

Topaz: a Firefox Protocol Extension for GridFTP based on Data Flow Diagrams

Richard Zamudio, Daniel Catarino, Michela Taufer, Brent Stearn, and Karan Bhatia

Abstract—As grid infrastructures mature, an increasing challenge is to provide end-user scientists with intuitive interfaces to computational services, data management capabilities, and visualization tools. The current approach used in a number of cyber-infrastructure projects is to leverage the capabilities of the Mozilla framework to provide rich end-user tools that seamlessly integrate with remote resources such as web/grid services and data repositories.

In this paper we apply rigorous software engineering tools, Data Flow Diagrams or DFDs, to guide the design, implementation, and performance analysis of Topaz, a GridFTP protocol extension to the Firefox browser. GridFTP servers, similar to FTP servers used on the Internet, provide a data repository for files and are optimized for grid use (support for very large file sizes, high-performance data transfer, third-party transfer, integration with Grid Security Infrastructure). Topaz provides users with a familiar and user-friendly interface with which to access arbitrary GridFTP servers by providing upload and download functionalities, as well as by obtaining and managing certificates.

Since it is integrated into the Firefox browser, Topaz introduces an additional layer of abstraction in the communication between client and server. In this paper, we compare upload and download transmission times of our protocol extension with other traditionally used tools such as UberFTP, the *globus-url-copy* command, an FTP client, and the Secure Copy Protocol (*scp*) command. DFDs enable us to identify the causes of performance slow-downs in Topaz when transferring small files with respect to grid tools such as UberFTP and the *globus-url-copy* command. For large files, Topaz performs as well as the traditionally used tools providing, in addition, scientists with a user-friendly interface.

Index Terms— Distributed computing, Data communication, Data access and management, Middleware and toolkits, Grid integration issues.

Manuscript submitted April 21, 2006. This work was supported in part by the National Science Foundation under Grants #SCI-0438430, SCI: NMI Development: The Computational Chemistry Prototyping Environment, and #EIA-0080940, MII: Graduate Education for Minority Students in Computer Science and Engineering: Extending the Pipeline.

Richard Zamudio is with the Computer Science Department, University of Texas, El Paso, TX 79968 USA (e-mail: rzamudio@utep.edu).

Daniel Catarino is with the Computer Science Department, University of Texas, El Paso, TX 79968 USA (e-mail: dcatarino1@utep.edu).

Michela Taufer is with the Computer Science Department, University of Texas, El Paso, TX 79968 USA (corresponding author: 915-747-6957; fax: 915-747-5030; e-mail: mtaufer@utep.edu).

Brent Stearn is with the San Diego Supercomputer Center, University of California, San Diego, CA 92093 USA (e-mail: flujul@sdsc.edu).

Karan Bhatia is with the San Diego Supercomputer Center, University of California, San Diego, CA 92093 USA (e-mail: karan@sdsc.edu).

I. INTRODUCTION

AS grid infrastructures mature, an increasing challenge is to provide end-user scientists with intuitive interfaces to access computational services, data management capabilities, and visualization tools [1]. Grid infrastructure tools are typically designed and deployed on server-class systems running a version of Unix. Meanwhile, end-users typically have less powerful desktops or laptops, for their day-to-day work, running a desktop OS. Therefore, end-users cannot easily access the server resources because their desktops and laptops are not integrated into the grid. One approach taken in [2] is to develop compatible grid protocols for desktop operating systems such as Windows so that these desktop resources become first class grid resources. An alternative approach is to develop light-weight portable tools that will extend the grid onto the desktop fully under the control of the end-user. This latter approach may be advantageous as it does not require the full grid stack to be developed and maintains control of the machine for the end-user.

One particular challenge is the integration of data management capabilities across the desktop to the server: the data that a user employs is typically on his or her own local machine and used in day-to-day work such as email, writing reports, basic analysis using spreadsheets, or other desktop software. Within the grid, this data can be accessed using high-performance and secure grid protocols such as GridFTP [3]. However, from the desktop (which is outside of the formal grid) the user must employ different protocols to upload the data to a repository. After the grid calculation, again the grid infrastructure uses one protocol to move the data to a repository and another protocol for downloading it to the desktop.

