

# Bridging Organizational Network Boundaries on the Grid

J. Tan<sup>†</sup>, D. Abramson<sup>†</sup> and C. Enticott<sup>§</sup>.

<sup>†</sup>School of Computer Science and Software Engineering, Monash University,  
<sup>§</sup> CRC for Enterprise Distributed Systems Technology (DSTC)

900 Dandenong Rd, Caulfield, Australia, 3145  
{jefferson.tan, david.abramson, colin.enticott}@csse.monash.edu.au

**Abstract—** The Grid offers significant opportunities for performing wide area distributed computing, allowing multiple organizations to collaborate and build dynamic and flexible virtual organisations. However, existing security firewalls often diminish the level of collaboration that is possible, and current Grid middleware often assumes that there are no restrictions on the type of communication that is allowed. Accordingly, a number of collaborations have failed because the member sites have different and conflicting security policies.

In this paper we present an architecture that facilitates inter-organization communication using existing Grid middleware, without compromising the security policies in place at each of the participating sites. Our solutions are built on a number of standard secure communication protocols such as SSH and SOCKS. We call this architecture Remus, and will demonstrate its effectiveness using the Nimrod/G tools.

## I. INTRODUCTION

THE Grid refers to “a distributed computing infrastructure for advanced science and engineering”[11]. As can be expected, Grid applications perform advanced computations through distributed processing. However, unlike the first generation of distributed computing paradigms, the Grid has an explicit focus in enabling *virtual organizations* through which shared and coordinated computations are possible across several institutions. A virtual organization (VO) refers to a group of people and/or organizations working together to share computations and resources.

As in any networked environment, security is an issue. Many VOs are formed with the Internet as the underlying communication infrastructure. As such, security mechanisms must be in place to protect each organization from potentially costly intrusion and sabotage. However, as they protect resources on the network, such mechanisms can also hamper accessibility. Securing resources is crucial, but we also seek to avoid counter-productivity.

Security issues studied in Grid applications include issues of authentication [18]. There is an obvious need to ascertain

the identity and authorization levels of a given user before access is allowed. Others have addressed problems with delegation [15] and how entities may share access rights to grid resources. Another key issue that has been addressed is the requirement that Grid computations can use the network while remaining consistent with prevailing security policies for the networks involved [10]. The mishandling of interoperability between domain security policies, or its total absence, can break grid computations. Firewalls, “choke points” by which to protect against unauthorized access over the network [6], can block grid applications that need to go on the network.

We refer to this as the *grid security interoperability* problem. Connections may be initiated successfully through one firewall then then fail to reach the intended second party. Somewhere in the interconnected networks involved, a mismatch occurs between what is allowed on one end and what is not allowed on the other. Grid applications must interoperate with the local security policies of all participating institutions. One cannot force local security policy changes on demand. That would break the autonomy of institutional domains and would be highly impractical [10]. Firewall restrictions are functions of an institution's security policy. Changing the former requires changing, or at least verifying the proposed changes against, the latter. One can understand how reluctant institutions would be to modify firewall restrictions every so often. This may prove impractical due to the dynamic nature of many grid applications. The burden of flexibility may be shifted to grid applications, but this, too, is not practical, particularly if this requires coding changes. Furthermore, institutions would be unhappy to find that the grid application components they are hosting are capable of firewall evasion. On the other hand, deploying agents to handle local security for “visiting” or guest clients is a good alternative [10]. It is a third party player that can conform to the security policy of the hosting institution and it can be designed to be unobtrusive and flexible. Using this philosophy, we designed a communication system that features grid security interoperability. Given recent

technology in advanced network communications through proxies, tunnelling and port forwarding, we developed a communication infrastructure one level above actual transport modes for service requests and responses. We implement it on top of multiple communication protocols with off-the-shelf or pre-installed utilities, e.g., SSH or SOCKS. We facilitate interoperability with security policies on behalf of the grid application components and avoid burdening either the security policymakers or the grid application programmers. By resorting to pre-installed utilities, local administrators retain control over how we get across the network.

