

Using Just-in-Time to Enable Optical Networking for Grids

Steven R. Thorpe and Daniel S. Stevenson
MCNC Research and Development Institute
Advanced Networking Research Division
3021 Cornwallis Road
Research Triangle Park, NC 27709 USA
{thorpe, stevenso}@mcnc.org

Gigi Karmous Edwards
MCNC Grid Computing and Networking Services
Advanced Technology Group
3021 Cornwallis Road
Research Triangle Park, NC 27709 USA
gigi@mcnc.org

Abstract

This paper presents an overview of the Just-In-Time (JIT) control plane and GridJIT service that has been developed for optical networks and describes several related projects. We believe there is a synergy between Grid computing needs and JIT networking: data grids typically require large, fast, and latency-sensitive data flows; we will show that the JIT control plane provides the network infrastructure to meet those needs. In this paper we: i) provide an overview of the JIT protocol and control plane, ii) discuss the relevance of high demand Grid applications to JIT, iii) discuss the advantages of using JIT in an all-photonic network vs. an OEO-based optical network, iv) present our thoughts on future Grid networking that can be realized using JIT, and v) describe Open Grid Services Infrastructure (OGSI), HDTV, and GridFTP-related projects that were built using JIT.¹

1. Introduction

Many of today's compute- and data-intensive e-science applications are looking to Grid-based technologies to meet their high demands. Until recently, the Grid community focused primarily on maximizing the availability, sharing, and utilization of resources such as CPU power and storage. Now, many in the Grid community [15, 10, 16, 12] are starting to regard the network as another vital Grid resource, to be used to provide large, fast data flows with minimal latency and jitter. MCNC Research and Development Institute (MCNC-RDI)² and North Carolina State University (NCSU) have developed a Just-In-Time (JIT) control plane, signaling scheme, and various software and hardware components that are synergistic with these needs.

¹This work was partially supported under NASA (National Aeronautics and Space Administration, <http://www.nasa.gov>) contract NAG2-1467.

²MCNC ANR Projects - Jumpstart. <http://jumpstart.anr.mcnc.org>

Across the globe, many government sponsored Grid research testbeds have been created for Grid-related research that involves extensive e-science. These testbeds are intended to address the concern of many in the scientific research community who think that today's production, experimental and research networks do not have the capabilities to meet the needs of some of the existing e-science and Grid applications[8]. Many of these applications have significant needs and constraints like determinism (guaranteed QoS), shared data spaces, real-time multicasting, large data transfers, latency requirements, and on-demand connections. To work effectively, such applications have shown significant improvements when using dedicated light paths (lambdas). Meeting these needs through use of dedicated end-to-end light paths is admittedly an expensive proposition unless one can hold the network resources for only the time needed to transfer the payload. We will discuss how this is possible using the JIT control plane. Another important requirement for these Grid applications is user/application-initiated network connections. Grid communities are exploring different control planes to address these requirements[3]; unique to the JIT approach is rapid reconfiguration to provide application-controlled multiplexing (ACM). We will discuss how JIT can meet some of these requirements; others are left for future work efforts.

The first application of the JIT control plane was on an all-photonic Dense Wavelength Division Multiplexing (DWDM) network. JIT also exhibits benefits in today's existing optical networking infrastructure, such as Multi Service Provisioning Platforms (MSPPs). In addition, the JIT control plane has been implemented in both hardware and software components for the optical switches and the client-side nodes. Initial prototypes of these components have been completed and integrated, and they are currently being tested, debugged, and refined within lab settings and also on a metro-area optical network.

A major focus of this paper is the GridJIT service and

how it allows Grid applications to access network resources through the JIT signaling and control plane. The Grid-JIT service exposes the ability to open and close JIT light paths through an Open Grid Services Infrastructure (OGSI)-compliant Grid Service. This allows Grid applications to request the networking resources on an on-demand basis for a specified time. After the transfer of data, the networking resources are released immediately, allowing other applications or data flow to access the resources. We contend that for global Grid collaborations, the sharing of scarce network resources among different high-end workstations is necessary, and the faster the connection creation and deletion occurs the greater the degree of multiplexing of different data streams between institutions.

The first deployment of JIT on a network outside of the lab was in October 2002 on the Advanced Technology Demonstration Network³, a high-performance networking testbed in the Washington, D.C. area. This network infrastructure included Lambda Optical Systems⁴ MEMS based all-photonic switches (a bufferless network). Several demonstrations have been conducted on ATDnet, including fast light path creation and deletion (approximately 12.5 ms reconfiguration times) between several HDTV video signals over the network. A key feature of JIT signaling is the ability to create a light path connection rapidly. The time scale of a switch's reconfiguration is dependent on the technology used⁵.

The project team has also focused on the definition of a JIT-friendly transport layer. Part of the overall strategy for transport protocols has been to provide early proof-of-concept uses of the JIT protocol by a user application. Toward this end and in collaboration with the U.S. Naval Research Laboratory (NRL) and SGI, we have modified an implementation of the Scheduled Transfer protocol (ST) [2] to run over JIT within kernel-space. A C version of the publicly available GridFTP protocol [5, 6], which is widely used within the high performance Grid community for transport of large datasets, was then modified to run over the JIT-enabled ST protocol. In addition, the GridJIT service discussed in this paper was integrated with an Java implementation of GridFTP to enable file transfer through JIT and an unmodified file transfer mechanism.

This paper will first briefly review the JIT protocol, along with our thoughts on the benefits that Grid environments can realize using JIT networking. Next we will describe our JIT protocol experiences from several different projects. Finally, the paper concludes with a short summary and a brief description of our anticipated future work.

³Advanced Technology Demonstration Network. <http://www.atd.net>

⁴Lambda Optical Systems. <http://www.lambdaopticalsystems.com>

⁵MEMS based optical switches reconfigure in a little over 10 ms, while novel optical switching technologies like polymer-based switches promise reconfiguration times in the order of nanoseconds. Existing MSPPs, like the Cisco 15454, reconfigure in the order of low 20s milliseconds.