Our approach has been to extend the grid infrastructure to the desktop environment through the use of lightweight, cross platform tools that provide direct access to grid protocols for the user. This project, called Topaz, provides a web browser extension that allows a user to access GridFTP servers directly from his or her desktop machine. The extension works with the Mozilla's Firefox browser and can support any platform that Firefox supports. The Mozilla framework provides a powerful base for development of end-user tools because it has support for CSS, JavaScript, XUL and even SOAP/Web Services [4]. The Mozilla framework also supports additional capabilities through the development of new XPCOM components written in C/C++. The Gemstone [5]-[7] and mozCellML [8] projects are using this approach in the

computational chemistry and the cell modeling communities respectively.

In this paper we present an innovative method to guide the design, implementation, and performance analysis of Topaz. The innovation of our approach lays in the application of rigorous software engineering tools such as Data Flow Diagrams (DFDs) to guide this engineering effort to extend the Firefox browser to include a user-friendly interface to data servers. Topaz is as easy to use as FTP clients but, at the same time, includes security techniques based on grid authentications as in the *globus-url-copy* command or UberFTP [9]. To our knowledge this is the first prototype of such an integration.

Topaz introduces an additional layer of abstraction in the communication between client and server. To quantify a possible overhead due to this abstraction, we compare upload and download transmission times of our protocol extension with other traditionally used tools such as UberFTP, the *globus-url-copy* command, an FTP client, and the Secure Copy Protocol (*scp*) command. DFDs are then used to analyze performance and identify the causes of slow downs.

The rest of this paper is so organized: Section II gives a brief introduction to the Globus Toolkit, GridFTP, and Mozilla. Section III introduces the Data Flow Diagrams and presents the design and implementation of the Firefox protocol extension using them. Section IV compares the performance of Topaz and other tools for data transmission. Section V discusses critical aspects related to Topaz and introduces future work. Section VI concludes the paper.

II. BACKGROUND AND RELATED WORK

The Globus Toolkit [10] has evolved into the standard for building grid systems, i.e., distributed systems that span multiple organizations incorporating heterogeneous resources while providing common security, job scheduling, data management facilities, information services, and a common runtime environment. It is the basis for a number of large-scale government sponsored scientific projects including Teragrid [11], Biological Informatics Research Network (BIRN), and many others. GridFTP [3] is one major component of the Globus Toolkit and provides file transfer capabilities from one grid node to another. It forms the basis for higher-level services such as the Globus Reliable File Transfer service (RFT) and the Data Replication Service (DRS). GridFTP is a protocol defined by Global Grid Forum Recommendation GFD.020, RFC 959, RFC 2228, RFC 2389, and a draft before the IETF FTP working group. The GridFTP protocol provides for the secure, robust, fast, and efficient transfer of (bulk) data. The Globus Toolkit provides the most commonly used implementation of that protocol, though other proprietary implementations do exist.

GridFTP, similar to the File Transfer Protocol (FTP), provides a client/server implementation for uploading and downloading files. The GridFTP protocol supports authentication using the Grid Security Infrastructure (GSI), authorization using gridmap files as well as SAML-based tools such as the CAS, and provides high performance through support for data striping across multiple server instances and support for parallel streams for data transfer. The Globus

Toolkit provides a server implementation along with a few client programs for accessing the server. The client programs include a command-line "*globus-url-copy url1 url2*" that takes a *gsiftp url1* of the form "*gsiftp://server/path/to/file*" and copies it to the server addressed by *url2*. The Globus toolkit also includes UberFTP, an interactive shell application that allows users to authenticate and upload/download files interactively.

Both GridFTP clients, *globus-url-copy* and UberFTP, require the Globus toolkit to be installed and hence are primarily server-based tools that are not directly used by most end-users. Instead, many grid projects provide web-based portals through which end-users can access GridFTP repositories from their desktops or laptops. The principle advantage of such an approach is that users need only a simple web browser installed on their local machine to access the file servers. However in this case the GridFTP protocol is used only between the portal server and the GridFTP server. The connection from the portal to the user's local machine uses HTTP and hence few of the advantages of GridFTP (restart, parallel file transfers, striping, etc.) are available.

