decrease.

UDT uses a flow-control window to limit the amount of unacknowledged packets. The UDT receiver calculates the packet arrival rate (AS) when it is time to feed back the ACK by using a median filter on the recent packet arrival intervals it recorded. It then attaches AS within the ACK packet. On the sender's side, if an ACK is received and the AS value is greater than 0, the size of the flow window, W, is updated as:

$$W = W * 0.875 + (RTT + SYN) * AS * 0.125$$

Rate control is used to obtain fast bandwidth discovery, fast loss-recovery, and intra-protocol fairness. Flow-control helps to reduce packet loss and oscillation and it helps to avoid congestion collapse. They both contribute to TCP friendliness.

During congestion, loss reports from the receiver can be dropped or delayed. When the sender continues to send new packets, it worsens the congestion. Flow control prevents this from happening. By reducing packet loss, flow control also helps to improve efficiency and fairness. Because a low loss-rate reduces the frequency of sending-rate decreases, it makes UDT less aggressive.

Rate control decides the throughput only when there are a small number of concurrent sources. As RTT and the number of concurrent flows increase, flow control limits the throughput. In such situations, flow control helps to reduce the oscillations that would be higher due to the high rate increase per RTT.

The flow window size starts from 2 as a slow start, and is updated to a number of acknowledged packets when the sender receives an ACK. Slow start ends when either the sender receives a NAK or it reaches the maximum window size, after which the congestion-control algorithm begins working. The inter-packet time is 0 during the slow start phase. It is set to the packet arrival interval when slow start ends.

4.3 Globus Toolkit

The Globus Project is developing the fundamental technologies needed to build these computational grids. Globus research focuses not only on the issues associated with building computational grid infrastructures, but also on the problems that arise in designing and developing applications that use grid services.

The Globus Project provides software tools that make it easier to build computational grids and grid-based applications. These tools are collectively called the Globus Toolkit. The Toolkit is used by many organizations to build computational grids that can support their applications. The composition of the Globus Toolkit can be pictured as the following three pillars. Security is the foundation common to all three pillars as shown in Figure-5.

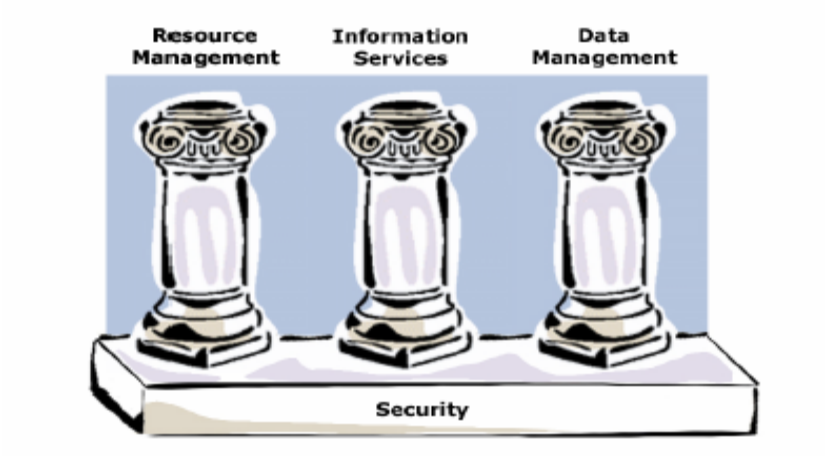

Figure 5: Composition of the Globus Toolkit

The first pillar of the Globus Toolkit provides Resource Management, which involves the allocation of Grid resources. It includes such packages as the Globus Resource Allocation Manager (GRAM) and Globus Access to Secondary Storage (GASS). The second pillar of the Globus Toolkit is for Information Services, which provide information about Grid resources and are the focus of this document. Such utilities include the Monitoring and Discovery Service (MDS), which provides the Grid Resource Information Service (GRIS) and the Grid Index Information Service (GIIS).

The third pillar of the Globus Toolkit is for Data Management, which involves the ability to access and manage data in a Grid environment. This involves such utilities as GridFTP and globus-url-copy, which are used to move files between grid enabled devices.

In the context of the Globus Toolkit, Information Services have the following requirements:

- Access to static and dynamic information regarding system components
- A basis for configuration and adaptation in heterogeneous, dynamic environments
- Uniform, flexible access to information
- Scalable, efficient access to dynamic data
- Access to multiple information sources
- Decentralized maintenance

As part of this information infrastructure, MDS provides directory services for grids using the Globus Toolkit. MDS uses an extensible framework for managing static and dynamic information about the status of a computational grid and all its components: networks, compute nodes, storage systems, and instruments.

4.3.1 An overview of Globus components

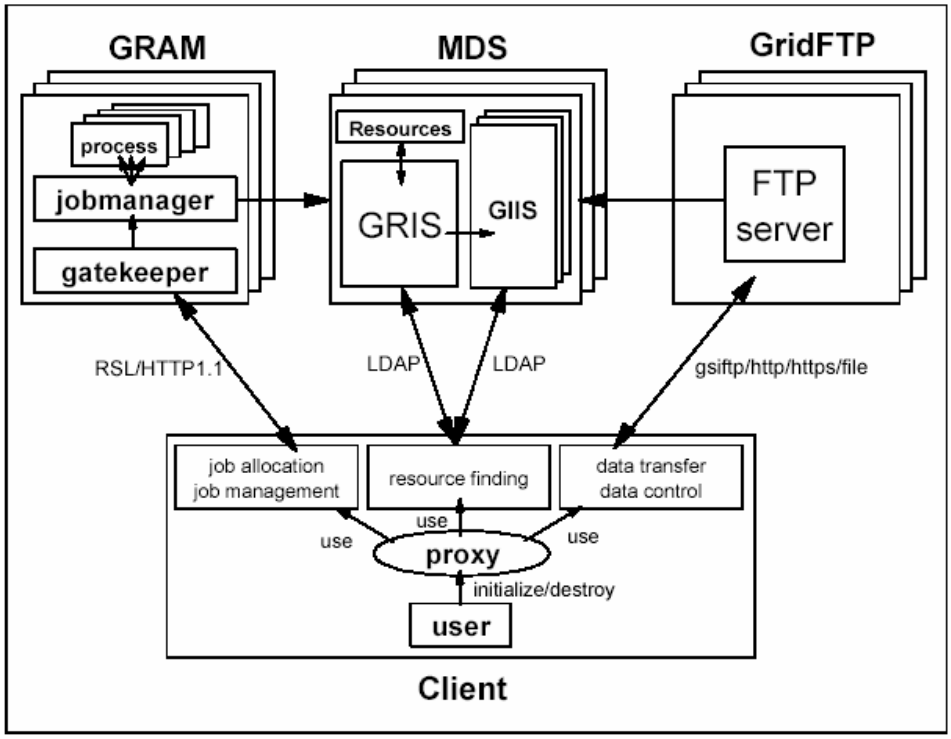

Figure 6: An overview of Globus components

4.3.2 How Globus Components Interacts

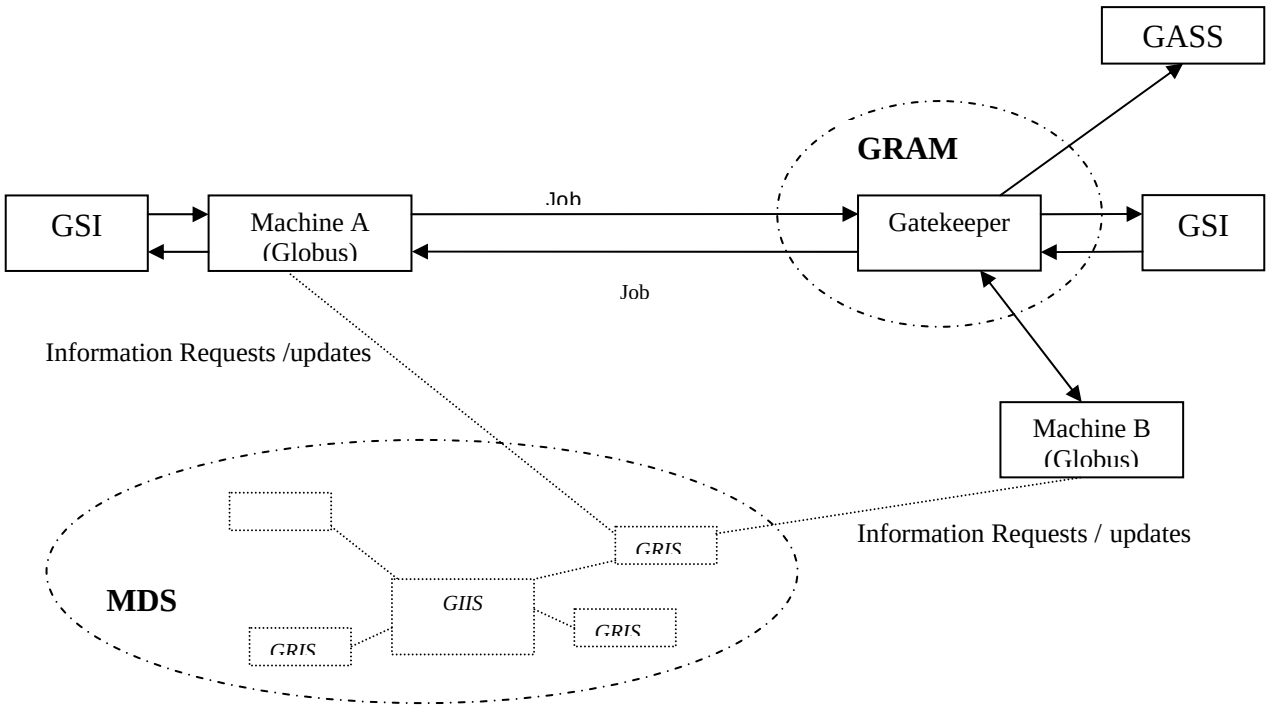

Figure 7: Interaction of Globus components

Keywords:

GSI : The GSI controls all the security arrangements (authorization etc)
MDS : The MDS is in charge of distributing information about machines
Gatekeeper : The gatekeeper controls interactions between machine and the jobs
GASS : The GASS server can be set up to provide access to other files during jobs
GRAM : Globus Resource Allocation Manager

4.4 GridFTP

GridFTP[13] provides a secure and reliable data transfer among grid nodes. The word GridFTP can refer to a protocol, a server, or a set of tools. This protocol is intended to be used in all data transfers on the grid. It is based on FTP, but extends the standard protocol with facilities such as multistreamed transfer, auto-tuning, and Globus based security.

As the GridFTP protocol is still not completely defined, Globus Toolkit does not support the entire set of the protocol features currently presented. A set of GridFTP tools is distributed by Globus as additional packages.

4.4.1 GridFTP server and client

Globus Toolkit provides the GridFTP server and GridFTP client, which are implemented by the in.ftpd daemon and by the globus-url-copy command, respectively. They support most of the features defined on the GridFTP protocol. The GridFTP server and client support two types of file transfer: standard and third-party. The standard file transfer is where a client sends the local file to the remote machine, which runs the FTP server. An overview is shown in Figure 8.