2. JIT Overview

In this section we present an overview of the JIT control plane and its potential benefits to the Grid community. For more details about the protocol, see [1, 7, 17, 18].

2.1. JIT Control Plane and Signaling Protocol

JIT is a reservation-based control plane whose distinguishing characteristics include relative simplicity, reduced latency and jitter, and ultra-fast switching capability. The high speed switching capability allows for application-controlled multiplexing (ACM). ACM involves having the application request the network resources for the time necessary to transfer the data payload. The basic premise of JIT is a true separation between control plane and data plane by having the header and the payload detached from one another. The header precedes the payload by a short period of time, called the offset time. The offset time calculation is dependent on several factors including switch reconfiguration times, physical distance to destination, and number of hops to destination. The offset time is used by the switch to process the header and reconfigure the switch per request prior to the arrival of the payload. The result is a bufferless network, when no OEO is necessary on the payload (all-photonic network).

The JIT signaling mechanism is based on a Tell & Go (TAG) protocol instead of Tell & Wait like TCP (Round Trip Time (RTT) for acknowledgment). Two reasons for JIT's ability to create and delete end-to-end connections rapidly are the absence of the round trip times from destination to source and the implementation of header processing in the hardware. Therefore, it is conceivable to have dedicated connections for time periods of milliseconds or shorter based on the application's needs. It is then arguable that a JIT-type connection can be from only a few packets to as large as a long lived-light path. Upon the completion of data transfer, the connection either times out or is torn down explicitly, depending on the characteristics requested by the originating node. This ability of applications to choose their connection time span is referred to as ACM.

Figure 1 illustrates implementation of these fundamentals when using the JIT signaling protocol to gate optical data transmission. As you move from left to right within the figure, you travel from the source node, which originates data, to the destination, or receiving node. Time advances as you move from top to bottom within the figure. SETUP, SETUP ACK, RELEASE, and KEEP ALIVE are signaling messages that are electronically processed at each node, ultimately directing the data to the appropriate next hop within the network. Processing of the signaling messages is done in hardware at each switch in order to keep time to a minimum. The data itself is not processed at each

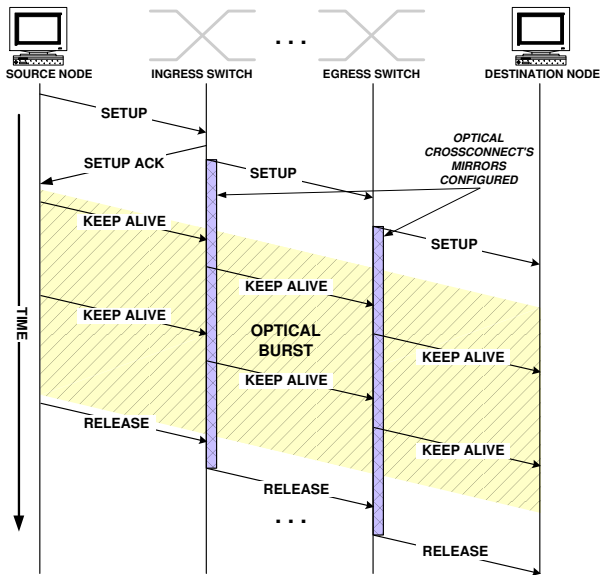


Figure 1. Example of an optical data transmission and its corresponding JIT signaling messages.

node; it simply passes through as directed by the switch. Note how the data catches up with its signaling messages as they traverse the network due to the signal processing time at each node. Note also that data transmission begins at the source node before the destination node has learned of the impending transmission. This example shows only one of many possible connection types supported by the protocol; additional types are described in [1].

At each node within the JIT network, finite-state machines maintain information on the status of each JIT connection on the node [18]. This state information is used to respond appropriately to signaling messages related to a connection. Multiple state machines have been designed to manage other scenarios including 1) source and destination client nodes; 2) ingress switch; 3) intermediate switch; and, 4) unicast and multicast.

2.2. JIT Routing and Forwarding

The JIT control plane relies on a distributed link state protocol for network topology and characteristics. The routing algorithm can be either distributed or centralized, based on the network's priority. For each end-to-end source and destination the routing algorithm produces a primary path as well as disjoint alternate paths. These paths are then distilled into individual forwarding tables for each switching node in the network. Header processing within a switch consists of a quick lookup in the forwarding tables for the

output port for the next hop. The majority of this is handled in hardware that provides ultra fast processing of the header messages. One consequence of JIT's TAG approach is the possibility of blocking at an internal switch node. A mechanism for reducing such an event is alternate next-hop look-ups in the forwarding tables, also known as deflection routing. If the primary next-hop port is busy, alternate next-hops are selected. This process reduces the blocking probability significantly.

2.3. Today's Grid Network Infrastructure

JIT provides a migration path from today's infrastructure to tomorrow's optical advances. The JIT Protocol Accelerator Card (JITPAC) described later in the paper works with existing MSPPs like Cisco's 15454 as well as many all-photonic switches. Future optical networks are envisioned as all-photonic bufferless networks with nano-second scale switch reconfiguration times and 100% wavelength conversion. The JIT control plane is poised to take advantage of these future technologies by allowing for sub-wavelength division multiplexing. In an all-photonic network the data plane is completely transparent, allowing for all format signals including analog. Today's OEO switch nodes require processing of data which includes handling of the various modulation formats, encoding, and protocol type.