The Mozilla project [4] also builds a toolkit, although with an entirely different focus. Created when Netscape made its Communicator product open-source, the Mozilla foundation provides a suite of open-source applications including the Firefox browser, Thunderbird email client, and Sunbird calendar, all of which are built on a common core also called Mozilla. From the start, the Mozilla core framework was designed to work on all modern platforms and to be highly modular in its architecture. Two key technologies used in the framework are XPCOM and XUL. XPCOM [12] is similar to CORBA and provides most of the functionality in Mozilla. Components can be written in C/C++, Python, and Javascript and are grouped into libraries that handle everything from filesystem manipulation, to security, XSLT, and rendering. XPCOM components are all cross-platform and new components can be added with a minimum of effort. XUL [13] is used to create GUIs in Mozilla. XUL is HTML-like in its simplicity yet Java Swing-like in its power; it can be combined seamlessly with CSS, SVG, Java applets and can access virtually any XPCOM component via a thin layer of Javascript. The ease of XUL and the robust, cross-platform nature of XPCOM combine to make Mozilla an ideal framework for rapid application development. Indeed, recent years have seen a proliferation of third-party applications like ActiveState's Komodo IDE built using Mozilla and over a thousand third-party extensions to Firefox and Thunderbird. According to current statistics [14], Firefox is the 2nd most popular browser, with over 12% of the market, and runs on all modern platforms.

III. DESIGN OF A FIREFOX PROTOCOL EXTENSION USING DATA FLOW DIAGRAMS

In this section we present a general overview of the Data Flow Diagrams (DFDs) and their applicability to design and guide the implementation of complex software systems (Section III.A). Using DFDs, we draw the data flow for both the Topaz download and upload functionalities (Section III.B).

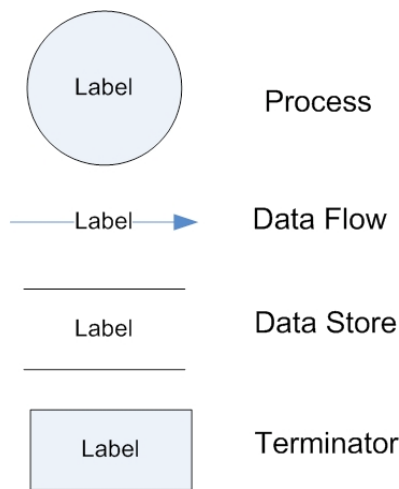


Figure 1. DFD keys.

A description of the software components of the Topaz module is presented in Section III.C. Sections III.D and III.E describe more in-depth the data and control flow for the download and upload respectively within the Topaz module as well as its implementation using DFDs.

A. Data Flow Diagrams

A Data Flow Diagram is a graphical tool in software engineering used for modeling information-processing systems, whole organizations, business planning, and strategic planning [15]. DFDs were introduced in the software engineering field in the mid 70's [16].

Typically DFDs are built using a combination of four components: processes, data flows, data stores, and terminators [17, 18]. A *process* is normally represented with a circle; it represents the transformation of inputs, e.g., packets of information or chunks of data, into outputs. Arrows that go in or come out of a process represent a *data flow*. A *data store* is drawn in a DFD using two parallel lines and represents a storage location from which we can retrieve or store data into, such as hard drives. A rectangle is used to represent a *terminator*, which is used to identify external entities. Traditionally, external entities are those that interact with the system by either providing or receiving data. In addition, each component has a label that describes its scope. Figure 1 shows a graphical representation of the DFD components. Note that DFDs differ from flowcharts because they emphasize the flow of data through the system rather than control and decision-making.

A system can be represented at different levels of abstraction by organizing the DFDs in a series of levels: each DFD level captures a different level of the system abstraction. The highest DFD level (usually called Level 0) represents the overall view of the whole system, where the system is normally represented by a single process and the major inputs and outputs are given by external entities. As we increase the DFD levels, we increase the level of details for representing the system or some of its components.

DFDs have been extensively used in industry and research laboratories to support the design and understanding of data

flow across various processes and subsystems in very large complex systems, as well as to analyze their real-time behavior. Examples of these applications include the communication validation in safety critical control systems such as an axis controller for an automobile [19], the modeling of radiation energy systems [20], and software requirement specifications [21].

B. Design of the Firefox Protocol Extension

The goal of the Topaz project is to extend Firefox with the following functionalities:

- Download (i.e., list as regular ftp, get file by clicking on name, drag and drop to desktop);
- Upload (i.e., put file by selecting from browser menu option, drag and drop to server);
- Remote upload-download (i.e., third-party transfer based on stripe transfer).