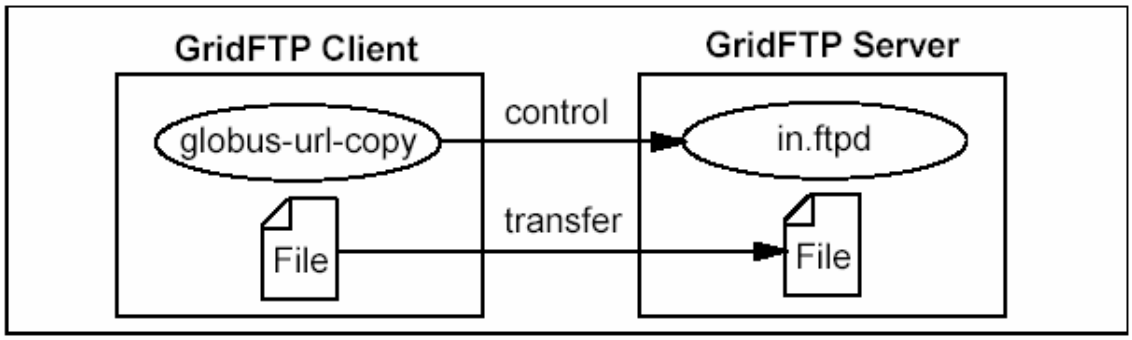

Figure 8: Standard File Transfer

Third-party file transfer is where there is a large file in remote storage and the client wants to copy it to another remote server, as illustrated in Figure 9.

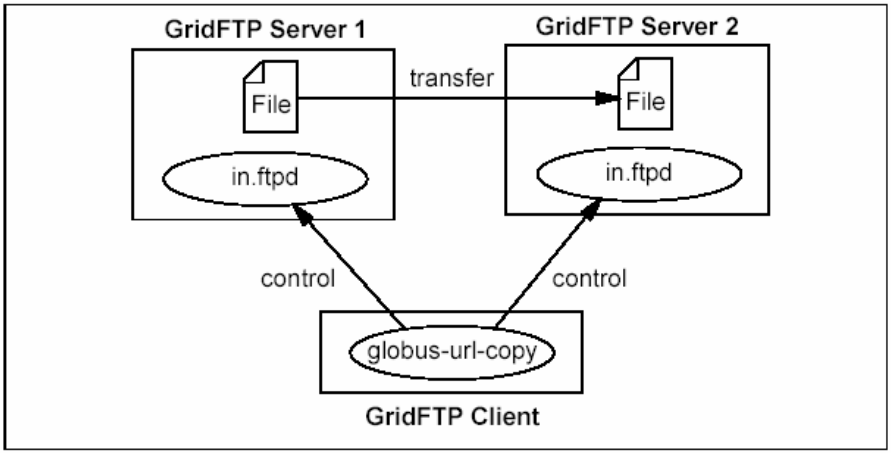

Figure 9: Third party file transfer

4.4.2 GridFTP tools

Globus Toolkit provides a set of tools to support GridFTP type of data transfers. The gsi-ncftp package is one of the tools used to communicate with the GridFTP Server.

The GASS API package is also part of the GridFTP tools. It is used by the GRAM to transfer the output file from servers to clients.

4.5 Globus XIO

The external view of the Globus XIO[15] architecture.

- Graphical Overview
- Framework
- Drivers
- Example
- Driver Interface

4.5.1 Graphical Overview

Globus XIO is broken down into two main components, framework and drivers. The following Figure 10 illustrates the architecture:

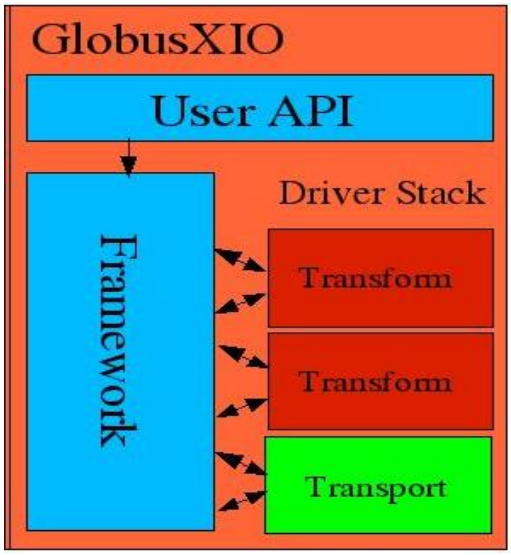

Figure 10: Globus XIO Architecture

4.5.2 Framework

The Globus XIO framework manages IO operation requests that an application makes via the user API. The framework does not work to deliver the data in an IO operation nor does it manipulate the data. All of that work is done by the drivers. The framework's job is to manage requests and map them to the drivers' interface. It is the drivers themselves that are responsible for manipulating and transporting the data.

4.5.3 Drivers

A driver is the component of Globus XIO that is responsible for manipulating and transporting the user's data. There are two types of drivers:

- **Transform drivers** manipulate the data buffers passed to it via the user API and the XIO framework.

- **Transport drivers** are capable of sending the data over a wire.

Drivers are grouped into stacks, that is, one driver on top of another. When an IO operation is requested by the Globus XIO, the framework passes the operation request to every driver in the order they are in the stack. When the bottom level driver (the transport driver) finishes shipping the data, it passes the data request back to the XIO framework. Globus XIO then delivers the request back up the stack in this manner until it reaches the top, at which point the application is notified that the request is complete.

In a driver stack there can only be **one** transport driver. The reason for this is simple. The transport driver is the one responsible for sending or receiving the data. Once this type of operation is performed, it makes no sense to pass the request down the stack, the data has just been transfered. It is now time to pass the operation back up the stack. the There can be many transform drivers in any given driver stack. A transform driver can manipulate requested operations as they pass. Some good examples of transport drivers are security wrappers and compression. However a transport driver can also add additional protocols. For example, a stack could consist of a TCP transport driver and an HTTP transform driver. The HTTP driver would be responsible for marshalling the HTTP protocol and the TCP driver would be responsible for shipping that protocol over the wire.

4.5.4 Example

The following picture illustrates a user application using Globus XIO to speak the GridFTP protocol across a TCP connection:

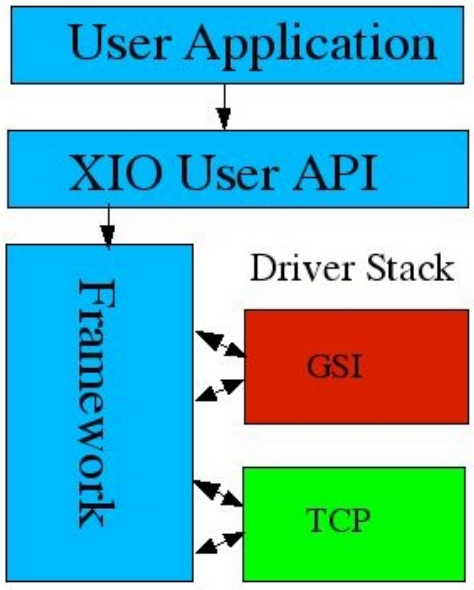

Figure 11: Globus XIO Example

The user has built a stack consisting of one transform driver and one transport driver. TCP is the transport driver in this stack, and as all transport modules must be, it is at the bottom of the stack. Above TCP is the GSI transform driver which performs necessary messaging to authenticate a user and the integrity of the data.

The first thing the user application does after building the stack is call the XIO user API function globus_xio_open().

The Globus XIO framework creates internal data structures for tracking this operation and then passes the operation request to the GSI driver.
The GSI driver has nothing to do before the underlying stack has opened a handle so it simply passes the request down the stack. The request is then passed to the TCP driver.

The TCP driver then executes the socket level transport code contained within to establish a connection to the given contact string.
Once the TCP connection has been established the TCP driver will notify the XIO framework that it has completed its request and thereby the GSI driver

will be notified that the open operation it had previously passed down the stack has now completed.

At this point the GSI driver will start the authentication processes (note that at this point the user does not yet have an open handle). The GSI driver has an open handle and upon it several sends and receives are performed to authenticate the connection. If the GSI driver is not satisfied with the authentication process it closes the handle it has to the stack below it and tells the XIO framework that it has completed the open request with an error.

If it is satisfied it simply tells the XIO framework that it has completed the open operation. The user is now notified that the open operation completed, and if it was successful they now have an open handle. Other operations work in much the same way:

4.5.5 Driver Interface

There is a well-defined interface to a driver. Drivers are modular components with specific tasks. The purpose of drivers in the Globus XIO library is extensibility. As more and more protocols are developed, more and more drivers can be written to implement these protocols. As new drivers are written they can be added to Globus XIO as either statically linked libraries or dynamically loaded libraries. In the case of dynamic loading, it is not even necessary to recompile existing source code. Each driver has a unique name according to the Globus XIO driver naming convention. A program simply needs to be aware of this name (this can obviously be passed in via the command line) and the Globus XIO framework is responsible for loading that driver.

4.5.6 Benefits

Globus XIO provides a framework that implements a simple Open/Close/Read/Write interface that is appropriate for most byte stream

oriented IO applications. The framework user API presents a uniform API, semantics, and error handling to the application. Variations in programming models due to the different underlying protocols and APIs are isolated in drivers and drivers may be “stacked” to get composite functionality.

The primary benefits to be gained from using Globus XIO include:

1. A uniform IO API that developers can learn and be comfortable with.
   A single, rational set of APIs, semantics, error responses, as well as problems and work around. Differences due to different network protocols or data access methods are isolated in the drivers making the application more stable and minimizing the programmer involvement with these differences.

2. Ability to quickly adapt to different transport protocols.
   Applications written with XIO can switch from TCP to other protocols with minimal changes to the application.

3. Ability to quickly adapt to different data access mechanisms.
   Files stored behind GridFTP servers, on HPSS or UniTree mass storage devices or any number of other vendor proprietary storage systems can all be accessed in uniform manner. This also provides providers of these access mechanisms a convenient vehicle for providing this access and immediately makes their access mechanisms available to any application utilizing XIO.

4. Provides a convenient framework for research and development.
   Researchers in new protocols and data access mechanisms can utilize the XIO framework during their research and development cycles. The XIO framework provides many of the facilities required to successfully conduct their research, but outside their interest and expertise.

Chapter 5

PROTOCOL FINALISATION

5.1 Related Work

Network researchers have been improving TCP for many years resulting in a series of TCP variations e.g. XCP (Explicit Congestion Control Protocol), NetBLT (Network Block Transfer). However none of these protocols are widely used today for grid applications; rather grid applications today generally use TCP parameter tuning and /or parallel TCP streams. In particular, parameter controlling the buffer size or maximum window size are often changed since the default values are usually too small for high BDP links. Unfortunately parameter tuning typically provides only modest improvements in throughput, and does not solve the fundamental problem of TCP congestion control.

Parallel TCP implementation such as Grid FTP and P-sockets increase throughput by using multiple parallel TCP connections. Unfortunately in practice, parallel TCP usually requires extensive tuning, which can be quite labor intensive. In addition parallel TCP exhibits performance and fairness shortcomings in lossy wide area networks.

Rate based protocols have been regarded as a better solution for congestion control than window based protocols. This idea can be found in NETBLT, VMPT (Versatile Message Transaction) and more recently SABUL (Simple Available Bandwidth Utilization Library) and Tsunami. SABUL uses UDP with a rate based congestion control mechanism.

Some of these approaches are compared below and will be realised and tested in laboratory in a simulated environment

- Comparison of TCP variants: Given in Appendix-A
- Comparison of UDP variants: Given in Appendix-B

- **Realization Approaches**

  Some of the approaches explained above will be tested in the lab. using the satellite emulator. These are TCP/IP (Default window), SABUL, GridFTP, TCP/IP (window scaling) and some others. Out of these tests have been conducted on TCP/IP with default window & window scaling, SABUL. Test results are presented in this document.

- **Performance measurement**

  The performance measurement parameter is mainly throughput in both lab. and satellite configurations. For measuring throughput, files of different sizes from 100 MB to 1 GB will be transferred from sender to receiver and time noted to transfer all the bytes. Throughput is calculated by dividing file size in bits with time taken in seconds to transfer the file.

5.2 Test plan

Testing will be done as follows:

- Testing in lab. with NISTNET.
- Testing through satellite.

For testing in lab, a test setup will be configured consisting of sender, receiver and satellite emulator to introduce satellite path delay and BER. For emulating delay and packet loss between sender and receiver, ‘ NISTNET ‘ software package will be used.