2.4. Meeting Grid Requirements

Some of the unique requirements of the Grid community are: i) very large data sets to be shared and transferred, ii) on-demand application- or user- initiated high-bandwidth connections, iii) coordination of network resources with compute resources such as CPU (or clusters) and storage. E-science applications are some of the main drivers for these high-end Grid applications [4, 9, 14]. Most large-scale e-science applications involve multiple collaborating institutions around the globe, each separated by WANs and MANs⁶. A collaborating institute may have several systems (high-end workstations) involved in data exchange with one or more institutions. Typical scenarios include a terabyte-scale file transfer or navigating through a visualization session, which demands new levels of network service that keep network latency and jitter low. In certain visualization scenarios, for example, data should ideally be streamed in uncompressed form because compressing images can accrue latency and jitter, thereby reducing the quality of the visualization.

⁶Some of the government supported research networks existing today support testbeds such as Starlight, ATDnet, CA*4net, NetherLight, etc. and provide the infrastructure necessary for international collaboration and are exploring mechanisms for improved interoperability.

To meet these requirements researchers have been establishing end-to-end light paths for data exchange that connect a single workstation to another at the other end. We contend that establishing a single high bandwidth light path between two workstations across a network is expensive and not sufficient for meeting the needs of most of these applications since they lack the capability of sharing the network resources among different workstations and applications. Therefore, the solution must provide a method of multiplexing multiple data exchanges between the collaborating institutions, with each data-exchange having a dedicated light path only for the amount of time necessary (time frames can be from microseconds to hours or days depending on the switching technology and application requirements). Since the JIT control plane offers ultra fast, ultra fine-grained, user- and application-initiated provisioning of light paths in full and sub-wavelength increments, we contend that if placed at the edges of these research networks, it can provide the multiplexing necessary for efficient use of a core network dedicated light path. This is described as a fan-in fan-out scenario. We distinguish the core light path from the edges by the dynamic nature of the light path setup and tear down. In other words, the initial setup of an end-to-end lightpath may include a core section of the lightpath which remains static for the duration of a data exchange session that can be configured using JIT signaling or other signaling protocols such as UCLP, GMPLS, *etc.* The assumption made is that reconfiguration time of the core section of the light path may be slower, and therefore less dynamic than what is required at the edges for ACM. The end-to-end lightpath can be completed using ultra fast JIT signaling. In this scenario, several workstations can connect to a JITPAC controlled switch or switches at the edge allowing the switching between the different workstations onto the core dedicated light path. Only one workstation will have use of the core light path at any given time, providing multiplexing based on ACM. The promise is that applications can realize the performance of dedicated channels but with the efficiency of dynamically sharing these facilities.

This clearly holds an advantage for Grid applications such as the parallelizable remote visualization steering shown in Figure 2. The researcher initiates a parallel visualization session on remote compute resources through Grid service. Multiple displays reside local to the researcher; together, the display panels render a single visualization object. Prior to object rendering, the large data set is divided over multiple servers for more efficient computation and I/O. Each display is updated from the server of servers which collects data from multiple, separate remote servers. Once the object is rendered on the displays, the user can start their analysis. User input at the client (e.g., "change this isosurface level", "rotate display", "analyze variable X", "animate over time or space") generates control commands

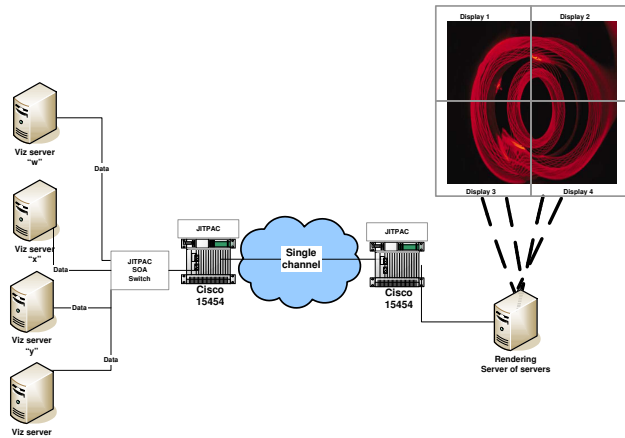


Figure 2. Distributed visualization over JIT.

that are passed to the remote servers. The appropriate control commands are then sent to each remote server, triggering large flows of data/geometries across the network to the client side and then rendered to the display wall. Each remote server updates its associated client-side server of servers in a near-realtime fashion. Each remote server will use the JIT control plane to initiate an end-to-end light path between itself and the client-side server of servers, send its update, and then release the resources for the next server to perform its update. Each update is latency and jitter sensitive, and the QoS is provided by executing the update via an end-to-end dedicated light path for the duration of the transfer. As the object is rotated through mouse movement, near-real-time object rendering occurs. This is a clear example of how ACM can meet the needs of the Grid community.

We believe that JIT will prove an effective mechanism for rethinking network fundamentals. This research is being approached without any legacy mindsets and with a 20-year technology lifetime in mind, during which rapid changes in technology are expected and will be taken advantage of. JIT is a simple protocol in which round-trip handshakes have been eliminated. It will have an optimized implementation with appropriate functional placement of optics, electronics, and software. It will eliminate data buffering within the network and keep intelligence concentrated at the network edges, thereby maximizing performance.