The rest of this paper is organized thus: Section II describes how network security hampers grid applications. Section III summarizes and classifies those conflicts. Section IV describes resolution strategies and presents the grid communication architecture that we have developed: Remus. Section V discusses a case study with Nimrod/G [4]. Section VI looks at related literature and Section VII presents our conclusions.

## II. BLOCKING GRID APPLICATIONS

The networks of institutions in a VO setting are protected using firewalls and other mechanisms. However, while access to their resources must be restricted, they need to be shared with remote components over the network. Typical firewall restrictions operate by checking packet destinations against a list of open, closed and filtered ports. This applies to inbound, outbound, and forwarded packets, relative to the protected domain. If a firewall is blocking incoming connections to a given port on a given host from outside the local network, then the services behind that port are inaccessible over the grid.

Let us consider the parameters for such problems. The VO is made up of a set of network domains  $VO = D_1, D_2, \dots, D_z$ . Each such network domain  $D_i$  is made up of nodes  $n_1, n_2, \dots, n_y$ . To simplify our discussion, we will also consider each node identifier  $n_j$  as the node's address. A connection is made to each node using a certain protocol  $\theta$  and a given destination port number  $1 \leq p_k \leq 65535$ . Assume for this discussion that we are only concerned with application-level protocols, e.g., SSH, TELNET, SMTP. The parameters for a connection  $K$  are therefore the destination address, port number and protocol type,  $K = \langle n_j, p_k, \theta \rangle$ . From such security policies surrounding a given node or its network, we can extract a finite list of possible connection parameter combinations and a complementary list of disallowed connections.

It is common to find several firewalls within the same organization, e.g., for each network or local firewalls for some nodes. Combining the various lists of the firewalls traversed makes for a potentially narrow window of accessibility. To simplify each situation between two nodes that need to connect, we instead consider a single list which results from compounding the security policies across the two nodes and all networks between them. Our concern is in specific connections to port  $p_j$  on node  $n_B$ , i.e.,  $n_B.p_j$ , from some source

port on node  $n_A$ . This list includes ports that are simply open but not bound to any pre-existing service, or those that are through some specific protocol. Note that we use the term "connection" loosely in this paper. We do not necessarily imply a connection-oriented protocol such as TCP. This may also include UDP (user datagram protocol), which is connectionless. We say that a node  $n_A$  is connected to port  $p_i$  on node  $n_B$  via protocol  $\theta$  when successful transport-level communication, e.g., TCP or UDP, begins and persists for the duration intended. The source port on  $n_A$  is irrelevant as long as a connection can be made.

A connection therefore requires three things. First, there must not be any restriction that will block connections from  $n_A$  to  $n_B$  on port  $p_i$ . Second, the connection should have been initiated by a component  $c_A$  and accepted by component  $c_B$ . Finally, it is necessary that both  $c_A$  and  $c_B$  can speak the same language via protocol  $\theta$ .

## III. CONNECTION PROBLEMS

Given that a successful connection  $K = \langle n_j, p_k, \theta \rangle$  involves three parameters, there are three corresponding types of conflicts that can hinder communications.

A *node conflict* occurs when security restrictions block connections to the target node. A node that hosts a database server may block any connection request from outside the local network, or it may reside within a hidden, private subnet, e.g., IP addresses of 192.168.\*.\*. It is also possible that a firewall or the router at the source will block outgoing connections to external or unauthorized networks.

A *port conflict* occurs when security restrictions block connections to the target port, implying that other ports instead are accessible. This is a common restriction that protects perpetually running services for which allowed connections are limited. For example, one might be running a private or Intranet file server that should only be accessible within a given group of users. It is also possible that a firewall at the source will block outgoing connections, e.g., to block access to e-mail delivery relays on port 25 (SMTP) unless it is the corporate SMTP server. This is a common measure to make sure that outgoing spam can be caught and blocked.