The test setup is shown in Figure 12. Both sender and the receiver consist of Pentium PCs installed with Redhat Linux version 9.0. Both PCs have inbuilt 100Mbps Ethernet cards and are connected through

this interface. For simulating path delay and packet drops between the sender and the receiver, NISTNET is used which can be installed on the sender PC as shown in Figure 12.

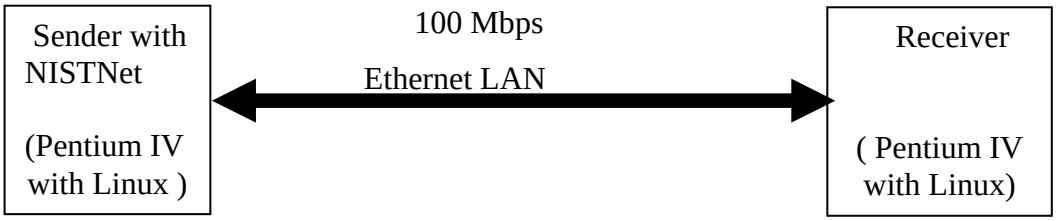

Figure 12: Test setup in lab. Environment

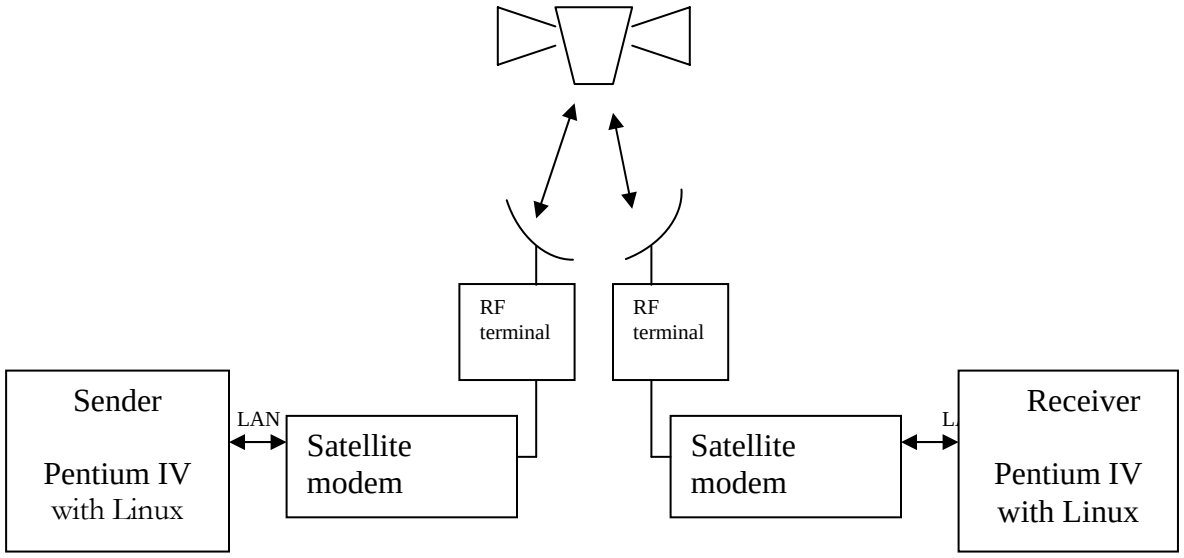

Figure 13: Test setup with Satellite

After testing the protocols in lab. with satellite emulator, these will be tested through satellite as shown in Figure 13.

5.3 Discussion & Analysis

- **Test measurements**

Test measurements have been taken in laboratory using the experimental set-up as shown in Figure 12. Different set of measurements were taken using TCP/IP with default window size, TCP/IP window scaling, SABUL and Tsunami. Different file sizes, RTTs and packet drops were used to emulate the satellite environment. Table-1 given below shows all these measurements.

Table 1. 0
Performance Analysis of protocols
File size - 1 GB, IPERF BW – 75.5 Mbps

Protocol	Delay	BER	Bandwidth Utilization
TCP/IP	500	10^{-6}	< 1 %
TCP/IP (extension)	500	10^{-6}	< 10%
SABUL	500	10^{-6}	70 %
TSunami	500	10^{-6}	22.40 %

Using standard TCP/IP protocol, throughput decreases from 30 Mbps to .840 Mbps when path delay is increased from 0 to 600 m.sec. Throughput further drops when error is introduced.

Using TCP/IP window scaling, throughput improves from .840 Mbps, for default window size of 65536 bytes to 3.89 Mbps with window size 4194304 bytes and to 5.27 Mbps for window size 134217728 bytes at RTT of 600 m.sec. However these throughput values are much less as compared to the theoretical expected values 67 Mbps and 2.14 Gbps at 600 m.sec. RTT.

SABUL protocol gives much better performance in a similar setup. Throughput measured using SABUL is around 40 Mbps at RTT of 0 to 1000 m.sec through NISTNET. Direct measurement between sender and receiver, without NISTNET, gives throughput around 60 Mbps. TCP/IP gives throughput of 77 Mbps in a similar setup. However, when error is introduced, SABUL throughput drops only marginally at .1% packet drop and by 15 % with 1% packet drop.

Chapter 6

IMPLEMENTATION PLAN OF FTP FOR SABUL/UDT

6.1 Overview of the project

As per literature survey and lab experiments, we found SABUL (Simple Available Bandwidth Utilization Library) –UDP variant as a promising protocol for this project. But SABUL does not provide FTP application. So we need to develop FTP application interface for SABUL. There are three approaches:

1) To develop driver interface for SABUL in GridFTP XIO
2) To change data connection of FTP from TCP to UDP
3) To develop SABUL-FTP application.

To understand these three approaches, the top-level architecture of our entire project is proposed as shown in Figure 14.

6.1.1 SGRM

The goal of SGRM is to facilitate efficient utilization of Satellite Grid resources by collectively managing and brokering computing and satellite communication resources. Negotiation with Workflow Manager of Front-end Environment for optimum resource commitment. Synchronize the availability of communication and compute resource s to complete user's task in defined time and QOS.

Satellite Grid Resource Broker is responsible for match making between user requirements and resource availability. It is also responsible for addresses load balancing (resource utilization optimization) coherence among communication resources and computing resources.

Satellite communication resource manager discovers and maintains the resource information required for communication using satellite. It reserves and allocates the communication resources and maintains the status. It is also responsible for resource optimization (load balancing) among the communication resources.

Grid Resource Manager discovers and maintains the computing resource information. It reserves and allocates the computing resources and maintains the status. It is also responsible for resource optimization (load balancing) among the computing resources.

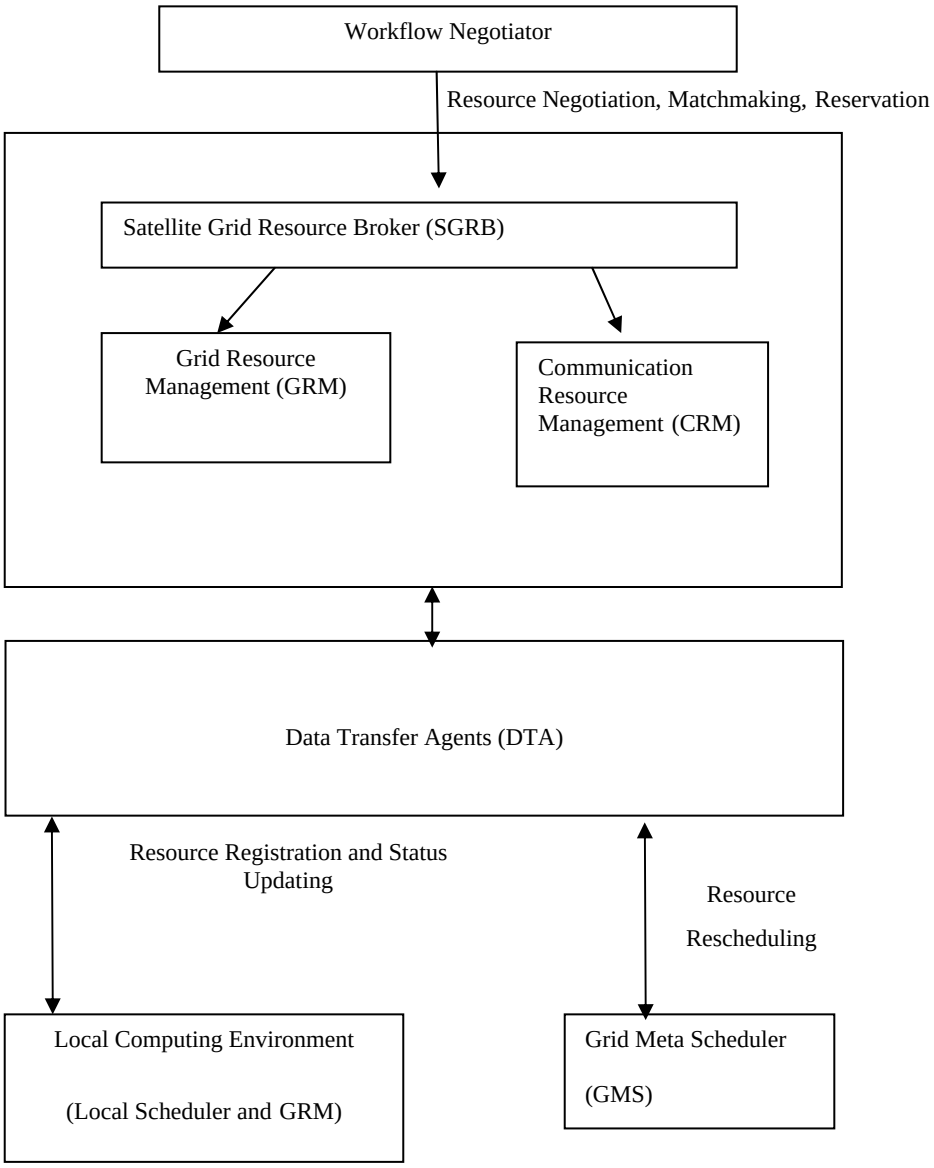

Figure 14: Top-level Architecture

6.1.2 Data Transfer Agents

They are responsible for point-to-point communication, data broadcasting, data multiplexing, data and acknowledgement synchronization over

asymmetric network. Different types of communication protocols will be developed as required by the Front-end Environment.

The prime responsibility of Data Transfer Agent is to transfer data/files to desired processing resources/terminals. DTA will receive notification for the data transfer well in advance by the Scheduler. DTA will examine the parameters like bandwidth, transfer time etc. specified by the scheduler for data transfer and on the bases of these parameters DTA will negotiate the communication resource i.e. satellite channel with Network Establisher. DTA will be designed to support different configuration requirements like point-to-point (unicast) data transfer and Broadcast/multicast data transfer for different service requirements like Round Robin FTP (RR_FTP), Multiple Algorithm Single Data (MASD) respectively.

The prime objective of Data Transfer Agent (DTA) is to transfer data between Grid terminals. The communication requirements will be different like point-to-point (unicast), broadcast/multicast, asymmetric etc. for different applications. DTA interfaces with GANESH(scheduler),Grid Resource Broker (SGRB) and Network Establisher (NE) and the major functions of DTA are as follows:

- Receive request from GANESH(scheduler) and SGRB for resource requirement.
- Handshaking with Network Establisher for requesting satellite channel resource based on resource request received from GANESH and SGRB.
- Transmit data from source terminal to destination terminal using satellite Channel.
- Acknowledgement and response notification protocol with GANESH, SGRB and NE.