2.5. JIT Deployment Infrastructure

Figure 3 illustrates an example of JIT hardware and software prototypes as deployed onto a network such as the ATDnet. The figure includes three client nodes, three optical network elements (OXCs), and three JIT Protocol Accelerator Cards (JITPACs). The OXCs, supplied by Lambda Optical Systems, contain mirrored crossconnect devices

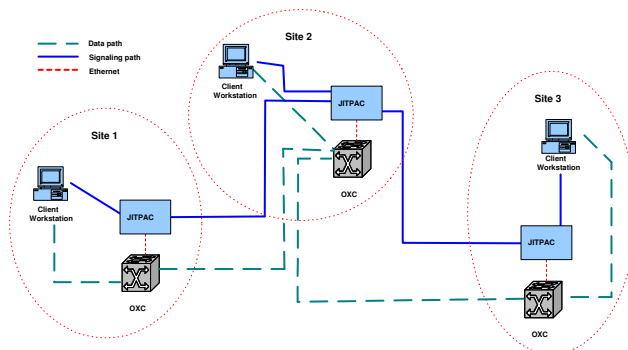


Figure 3. Prototype JIT deployment at sites.

that can be configured to direct incoming optical data to any available output fiber. Each OXC has an associated JITPAC, which sends mirror configuration commands to its OXC's embedded CPU using a remote procedure call (RPC) mechanism. The JITPAC device [13] is a system which integrates an embedded CPU, a Field Programmable Gate Array (FPGA), memory, and ATM and Ethernet adapters. The FPGA manages the state machines for each optical transmission passing through the OXC. The SwitchController, a software application running on the JITPAC's embedded CPU, shepherds communications among the FPGA, the external nodes within the network (through JIT signaling messages over ATM or UDP), and the OXC (through RPC calls over Ethernet). Note that while the out-of-band signaling channel in this deployment is carried over ATM or UDP, in future implementations we expect it to be allocated on a specific optical wavelength throughout the network on the same fibers as the data channels.

The client nodes shown in Figure 3 are responsible for: 1) managing client-side state machines for each JIT connection; 2) JIT signaling message construction and parsing; 3) transmission and receipt of JIT signaling messages; and, 4) transmission and receipt of optical data.

2.6. JIT API

The JIT API is a software library that provides data connection establishment and maintenance, and direct access to the various features of the JIT signaling protocol. It assumes that the JIT signaling protocol is used in conjunction with an appropriate data adaptation layer, as well as a higher-layer transport protocol. Therefore, one recommended use of the API is inside a kernel of an OS. It can also easily be made accessible to user-space applications, as it is written entirely in the C language.

The JIT API contains the software implementation of the finite state machines that maintain information on the sta-

tus of each JIT connection on the node. The API presents a view of the network to its upper layer within the stack (typically but not necessarily the OS kernel), allowing the upper layer to request a connection for a given transmission of data and then be told when the data can be transmitted. All related measurements are done in units of time, not in units of volume of traffic. The JIT API makes no attempt to enforce the transmission of the data at appropriate moments specified by the network through the signaling protocol. Instead, it simply notifies the upper layer when such a moment arrives, leaving the actual transmission of data on the fiber and the particular format of the transmission to the specific adaptation layer used in conjunction with the JIT signaling protocol.

The JIT API software library is used within each of the JIT study projects described in subsequent sections of this paper.

3. GridJIT Project Study: OGSi Accessible JIT Light Paths

The Globus Project's Open Grid Services Architecture (OGSA)⁷ and the Open Grid Services Infrastructure (OGSI) upon which OGSA is built represent an exciting evolution towards a Grid system architecture based on Web services concepts and technologies. These Grid middleware standards provide a common framework for connecting a wide variety of distributed computational, storage, and networking resources. OGSA allows different Grid technologies to easily leverage each other, using the common language provided by the OGSI implementation of the Globus™ Toolkit.

We believe it is highly desirable to provide access and control of JIT connections through an OGSI-based Grid service interface, in order to support interoperability of JIT networking with other Grid-aware technology that requires dynamically created light paths on demand. Therefore, we have developed such an OGSI service, which we call *GridJIT*. The GridJIT service is designed to run a single service instance on each JIT-enabled host and is able to communicate with a JITPAC to set up and tear down JIT light paths from that host. The underlying C libraries of the JIT API were wrapped into a C++ class, which itself was wrapped into Java bindings using SWIG⁸. These Java bindings were then exposed and deployed as the GridJIT service using the OGSI implemented by Globus Toolkit version 3.0.2. As the sample code below shows, it is straightforward to use the GridJIT service from a Java program:

```
// Connect to the GridJIT service
GridJITClient gridJITClient = new GridJITClient(
```

⁷Globus Project: Open Grid Services Architecture. <http://www.globus.org/ogsa>

⁸Simplified Wrapper and Interface Generator. <http://www.swig.org>

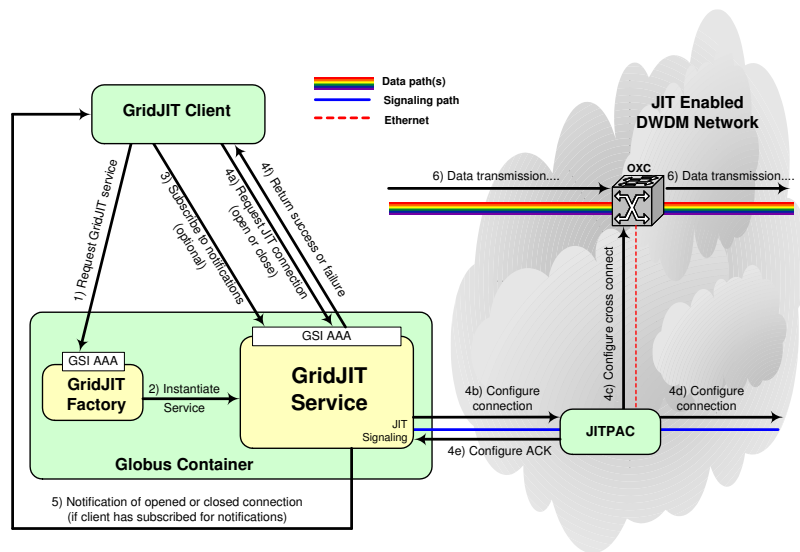


Figure 4. GridJIT event sequence.

```

"http://127.0.0.1:8080/ogsa/services/gridjit/GridJITFactoryService/gridjit";
// Start listening for notifications
gridJITClient.startListening();

// Open a light path from the GridJIT service's host to 128.109.179.252
IPAddress destAddress = new IPAddress("128.109.179.252");
UnsignedInt duration = new UnsignedInt(0); // explicit close required
LambdaIndex reqChan = new LambdaIndex(new UnsignedByte(6)); // lambda 6
JITConnection jc = gridJITClient.openLightPath(destAddress, reqChan, duration);

// Close the light path
gridJITClient.closeLightPath(jc);

// Stop listening for notifications
gridJITClient.stopListening();