In a port conflict situation, we may find that there is instead an accessible vacant port. This presents a *vacant port conflict* in which the port is not bound to any service. This is a conflict only in the sense that we are unable to communicate with a component on the other end through that vacant port.

Finally, a *protocol conflict* happens when a successful connection can be made to the node/port  $n_B.p_k$  but the service listening behind  $p_k$  is unable to communicate using the protocol that the connecting component can use. We might instead encounter some service *daemon*  $d(n_B, p_k, \theta')$  that communicates using a different protocol  $\theta'$ . A daemon is a generic term with UNIX origins for a long-duration process that runs in the background listening for connections over the network by which it can deliver specific services. For example, perhaps our client component wishes to transfer files

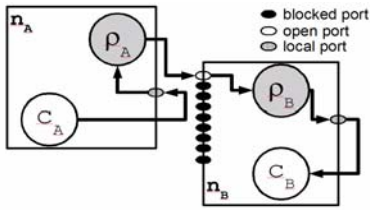


Fig. 1. Using the vacant port means we are free to bind it to our own rerouting component ( $\rho_B$ ).

via FTP (File Transfer Protocol) but the only open port on  $n_B$  is port 80 which can only accept HTTP (HyperText Transfer Protocol).

#### IV. CONFLICT RESOLUTION

To fix a vacant port conflict, we use a *rerouter*  $\rho_B$  on the vacant port of  $n_B$  to forward connections between components  $c_A$  and  $c_B$  using a protocol that they understand. In case component  $c_A$  can only access local ports or sockets, we can use a local rerouter  $\rho_A$ . Fig. 1 illustrates.

Dealing with a protocol conflict, recall that the component  $c_A$  on node  $n_A$ , and  $c_B$  on node  $n_B$ , wish to communicate using a common protocol  $\theta$ . We have no choice but to use a non-vacant port, as in Fig. 2, where the port is already bound to some daemon component  $d(n_B, p_k, \theta')$  on  $n_B$ , port  $p_j$ , that communicates using protocol  $\theta'$ . The assumption is that this daemon exists independently of  $c_B$ , and has no pre-existing functionality to communicate with it. The simplest solution is to use a rerouter  $\rho_A$  on node  $n_A$  to connect through the daemon using protocol  $\theta'$  and apply what we call a *plug-in solution* where the daemon hosts  $\rho_B$  as a "plug-in." For example, if this daemon happens to support web services [20], then we can implement  $\rho_B$  as a web service that  $\rho_A$  can talk to. It becomes the job of  $\rho_B$  to talk to  $c_B$  using protocol  $\theta$ . If the daemon is incapable of this, then local administrators may have to open a hole through the firewall to a preferred port.

Finally, if there are no directly accessible ports, we require a third-party *proxy node* to facilitate an indirect connection between  $n_A$  and  $n_B$ , as shown on Fig. 3. If possible, we want to use protocol  $\theta$  throughout, including connections to  $n_{proxy}$ . It may also be the case that we need multiple proxy nodes in a chain. Rerouters on multiple proxies, i.e.,  $\rho_2$  through to  $\rho_f$ , link up and forward the connection through. The complexity of the proxy chain is hidden, and we avoid having to reprogram  $c_A$  and  $c_B$  although communication slows down to some degree.

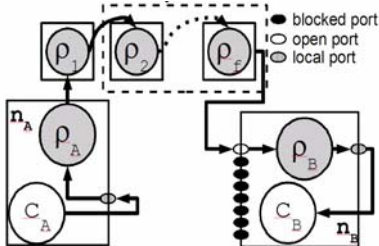


Fig. 3. Using one or more proxies to get around firewall restrictions. This presents a circuitous route but the connection is virtually direct, since the rerouters will hide the complexity from the  $c_A$  and  $c_B$ .

