

Optimizing Tunneled Grid Connectivity across Firewalls

J. Tan, D. Abramson and C. Enticott

Caulfield School of Information Technology,
Monash University

PO Box 197, Caulfield East, VIC 3145, Australia

{jefferson.tan, david.abramson, colin.enticott}@infotech.monash.edu.au

Abstract

Grids today generally assume that concurrent network connections are possible among many processors attached to high-capacity networks. However, inter-network boundaries dividing independent institutions often have firewalls, typically to restrict how many and which ports are accessible. In some cases, ports are opened indefinitely for Grid applications, but this compromises security significantly. On the other hand, solutions that manage port openings in an ad-hoc manner for applications are non-trivial to implement. An alternative firewall traversal technique is required that will provide manageable openings with less complexity involved. This is possible through proxies and managed tunnels using ports already authorized across the firewalls. We have developed a transparent connectivity mechanism for this, called **Remus**, which reroutes Grid connections through a tunnel on ports allowed across firewalls. However, a single tunnel presents a performance bottleneck. In this paper, we present the method by which Remus distributes several connections over *multiple* tunnels, improving throughput as a result. Rerouting wrappers hide the tunneling from applications, intercepting outgoing connections and rerouting them transparently. Well-known and mature tools and protocols, such as SSH and/or SOCKS, are utilized, instead of imposing customized, non-standard mechanisms. Results of our experiments are also presented for large file transfers over a Globus-based Grid that uses Remus.

Keywords: grids, firewalls, tunnels, firewall virtualization

1 Introduction

The *Grid* has been defined as “a distributed computing infrastructure for advanced science and engineering” (Foster, Kesselman and Tuecke 2001). *Grid applications* exploit the distributed processing capacity of Grids across networks of independent institutions collaborating within the context of a *virtual organization* (VO). The Internet is widely used as the interconnection infrastructure on top of which the internetworking of VO members is built. However, the Internet is full of threats, so boundaries between the networks of VOs are often protected with firewalls. This becomes a problem for VOs since firewalls restrict internetworking capabilities and

therefore restrict the level of collaboration possible. We end up with a Grid connectivity problem where “benign applications cannot traverse a firewall” (Rosmanith and Kranzlmuller 2004). Many Grid applications work under the assumption that there are no such restrictions. While it is possible to reprogram these applications in order to adapt to these restrictions, this can be expensive. It also presents an unwanted situation of applications being sidetracked by firewall considerations when they should instead concentrate simply on what the applications are meant to do. The point must also be made that it won’t be easy for Grid applications to adapt to all the security policies of all the networks on the Grid. The code that must be added to these applications to address firewalls will add to application overhead and complexity. On the other hand, firewalls are necessary extensions of organizational security policies that must be dealt with. We have studied the likely scenarios resulting from security interoperability and explored or developed methods that facilitate connectivity despite such restrictions.

A relatively new solution that provides many desirable characteristics is the use of *firewall virtualization* (Monga, Niederberger and Metsch 2008, IBM 2003). The overall security of a given site is organized as a set of virtualized firewall domains. This fits into the structure of many organizations. Instead of relying on centralized security management, a given constituent group or enterprise becomes responsible for its own security management. Grid administrators become responsible for managing the firewall configuration for portions of the network that are participating in the Grid. Authenticated and authorized Grid users are to be provided with connectivity, while everyone else is excluded. On the other hand, the implications must not be underestimated. Firewalls will require heavy customization in order to effect virtualization. Whereas they cannot be customized internally, they must be controllable remotely from another host where virtualization software is running. There are now a few virtualization-capable firewall solutions in the market, but even those would require significant development work in order to deliver firewall virtualization that operates *with* the Grid. User authentication and authorization within the Grid security infrastructure must be implemented, as well as ad-hoc, temporary and selective firewall openings, i.e., *pinholes* (D-Grid Integrationsprojekt 2006, Green, Gallo and Miller 2004). Firewalls presently in use were not designed with virtualization capabilities, and will be even harder to virtualize in the manner described, particularly with the lack of remote management extensions for software-driven reconfiguration on demand.