In our engineering effort, rigorous software engineering tools such as the Data Flow Diagrams support us to drive the design, implementation, and performance analysis of Topaz, our Firefox protocol extension. Currently our protocol extension supports download and upload functionalities. Remote upload-download is work in progress and is not covered in this paper.

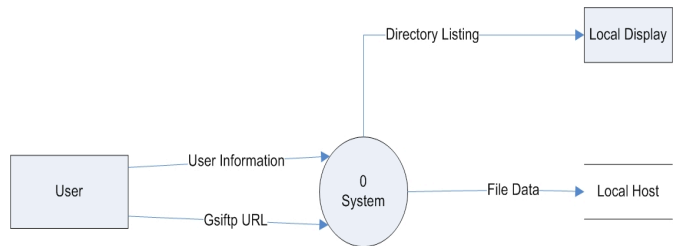


Fig. 2. Level 0 DFD for download.

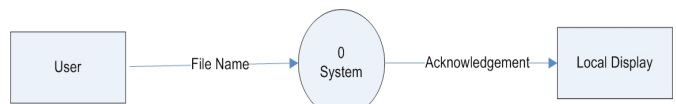


Figure 3. Level 0 DFD for upload.

Figures 2 and 3 provide the highest DFD levels (Level 0) of the two major functionalities: download and upload respectively. Figures 4 and 5 present Level 1 for the same functionalities.

In Figure 4, the user provides a gsiftp URL and user information (i.e., username and password) to Firefox (if such data has not already been provided in previous downloads). Firefox then forwards the user information and the URL to Topaz. If the user does not have a valid proxy credential, Topaz obtains one from an appropriate certificate authority using the user's certificate. Once a valid proxy has been obtained, Topaz uses the Globus FTP client API to establish a GridFTP control connection to the GridFTP server and authenticate the user. At this point, Topaz determines whether the URL represents a file or directory and issues the corresponding request to establish a data connection to download the content. Globus fetches raw data from server

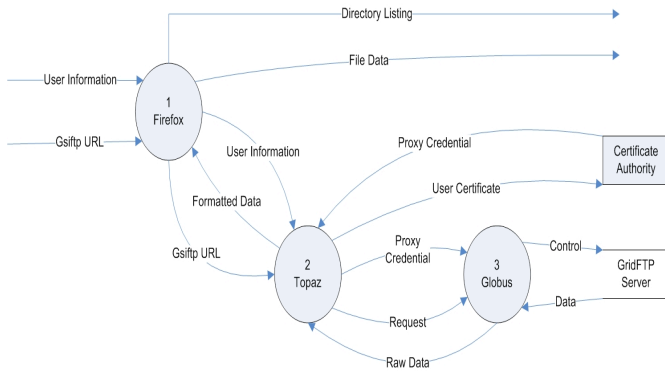


Figure 4. Level 1 DFD for download.

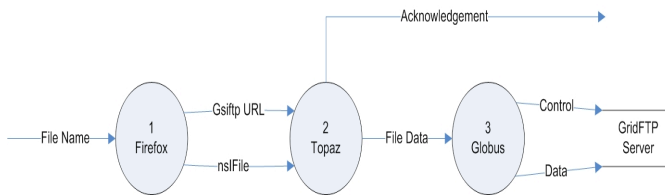


Figure 5. Level 1 DFD for upload.

and forwards it to Topaz that formats the data appropriately for the request and forwards the formatted data to Firefox. For list requests, Firefox converts the formatted list data (in application index format) to HTML and displays the listing in the browser window. For get requests, Firefox displays the raw data in the browser window or saves it to a local file if “Save Link As ..” is selected from the pop-up menu. In case of failure, Topaz instructs Firefox to display a proper message.

In Figure 5, the user selects the file to upload through the Firefox file dialog. The file object and current gsiftp URL from the location bar are passed to Topaz. Topaz establishes a control connection, authenticates the user, and opens a data connection for upload. Topaz also opens the file and sends the data over the data connection to GridFTP server.

While the version of Topaz presented in this paper is used exclusively through the Firefox browser, it can also be packaged as a standalone command-line tool, a standalone application GUI, or as part of any other Mozilla based application.