```

The first GridJIT implementation was completed as of this writing (Spring 2004). Figure 4 shows the simple sequence of events generated when a client requests a light path from the GridJIT service. GridJIT was tested using a software-only version of the JITPAC called the *SoftJITPAC*. Once GridJIT was in place, it was a simple task to write a short client-side Java program which 1) connects to GridJIT service instances on two different hosts; 2) simultaneously requests light paths from each GridJIT service to the other; 3) once the light paths are open, uses the COG kit's⁹ Java GridFTP client library to initiate a file transfer between the two nodes using the GridFTP protocol, over the "light path"¹⁰ opened by step 2; and, 4) simultaneously closes down both light paths.

Now that the initial GridJIT service is in place, it should be straightforward to interoperate with other OGSi services and use GridJIT for their light path needs. In fact, the GridJIT service could even interoperate with other Grid services

⁹Java Commodity Grid Kit 1.1 Alpha. <http://www-unix.globus.org/cog/java/1.1a>

¹⁰Since we don't yet have an actual cross connect device at MCNC-RDI, we're using a "SoftJITPAC" and a "Virtual OXC" instead. The GigE cards are directly linked to each other instead of through a JIT switch.

to concatenate multiple light paths end-to-end, each created with different underlying architectures, resulting in a single light path across different domains. For example, the User Controlled Lightpaths Project¹¹ sponsored by CANARIE Inc.¹² has a standard Grid interface that allows Grid applications to directly create end-to-end communication pipes for large data transfers. Also, the DWDM-RAM project [11] dynamically creates light paths through a Grid service interface.

Our initial results show a typical GridJIT configuration time on the order of 160-300 ms from when a GridJIT client program requests a new light path until the path is opened. The best-case time is approximately 110 ms, while the worst-case time is approximately 600 ms¹³. With optimization, we expect these numbers to decrease significantly. The majority of this setup time is spent within OGSi-land, while only a few tens of ms are spent within the setup of the JIT control plane. Note that we haven't yet tested GridJIT with data moving across the data plane, so it is possible that configuration times will go up slightly in that case. Timing also depends on factors such as the distances between the network nodes and the types of network elements in use - all of which we still need to explore in greater detail.

As shown by Figueira *et al.* in [11], even with optical path configuration times on the order of 48 seconds, the optical network is still a better option than the traditional

¹¹User Controlled Lightpaths Project. <http://bbr.uwaterloo.ca/canarie/>

¹²<http://www.canarie.ca>

¹³This does not include the initial one-time overhead associated with instantiation of the GridJIT service within its globus container, because once an instance is created it remains memory resident for the life of the container.

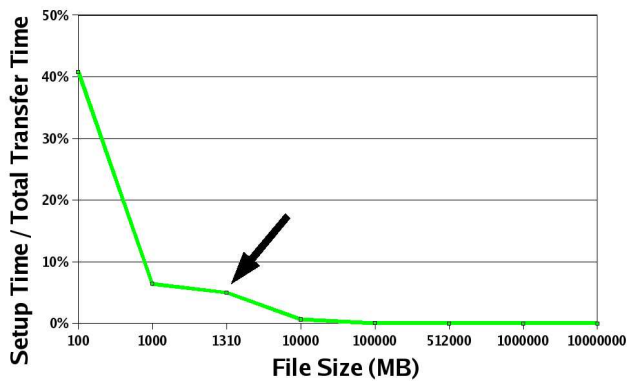


Figure 5. GridJIT setup overhead is insignificant at 1310 MB. (Note: the scale on the horizontal axis is non-linear.)

packet switched network for file transfers of datasets on the order of 1.9 to 4.5 GB or greater. [11] also calculated the time spent in path setup as a percentage of the total transfer time for different files sizes. A threshold of 5% or less for setup time compared to total time was chosen as the point below which setup overhead becomes insignificant. Figure 5 shows this threshold as calculated for GridJIT. 920 Mbps throughput was assumed, along with a 600 ms JIT worst-case setup time. From the figure we can see that for 1310 MB file size or greater, GridJIT setup overhead is deemed insignificant (5% of total transfer time).

Both the DWDM-RAM and UCLP Grid service implementations have turnaround times on the order of several tens of seconds for opening a connection, as opposed to several hundred ms for the GridJIT service. Nonetheless, we believe that interoperability between GridJIT and these other services would have the potential to extend and speed up connection possibilities for those end-users and applications that can benefit from self-initiated light paths.

Our future GridJIT related plans include more detailed study of the processing times that can be expected in various scenarios¹⁴ and across various parts used by the GridJIT stack¹⁵.

¹⁴e.g. All-photonic MEMS based switching, Cisco 15454s, Glimmerglass Sys 300s, All-photonic SOA-based switching, etc.

¹⁵e.g. GridJIT client, OGSi-land within the GridJIT Service, SWIG, notifications, C/C++ JIT API Wrapper, Software vs. Hardware JITPAC processing, Switch reconfiguration, local/metro/wide area JIT networks, etc.

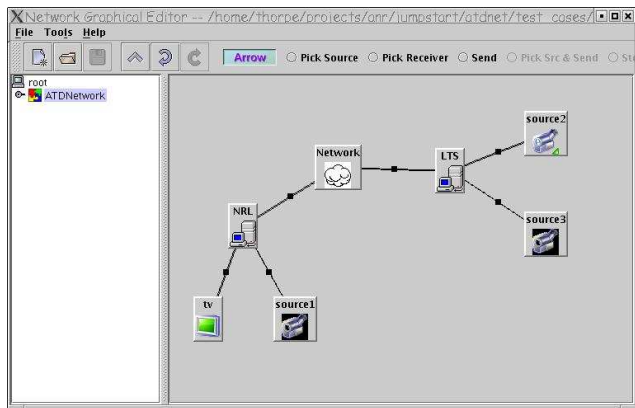


Figure 6. Starfish GUI application used to drive HDTV demo on the ATDnet.