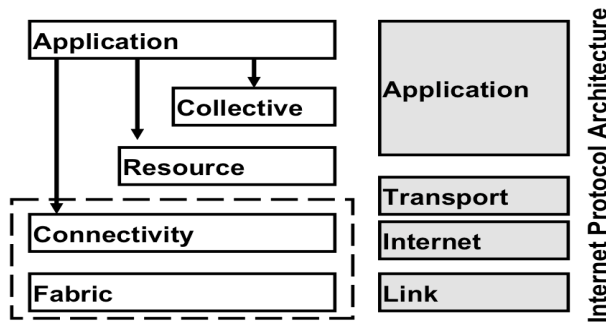


Figure 1. This Grid Architecture from “The Anatomy of the Grid” (Foster, Kesselman and Tuecke 2001) provides a side-by-side perspective and associates Grid and fabric connectivity with TCP/IP.

A more common solution in general use today, from before the inception of Grids, is to use *tunneling*, in which a single connection is used to carry a different set of one or more tunneled connections. Firewalls are typically used to implement a minimal access policy, such that inbound, outbound, or both kinds of traffic, are restricted to a small number of acceptable protocols only. These protocols often rely on predetermined TCP or UDP ports, which may be the only ports that firewalls will allow access to or from. For example, secure shell (SSH) logins are considered safe, requiring first an authorized user account, and being secured by strong encryption mechanisms (Ylonen 1996), making SSH connections one of those few that firewall administrators are more likely to allow. As it happens, SSH is also capable of tunneling multiple connections across, transporting each connection as a virtual TCP channel sharing the same SSH session. Among other things, an inevitable side effect of several channels within one tunnel is a slower connection for each of those channels. None of those tunneled connections will achieve the maximum data transfer rates available to direct connections. On the other hand, given that tunneling is the only option in certain situations, it would be beneficial to find some means of optimizing tunneled connectivity.

This paper describes how we achieve this by distributing the load of tunneled connections across several tunnels. We do so with two design goals. First, the mechanics of tunneling and balancing network load across multiple tunnels must be hidden from Grid applications. Second, it should be possible to construct the multi-tunnel infrastructure using commodity off-the-shelf (COTS) tools, avoiding the need to build everything up from scratch for a limited set of environments only. To illustrate the impact on network performance from using multiple tunnels, we will present data transfer measurements for Globus file transfers with and without multiple tunnels.

2 Using Multiple Tunnels

2.1 REMUS

Here we will provide a brief background of our previous work on tunneling for Grid connectivity, which we improve upon in this paper in order to enhance tunneling performance.

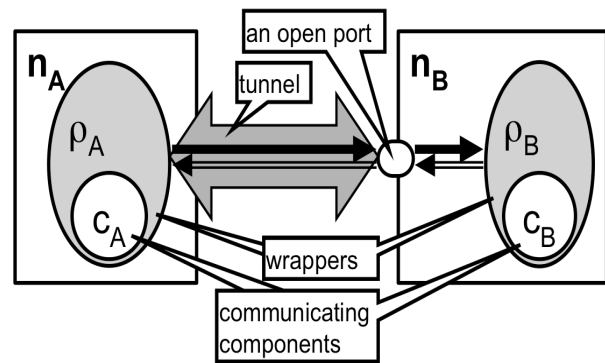


Figure 2. The abstract model of Remus illustrates the two central ideas of diverting connections unobtrusively via wrappers and rerouting.

Figure 1 shows the architecture of the Grid as described in a Grid anatomy by Foster, Kesselman, and Tuecke (2001). We use this particularly anatomy because this is the basis for the Globus Toolkit (Globus Alliance website, 2008), a dominant technology in many of today’s computational Grids. In shaded layers within the Grid architecture, the Internet protocol architecture is shown, which consists of the Transport, Internet and Link layers. We point out that there are therefore two modes of connectivity if the entire network infrastructure is considered from Grid to bottom. We use the term *fabric connectivity* to refer to the basic networking capability of typical networked host, e.g., file servers, desktops, etc., which exists prior to the host becoming part of a Grid. Such hosts form the *fabric layer* seen at the bottom of the Grid architecture of **Figure 1**. TCP/IP is typically assumed, being the most common. Fabric connectivity is usually not managed nor even a conspicuous concern on the Grid, being simply assumed. Grids are commonly built on top of existing computing platforms that communicate using TCP/IP, typically with pre-existing clusters and wide area networks that rely on the Internet.