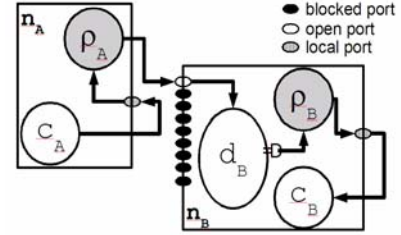


Fig. 2. Protocol conflict on a non-vacant port already bound to a daemon process. Our best option is to plug-in the rerouter to the existing daemon, assuming that it accepts plug-in components

These conflict resolution strategies are facilitated through a **Rerouting and Multiplexing System (Remus)** that we are developing. The principles behind Remus are well-established and we reuse pre-existing communication utilities underneath. Establishing the link between components may involve tunnelling one protocol under another. Remus was developed with grid applications in mind. We expect situations that may be peculiar in two ways: grid applications may need to communicate using multiple channels using multiple protocols, and the same grid application may be deployed over different VO scenarios, presenting different security incompatibilities. We therefore developed Remus to be unobtrusive, flexible and capable of multiple strategies that can be used as needed.

The illustrations in Fig. 4 illustrate the Remus architecture in two configurations. At least one rerouter must be active on each side of the firewall(s) and network(s) in between, with some direct or indirect path between them. Fig. 4(a) illustrates a simple connection that pipes port connections across two or more firewalls. A *star deployment* as shown on Fig. 4(b) is also possible. A rerouter "hub" can channel connections for multiple components ( $a$  through  $x$  in the boxes) on different nodes, as long as the hub host is accessible to them. To support some network implementations with segments, Remus rerouters may also use proxy chaining techniques [1] with multiple hops across a chain of rerouters (see Fig. 3).

#### V. CASE STUDY: NIMROD/G AND REMUS

##### A. Nimrod/G

We tested the applicability of Remus through Nimrod, a middleware that sits above and harnesses the computational power of several loosely-coupled processors in processing experiments [3]. Nimrod experiments are *parametric applications*, involving a large number of tasks that process a large range of parameters. Useful areas include design and engineering, as both areas benefit from the simulation of their target product over several different situations to study the effects of changing parameters, e.g., weather conditions, load, etc. The large volume of data and processing involved prompts us to exploit distributed computing over a large number of processors. A *cluster* of such processors is a typical environment which support distributed computing.

Nimrod/G [4] is a Grid-enabled version of Nimrod that exploits the use of various grid-middleware services in order to use computational grids of various standards, e.g., Condor

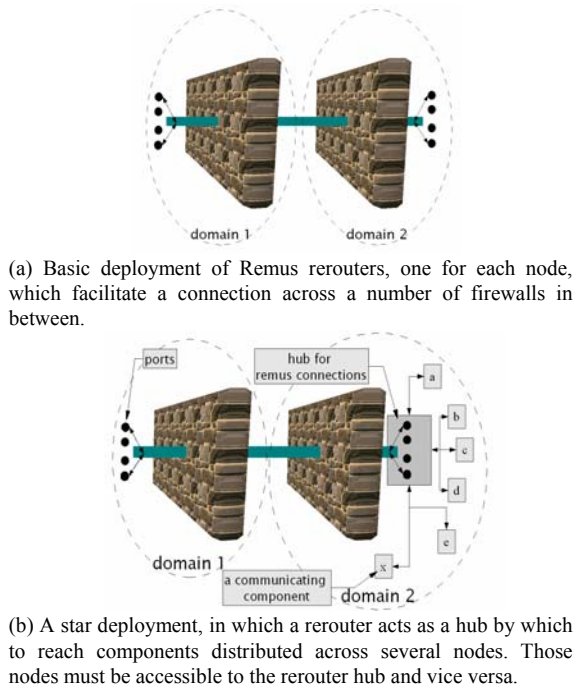


Fig. 4. Remus deployment configurations.