4. HDTV Project Study: HDTV Transmission Gated By JIT

For ATDnet's ongoing demonstrations of JIT, three different 1.5 Gbit/second HDTV signals are optically transmitted through the network and gated by JITPACs, whose switches direct only one of the three signals to NRL's HDTV display at any given time.

The user-space JIT signaling proxy application, which we call the *clientEmulator*, is run on Linux machines with one instantiation for each HDTV source. The clientEmulator is a standalone C application that uses the JIT API library. It can be controlled through commands sent to its standard input stream, or alternatively, over a socket stream. Each clientEmulator is used to generate JIT signaling messages on behalf of its client application - in this case, the HDTV signal. The JIT signaling messages cause the JIT-PACs within the network to reconfigure their mirrors appropriately to allow the HDTV signal to reach its final destination: the HDTV display at the NRL.

To provide a human-friendly GUI to drive the HDTV demo, we employ a Java application called *Starfish*. This application was built using in-house code on top of the J-Sim¹⁶ and BRITE¹⁷ packages. Starfish provides a graphical environment for building a representation of a JIT network's topology. Once the topology is built, a point-and-click interface can be used to set up and tear down JIT connections within the network. The instructions to create or remove JIT connections are then sent over socket streams to clientEmulators, which are connected into the network's JIT control

¹⁶J-Sim (formerly known as javasim): A Component-based, Compositional Simulation Environment. <http://www.j-sim.org>

¹⁷BRITE: An Approach Towards Universal Topology Generation. <http://www.cs.bu.edu/brite>

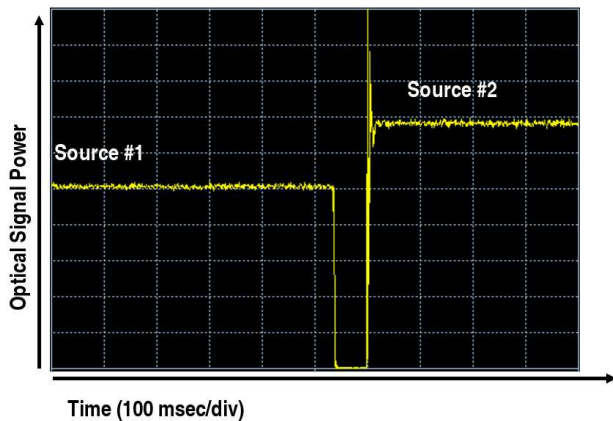


Figure 7. Switching time between HDTV sources during ATDnet demonstration. This includes an artificially introduced 50 ms delay external to JIT. (Measurements and image courtesy of Linden Mercer at the U.S. Naval Research Laboratory.)

plane through the JIT API library.

Figure 6 shows a screen capture of the Starfish GUI configured to drive the HDTV demo on the ATDnet. The icons labeled “source1”, “source2”, and “source3” represent the three HDTV sources and their associated JIT proxy clientEmulator applications. The “NRL” and “LTS” icons represent the JITPACs associated with the OXC devices at each facility. Finally, the “TV” icon is the destination node for receipt of HDTV signals. It is a simple process within Starfish to click from one HDTV source to another, with each click generating commands to the clientEmulators to first tear down the prior JIT connection, and then set up a different JIT connection for the new HDTV source.

We discovered, however, that when rapidly sending a sequence of tear-down and setup commands, Starfish couldn't guarantee that the tear-down request would arrive at its clientEmulator before the setup request arrived at its clientEmulator, which typically runs on an entirely different machine. Which command arrives first is subject to the current conditions on the IP network used by Starfish to send the commands. If a new HDTV signal's JIT SETUP message were to arrive before the old HDTV signal's JIT RELEASE message arrived, the SETUP (and its corresponding HDTV signal) would get blocked within the JIT network. This block occurs because the HDTV display and all of the HDTV sources are using exactly the same wavelength in the data plane, but only one connection at a time can use that wavelength all the way to the destination HDTV display. To eliminate the possibility of this IP network induced

blocking, we introduced an artificial delay after sending the tear-down request from Starfish and before sending the subsequent setup request. After experimentation with various durations for the delay, we settled on 50 ms, which completely avoided blocking within the JIT network. We suspect this number could be reduced, but it sufficed for the purposes of the demo.

Figure 7 graphs two different HDTV signals received during the HDTV demo, each from a different source. The trough in the middle of the graph, approximately 75 ms long, is the time during which no signal was available for display. Allowing approximately 50 ms for the artificially induced delay leaves approximately 25 ms for the reconfiguration time through the two OXC devices.

5. GridFTP Project Study: GridFTP over ST-over-JIT Integration

As we mentioned earlier, part of the overall strategy for JIT transport protocol research has been to identify suitable user applications that could be modified to use the JIT signaling protocol transparently. The GridFTP protocol's *globus-url-copy* client application and *in.ftpd* server daemon, which enable data transport for the Grid computing community, were identified as suitable initial target applications since they are used for large, high-speed data flows.