Given the stacked architecture in which TCP/IP is the basis for fabric connectivity, Grid connectivity is easily affected by fabric layer connectivity issues. One issue that concerns us is how firewalls can easily break Grid applications. Many have investigated this problem, where Grid services are inaccessible due to firewall restrictions (Denis et al 2004, D-Grid Integrationsprojekt 2006, Graupner and Reimann 2002, Green, Gallo and Miller 2004, Rosmanith and Kranzlmuller 2004, Son, Farrellee and Livny 2005, Son and Livny 2003, Tan, Abramson and Enticott 2005, Tanaka et al 2000). We have investigated this problem in the past, and proposed an architectural solution, a *virtual connectivity layer*, which promises several means of connectivity while minimizing the impact on Grid applications programming and firewall administration. We consider it crucial to reroute Grid applications transparently on top of *authorized* cross-firewall channels. Transparency is necessary in order to avoid burdening applications with firewall considerations. At the same time, any means of firewall traversal used must be authorized, compliant with the security policies of the institutions involved. Finally, since many firewall traversal methods work well for different situations, the ideal cross-firewall facility must

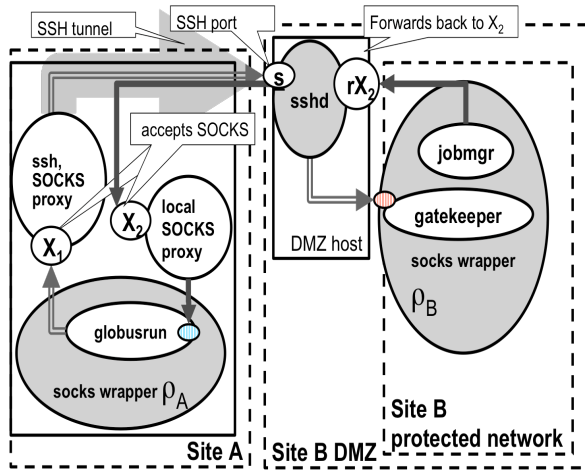


Figure 3. Globus connections are enabled across firewall via SOCKS wrappers. The SOCKS connections are in turn tunneled through a single SSH session with dynamic port forwarding.

be able to adapt as necessary, e.g., the protocol, proxies and ports to use.

Figure 2 illustrates **Remus**, our **Rerouting and multiplexing** system which uses *wrappers* in order to transparently trap and reroute connections across firewalls using whatever means is allowed by the environment and the prevailing security policies. This means that the application message that cannot get across directly to its destination due to firewall restrictions is diverted across the tunnel instead. There are situations where tunneling is considered a practical option, particularly where the number of permanent openings across firewalls should be minimal. The broad bi-directional arrow illustrates a tunnel that provides a virtual link that the firewall(s) will permit. This facilitates multiplexing several socket connections within the tunnel. Wrappers are also required to provide unobtrusive rerouting.

Figure 3 illustrates the Remus tunneling approach when the situation allows us to employ SSH for the tunneling and SOCKS (Inferno Nettverk A/S 2008, Leech et al 1996) for transparent rerouting of all connections between Globus client and server processes. We have used Remus to assist the Nimrod/G meta-scheduler (Buyya, Abramson and Giddy 2000) in traversing firewalls, allowing it to access Globus computational resources and allowing those resources to connect back to Nimrod/G (Tan, Abramson and Enticott 2005). Nimrod agents are launched on Globus compute nodes via Globus job submissions, and Nimrod agents must make direct connections back to the Nimrod servers in order to accept experiment tasks and input data. Firewall issues made this impossible for Nimrod agents to do, which prompted our work on rerouting connections via tunnels. However, the purpose of this paper was to take a close look at the throughput degradation caused by tunneling. We achieve this using GridFTP, the file transfer protocol from Globus. We are able to observe the effects of different tunneling configurations on throughput involving large file transfers.

The mechanics of this setup are shown in **Figure 4**, where one tunnel is used for each direction of the

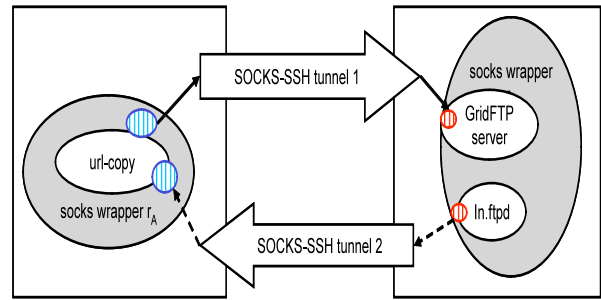


Figure 4. Tunneling connections with one tunnel going in each direction is simple, but the tunnels become bottlenecks.