Moreover DTA will have decision-making algorithm, which will decide the suitable protocol for data transfer, continue/discontinue the on-going connection and the type of connectivity. This decision-making algorithm

works on the bases of either input received from scheduler or from the performance evaluation results obtained for different protocols. For selection of data transfer protocol, DTA will receive inputs like file size from scheduler. DTA will decide the suitable protocols for data transfer e.g. it will use GridFTP for large file or SABUL FTP/RBUDP FTP for small files. DTA will handshake the next transfer requirement with scheduler during the ongoing connection. If the next transfer request is for same pair of terminals then without disconnecting the satellite connection DTA will use it for the next file transfer. DTA will also decide type of connectivity like point-to-point, broadcast, multicast etc. from the command inputs received from scheduler. DTA has two interface one for FTP and another for GridFTP(See Figure 15)

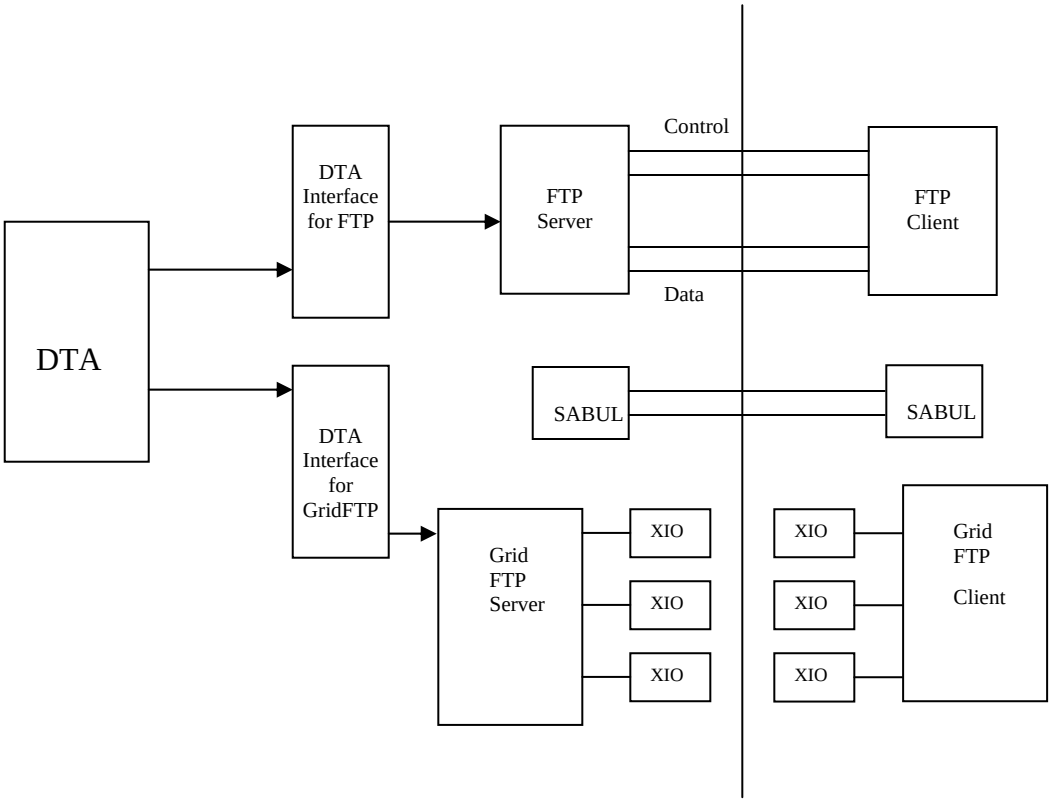

Figure 15: DTA Interface

As shown in Figure 15, three approaches to develop FTP application interface for SABUL is clearly understood.

6.2 SABUL-FTP Application

In the third semester a test SABUL-FTP application had been developed using C++. It implements following FTP commands:

- Connect
- Get
- Put
- Help
- Close

This approach was adopted to understand SABUL-connection in detail. As shown in the above Figure 15, DTA interface for GridFTP was already available. If driver interface for SABUL in GridFTP XIO is developed then I can easily interface with GridFTP DTA interface.

Chapter 7

DEVELOPMENT OF DRIVER INTERFACE FOR SABUL/UDT IN GRIDFTP XIO

7.1 Testing and Performance Evaluation of UDT using Globus-XIO

SABUL (Simple Available Bandwidth Utilization Library) is designed for data-intensive applications in high bandwidth-delay product networks with user level implementation and control. UDT (User-based Data Transfer) is the successor of SABUL. It removes the TCP connection from SABUL and enhances the congestion/flow control. UDT implements slow start and AIMD control scheme for flow.

The Globus Toolkit 4.0.1 includes the UDT driver. This UDT driver is needed to be studied and tested so that our confidence level increases then it will be become easy to develop and test the SABUL driver. If while testing encouraging results are obtained then UDT can also be considered as one of the candidate protocol like SABUL for our project.

With the globus_xio_example.c (Appendix- D Listing-1), we were not able to test UDT driver and only file (IO) driver was tested. So the following code globus_xio_udt_file.c (Appendix-D Listing-2) is used to test UDT driver.

Results are tabulated in Table 2 as shown below.

Table 2.0
Performance Analysis of UDT Protocol using Globus-XIO
Buffer size = 50000 bytes
File size – 1.1 GB, IPERF BW – 75.5 Mbps

Delay (ms)	BER	Throughput (Mbps)	%Bandwidth Utilization
0	0	30	39.74
100	0	14	18.54
100	10^{-6}	2.92	3.87

The above results are not at all accepted in the Grid-Satellite environment where delay can be as high as 500ms. So the program was debugged and it was concluded that throughput can be increased if the disk-access time is we

optimized[17]. This can be achieved by optimizing the buffer size. In the above program the buffer size is 50000 bytes. Test analysis was carried out with different buffer sizes and optimum bandwidth-utilization was found at 40MB buffer-size as shown in Figure16.

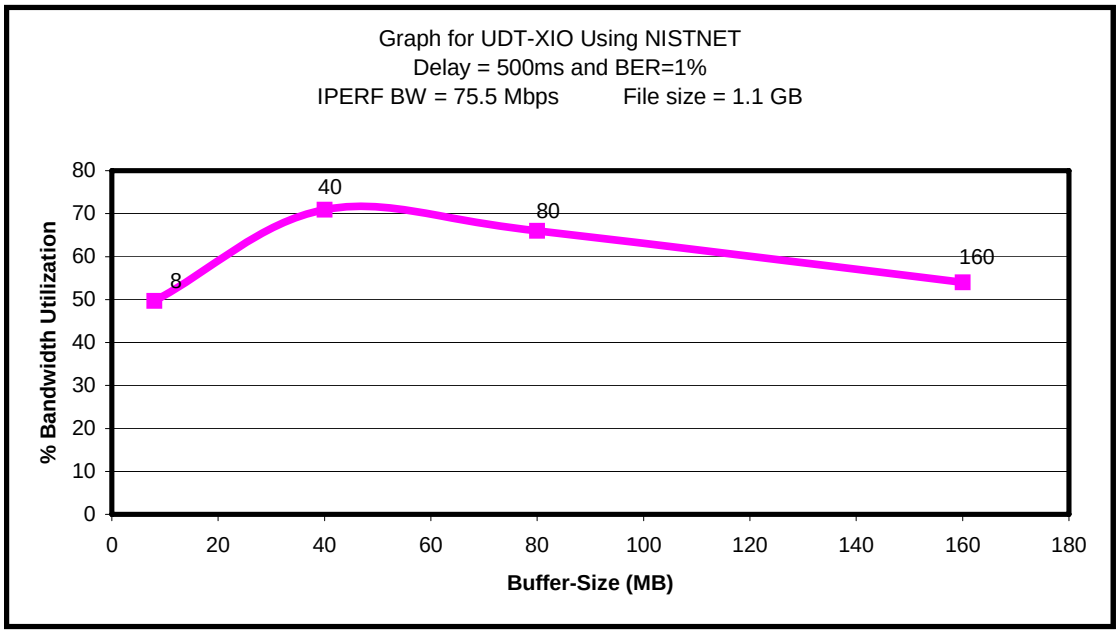

Figure 16: Graph of UDT-XIO for different buffer-sizes

In the globus_xio_udt_file.c program (Appendix-D, Listing 2) buffer-size was increased first to 8MB and then to 40MB by which the performance had increased as shown in Table 3.0 and Table 4.0 respectively. Graphs for Table 2.0 and Table 3.0 are plotted in Figure 17 and Figure 18 respectively.

Table 3.0
Performance Analysis of UDT Protocol using Globus-XIO
Buffer Size = 8 MB
File size – 1.1 GB, IPERF BW – 75.5 Mbps

Delay (ms)	% BER (10^{-6})	% Bandwidth Utilization
0	0	89.45695
0	0.1	89.66887
0	0.3	89.45695
0	0.5	89.45695
0	0.7	89.23179
0	1	89.45695
500	0	47.94702
500	0.1	48.13245
500	0.3	48.39735
500	0.5	48.21192
500	0.7	48.19868
500	1	47.82781
1000	0	40.78146
1000	0.1	40.34437
1000	0.3	39.70861
1000	0.5	35.96026
1000	0.7	39.43046
1000	1	38.72848

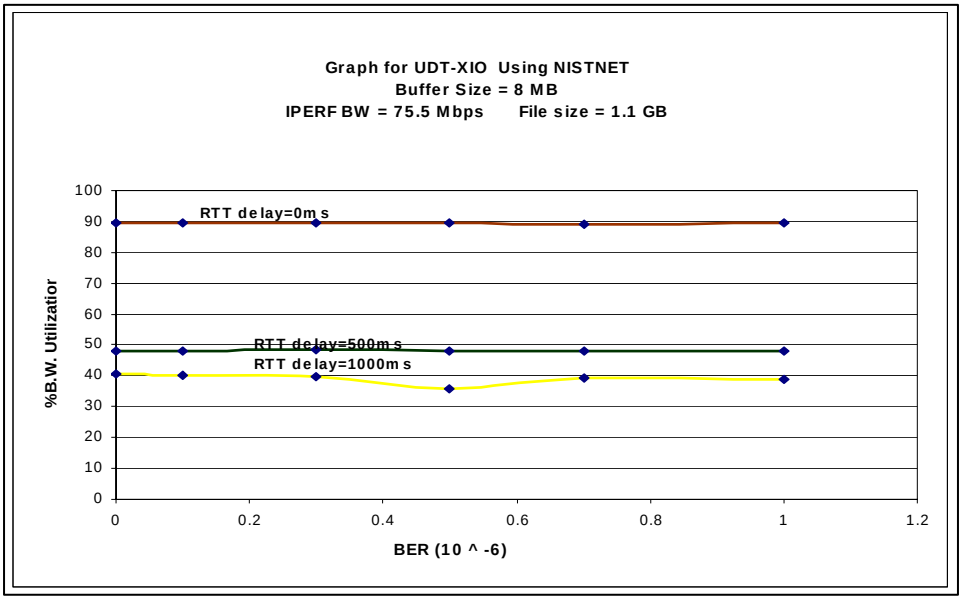

Figure 17: Graph of UDT-XIO (8MB)

Table 4.0
Performance Analysis of UDT Protocol using Globus-XIO
Buffer Size = 40 MB
File size – 1.1 GB, IPERF BW – 75.5 Mbps

Delay (ms)	% BER (10^{-6})	% Bandwidth Utilization
0	0	87.61589
0	0.1	87.86755
0	0.3	87.61589
0	0.5	88.63576
0	0.7	90.21192
0	1	90.59603
500	0	78.27815
500	0.1	78.47682
500	0.3	78.27815
500	0.5	78.27815
500	0.7	78.27815
500	1	78.27815
1000	0	68.43709
1000	0.1	68.35762
1000	0.3	68.82119
1000	0.5	68.6755
1000	0.7	68.43709
1000	1	68.37086