C. Software Components of the Protocol Extension

The Mozilla framework is modular in nature and open to modifications and additions. Firefox retains this flexibility by enabling extensions, like Topaz, to enhance its core functionality as shown in Figure 4 and Figure 5. Extensions are packaged for distribution into a special zip file called an XPI that contains compiled binaries and/or Javascript, GUI files, and extension metadata. Topaz is installed with a single click via the Firefox Extension Manager that downloads the XPI and configures the browser to make use of the new protocol extension.

Topaz comprises five software components: the protocol handler, the login manager, the upload manager, the channel, and the stream. The protocol handler is responsible for processing gsiftp URLs in Firefox and is used only for downloads. The stream handles data transfer from GridFTP

servers (download), or to GridFTP servers (upload). For downloads, the channel handles data flow between the browser and the stream; the login manager manages the users proxy credential. For uploads, the upload manager creates a stream and provides it with the current URL.

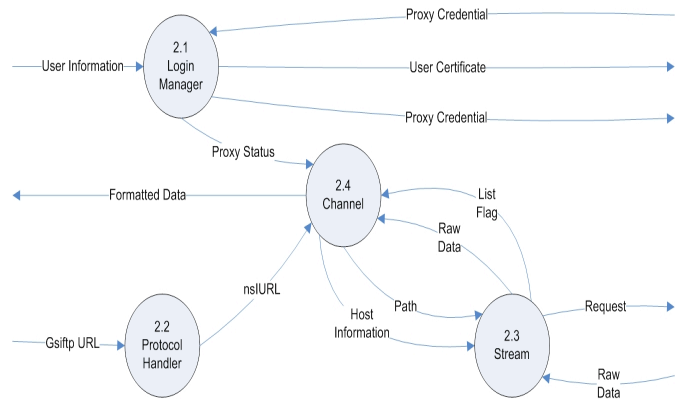


Figure 6. DFD Level 2 for download.

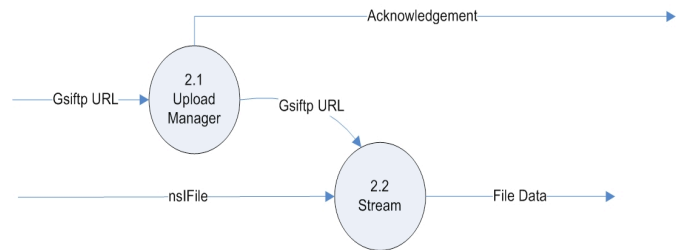


Figure 7. DFD Level 2 for upload.

Figures 6 and 7 show Level 2 of the data flow within the Topaz module for download and upload respectively. For each functionality, a different data flow takes place among the software components. An in-depth description of the data flow for the download and the upload is presented in Section III.D and Section III.E respectively.

D. Download Functionality

The detailed DFD of the download functionality is shown in Figure 6. Once Firefox creates the Topaz module, it passes the gsiftp URL to the protocol handler. The protocol handler creates an *nsIURL* object from the URL string. This is an XPCOM object that contains the URL information and the appropriate methods to parse it. The protocol handler creates a channel for the transfer and passes it the URL object. The channel creates a login manager to verify that the user has a valid proxy credential. If a valid proxy does not exist, the login manager presents the user with a dialog box to enter a username and password and to select a certificate authority in order to obtain a proxy. Possible authentication servers are: GAMMA servers [22] and MyProxy servers [23]. Once the proxy is obtained, it is stored in a standard location, i.e., the *.globus* directory in the users home directory.

After the login manager returns successfully, the channel calls *nsIURL* methods to parse the GridFTP server name and the path of the file or directory being requested. The channel

creates a stream and passes it the host and path information. Like FTP, GridFTP uses separate connections for transmitting control information and data, however, it provides additional security by allowing these connections to be encrypted. By default, the control connection is encrypted and the data connection is not. The control connection is established by the stream using the Globus function `globus_fip_control_connect()`. The stream then calls `globus_fip_control_authenticate()` to authenticate the user on the GridFTP server using the proxy in the user's `.globus` directory.

At this point, the stream obtains the size of the content being downloaded using the `globus_fip_client_size()` call. In order to determine whether the content is a file or directory listing, the stream issues the `globus_fip_client_get()` call to open a data connection and `globus_fip_register_read()` to get a portion of the content. If this first call returns data then the URL corresponds to a file and the download can continue. This data is passed to the channel and then to Firefox.