exchange between Globus clients and servers. This requires login accounts on each host. It is also possible to have the accounts on hosts other than the actual Globus client and server hosts, but these intermediate hosts must have direct accessibility to the Globus hosts, which is likely if they share the same network or administrative domain. To simplify the discussion, let us assume that both the Globus client and server hosts are accessible only for SSH logins, either by an active SSH agent, to avoid repetitive password entry, or through a GSI (Grid Security Infrastructure) proxy certificate if GSSAPI (Generic Security Services Application Program Interface) is supported, which is commonly true for Globus grids. If so, the tunnel persists for the duration of the certificate, possibly for many days. Either way, when Remus successfully creates the SSH session for any direction, a SOCKS proxy tunnel is also created. SSH refers to this as *dynamic port forwarding*. A local SOCKS proxy port is now ready for SOCKSv4 connections. This completes the first layer of connectivity that provides SOCKS accessibility. A second layer is now necessary in order to divert all connections through the tunnel instead of directly between the Globus client and server (or vice versa). This is important since we assume a firewall situation where direct connections are blocked. This rerouting layer works by preloading a SOCKS library file prior to executing Grid transactions over the network, e.g., job management or data transfer operations. This is achieved in Linux by setting the environment variable `LD_PRELOAD` to the full path of the SOCKS library file. We used **Dante** (Inferno Nettverk A/S 2008), a free SOCKS implementation that is able to “socksify” processes by substituting SOCKS-enabled transport instead of direct ones. Note that Dante must be configured to use the SOCKS port created by the SSH sessions that enable SOCKS-based tunneling.

Figure 5 presents the results of an experiment in transmitting large files via GridFTP, the standard Globus protocol for file transfers. Multiple transfers were conducted for each of several files with increasing sizes from 50MB to 250MB. The transfers were conducted across two networks with firewall restrictions in between. In transferring a 250 MB file, direct transfer rates (top graph) exceeds 4.2 MBps. Remus tunnel throughput goes up to about 3.8 MBps (bottom graph). Note the top graph dip at direct 200MB file transfers, which consistently occurs in our tests. This may be due to TCP congestion control, as “window full” notifications were appearing among the packets we captured.

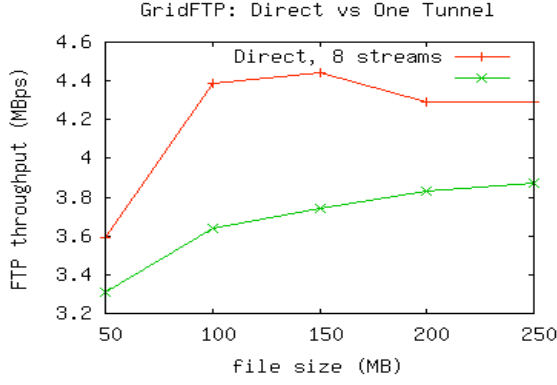


Figure 5. Throughput graphs for Direct GridFTP (higher throughput) and via Remus tunneling (lower throughput) are shown, both using 8 parallel GridFTP streams. However, Remus forced all 8 streams into one tunnel.

In all tests, we used GridFTP’s parallel stream capabilities, 8 streams in this experiment. Remus used one SOCKS+SSH tunnel. Note that the transfer loads of the virtual channels add up. Each stream to be tunneled translates into a new sub-channel, which must be tracked using control data to identify the sub-channel. The packet streams must also be coordinated, as with round robin. Finally, with more processes running, the arrival rate of packets to be tunneled goes up. These all contribute to how tunneling through one tunnel results in significantly lower throughput.

2.2 Multiple Tunnels

Tunnelling slows things down, but we must use tunneling in certain situations. We have come across two separate cases when trying to make resources behind a firewall accessible to a Nimrod/G meta-scheduler (Buyya, Abramson and Giddy 2000). Grids involving autonomous institutions connected across the Internet cannot mean that firewalls and security policies are to be compromised. They should still apply when an organization participates in Grids with other organizations. One possible solution is to lift the restrictions over a few ports, perhaps the minimal requirements of the Grid middleware, but this can limit the degree of distributed transactions.

We mitigate the situation through optimization techniques. In our approach, there is no upper limit to the number of actual socket connections crossing the firewall. First, if one TCP stream is too constricting for N connections, why not use N multiple tunnels? If the single tunnel is indeed a bottleneck, then using several tunnels should cause an improvement because each tunnel exists as an independent socket connection.