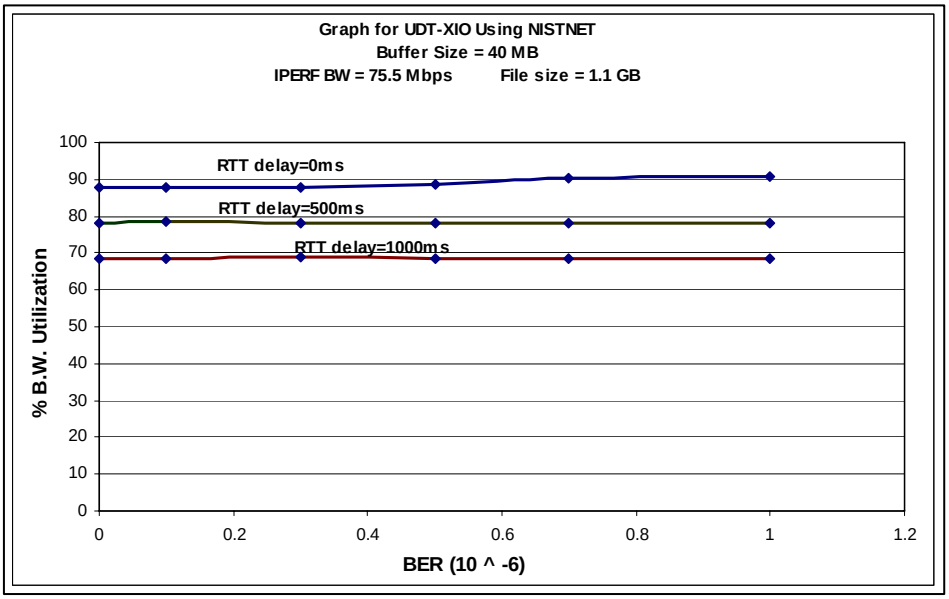

Figure 18: Graph of UDT-XIO (40 MB)

7.2 SABUL-XIO Driver Development

7.2.1 Architecture of Globus-XIO for SABUL

There are two important concepts in the architecture of Globus XIO (see Figure 9 and Figure 10), the driver and the stack. The driver is the abstracted component where all the protocol specific implementation exists. The protocol developer can focus on the interesting details of the protocol they are prototyping or producing for production. The driver must implement a well-defined set of interface functions, with a well-defined set of semantics associated with them. Essentially, these functions consist of Open, Close, Read, and Write. For instance, a driver developer defines an open function, and implements the code specific to their protocol for opening. The same steps are taken for the read, write and close functions. When implementing these functions the developer does not need to be concerned with user error checking, this has all been done by the framework and all parameters are guaranteed to be good. This and all other user interface issues are removed from the concern of the driver developer.

There are two types of drivers, transform drivers and transport drivers. Transport drivers are those that actually move data into or out of the process space. Examples of this are TCP and UDP. Transform drivers are those that manipulate, examine, frame, or change the data, or in other words, drivers that take any action other than moving the data across the process boundary. Examples would be compression, logging, and HTTP.A particular protocol implementation can involve many drivers; this is where the concept of a stack of drivers comes into play. In a stack there must only be one transport driver, and it must be at the bottom of the stack. This stands to reason since the transport driver is what actually moves the bits on a wire. There can be any number of transform drivers on a driver stack. As data is written it moves from the user to the first driver on the stack. The data is moved down the stack, through the effects of each transform driver (if any

are used) to the transport driver. The transport driver then ships the data over the wire. Data follows a similar path for reads, only in the opposite direction, from wire to transport driver, and up the stack to the user. The protocols that make use of multiple transport drivers have to construct multiple stacks. For example, SABUL protocol (as shown in Figure 19) that uses TCP for exchanging control information and UDP for transferring the actual data, has to have two different stacks: one with TCP as a transport and the other with UDP as the transport driver.

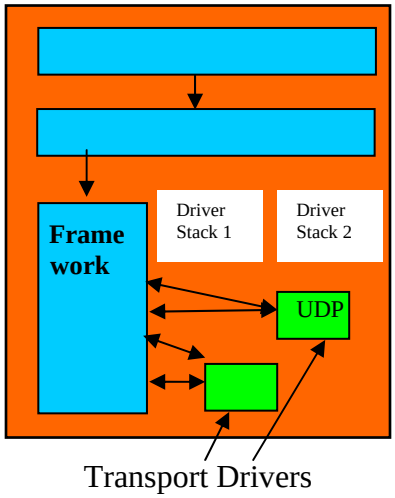

Figure 19: Globus XIO Example for SABUL Protocol

7.2.2 Globus-XIO Driver for SABUL

The Globus Toolkit 4.0.1 contains the UDT–XIO driver code and it consists of following files:

1) globus_xio_udt_attr.c
2) globus_xio_udt.c
3) globus_xio_udt_close.c
4) globus_xio_udt_control.c
5) globus_xio_udt_open.c
6) globus_xio_udt_read.c
7) globus_xio_udt_write.c
8) globus_xio_udt_misc.c

I studied these files and found the file where there is the return control path from client to the server. This was defined in the file globus_xio_udt.c. Then I changed the driver for that path from UDP to TCP.

7.3 Performance Evaluation of SABUL using Globus-XIO

Results are tabulated as given below in Table 5.0 and Table 6.0 and graphs are plotted in Figure 20 and Figure 21 respectively.

Table 5.0
Performance Analysis of SABUL Protocol using Globus-XIO
Buffer Size = 8MB
File size – 1.1 GB, IPERF BW – 75.5 Mbps

Delay (ms)	%BER (10^{-6})	% Bandwidth Utilization
0	0	89.66887
0	0.1	89.66887
0	0.3	89.66887
0	0.5	89.01987
0	0.7	89.66887
0	1	88.80795
500	0	46.91391
500	0.1	48.22517
500	0.3	47.57616
500	0.5	47.45695
500	0.7	46.78146
500	1	47.15232
1000	0	41.48344
1000	0.1	41.8543
1000	0.3	41.29801
1000	0.5	41.57616
1000	0.7	41.48344
1000	1	41.01987

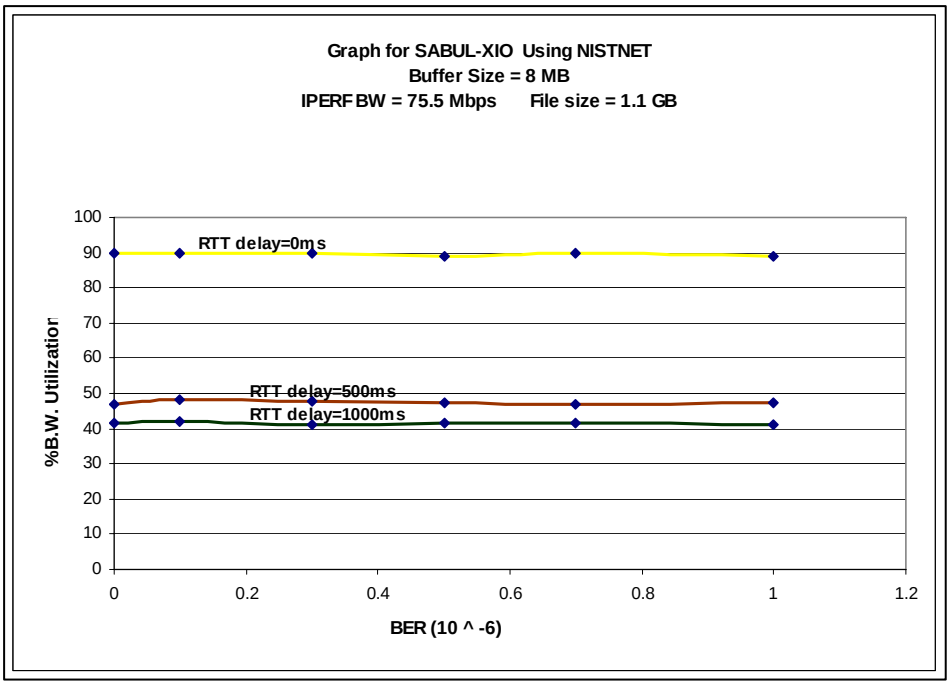

Figure 20: Graph for SABUL-XIO (8 MB)

Table 6.0
Performance Analysis of SABUL Protocol using Globus-XIO
Buffer Size = 40MB
File size – 1.1 GB, IPERF BW – 75.5 Mbps

Delay (ms)	% BER (10^{-6})	%Bandwidth Utilization
0	0	89.93377
0	0.1	90.33113
0	0.3	89.80132
0	0.5	85.0596
0	0.7	88.25166
0	1	89.15232
500	0	77.88079
500	0.1	77.68212
500	0.3	77.72185
500	0.5	77.88079
500	0.7	78.17219
500	1	77.2053
1000	0	68.13245
1000	0.1	68.05298
1000	0.3	67.16556
1000	0.5	67.76159
1000	0.7	67.45695
1000	1	67.82781

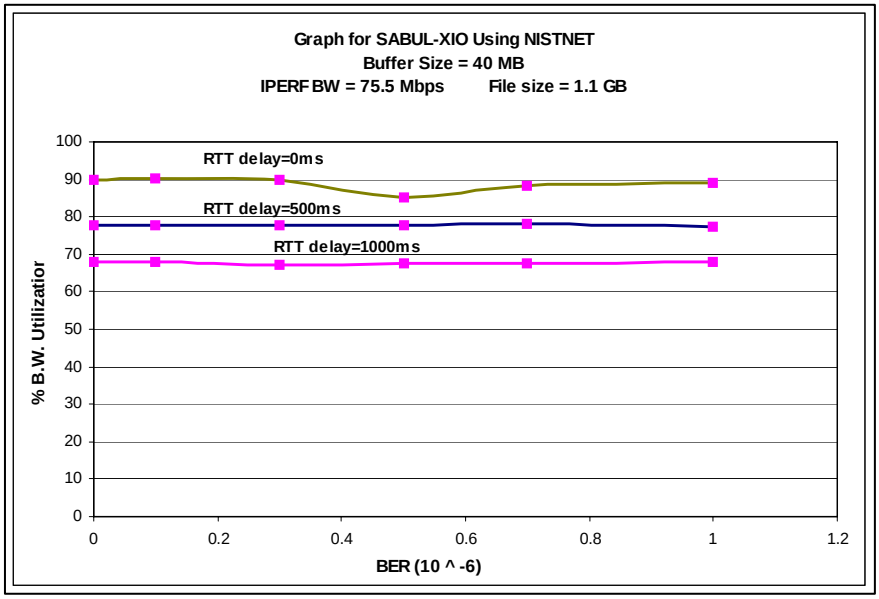

Figure 21: Graph for SABUL-XIO(40 MB)

7.4 Interfacing with DTA

The detail explanation of DTA Interface and its working is given in Section 6.1.2. As with GridFTP two programs are needed to be developed: Server and Sender. Server program is needed to be manually started at the terminals. As for example suppose Machine-A needs a file from Machine-B. Server is started at both the machines and Sender at any one of the machine or at different machine (say Machine-C). Sender will send all the parameters to the machine where Source file resides (here Machine-B). Then Machine-B sends parameters (filename and destination directory) to Machine-A. At Machine-A and Machine-B threads for receiving data and sending data are started respectively. Threads will transfer the file and exit. Server and Sender programs for UDT-XIO driver had been developed as this driver is already built available with Globus-Toolkit 4.0.1.

Chapter 8

CONCLUSION AND SCOPE OF FUTURE WORK

8.1 Conclusion

- As per literature survey and test analysis, performance of SABUL is good at high delay and BER.
- While installing Globus-Toolkit-4.0.1 we found UDT-XIO driver was already available. UDT is the successor of SAUL and test anlysis shows that performance of UDT is comparable to SABUL.
- Test Analysis shows that if buffer size is small then throughput of SABUL/UDT decreases. So the buffer size is needed to be optimized and optimum throughput and bandwidth-utilization is achieved at 40MB buffer size as shown in Figure 16.