On the contrary, if the read call does not return any data the URL corresponds to a directory. In this case, the stream calls `globus_fip_client_list()` in order to begin a directory listing download. The directory data is retrieved using the `globus_fip_register_read()` function. The data is passed to the channel, along with a flag indicating that this is a directory listing. When this flag is detected, the channel creates an XPCOM `nsIStreamConverterService` to convert the directory listing into HTTP-index format. This format is designed to provide a machine-readable directory listing. The data is then passed to Firefox that displays it as a web page.

E. Upload Functionality

The detailed DFD of the upload functionality is shown in Figure 7. In the upload, the check for a valid proxy is done a-priori by doing a directory listing on a GridFTP server using Topaz as described in Section III.D. A file upload is initiated when a user selects "Upload File" from the "File Menu". This menu item is added to the existing Firefox "File Menu" using a XUL overlay. The Topaz module creates an upload manager to process the request. The upload manager gets the current URL string from the location bar of the current browser window. It then creates a stream and passes it the URL string.

At this point, the stream creates an XPCOM `nsIFilePicker` to prompt the user for the file to upload. If a valid file is given, it is opened for reading. The stream then creates a `nsIURL` object from the URL string and parses the GridFTP server name and path where to upload the file. The stream uses the `globus_fip_control_connect()` call to open a control connection to the specified server. The user is authenticated on the GridFTP server with the `globus_fip_control_authenticate()` call using the proxy in the `.globus` profile. If the authenticate request is successful, the stream issues a `globus_fip_client_put()` call to establish a data connection to the GridFTP server; otherwise a proper error message is generated. The data is read from the file and sent to the GridFTP server using the `globus_fip_client_register_write()` call.

IV. PERFORMANCE EVALUATION

Since Topaz is built on top of Globus libraries and can be integrated into the Firefox browser, we introduce an additional layer of abstraction in the communication. To quantify any possible overhead, we compared the transfer times in seconds for uploading (i.e., get) and downloading (i.e., put) data between a client at the University of Texas at El Paso and a GridFTP server at the San Diego Supercomputer Center using different transmission tools: Topaz, the `globus-url-copy` command in Globus 4.0.1, UberFTP 1.13, LFTP version 3.2.1-2, and the Secure Copy Protocol command `scp`. We used a set of files with different sizes (i.e., 32KB, 256KB, 2MB, 16MB, 128MB, and 1024MB) and we ran the different upload and download experiments three times for each protocol and each file size. The numbers reported in this section are the average values of those measurements.

For each tool, we measured the time from when the command is issued by the user until the completion of the transfer. In particular, for Topaz, we measured the time from when the channel is created until the stream is finished with the transfer for downloads (Figure 6) and from when the stream receives the filename until it is finished with the transfer for uploads (Figure 7). For `globus-url-copy` we measured the time from when the command is issued until its completion. UberFTP, LFTP, and `scp` give the transfer time after each operation and this is the time reported in this paper. While running our performance measurements, we used default configurations for the several tools and no optimization has been applied: this is because we wanted to emulate the usage conditions for most scientists that normally have neither the needed expertise nor the time to configure these tools with high levels of optimization. Since Topaz deploys GridFTP, it can benefit from the several optimizations available for this protocol, e.g., multi-streaming, buffer size tuning. This paper focuses on the additional overhead that our extension can cause rather than its and other protocols' optimizations.

The results of our comparisons for the download and upload transfer times are presented in Figures 8 and 9 respectively. In particular, Figure 8 shows the average transfer time (y-axis, in logarithmic scale) measured for downloading files with different sizes (x-axis) and Figure 9 presents the same measurement, the transfer time, for the upload of files with different sizes. Both of the figures show that although Topaz is characterized by an initial transfer time that is larger than the other tools, in both cases, for the download and the upload, this gap tends to decrease as the size of the file increases and for files larger than 16MB the time becomes the same.