3 Prototype, Experiments and Results

Figure 6 shows the use of multiple tunnels, each running on its own TCP stream, carrying a portion of the total connections that require tunneling across. In this setup, connections from the client to the server use one tunnel, but the greater share of connections, which go from the server to the client, use several tunnels. The testbed involved a cluster as a Globus Toolkit 2.4 resource and a

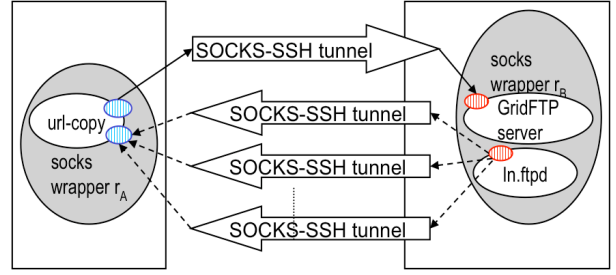


Figure 6. Distributing connections across several tunnels (multi-tunneling) reduces throughput degradation, compared to using single tunnels.

client three gateway hops away, over a distance of 10 km. The round trip time average between them is 0.706 ms. Both the cluster head node and the client host are Pentium 4 Linux servers running at 3 GHz and 2.8 GHz, with 384 MB and 1 GB of RAM, respectively. The tunnels are constructed as illustrated, with SSH using the blowfish cipher with no compression. Dante 1.1.19 is used to provide the SOCKS wrapper around Globus client and server processes.

The results presented in **Figure 7** validate our expectations. In the experiments conducted, files at 10, 100, 200, 250 and 500 MB in size were transferred via GridFTP. Five transfers were simultaneously made for each graduation in size. As with earlier experiments, we ran 8 parallel GridFTP streams for each file transfer. Without tunneling, this would involve 8 parallel socket connections. With tunneling, these socket connections are carried across as virtual channel connections in parallel. Each of the data transfers were GET operations, meaning that files were being downloaded from the server down to the client. This involves parallel socket connections initiated from the GridFTP server towards the client site, for which reason we provided multiple streams from the server to the client. A smaller volume of traffic over a few sockets proceeds from the client to the server, for which we only provided one tunnel in that direction. We also decided not to tunnel the request connections from the client to the GridFTP service on one port, since this involves few connections and a low volume of traffic.

The results show the effect of setting different tunnel-to-socket ratios. Using multiple tunnels provides better performance compared to the use of a single tunnel. We performed experiments using two multi-tunnel configurations, the top two graphs in the figure. At mid-level improvement, each tunnel carried up to five socket connections (1-to-5). In transferring 500MB files, this configuration increased throughput from about 1.3 MBps using just one tunnel to 1.68 MBps. Even better performance was achieved at 1.78 MBps when we dedicated one tunnel for each socket connection (1-to-1). The socket-to-tunnel ratio affects the maximum number of connections being crammed inside one tunnel. This will also affect how many tunnels will be required, based on how many distinct socket ports will be involved, as dictated by the Globus TCP port range parameter. This parameter allows Globus to limit the ports to be used in network connections. The Remus configuration for multi-tunneling does not require advanced knowledge about the nature of the connections that an application will require,

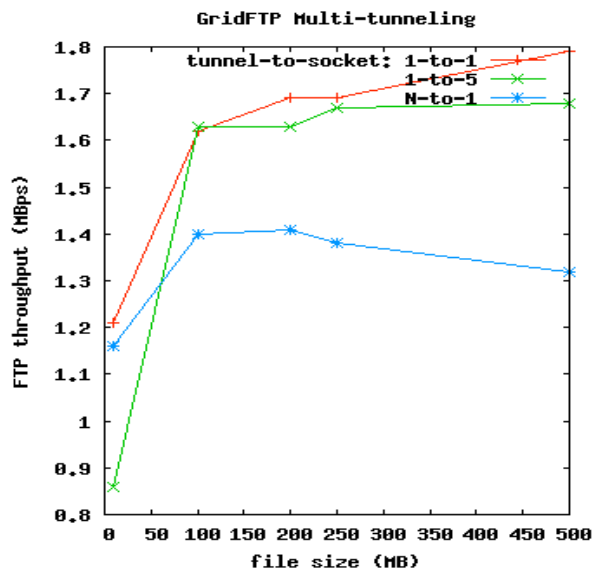


Figure 7. These results compare the transfer rates achieved through multiple tunnels and those through a single tunnel. Performance is best when one tunnel is dedicated for each TCP socket connection. It is worst when only one tunnel is used for all socket connections.

neither in the actual number nor the particular destination ports. The wrapper must simply be configured so that a range of destination ports are mapped to each available SOCKS proxy.