[7] and Globus [12]. Nimrod/G makes use of multiple remote resources through grid-middleware services. Such services have been accessed by Nimrod/G via the Internet across different institutions. However, as with many grid applications, firewalls have caused problems. In one instance, before we started using rerouters, firewalls made it impossible to deploy Nimrod/G.

There are three layers of communicating components in Nimrod/G. The *computational resource layer* consists of the actual resources used to perform experiments. The *resource management layer* oversees the usage of those resources. Finally, the *experiment management layer* contains Nimrod/G itself, which is the experiment and job manager through which experiments are submitted and from which results are finally obtained. The resource management layer may consist of several independent resource managers that Nimrod/G may invoke to make use of the resources managed by the resource managers. Note that the other components of Nimrod/G, e.g., pre-processors and schedulers, are not described here. We wish to focus on those components that will use the network for which Remus makes a difference.

At each of the two abstractions between the three layers, firewalls can get in the way. The firewall between the resource management components and the computational resources they manage is probably easy to traverse. This is because these components, the networks that link them, and network resources including firewalls, are usually under the same network manager. On the other hand, the uppermost Nimrod/G layer is probably on another network, so the firewall can get in the way by blocking traffic between Nimrod/G and resource managers. For example, Nimrod/G at Monash University is configured to use computational

resources in other institutions. Experiments submitted via Nimrod/G translate into jobs to be submitted by Nimrod/G to a resource manager in another institution. While current security policies are lax, this is expected to change in the future and is not usually how grid situations are.

When Nimrod/G submits jobs to the resource manager, this is in the form of a set of *Nimrod agents*. These agents encapsulate and manage individual jobs that are to be processed. After an agent is deployed on a given computational node, it communicates with Nimrod/G for two reasons: for experiment control directives and for input and output file transfers. The agent actually talks to two Nimrod/G components: a database server for experiment and agent control, and a file server for file transfers. This necessitates a connection from the agent to two distinct ports behind which the two Nimrod/G components can each listen [4]. The firewall protecting the computational resources may block these connections initiated from those computational nodes.

Nimrod/G can also deploy a *Nimrod proxy* component that can sit on a “launch” or “front” node, listening behind two ports to mimic the ports behind which the Nimrod/G database and file servers listen for connections. The agents on the computational nodes talk to the Nimrod proxy as if they were talking to the database and file servers. However, the firewall protecting the computational resources may still block connections leaving the node hosting the Nimrod proxy. After all, the Nimrod proxy still needs to connect to the same two ports of Nimrod/G. The Nimrod proxy may be repositioned, but it must be situated so that the agents are not blocked from reaching it by the firewall that protects the computational nodes.

### B. Using Remus

Fig. 5 illustrates how Remus is used. Circles are processes and boxes drawn with solid lines are nodes. The ellipses drawn with dashed lines represent departmental or institutional domains. Boxes drawn with dashed lines represent networks. The solid bar which links  $\rho_A$  and  $\rho_B$  represents a tunnelled link. Nimrod agents that process experiments on compute nodes are drawn as solid dots inside shaded squares, the squares being compute nodes.

To the left of the firewall is a single deployment of Nimrod/G within a single institutional domain, i.e., ours. To simplify the discussion, we only show two components of Nimrod/G running on the Nimrod/G node: the file server (FS) and the database server (DBS). Focusing only on the left domain, i.e., our own resources, it is straightforward for agents to interact with both the FS and DBS components. This is not so when dealing with external institutions. One or more firewalls would stand between Nimrod/G and the agents running on the external network. Note in Fig. 5 how the Nimrod proxy ( $NP_x$ ) marshals all the connections from the agents to Nimrod/G via a single channel. However, the Nimrod proxy is not designed to deal with tunnelling and varying firewall restrictions. A rerouter that can exploit several strategies is necessary to deal with that.