8.2 Scope of Future Work

- Actual field-testing with satellite is to be done whenever the resources are available in the future.

REFERENCES

[1] Foster I., Kesselman, C., Nick, J. and Tuecke, S. *The Phsiology of the Grid: An Open Grid Service Architecture for Distributed Systems Integration*, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[2] Ian Foster *What is the Grid? A Three Point Checklist*, July 20, 2002

[3] ISRO's Internal Document, *Grid PDR*

[4] SGRM (Satellite Grid Resource Manager)

[5] Jason Leigh, Chaoyue Xiong, Eric He, Rajkumar Kettimuthu,Yunhong Gu, Sanjay Hegde, Michael Welzl, Pascale Vicat-Blanc Primet, Jason Leigh, Chaoyue Xiong *Survey of Protocols and Mechanisms for Enhanced Transport over LONG FAT PIPES*

[6] Yunhong Gu and Robert Grossman*, SABUL: A Transport Protocol for Grid Computing*

[7] Yunhong Gu, Xinwei Hong, Marco Muzzuco, Robert Grossman *Rate Based Congestion Control Over High Bandwidth/Delay Links*

[8] Faster Bulk Transfer Starring: *UDP* , *Study and Evaluation of 3 UDP protocols*, www.csm.ornl.gov/~dunigan/netperf/udp/

[9] Tsunami, http://www.indiana.edu/~anml/anmlresearch.html

[10] Globus Toolkit 4.0.1 Installation, http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html

[11] Yunhong Gu and Robert L. Grossman, *UDT: An Application Level Transport Protocol for Grid Computing*

[12] Yunhong Gu and Robert Grossman, *Using UDP for Reliable Data Transfer over High Bandwidth-Delay Product Networks*

[13] GridFTPOverview, GridFTP Update January 2002

[14] GridFTP Development Guide, http://www.globus.org/toolkit/docs/4.0/data/gridftp/developer-index.html#s-gridftp-developer-archdes

[15] GlobusXIO, http://www.globus.org/toolkit/docs/4.0/common/xio/developer-index.html

[16] William Allcock, John Bresnahan, Rajkumar Kettimuthu and Joseph Link, *The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid*

[17] Robert L. Grossman, Yunhong Gu, David Hanley, Xinwei Hong and Parthasarathy Krishnaswamy, *Experimental Studies of Data Transport and Data Access of Earth Science Data over Network with High Bandwidth Delay Products*, October 2003 http://www.rgrossman.com/dl/journal-031.pdf

[18] NISTNet Home Page: http://sand.ncsl.nist.gov/nistnet

[19] Iperf 1.7.0: http://dast.nlanr.net/Projects/Iperf

APPENDIX – A

Table 7.0
Comparison of TCP variants

	HS TCP	Scalable TCP	FAST TCP	XCP	CADPC/ PTP	GridFTP
Principle	Modify TCP response function when congestion window is larger than a threshold.	More aggressive in congestion control.	Based on TCP Vegas.	Decouple efficiency control and fairness control. Former uses MIMD and latter uses AIMD.	Congestion control based on feedback from routers.	Multiple parallel TCP streams
Operation Mode	Memory-to-memory. Kernel space.	Memory-to-memory. Kernel space.	Memory-to-memory.	Memory-to-memory.	Memory-to-memory.	File transfer protocol
Authentication	No.	No.	No.	No.	No.	GSI and Kerberos
FTP syntax	No.	No.	No.	No.	No.	Yes.
Congestion Control	Yes.	Yes.	Yes.	Yes.	CADPC	Yes.
Need to modify router software	No.	No.	Better if using AQM routers	Yes.	Yes.	No.
Fairness	Need more investigation.	No.	Need more investigation.	Yes.	Max-min fairness	Yes.
TCP Friendly	No when new TCP response function is triggered.	Need more investigation.	Need more investigation.	Yes.	No.	Yes.
Predictable Performance Model	Yes.	Yes.	Need more investigation.	Yes.	Yes.	No.
Simulation and Implementation	Both	Implementation	Not published.	Both.	Both.	Implementation.

Appendix – B

Table 8.0
Comparison of UDP variants

	RBUDP	**Tsunami**	**SABUL**
Principle	UDP data + TCP control.	UDP data + TCP control.	UDP data + TCP control.
Operation Mode	Disk-to-disk, memory-to-memory.	disk-to-disk (i.e. file transfer protocol, not general transport)	Disk-to-disk, memory-to-memory.
Authentication	No.	Yes but very simple.	No.
FTP syntax	No.	Partially. Only a few commands like connect, get, close, etc.	Partially. Only supports one connection one time.
Application Programming Interface	Yes.	No.	Yes.
3rd Party Transfer	No.	No.	No.
Congestion Control	Optional. Limited congestion control can be turned on.	Limited. Sending rate is reduced when loss rate is more than a threshold.	Yes.
Fairness	N/A	N/A	Yes.
TCP Friendly	No.	No.	Yes.
Predictable Performance Model	Yes.	No.	No.
Implementation	Available.	Available.	Available.

Appendix – C

Installation Of Globus Toolkit 4.0.1

This section describes the installation of Globus Toolkit 4.0.1.

Starting point for this document is a fresh installation of RedHat 9 workstation.

Expected is that you are the root user of this RedHat system. Then we will install JAVA, Ant, Globus toolkit software, generate and sign certificates.

C.1 Installing Supporting Software

C.1.1 The Java language

Since the underlying code of the globus toolkit (GT4) is written in Java, you have to install the Java platform on every machine running the Toolkit. The recommended version is 1.4.2, and you can download this from the Sun website at http://java.sun.com/j2se.

The major steps for installing Java are:

1. Install java in path /usr/local/j2sdk1.4.2_10
2. Set environment variables

The java binary for Linux is then copied and installed. Add execute rights to the bin-file if needed.

```
root # cp j2sdk-1_4_2_10-linux-i586.bin /usr/local
root # cd /usr/local
root # ./j2sdk-1_4_2_10-linux-i586.bin
root # rm j2sdk-1_4_2_10-linux-i586.bin
```

To use the Java installation, edit the shell login script, in our situation this is "~/.bashrc".

```
root # cd ~
root # vi .bashrc
```

Add a JAVA_HOME and change the PATH environment setting.

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_10
export PATH=$JAVA_HOME/bin:$PATH
```

Then you need to login again to use the new settings. And you can verify that your environment uses the correct java version.

```
root # java –version
```

C.1.2 The Ant builder

Ant is a Java-based build tool required for the GT3 installation and development of grid services. With build scripts, used by Ant, we can install, compile and copy files and applications. Ant is required for the installation of Grid services. The recommended version: 1.6.0 and can be retrieved from http://jakarta.apache.org/ant.

The main tasks to install Ant are;

1. Install Ant in path /usr/local/apache-ant-1.6.2
2. Set the environment variables

Now Ant is installed, and you can modify the already mentioned ".bashrc" file to use Ant.

```
export JAVA_HOME=/usr/local/j2sdk1.4.2_10
export ANT_HOME=/usr/local/apache-ant-1.6.2
export PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$PATH
```

To use these settings, login again.

C.2 The user logins

C.2.1 Adding users

The root user did our work up to this point. To add a new user you can use a graphical tool in RedHat. Use the RedHat start-menu and select in the System Settings the Users and Groups. On the Users tab-page click the button to add a user and type "globus" for the name and password. Select create a private group. This will result in a user that is placed in a group with the same "globus" name.

C.2.2 The .bashrc login script

During the previous installations the ".bashrc" script was gradually extended. Here the final result is presented, and this script will be copied to the home directories of the other users, in our case "globus".

Note that it contains more settings than we prepared, but we need them in advance to the globus installation.

The resulting ".bashrc" script looks like;

```
# .bashrc
# User specific aliases and functions
# Globus, java, ant, junit, ssl settings
export JAVA_HOME=/usr/local/j2sdk1.4.2_10
export ANT_HOME=/usr/local/apache-ant-1.6.2
export GLOBUS_LOCATION=/usr/local/globus
PATH=$JAVA_HOME/bin:$ANT_HOME/bin:$GLOBUS_LOCATION/bin:$PATH
. $GLOBUS_LOCATION/etc/globus-user-env.sh
#alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
```

In the example below, the user globus is logged-in. Then it switches to root without changing the current working directory (do not specify the minus) and copies the ".bashrc" file to the globus home directory. It checks to see if the user permissions and owner are OK, and they are.

```
$globus [globus@localpc globus] $  pwd
                                    /home/globus
$globus [globus@localpc globus] $ su root
                                  Password:

#root [root@localpc globus] #  cp /root/.bashrc .
 cp: overwrite `./.bashrc'? y
#root [root@localpc globus] #  exit

$globus [globus@localpc globus] $ ls -al .bashrc
-rw-r--r--  1   globus globus  599    Oct 22 13:48     bashrc
```

Please ignore the error message for the "globus-user-env.sh" for now, you will install the globus software in just a moment.

C.3 Installing Globus Toolkit on the first machine

Now all the preparations are finished and the Globus Tookit 4 (GT4) software can be installed. This process is some time consuming.

C.3.1 Installation of the source code

Ok, the root user will create a new folder and install the globus software in it. If you get a message “invalid user” at the chown command, check the previous “adding users” chapter.

```
root # pwd
        /root
root # mkdir /usr/local/globus
root # chown globus:globus /usr/local/globus
root # su – globus
```

Make sure the “.bashrc” file for the globus user is the same as the one we prepared earlier. So you will get a “no such file or directory” warning like described earlier at login or su.

The software (gt4.0.1-all-source-installer.tar.gz) is copied from the site http://www.globus.org/toolkit/downloads/4.0.1/#source, untarred and installed by the globus user. The sequence of commands given as below, after the source is untarred.

```
globus $ cd gt4.0.1-all-source-installer

globus $ source .bashrc

globus $ ./configure –prefix=$GLOBUS_LOCATION

globus $ make
```

This is very time-consuming process and may consume three to five hours.

```
globus $ make install
```

Now the software is installed at GLOBUS_LOCATION (i.e. /usr/local/globus) and next step is generation and signing of security certificates.

C.4 Certificate Authority and user certificates

C.4.1 Quick introduction

Globus is using a security infrastructure with certificates per user, so users or processes running for this user can be identified using this certificate. A user can generate a certificate for him/herself, but to really use it, the certificate must be signed by someone else, a certificate authority (CA). You can visualize this as a hierarchy of people signing for other people.

There are commercial CA's that will sign your organisation's certificate for some cash and prove that you are who you say you are. That signed certificate can then be used to sign users within your organisation.

For demo and easy installation purposes, the globus installation offers us a way to avoid the above procedures, but you should definitely consider an official certificate for production systems.

These steps will be done:

- Setup a CA as globus
- Generate a host certificate as root, sign by globus

The next step is to generate and sign a certificate for your host machine.

C.4.2 Creating a simple CA

The globus installation provides a simpleCA option for demo purposes and we will use this.

Normally the following should be generated once per grid.

globus $ source $GLOBUS_LOCATION/etc/globus-user-env.sh

globus $ $GLOBUS_LOCATION/setup/globus/setup-simple-ca

You have to answer some questions, and you can accept the default answers. Except for the passphrase, which is in our case "pravina". Note that you shouldn't use spaces.

This process generates a CA that is stored in;

/home/globus/.globus/simpleCA/globus_simple_ca_c7881362_setup-0.17.tar.gz

And the c7881362 is a short identification hash for your certificate.

```
globus $ su root
         Password:

root  # /usr/local/globus/setup/globus_simple_ca_c7881362_setup/setup-gsi -default
```

C.4.3 Creating a certificate for your host

The host name (eg “localpc”) is used to request a certificate for it. Hostname can be checked by the following command.