Note that a bottleneck effect can be expected from the multi-tunneling mechanism. Each of the tunnels is an independent SSH tunnel, necessitating an independent SSH session that involves one socket connection that originates from some port on the SSH client host and terminates at the same specific port (tcp/22) on the SSH server host. The SSH server is therefore working hard as it manages each of the N individual SSH connections. However, the actual connections are only indirectly managed by the SSH server. A child SSH process that is spawned for that purpose maintains each connection. We therefore believe that the bottleneck effect is tolerable up to a certain extent, despite what appears to be a $1:N$ ratio between the SSH server port 22 and the number of connections from SSH clients. An in-depth analysis is beyond the scope of this paper, but we can offer conjecture.

Both the server and client implement TCP/IP. Each layer in the protocol architecture is a separate subsystem. The connection of upper subsystems to its counterpart on the other host, e.g., application to application, is only an abstraction. The real connection is between physical devices. The first bottleneck is thus the slowest network interface, link or intermediate system, such as a modem, switch or router. The second bottleneck is not actually the port at all, this being a mere abstraction, but rather the application that upholds the tunnel. SSH tunnels terminate on the server side at the same port, which is port 22. The client and server pair that establishes a connection, e.g., a login with or without a tunnel, shares a distinct *session*. We distinguish between the SSH daemon that accommodates requests and, for each session, what we refer to as a distinct *session server*. Therefore, while

all connections are delivered by the TCP subsystem to a single SSH server process on port 22, the *SSH daemon* will hand the connection over to the appropriate session server. In the other direction, there is always a distinct SSH client that initiated the session to begin with. Thus, the only logical bottleneck will be the SSH daemon, arbitrating on behalf of session servers. The arbitration process adds a layer of complexity and communication overhead. Each session server also deals with decryption, possible decompression, and the demultiplexing of several virtual channels in one tunnel. An in-depth analysis will be necessary in order to determine the various types of overhead in exact terms, but it is beyond the scope of this paper.

Remus has been applied successfully for both GSIFTP and GRAM services of Globus Toolkit 2.4. We have also applied it to Nimrod/G 3.01, for which the Nimrod agents running on computational nodes were socksified in order to let them exit the secure network through a SOCKS proxy on the firewall host or on the head node of the cluster. Nimrod agents generally use the network in bursts only, such as when downloading experiment plan files and executables, or when uploading the results of experiments. While possible, it is not common for these uploads and downloads to involve the volume of data we used in the experiments here, which is why the experiment focused on GridFTP using large files. The goal of our measurements in this paper was to illustrate that the use of multiple tunnels provides higher throughput than the use of singular tunnels.

4 Related Works

Firewalls are now considered to be necessary components on the Internet to protect organizational networks. These devices enforce official security policies of a given organization. There are now several firewall traversal tools today which are used either openly or covertly. Applications seeking to get across the firewall for legitimate purposes will require such measures.

The secure shell (SSH) (Ylonen 1996) is one of those tools commonly used for tunneling, and has been recommended specifically for Grid firewall traversal (Graupner and Reimann 2002, Welch 2008). It is expected that many secure networks allow SSH connections if they allow anything at all, since SSH is quite secure for both logins and file transfers. The most basic form of SSH tunneling is through *port forwarding*, where several unidirectional channels are configured at session startup, so that the beginning of the channel is a local socket bound to the SSH client, and the terminus is a remote socket bound to the SSH server. As mentioned previously, this remote socket bound by the SSH session server is accessible only through the SSH daemon which manages all connections going through port 22. An SSH session may bind several pairs of mapped (local and remote) ports, 1-to-1. The local port will forward connections across to the counterpart remote port on the remote destination host. Another form of tunneling in SSH which we use is *dynamic port forwarding*, where the SSH client binds only one port which forwards connections, using the SOCKS protocol (Lee 2005, Leech et al 1996), to any number of remote ports on the SSH server site. In both static and dynamic port forwarding,

connections tunneled through SSH sessions may be made to any port on any site that is accessible to the SSH server site, even one that is not the SSH server site itself. Dynamic port forwarding provides more flexibility than static port forwarding, but SOCKS protocol support is typically limited to SOCKS v4. SOCKS v5 introduces additional features. The SOCKS protocol is also considered a firewall traversal mechanism on its own, assuming that there is network access to a *SOCKS proxy*, which facilitates connections. In dynamic port forwarding, the SSH client behaves like a limited SOCKS proxy. A full-featured SOCKS proxy is capable of other things, including the ability to chain several SOCKS proxies together.