For small files, less than 2MB, the time gap between `globus-url-copy` command and Topaz is constant. The difference for the download times is equal to 1.8 seconds. The time gap between upload times is smaller and equal to 0.7 seconds. The reason for the additional overheads in the Topaz functionalities can be found in Figures 4 and 5 that show the additional layer of abstraction of Topaz on top of the Globus libraries. The DFDs in Figures 6 and 7 depict the reason for a larger overhead in the download functionality. Note in these figures how the download functionality has a more complex transfer structure than the upload functionality. Indeed the

download requires the generation of a channel by the protocol handler as well as the generation of the login manager and the stream by the channel. On the contrary, for uploads, the upload manager generates only the stream.

V. DISCUSSION AND FUTURE WORK

There are several advantages in using Firefox to create a GridFTP client. The browser provides users with an easy to use interface that most users are already familiar with. Firefox provides a number of features that are available to the protocol such as drag and drop support and dialog boxes for saving files and providing passwords. Since Firefox handles the user interface, it makes it easier and faster to develop a GridFTP client across multiple platforms.

There are also some challenges in integrating Topaz into the Mozilla framework. Mozilla is still largely under development

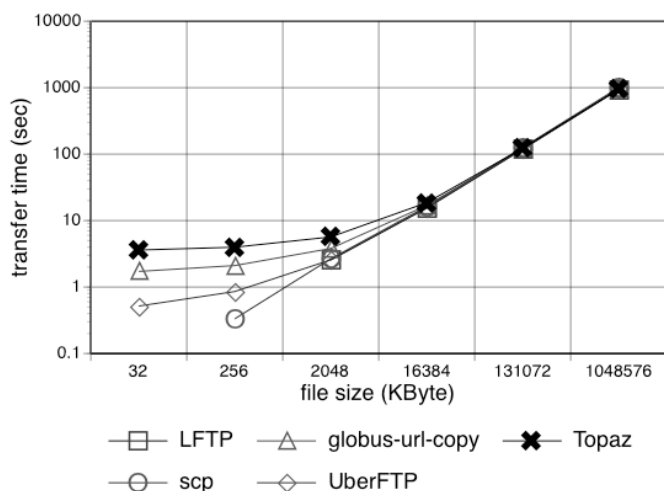


Figure 8. Transfer rate of download (get).

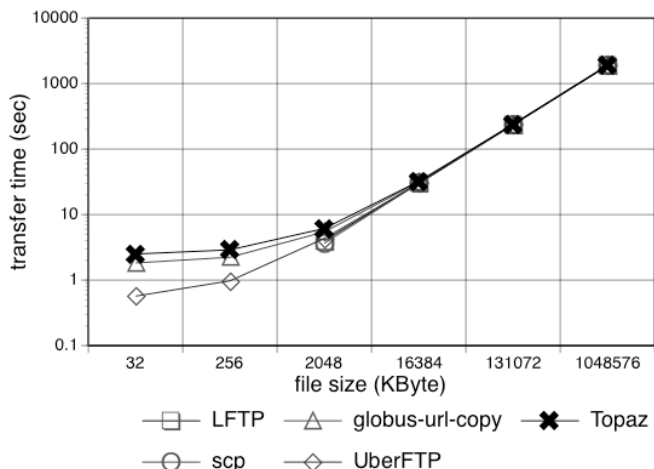


Figure 9. Transfer rate of upload (put).

and has a number of unfrozen interfaces, i.e., that may be changed in future releases. Because of this, interface references within Topaz will need to be maintained. Moreover, the dynamic nature of Mozilla can make it challenging to find good documentation.

In order for users to take full advantage of the GridFTP protocol, we are working on the integration of its third-party transfer functionality into Topaz. A third-party transfer will allow Topaz to connect to two GridFTP servers simultaneously and initiate a direct file transfer between the servers. This will be accomplished by dragging a file from one browser window and dropping it in the other. Lastly, future work will include the integration of the GridFTP parallel streams for data transfer.

VI. CONCLUSION

This paper presents Topaz, a Firefox protocol extension for GridFTP. Topaz provides a simple and secure access to grid technology. It can run as a stand-alone tool or can be integrated in the Mozilla framework. Rigorous software engineering tools such as Data Flow Diagrams guided its design and implementation. The diagrams also helped us to address the performance evaluation of Topaz. Our performance evaluation shows that the initial overhead due to the additional layer of abstraction introduced by Topaz becomes insignificant as the size of the transferred files increases, i.e., above 16MB. This makes Topaz a well-suited, user-friendly, and secure tool for large file transfers. Topaz is an in-progress, open-source project and future work includes support for third-party transfers and parallel stream data transfers.