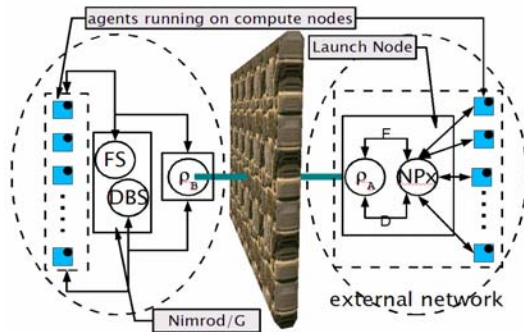


Fig. 5. Nimrod/G with a Remus connection via rerouters that allow the Nimrod proxy ( $N_{Px}$ ) on the external network to connect to the Nimrod/G file and DB servers ( $FS$  and  $DBS$ ).

In Fig. 5,  $\rho_A$  sets up two listening sockets that the Nimrod proxy will be fooled into using. When connections are made through those sockets,  $\rho_A$  shoots the connection through to the other side. This was originally implemented using SSH underneath, i.e.,  $\rho_A$  is the SSH client and is the  $\rho_B$  SSH server. The connections are passed along via port forwarding to the Nimrod DB and file servers. We can automate the initial connection with passphrase-less login and disable shell sessions on the other end for security. Note that an account must be set up, and SSH connections must be direct connections.

More recently, we implemented Remus rerouters with SOCKS proxy connections. Our prototypical but successful setup was implemented with **Dante** [8] using the **socksify** wrapper utility at the core of rerouter  $\rho_A$  for the Nimrod agents. The  $\rho_B$  rerouter for the database and file servers is implemented using the SOCKS proxy server. Fig. 5 shows how  $\rho_B$  is on a third party node (in a standalone square). The actual connection required only one TCP port. While it is better to marshal all connections through the Nimrod proxy connecting via a local rerouter, the current prototype allows the agents to connect without the Nimrod proxy. If necessary, we could use the Nimrod proxy which alone needs to be wrapped within **socksify**. Note that this wrapper does not affect the listening ports of the wrapped component. For example, the Nimrod proxy, wrapped around with **socksify**, will nonetheless be accessible to the agents, just as it was without the wrapper around it. This is because this wrapper alters outgoing connections only. This implementation illustrates how a proxy solution works, where connections go through  $\rho_B$  on a third party node. SOCKS has an automated reconnection feature, making it quite reliable. Agent-to-Nimrod/G connections are short, and Nimrod/G servers and agents can withstand disconnections.

Setting up rerouters is simple. Scripts will set up the rerouters either as SSH or SOCKS clients. We assume that appropriate servers for SSH and SOCKS are available. Configuration parameters, e.g., ports and IP addresses, are given as command-line parameters. They need not be invoked by a superuser as long as the listening ports are set up above port 1023. Note that Nimrod/G components only use TCP

Communication using Remus will suffer some degradation.

We have not taken measurements, but any attempt to tunnel or use proxies suffers from overhead. Compression helps but the improved speed may not match direct connection speeds.

## VI. RELATED WORK

The sort of firewall-related problems dealt with in this paper have been discussed widely under various headings, e.g., *covert channels* [5]. This was originally more of a concern among Internet users. Several tools exist to evade firewalls. SOCKS and web proxies are the most common, and anonymous proxies make them easy to use without special utilities at the client. Several utilities and protocols can also work together, e.g., SSH port forwarding tunnelled via web proxy using **proxytunnel** (<http://proxytunnel.sourceforge.net>).