We have not found Grid-specific utilization of multiple tunnels, but there are plenty of other works that address the general firewall issues for the Grid. Some have tried customizations of Globus in order to solve the firewall problem. One such solution uses a **Nexus Proxy** that has access to Globus resources but is situated outside the firewall, making it accessible to remote hosts (Tanaka et al 2000). Globus transactions may then use this proxy to connect to the protected Globus resources. On the other hand, solutions avoiding code changes or different builds are preferred, where legacy applications and Grid subsystems need not be reprogrammed. **XRAY** (Son, Farrellee and Livny 2005) uses a mutually accessible relay, the GCB broker, in order to facilitate communications between client and server applications. As with the Nexus Proxy, certain XRAY agents will be accessible from both sides of the firewall, providing a point of entry after authentication requirements are met. Other forms of tunnels have been built but for other purposes, such as **glogin** (Rosmanith and Kranzlmuller 2004), which erects a tunnel through a job, submitted as a normal Globus job. This tunnel, however, is intended to provide interactivity to users, giving them virtual login access to Grid compute nodes. It will fail if the firewall, typically that which protects the computational resources, blocks connections between user hosts and the glogin job on the compute node.

Other solutions attempt to deal with firewall devices directly, which may be requested to temporarily remove restrictions as required by Grid transactions and services (D-Grid Integrationsprojekt 2006, Quittek, Stiernerling and Srisuresh 2008, Rosmanith and Kranzlmuller 2004). From a non-Grid perspective, this is possible through efforts to standardize the operations, semantics and manageability of *middleboxes*, which pertain to managed devices between networks (Quittek, Stiernerling and Srisuresh 2006). As earlier mentioned, this *firewall virtualization* approach provides Grid administrators with an administrative context by which to manage firewall restrictions affecting Grid resources (Monga, Niederberger and Metsch 2008). On the other hand, existing infrastructure will have to be modified heavily in order to address the authentication and authorization requirements of the Grid on both users and applications. Customized Grid services must accommodate ad-hoc and temporary relaxation of firewall restrictions for certain applications or users for an appropriate amount of time. The lifetime of such *pinholes* must also match an

appropriate time characteristic of users and/or applications, e.g., the lifetime of the proxy certificate.

While all the above solutions are successful, these non-standard, custom-built mechanisms do not present generic solutions for typical Grids. Remus is designed to use generic mechanisms such as SOCKS and SSH, and uses wrappers to avoid the need to reprogram Grid applications or middleware with firewall traversal capabilities.

5 Conclusions

We have presented our work in optimizing tunneling as a firewall traversal mechanism for Grids. Tunneling is a viable mechanism when Grids involve networks that are protected by firewalls. Since firewalls restrict the number of ports accessible, and assuming that removing firewalls or lowering their levels of restriction are not options, tunneling is the simplest alternative. Since tunneling necessary involves overheads and bottlenecks that will reduce data transfer throughput, we sought means to minimize the expected slowdown. We have therefore devised the use of multi-tunneling, which is a simple, static form of load balancing. We illustrated how we are able to transparently reroute Globus data transfer connections through several tunnels. Outgoing connections are statically load-balanced by assigning a mapping between destination ports and SOCKS proxy ports. Our graph shows that this improves transfer rates when compared to using a single tunnel. We further showed that the socket-to-tunnel ratio impacts the performance of load-balanced tunneling.

Several issues remain open to future research however. It would be useful to determine how to best set the socket-to-tunnel ratio in order to reduce the number of tunnels while maintaining a healthy network throughput. It would also be advantageous to look at more sophisticated means of load balancing. The prototype setup used in the experiment uses static load balancing, in which a specific set of one or more socket ports are assigned to a specific tunnel. A smarter load balancer might be able to assign tunnels in a more intelligent manner. Finally, there is also the possibility that TCP congestion control has a significant effect on the tunneled transports. When the tunnel stream fills up with too many small packets caused by the traffic of several virtual channels, TCP throttling for congestion control may well kick in (Nagle 1984). We have observed this in our experiments but must study this in greater depth.

6 References

- Buyya, R., Abramson, D., and Giddy, J. (2000): Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid. *Proc. HPC Asia 2000*, Beijing, China.
- Denis, A., Aumage, O., Hofman, R., Verstoep, K., Kielmann, T. and Bal, H. E. (2004): Wide-Area Communication for Grids: An Integrated Solution to Connectivity, Performance and Security Problems. *Proc. 13th IEEE Int. Symposium on High Performance Distributed Computing (HPDC-13 '04)*, pp. 97-106.