```
root # echo $HOSTNAME
       localpc

root # grid-cert-request -host ‘`hostname`‘
```

That process generated three files in the “/etc/grid-security/” folder. These are

- “hostkey”, a private key for your host,
- “hostcert_request”, the certificate request for your host, and
- “hostcert” the signed certificate for your host.

But, you see that this “hostcert” is still empty because we only generated a request yet, and did not sign our request.

Here we will sign the “hostcert_request” using the simpleCA certificate we created in the previous section. If your organisation uses an official certificate, use that one. Sign it using the simpleCA as globus user and do not worry about the error. The signed certificate is placed somewhere else and we will copy that certificate to the correct location later. And enter the CA’s passphrase as entered creating the CA.

```
root # exit

globus $ cd /etc/grid-security

globus $ grid-ca-sign -in hostcert_request.pem -out hostsigned.pem
```

Enter password for the CA key: (Enter the CA's passphrase as entered creating the CA)

The new signed certificate is at: /home/globus/.globus/simpleCA//newcerts/01.pem

ERROR: Cannot create the requested output certificate file hostsigned.pem
Instead manually copy and rename the new signed certificate at /home/globus/.globus/simpleCA//newcerts/01.pem

Now overwrite the empty "hostcert" file with the signed host certificate request.

globus $ su root
Password:

root # cd /etc/grid-security

root # cp /home/globus/.globus/simpleCA/newcerts/01.pem hostcert.pem

Verify the permissions on the files in this folder, the "hostcert" should be owned by root and read-only for other users. The "hostkey" should be read-only for the root – no-one else.

The host certificate is now ready.

C.4.4 Creating a certificate for your user

Now there is a globus user acting as a certificate authority, a root user with the host certificate, and next step is to create a user certificate.

Our user is named "globus", and was already added to your system in a previous section, as you did with the ".bashrc" script.
Login with this user and request for a certificate.

globus $ grid-cert-request –force –cn "Pravina"

This will create three files in "/home/globus/.globus/". The principle is like generating certificate for the host, but these are user certificates.

Three pem files are the key, certificate request and certificate for this user. These are:

- The userkey.pem is private for the user, so no one else is allowed to read or use this userkey.
- The usercert_request.pem will be sent to the CA (globus user) to sign it.
- The usercert.pem which is empty.

After the globus user signed the usercert_request and produced a signed certificate for the globus user, globus will sent the certificate back to the globus user. This signed certificate will be the public certificate for the globus user.

```
globus $ pwd
        /home/globus/.globus

globus $ ll
total 8
-rw-r--r--  1  globus  globus      0    Jan 20 14:34  usercert.pem
-rw-r--r--  1  globus  globus  1412   Jan 20 14:34  usercert_request.pem
-r--------   1  globus  globus   951   Jan 20 14:34  userkey.pem
```

Because of user permissions the globus user has to be in this "/home/globus/.globus/" folder to sign the usercert_request. Again, like with signing the host certificate, there will be an error but we will copy the resulting file later ourselves. Enter the passphrase for the CA to sign the certificate.

```
globus $ pwd
        /home/globus/.globus

globus $ grid-ca-sign -in usercert_request.pem -out signed.pem

Enter password for the CA key:

The new signed certificate is at: /home/globus/.globus/simpleCA//newcerts/02.pem

ERROR: Cannot create the requested output certificate file signed.pem
```

Instead manually copy and rename the new signed certificate at

/home/globus/.globus/simpleCA//newcerts/02.pem

Now the certificate is signed and placed in the 02.pem file. This file needs to be copied to the /home/globus/.globus/usercert.pem file, but this cannot done directly by the globus user. So the root user is used for this. Note that you have to enter this on one single line and it will overwrite an existing empty file.

```
root $  cp /home/globus/.globus/simpleCA//newcerts/02.pem      \
        home/globus/.globus/usercert.pem
```

This way, the permissions and ownership of the file stays the same, and no chmod or chown has to be changed. This results in the "/home/globus/.globus/" folder containing:

```
globus $ ll
total 12
-rw-r--r--  1  globus  globus   2738  Jan 20 15:45      usercert.pem
-rw-r--r--  1  globus  globus   1412       Jan 20 14:34 usercert_request.pem
-r--------    1  globus  globus    951       Jan 20 14:34 userkey.pem
```

You are now finished creating signed certificates.

C.4.5 Test your user certificate

To verify that you did the right thing, you can verify your certificate with your key with your CA.

```
globus $ pwd
         /home/globus/.globus

$ grid-proxy-init -debug –verify

User Cert File: /home/globus/.globus/usercert.pem
User Key File: /home/globus/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u500
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-localhost.
localdomain/OU=localdomain/CN=Pravina
Enter GRID pass phrase for this identity:
Creating proxy .............++++++++++++
..............……..........................++++++++++++
Done
Proxy Verify OK
```

Your proxy is valid until: Mon Jan 23 23:24:37 2006

C.4.6 Connect users to certificates

The root user now has to edit a file and add all users, in our case only the "globus" user with its certificate and username. But first, let's see what should be entered in that file later.

```
globus $ whoami
        globus

globus $ grid-cert-info -subject
/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=Pravina
```

A file containing entries mapping certificate subjects to local user names. This file can also serve as a access control list for GSI enabled services and is typically found in /etc/grid-security/grid-mapfile if user is root-user else $HOME/.gridmap for globus user.

```
globus $ vi /home/globus/.gridmap
```

In the file there is one single line with a mapping from the certificate subject to the username. Enter a quoted subject, a space and the username. Note that the following line might be wrapped in your print but it is a singleline.

```
"/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=Pravina" globus
```

The system is now ready to be started! Thus in one system we have installed server and user (client-globus).

C.5 Installing Globus on the another machine

Do all the steps on the second machine except C.4.1,C.4.2,C.4.3.

C.5.1 Creating a certificate for user on the another machine

So now there is a globus user acting as a certificate authority, a root user with the host certificate on the first machine, and now on the another machine a user certificate will be created.

On the another machine the username named “globus” is already and also create .bashrc.

Login with this user and request for a certificate.

globus $ grid-cert-request –force –cn “PRM”

This will create three files in “/home/globus/.globus/”. Three pem files are the key, certificate request and certificate for this user. These are:

- The userkey.pem is private for the user, so no one else is allowed to read or use this userkey.
- The usercert_request.pem will be sent to the CA of the first machine (globus user) to sign it.
- The usercert.pem which is empty.

```
globus $ pwd
/home/globus/.globus

globus $ ll
total 8
-rw-r--r--1        globus globus            0      Jan 27 17:18  usercert.pem
-rw-r--r--         1        globus globus  1412    Jan 27 17:18    usercert_request.pem
-r--------         1        globus globus   951     Jan 27 17:18  userkey.pem
```

Because of user permissions the globus user has to be in this “/home/globus/.globus/” folder to sign the usercert_request. Now copy usercert_request.pem from sencond machine to first machine using FTP.
Here on the first machine, we will sign the “usercert_request” of the second machine using the simpleCA certificate of the first machine that we created in the section C.4.2. If your organisation uses an official certificate, use that one. Sign it using the simpleCA as globus user and do not worry about the error. The signed certificate is placed somewhere else and we will copy that certificate to the correct location later. And enter the CA’s passphrase as entered creating the CA.

```
globus $ pwd
        /home/globus/.globus
globus $ grid-ca-sign -in usercert_request.pem -out signed.pem

Enter password for the CA key:
The new signed certificate is at: /home/globus/.globus/simpleCA//newcerts/03.pem

ERROR: Cannot create the requested output certificate file
signed.pem
```

Instead manually copy and rename the new signed certificate at /home/globus/.globus/simpleCA//newcerts/03.pem

Now the certificate is signed and placed in the 03.pem file. This file needs to be copied to the /home/globus/.globus/usercert.pem file on the second machine so do this with FTP. In the second machine as the root user exceute following command.

```
root $  cp /home/globus/.globus/simpleCA//newcerts/03.pem   \
        home/globus/.globus/usercert.pem
```

This way, the permissions and ownership of the file stays the same, and no chmod or chown has to be changed.

This results in the "/home/globus/.globus/" folder containing:

```
globus $ ll
total 12
-rw-r--r--        1       globus        globus  2738    Jan 30 10:09   usercert.pem
-rw-r--r--       1       globus globus  1412    Jan 27 17:18   usercert_request.pem
-r--------        1       globus globus    951    Jan 27 17:18   userkey.pem
```

You are now finished creating signed certificate of the user on the second machine.

C.5.2 Test your user certificate on the second machine

To verify that you did the right thing, you can verify your certificate with your key with your CA..

```
globus $ pwd
        /home/globus/.globus


globus $ grid-proxy-init -debug –verify

User Cert File: /home/globus/.globus/usercert.pem
User Key File: /home/globus/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u500
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCAlocalhost.
localdomain/OU=localdomain/CN=PRM
Enter GRID pass phrase for this identity:
Creating proxy .....++++++++++++
.....++++++++++++
Done
Proxy Verify OK
Your proxy is valid until: Tue Jan 31 23:24:37 2006
```

C.5.3 Connect both users to certificates

The root user now has to edit a file and add all users, in our case only the “globus” user with its certificate and username. But first, let’s see what should be entered in that file later.

```
globus $ whoami
        globus

globus $ grid-cert-info -subject
/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=PRM
```

In the home directory of the globus user (/home/globus) of the second machine, create .gridmap file.

```
globus$ vi /home/globus/.gridmap
```

In this file, add two lines with a mapping from the certificate subject to the username. Enter a quoted subject, a space and the username. Note that the each line might be wrapped in your print but it is a singleline.

"/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=Pravina" globus
"/O=Grid/OU=GlobusTest/OU=simpleCA-localhost.localdomain/OU=localdomain/CN=PRM" globus

Add second line in .gridmap file of the first machine.

The second system is also ready to be started! Thus in first system server and user (client-globus) is installed and in second system (client-globus) is installed.

Appendix – D

Globus-XIO Examples

Globus-XIO examples implements Globus-XIO user APIs.

Listing-1 globus_xio_example.c

```
#include ”globus xio.h”

int main( int argc, char * argv[])
{
        globus_result_t res;
        char * driver_name;
        globus_xio_driver_t driver;
        globus_xio_stack_t stack;
        globus_xio_handle_t handle;
        globus_size_t nbytes;
        char * contact_string = NULL;
        char buf[256];
        contact_string = argv[1];
        driver_name = argv[2];

        globus_module_activate(GLOBUS_XIO_MODULE);

        res = globus_xio_driver_load(driver_name, &driver);
        assert(res == GLOBUS_SUCCESS);

        res = globus_xio_stack_init(&stack, NULL);
        assert(res == GLOBUS_SUCCESS);

        res = globus_xio_stack_push_driver(stack, driver);
        assert(res == GLOBUS_SUCCESS);

        res = globus_xio_handle_create(&handle, stack);
        assert(res == GLOBUS_SUCCESS);

        res = globus_xio_open(handle, contact_string, NULL);
        assert(res == GLOBUS_SUCCESS);
        do
        {
                res = globus xio read(handle, buf, sizeof(buf) - 1, 1,&nbytes, NULL);
                if(nbytes > 0)
                {
                        buf[nbytes] = ’\0’;
```

```
                fprintf(stderr, ”%s”, buf);
            }while(res == GLOBUS_SUCCESS);

            globus_xio_close(handle, NULL);
            globus_module_deactivate(GLOBUS_XIO_MODULE);
            return 0;
        }
    }
```

The above example is explained in following steps:

Step 1: Activate Globus

XIO was developed by the Globus Alliance and so inherits some Globus Toolkit semantics. Accordingly, as with all Globus Toolkit programs, you must first activate the Globus module. Until activation is complete, no XIO function calls can be successfully executed. The module is activated with the following line:

```
globus_module_activate(GLOBUS XIO MODULE);
```

Step 2: Load Driver

The next step is to load all the drivers needed to complete the I/O operations in which you are interested. The function globus_xio_load_driver() is used to load a driver. To successfully call this function, you must know the name of all the drivers you wish to load. For this example we want to load only the file io driver. The prepackaged file io driver's name is “file.” This driver is loaded with the following code:

```
globus_result_t res;
globus_xio_driver_t driver;
res = globus_xio_load_driver(&driver, ”file”);
```

If upon completion of the above function call, the variable res is equal to GLOBUS SUCCESS, then the driver was successfully loaded and can be referenced with the variable “driver.”

Step 3: Create Stack

Once globus xio is activated and a driver loaded, you need to build a driver stack. In our example the stack consists of only one driver, the file driver. The stack is established with the following code (building from the above code snips):

```
globus_xio_stack_t stack;
globus_xio_stack_init(&stack);
globus_xio_stack_push_driver(stack, driver);
```

Step 4: Opening the Handle

Once the stack is created, you can open a handle to the file, in one of two ways. The first way is a passive open. An example is a TCP listener. The open is performed passively and then it waits for some other entity to act on it. The other alternative is an active open. An example is a TCP connect. The user, who initiates the open, performs the open actively. This example has an active open. To create a handle for an active open, you first initialize the handle object and then open it with the contact information. The following code illustrates this:

```
globus_xio_handle_t handle;
globus xio handle create(&handle, stack);
res = globus xio open(handle, ”/tmp/junk.txt”, NULL);
```

Step 5: The Payoff

Now that with an open handle to a file, data can read or write to it with either: globus xio read() or globus xio write() respectively. Once you are finished performing I/O operations on the handle, you should call globus_xio_close(handle).

All this may seem like a lot of effort for simply reading a file. The advantages become clear, however, when we need to use other drivers. In the above example, it would be trivial to change the I/O operations from file I/O to TCP, HTTP, or FTP.

All we would need to do is change the driver_name string passed to globus_xio_load_driver() and change the contact_string passed to

globus_xio_[register]_open(). Both can be done easily at runtime, as the program globus xio example.c given above demonstrates.

- ## Steps to run globus_xio_example.c

1) Use globus-makefile header command to generate macros that you can include in a makefile.

```
$ globus-makefile-header -flavor=gcc32dbg globus_xio > header
```

2) Create makefile (vi makefile).

3) Paste all contents of header in this makefile.

4) Then in makefile file type the following

```
all:
    $(GLOBUS_CC) $(GLOBUS_INCLUDES) -o globus globus_xio_example.c \
            $(GLOBUS_LDFLAGS) $(GLOBUS_PKG_LIBS)
```

Note: to give tab before $(GLOBUS_LDFLAGS) otherwise you will get error Invalid separator.

5) Then run makefile by coming following command

```
$ make
```

6) Now your globus_xio_example.c's executable is generated as globus.

7) To see the output of globus_xio_example.c give the following command

```
$ ./globus file:/tmp/temp.txt file
```

here argv[1]=contact string ie where file temp.txt is stored
and argv[2]=driver name

Listing-2 globus_xio_udt_file.c

```
#include "globus_xio.h"
#include "globus_xio_util.h"
#include "globus_xio_udt.h"

#define CHUNK_SIZE 50000
#define FILE_NAME_LEN 25

void test_res(globus_result_t      res)
{
   if(res == GLOBUS_SUCCESS)
   {
      return;
   }
   fprintf(stderr, "ERROR:%s\n" ,  globus_error_print_chain(globus_error_peek(res)));
   globus_assert(0);
}

void help()
{
       fprintf(stdout, "globus-xio-udt-bw [options]\n"
       "-----------------\n"
       "using the -s switch sets up a server.\n"
       "specify -c <contact string> to communicate with the server\n"
       "options:\n"
       "-c <contact_string> : use this contact string (required for client)\n"
       "-s : be a server\n"
       "-p : server port (optional)\n"
       "-f : file name\n");
}

int main(int  argc, char ** argv)
{
   globus_xio_driver_t      udt_driver;
   globus_xio_stack_t       stack;
   globus_xio_handle_t      xio_handle;
   globus_xio_server_t      server;
   globus_xio_attr_t        attr = NULL;
   char *cs = NULL;
   globus_result_t          res;
   int     ctr,    rc;
   globus_bool_t            be_server = GLOBUS_FALSE;
   char   filename[FILE_NAME_LEN];
   FILE *fp;
```

```
rc = globus_module_activate(GLOBUS_XIO_MODULE);
globus_assert(rc == GLOBUS_SUCCESS);
res = globus_xio_driver_load("udt", &udt_driver);
test_res(res);
res = globus_xio_stack_init(&stack, NULL);
test_res(res);
res = globus_xio_stack_push_driver(stack, udt_driver);
test_res(res);

if (argc < 4)
{
   help();
   exit(1);
}
for(ctr = 1; ctr < argc; ctr++)
{
   if(strcmp(argv[ctr], "-h") == 0)
   {
      help();
      return 0;
   }
   else if(strcmp(argv[ctr], "-c") == 0)
   {
      if (argc < 5)
      {
               help();
               exit(1);
      }
      cs = argv[ctr + 1];
      ctr++;
   }
   else if(strcmp(argv[ctr], "-s") == 0)
   {
      be_server = GLOBUS_TRUE;
   }
   else if(strcmp(argv[ctr], "-p") == 0)
   {
         if (argc < 6)
         {
              help();
              exit(1);
          }
      test_res(globus_xio_attr_init(&attr));
      test_res(globus_xio_attr_cntl( attr, udt_driver, \
      GLOBUS_XIO_UDT_SET_PORT, atoi(argv[ctr + 1])));
    }
```

```
    else if(strcmp(argv[ctr], "-f") == 0)
    {
          if (ctr + 1 < argc)
          {
                strcpy(filename, argv[ctr + 1]);
           }
           else
           {
                help();
                exit(1);
            }
      }
}
if (!be_server && (!cs || !*cs))
{
   help();
   exit(1);
}
if(be_server)
{
   char    buffer[CHUNK_SIZE + 1];
   int      nbytes, i, x;
   res = globus_xio_server_create(&server, attr, stack);
   test_res(res);
   globus_xio_server_get_contact_string(server, &cs);
   fprintf(stdout, "Contact: %s\n", cs);
   res = globus_xio_server_accept(&xio_handle, server);
   test_res(res);
   res = globus_xio_open(xio_handle, NULL, attr);
   test_res(res);
   fp = fopen(filename, "r");
   while(!feof(fp))
   {
         for(i= 0; i< CHUNK_SIZE + 1; i++)
               buffer[i] = '\0';
          x = fread(buffer, CHUNK_SIZE, 1, fp);
          nbytes = strlen(buffer);
          res=globus_xio_write(xio_handle,buffer,nbytes, nbytes , &nbytes, NULL);
          test_res(res);
    }
    fclose(fp);
 }
 else
 {
     char    buffer[CHUNK_SIZE];
     int      nbytes, i;
```

```
        res = globus_xio_handle_create(&xio_handle, stack);
        test_res(res);
        res = globus_xio_stack_destroy(stack);
        test_res(res);
        res = globus_xio_open(xio_handle, cs, attr);
        test_res(res);
        fp = fopen(filename, "w");
        while(1)
        {
          for (i=0; i<CHUNK_SIZE; i++)
                          buffer[i] = '\0';
          res = globus_xio_read(xio_handle, buffer, sizeof(buffer), \
                    sizeof(buffer), &nbytes, NULL);
          fputs(buffer, fp);
          if (res != GLOBUS_SUCCESS)
                  break;
         }
        fclose(fp);
    }
    res = globus_xio_close(xio_handle, NULL);
    test_res(res);
    res = globus_xio_driver_unload(udt_driver);
    test_res(res);
    rc = globus_module_deactivate(GLOBUS_XIO_MODULE);
    globus_assert(rc == GLOBUS_SUCCESS);
    return 0;
}
```

Appendix – E

Globus-XIO Driver Guide

Driver is an abstracted component where all the protocol specific implementation exists. Driver must implement well-defined set of interface functions (like Open, Read, Write and Close) with well-defined set of semantics associated with them.

E.1 Data Structures

There are three data structures that need to be understood for writing XIO driver. These are: attribute, target, and handle. The driver defines the memory layout of these data structures but the globus_xio framework imposes certain semantics upon them. It is up to the driver how to use them, but globus_xio will be expecting certain behaviors.

E.2 Attributes

Each driver may have its own attribute structure. The attribute gives the globus_xio user API an opportunity to tweak parameters inside the driver. The single attribute structure is used for all possible driver specific attributes:

1. Target attributes
2. Handle attributes
3. Server attributes

How each of these can use the attribute structure will be unveiled as the tutorial continues. For now it is simply important to remember there is attribute structure used to initiate of the driver ADTs.

A driver is not required to have an attribute support at all. However if the driver author chooses to support attributes the following functions must be implemented:

```
typedef globus_result_t
(*globus_xio_driver_attr_init_t)(void **out_attr);

typedef globus_result_t
(*globus_xio_driver_attr_cntl_t)
        (void *attr, int cmd, va_list ap);

typedef globus_result_t
(*globus_xio_driver_attr_copy_t)(void **dst, void *src);

typedef globus_result_t
(*globus_xio_driver_attr_destroy_t)(void *attr);
```

E.3 Target

A target structure represents what a driver will open. It is initialized from a contact string and an attribute. In the case of a file driver, the target simply holds onto the contact string as a path to the file.

The file driver implements the following target functions:

```
globus_result_t globus_xio_driver_file_target_init(
 void **out_target, void *target_attr,
 const char *contact_string, globus_xio_driver_stack_t stack)
```

and

```
globus_result_t globus_xio_driver_file_target_destroy(
        void *target)
```

E.4 Handle

The most interesting of the three data types discussed here is the handle. All typical IO operations (open, close, read, write) are performed on the handle. The handle is the initialized form of the target and an attr. The driver developer should use this ADT to keep track of any state information they will need in order to perform reads and writes.

The reader should review the following functions in globus_xio_driver_example.c in order to see how the handle structure is used:

- globus_xio_driver_file_open()
- globus_xio_driver_file_write()
- globus_xio_driver_file_read()
- globus_xio_driver_file_close()

Appendix – F

Tools Used in this Experiment

F.1 NIST Net

NIST Net[18] provides the ability to emulate common network effects such as packet loss, duplication or delay; router congestion; and bandwidth limitations. It is designed to offer sufficient capabilities and performance to reproduce a wide range of network behaviors (forwarding rates of up to 1 Gbps, satellite-like delays or longer, asymmetric characteristics), while requiring only commodity PC hardware and operating systems. NIST Net has been used for emulation up to line rate over 100Mbps Ethernet with typical "throw-away" machines (200 MHz Pentium-class processors and PCI-based 10/100 Ethernet cards). On current generation machines, NIST Net has been successfully used at line rate with gigabit Ethernet cards and 622 Mbps OC-12 interfaces.

Emulation, as defined here, is a combination of two common techniques for testing network code: *simulation*, which we can define as a synthetic environment for running representations of code; and *live testing*, a real environment for running real code. In these terms, emulation is a "semi-synthetic" environment for running real code. The environment is semi-synthetic in the sense that it is a real network implementation (in the case of NIST Net, the Linux 2.xx kernel) with a supplementary means for introducing synthetic delays and faults. Emulation thus offers many of the advantages of both simulations (a controlled, reproducible environment, which is relatively quick and easy to assemble) and live testing (real code in a real environment, which obviates any questions about the fidelity of the representation).

F.2 Iperf

While tools to measure network performance, such as ttcp, exist, most are very old and have confusing options. Iperf[19] was developed as a modern alternative for measuring TCP and UDP bandwidth performance.

Iperf is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay jitter, datagram loss.