Many of the client machines on the ATDnet are SGI workstations, while others on the ATDnet and most machines at MCNC-RDI are Linux workstations. For this project, therefore, code portability was an objective, targeting both the Linux and Irix kernels in a cross platform fashion with the same code base. We decided to pursue the integration of the JIT API underneath the ST protocol, which would serve as an adaptation layer between user-space applications and the JIT API. Reasons for this decision included 1) ports of SGI's high-bandwidth, low-latency ST protocol are available in both targeted kernels; 2) ST's performance characteristics are expected to be more desirable than TCP (you can consider ST to enable "remote DMA" access over the network); and, 3) porting TCP to JIT is anticipated to be more difficult than porting ST to JIT. Upon completion of the ST/JIT integration step, any user-space applications that could use the ST libraries for data transfers could easily be modified to use the JIT-enabled ST, making the JIT signaling protocol available transparently. Ports of the GridFTP protocol were subsequently modified to use this JIT-enabled version of ST for data transfer, as described below.

ST [2] was developed by SGI as an underlying technology to support its GSN™ networking solution, which aimed to meet the need for high-bandwidth and low-latency performance with minimal CPU use. After developing its

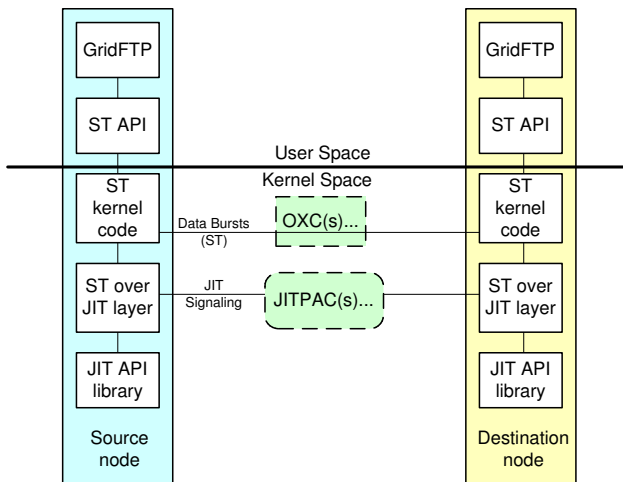


Figure 8. Components of the initial GridFTP over JIT prototype implementation.

proprietary Irix implementation, SGI also released an open-source Linux ST implementation.

The GridFTP protocol [5, 6] is more than a simple extension to the widely-implemented IETF standard FTP protocol. It introduces concepts that go well beyond the standard FTP procedures. It provides a fast, secure, efficient, and reliable data transport protocol for the Grid community. Currently, GridFTP supports Grid Security Infrastructure (GSI)¹⁸ and Kerberos authentication, third-party control of data transfer with GSSAPI security, parallel or striped data transfer, partial file transfer, and others. Client and server-side implementations of the protocol are included with the Globus Toolkit releases.

For the adaptation protocol we targeted ST over IP running on GigE (Linux) or ST directly on GigE (Irix). JIT signaling is done over a separate link (UDP or ATM).

Figure 8 illustrates the various components involved with this effort. Between the two client nodes is one or more JITPACs, which provide signaling messages back to the source node to indicate when data can be transmitted. On the testbed used for this GridFTP/JIT integration effort at MCNC-RDI, we didn't yet have an available OXC device. Instead, data was sent over a direct (optical) GigE connection between the two client nodes, through a "Virtual OXC." That is, data transmission was gated by the JITPAC, which was configured to allocate wavelengths as if an OXC was in fact attached.

At the top of the stack in Figure 8 is user-space GridFTP protocol software. Version 2 of the open source Globus

Toolkit (GT2)¹⁹ was used as a starting point. The GridFTP implementation's data sending code was modified to call the user-space ST API library, which in turn invokes the JIT-enabled ST implementation that is within kernel-space, rather than the standard TCP.

The modified ST kernel code, in turn, relies on the lower level ST-over-JIT layer to gate its data transmission. The ST code was modified such that before it begins a data transfer, it confirms that a JIT light path is open. As long as there is an open light path, it is able to send the data out.

The primary purpose of the ST-over-JIT layer is to simplify this use of the JIT API library by the higher level ST code. The details of using the JIT API library are encapsulated within the ST-over-JIT layer, thereby exposing the ST layer kernel code to an API with only two functions: *jit_open_light_path()* and *jit_close_light_path()*. Another purpose of the ST-over-JIT layer is to send and receive JIT signaling messages on behalf of the JIT API below it.

Initial code development of the ST-over-JIT-enabled GridFTP is 99% complete at the time of this writing (Spring 2004). The remaining work involves final testing and debugging. To a large extent, progress in this area has been limited not for technical reasons but rather for scheduling and administrative reasons across participating organizations.

Preliminary results have revealed these limitations: On the Linux platform, the ST port's implementation proved to be somewhat unreliable. In addition, it is not optimized to write directly to GigE, as the Irix port is. The Linux ST writes instead to IP over GigE, which slows it down. More generally, the ST protocol itself is not widely implemented, and TCP/IP is still the dominant protocol for the Internet.

We plan to focus on the Irix OS to complete this proof-of-concept implementation. However, we have already shown that the JIT protocol can be successfully implemented within the kernel-space software stack.

6. Conclusions and Future Work

This paper has described the JIT networking concepts that can help meet Grid networking requirements for large, fast, and latency sensitive data flows. We have also reviewed OGSi, HDTV, and GridFTP projects that use JIT to gate data transfer from within both standalone and Grid-based applications, and from within both user-space and kernel-space.

The project team plans future efforts in several areas that will further the development of JIT and truly allow higher performance and lower cost to benefit networked Grid applications. We intend to pursue integration of the GridJIT service into research and production networks, including

¹⁸Overview of the Grid Security Infrastructure. <http://www.globus.org/security/overview.html>

¹⁹Available via anonymous ftp to <ftp://ftp.globus.org/pub/gt2/2.0>

interoperating with other Grid services where possible for the coordination with other resources such as CPU, storage, and network control planes accessed through mechanisms other than JIT. We also plan to do more detailed timing tests of various GridJIT scenarios across all of the components within the network. JITPAC drivers currently exist or are under development for switches from Lambda Optical Systems, Glimmerglass, Calient, and Cisco, and other drivers will likely be developed in the future. A JIT LAN system is being designed that will use a passive star coupler. This system can dramatically reduce the optical configuration setup time within a LAN environment, allowing light path communications to more than one peer simultaneously. After using GigE initially, this system is eventually expected to employ 10GigE technology, saving significant costs when compared to buying a 10GigE switch. JIT network cards are being developed that will implement the client-side state machines within hardware, along with tunable lasers. Next generation JITPACs with switching times on the order of nanoseconds are being developed to decrease light path configuration time within a JIT WAN.