- D-Grid Integrationsprojekt (DGI) (2006): *Dynamic Firewalls Configuration, Ver. 1.0*, a technical report. https://www.d-grid.de/fileadmin/user_upload/documents/DGI-FG3-5/FG3-5_Dynamic_Firewalls.pdf. Accessed 14-Sep-08.
- Floyd, S. (2001): A report on recent developments in TCP congestion control. *IEEE Communications Magazine*, 39(4), pp. 84–90.
- Foster, I., Kesselman, C., and Tuecke, S. (2001): The anatomy of the Grid: enabling scalable virtual organizations. *Int. Journal of Supercomputer Applications*, 15(3).
- The Globus Alliance website, <http://www.globus.org>. Accessed 14-Sep-08.
- Graupner, S., and Reimann, C. (2002): *Globus grid and firewalls: issues and solutions in a utility data center environment*. Hewlett-Packard Co., Technical Report HPL-2002-278
- Green, M. L., Gallo, S. M. and Miller, R. (2004): Grid-Enabled Virtual Organization Based Dynamic Firewall. *Proc. 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*, Washington DC, USA, pp. 208–216.
- IBM (2003): *IBM Managed Hosting – virtualized managed firewall and load balancing services* (white paper). IBM. <http://www-935.ibm.com/services/us/ebhs/pdf/whitepaper-virtual-firewall-and-load-balancing.pdf>. Accessed 14-Sep-08.
- Inferno Nettverk A/S. *Dante: A free SOCKS implementation*. <http://www.inet.no/dante/>. Accessed 14-Sep-08.
- Lee, Y.-D.. (2005). *SOCKS: A protocol for TCP proxy across firewalls*. NEC Systems Laboratory. <http://archive.socks.permeo.com/protocol/socks4.protocol>. Accessed 14-Sep-08.
- Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and Jones, L. (1996): *SOCKS protocol v. 5*. IETF RFC 1928, <http://tools.ietf.org/html/rfc1928>. Accessed 14-Sep-08..
- Monga, I., Niederberger, R., and Metsch, T. (2008): Firewall Virtualization for Grid Applications. OGF23, FVGA-WG-BoF, Open Grid Forum.
- Nagle, J. (1984): *Congestion Control in IP/TCP Internetworks*. IETF RFC 896. Accessed 14-Sep-08..
- Quittek, S., Stiernerling, M., and Srisuresh, P. (2008). *Definitions of Managed Objects for Middlebox Communication*, Internet-Draft, IETF (Internet Engineering Task Force), <http://www.rfc-editor.org/rfc/rfc5190.txt>. Accessed 14-Sep-08.
- Rosmanith, H., Kranzlmüller, D. (2004): glogin - A Multifunctional, Interactive Tunnel into the Grid," in *Proc. 5th IEEE/ACM International Workshop on Grid Computing (GRID'04)*, Washington DC, USA, pp. 266–272.
- Son, S., Allcock, B., Livny, M. (2005): CODO: Firewall Traversal by Cooperative On-Demand Opening. *Proc. 14th IEEE Int. Symposium on High Performance Distributed Computing, (HPDC'05)*, pp. 233–242.
- Son, S. and Livny, M. (2003): Recovering Internet symmetry in distributed computing. *Proc. 3rd IEEE/ACM Int. Symposium Cluster Computing and the Grid (CCGrid 2003)*, Washington DC, USA, pp. 542–549.
- Tan, J., Abramson, D., and Enticott, C. (2005): Bridging organizational network boundaries on the Grid. *Proc. 6th IEEE/ACM Int. Workshop on Grid Computing (Grid 2005)*, Seattle, Washington, pp. 327–332.
- Tanaka, Y. Hirano, M. Nakada, H., and Sekiguchi, S. (2000): Performance evaluation of a firewall-compliant Globus-based wide-area cluster system. *Proc. IEEE Int. Symposium on High-Performance Distributed Computing 2000 (HPDC 2000)*, Washington DC, USA, pp. 121–128.
- Thorpe, S. (2005). [Online, an email sent to the globus-discuss mailing list about running a GridFTP server behind a NAT firewall]. http://www-unix.globus.org/mail_archive/discuss/2005/01/msg00216.html.
- Welch, V. (2008): *Globus Toolkit Firewall Requirements*. The Globus Alliance, <http://www.globus.org/toolkit/security/firewalls/>. Accessed 14-Sep-08.
- Ylonen, T. (1996): SSH -- Secure login connections over the Internet. *Proc. 6th USENIX Security Symposium*, San Jose, CA, pp. 37–42.