ACKNOWLEDGMENT

Financial support through the National Science Foundation (grants # SCI-0438430, SCI: NMI Development: The Computational Chemistry Prototyping Environment; # EIA-0080940, MII: Graduate Education for Minority Students in Computer Science and Engineering: Extending the Pipeline; and # SCI-0506429, DAPLDS - a Dynamically Adaptive Protein-Ligand Docking System based on multi-scale modeling) is acknowledged. We wish to thank Andre Kerstens for his help in running the performance tests as well as Trilce Estrada and Luis Perez for their comments.

REFERENCES

- [1] Foster, I., Kesselman, C. and Tuecke, S., The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 2001.
- [2] Feng, J., Cui, L., Wasson, G., and Humphrey, M., Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET. In Proceedings of the 2005 Grid Workshop (Associated with Supercomputing 2005), November 2005.
- [3] Allcock, W., Bresnahan, J., Kettimuthu, R., Link, M., Dumitrescu, C., Raicu, I., and Foster, I., The Globus Striped GridFTP Framework and Server. Proceedings of SC2005, ACM/IEEE Supercomputing 05, November 2005.
- [4] Mozilla - An Open-source Web Browser. <http://www.mozilla.org>
- [5] Baldrige, K.K., Greenberg, J.P., Sudholt, W., Bhatia, K., Mock, S., Amoreira, C., Potier, Y., and Taufer, M., The Computational Chemistry Prototyping Environment. Proceedings of the IEEE Special Issue on Grid Computing, 2004.
- [6] Greenberg, J.P., Katz, M., Bruno, G., Sacerdoti, F.D., Papadopoulos, P., and Baldrige, K., Incorporation of Middleware and Grid Technologies to Enhance Usability in Computational Chemistry Applications. International Conference on Computational Science, 2004. Krakow, Poland: Springer - Verlag.
- [7] GEMSTONE - Grid Enabled Molecular Science Through Online Networked Environments. <http://grid-devel.sdsc.edu/gemstone>

- [8] Miller, A.K., mozCellML, Unpublished results, 2004.
- [9] UberFTP – an Interactive GridFTP-enabled ftp client.
<http://dims.ncsa.uiuc.edu/set/uberftp>
- [10] Globus – An Open-source Software Toolkit used for Building Grid Systems and Applications. <http://www.globus.org>
- [11] TeraGrid - Open Scientific Discovery Infrastructure to Create an Integrated, Persistent Computational Resource. <http://www.teragrid.org>
- [12] XPCOM - Cross Platform Component Object Model.
<http://developer.mozilla.org/en/docs/XPCOM>
- [13] XUL - XML User Interface Language. <http://www.xulplanet.com>
- [14] Janco Associate Inc., Firefox Snags Over 12%. Press-Release, January 2006.
- [15] Yourdon, E., Modern Structured Analysis. Prentice Hall, 1989.
- [16] Yourdon, E. and Constantine, L., Structured Design. New York: YOURDON Press, 1975.
- [17] DeMarco, T., Structured Analysis and Systems Specification. Englewood Cliffs, NJ: Prentice-Hall, 1979.
- [18] Gane, C. and Sarson, T., Structured Systems Analysis: Tools and Techniques. Englewood Cliffs, NJ: Prentice-Hall, 1978.
- [19] Herard J. et al., Validation of Communication in Safety Critical Control System. Technical Report TR 543. Published by Nordtest. URL:<http://www.nordtest.org>
- [20] Barkstrom, B. and Wielicki, B., Clouds and the Earth's Radiant Energy System (CERES). Algorithm Theoretical Basis Document. Atmospheric Sciences Division, NASA, 1996.
- [21] Grubb, T. et al., Landsat 7 Processing System (LPS). Software Requirements Specification. Goddard Space Flight Center, 1995.
- [22] Bhatia, K., Mueller, K., Chandra, S. GAMA: Grid Account Management Architecture. In Proceedings of the First IEEE International Conference on e-Science and Grid Computing, December 2005.
- [23] Basney, J., MyProxy Protocol. Global Grid Forum Experimental Document GFD-E.54, November 2005.