7. Acknowledgments

The authors gratefully acknowledge the valuable assistance of Carla Merrill, Bryan Bush, Linden Mercer, Art Goldman, Hank Dardy, Wai Ng, Eric Kinzie, Arnold Bragg, Ilia Baldine, Hongjie Xin, Joel Hernandez, Raghavendra Uppalli, Xiaoyong Wu, Mark Cassada, Mike Pratt, Mru-gendra Singhai, A. Halim Zaim, Lavanya Ramakrishnan, Bonnie Hurst, Sue Gambill, Keith Jackson, Mark Johnson, Jason Leigh, Tom DeFanti, and Borja Sotomayor.

References

- [1] Jumpstart JIT Signaling Definition, May 2003. http://jumpstart.anr.mncn.org/Docs/03_05_12/SignalingDefinition.pdf.
- [2] Scheduled Transfer Protocol. A whitepaper, 1998. http://www.sgi.com/peripherals/networking/st_whitepaper.pdf.
- [3] Workshop on Optical Control Planes for the Grid Community, Apr. 23 2004. <http://www.mncn.org/mncnopticalworkshop>.
- [4] CANARIE Optical Border Gateway Protocol Website. <http://obgp.canet4.net>.
- [5] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, and S. Tuecke. GridFTP Protocol Specification. GGF GridFTP Working Group Document, Jan. 2002. <http://www.globus.org/research/papers/GridftpSpec02.doc>.
- [6] W. Allcock, J. Bresnahan, I. Foster, L. Liming, J. Link, and P. Plaszczac. GridFTP Update January 2002. Technical Report, Jan. 2002. <http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf>.
- [7] I. Baldine, G. Rouskas, H. Perros, and D. Stevenson. Jumpstart - a Just-In-Time Signaling Architecture for WDM Burst-Switched Networks. *IEEE Communications*, page 82, Feb. 2002.
- [8] T. DeFanti, C. de Laat, J. Mambretti, K. Neggars, and B. S. Arnaud. TransLight, A Global-Scale LambdaGrid for E-Science. *Communications of the ACM*, pages 34–41, Nov. 2003.
- [9] E. M. et al. Generalized Multi-Protocol Label Switching Architecture. IETF Draft, May. 2003. <http://www.ietf.org/proceedings/03nov/I-D/draft-ietf-ccamp-gmpls-architecture-07.txt>.
- [10] T. Ferrari, G. Karmous-Edwards, M. Leese, P. Mealor, I. Monga, and V. Sander. Grid Network Services Use Cases (DRAFT informational pdf document), May 2004. <http://forge.gridforum.org/projects/ghpnr-g/document/draft-ggf-ghpn-netservices-usecase-0/en/4>.
- [11] S. Figueira, S. Naiksatam, H. Cohen, D. Cutrell, P. Daspit, D. Gutierrez, D. B. Hoang, T. Lavian, J. Mambretti, S. Merrill, and F. Travostino. DWDM-RAM: Enabling Grid Services with Dynamic Optical Networks. In *Workshop on Grids and Networks held in conjunction with CCGrid 2004, the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, April 2004.
- [12] M. Hasan and W. Clark. Network Service Interfaces to Grid (DRAFT informational pdf document), May 2004. <http://forge.gridforum.org/projects/ghpnr-g/document/draft-ggf-masum-grid-network-services-0/en/1>.
- [13] P. Mehrotra, I. Baldine, D. Stevenson, and P. Franzon. Network Processor Design for Optical Burst Switched Networks. In *Proceedings of the International ASIC/SOC Conference*, Sept. 2001. IEEE, 5 pp.
- [14] H. Newman, M. Ellisman, and J. Orcutt. Data-Intensive e-Science Frontier Research. *Communications of the ACM*, pages 68–77, Nov. 2003.
- [15] V. Sander, W. Allcock, P. CongDuc, J. Crowcroft, M. Gaynor, D. Hoang, I. Monga, P. Padala, P. Vicat-Blanc, M. Tana, and F. Travostino. Networking Issues of Grid Infrastructures (DRAFT informational pdf document), Apr. 2004. <http://forge.gridforum.org/projects/ghpnr-g/document/draft-ggf-ghpn-netissues-3/en/1>.
- [16] D. Simeonidou, R. Nejabati, B. S. Arnaud, M. Beck, P. Clarke, D. Hoang, D. Hutchison, G. Karmous-Edwards, T. Lavian, J. Leigh, J. Mambretti, V. Sander, J. Strand, and F. Travostino. Optical Network Infrastructure for Grid (DRAFT informational pdf document), Mar. 2004. <http://forge.gridforum.org/projects/ghpnr-g/document/draft-ggf-ghpn-opticalnets-1.pdf/en/1>.
- [17] J. Y. Wei and R. I. McFarland. Just-In-Time Signaling for WDM Optical Burst Switching Networks. *Journal of Lightwave Technology*, pages 2019–2037, Dec. 2000.
- [18] A. Zaim, I. Baldine, M. Cassada, G. Rouskas, H. Perros, and D. Stevenson. Formal Description of the Jumpstart Just-in-Time Signaling Protocol Using EFSM. In *OptiComm 2002*, pages 160–173, May 2002. Online PDF is available at <http://jumpstart.anr.mncn.org/Opticomm2002.pdf>.