As they pertain to grid applications, problems caused by security measures have been documented. Many have focused on authentication issues, e.g., the Grid Security Infrastructure (GSI) [13]. Some have mentioned firewall issues [10], [15], [19] and [21]. For example, when dealing with the Globus toolkit, it has been noted that one port must be open for the Gatekeeper but a whole range of other ports are necessary for clients that call back [15]. Grid services can be invoked from a host in a demilitarized zone (DMZ) in front of the firewall. However, this places the DMZ host at risk. HTTP tunnelling is also possible through a web proxy, but, this assumes that web proxies are accessible. The web proxy might also be configured to block tunnelling. In some protected networks, the web proxy may ignore requests to sites external to the institutional domain. SSH tunnelling is a favoured approach, using port forwarding as we have done for Nimrod/G. The assumption is that there is either one firewall-free host involved, or, if both communicating hosts are behind firewalls, there is a third-party host that is firewall-free. This third-party host must be protected as it presents a possible backdoor. However, it is probably easier to protect a host running nothing but SSH as compared to one that runs Grid services. The practical use of SOCKS proxies has also been noted [15], with the assumption that communicating components must be capable of using SOCKS. Linking them to or rebuilding them with SOCKS libraries is sometimes necessary [19].

A proxy approach in tandem with a specialized resource manager was proposed for a wide-area cluster system in [19]. It used the Nexus proxy that was built into Globus. Job requests are received by a component outside the firewall. The resource allocator inside the firewall looks into which resource to send the job to. When a decision is reached, the job is sent to the proper resource through a local Q server (for queuing). However, this approach had limitations. The Nexus proxy only supports TCP, and is built into Globus itself, and it does not support multiple protocols.

We must also note the significance of *web services* [20] which provides a standard means to locate and communicate with applications over a network. It is significant in our discussion because web services are also built on top of

several possible communication protocols, e.g., HTTP, SMTP and FTP. Data is packaged using a machine-readable format such as XML and delivered via SOAP (Simple Object Access Protocol) messages [20]. However, since security management remains an issue, whereas security managers normally watch ports and protocols, they will simply shift to watching request headers in a web services infrastructure. There are also concerns about the performance drawbacks of web services, due to the focus of SOAP on flexibility [9]. On the other hand, Remus rerouters focus on port forwarding as the means to interoperability. Rerouters are invisible to the communicating components. The only assumption we make is that they communicate using TCP or UDP ports.

## VII. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we presented the problems caused by firewalls to grid applications. We also offered solutions that we have developed, with the Nimrod/G case study as a proof of concept for one of those solutions. We believe that it is not completely up to the grid applications to harmonize with existing security policies for the institutions involved in a VO undertaking. There is a practical necessity to avoid costly changes, especially code changes, in conforming to intra-institutional security policies. Neither is it practical for network security policies and devices to be to cater to every requirement of grid applications. While there are many well-known firewall evasion utilities around, we did not see any multi-pronged solutions that are specifically applied to grid applications.

We proposed, developed and tested a rerouting and multiplexing system called Remus, to address the problem. Rerouters will be stationed where grid application components can reach them. It is up to the rerouters to set up the links between the communicating grid components across one or more firewalls. Prototypical implementations using SSH and SOCKS were successfully tested on Nimrod/G.

Further refinements and enhancements are on-going. We will explore plug-in solutions, in which rerouters are service or component plug-ins to pre-existing daemons. We must also look at proxy chaining, and GSS-API support for encrypted communications on SOCKS. We will also study persistent link features to support grid application components that act as senders and listeners at different stages. We plan to add modules that will allow rerouters to communicate through other means, e.g., web services using SOAP, tunnelling over UDP, and other means that may be more appropriate for a given situation. We will also look into problems with complicated network topologies and advanced firewall restrictions.

## ACKNOWLEDGMENTS

The authors would like to acknowledge their colleagues who have contributed to this work, in particular, Mr Slavisa Garic. This project is supported by the Australian Partnership for Advanced Computing (APAC). Nimrod/G is supported by

the CRC for Enterprise Distributed Systems Technology (DSTC). Colin Enticott is supported by the Australian Government Department of Communications, Information Technology and the Arts under a GrangeNet award.

## REFERENCES

- [1] B. Aboba and J. Vollbrecht. "Proxy chaining and policy implementation in roaming." RFC 2607, June 1999.
- [2] Abramson, D., Barak, A and Enticott, C. "Job Management in Grids of MOSIX Clusters", 16th International Conference on Parallel and Distributed Computing Systems, August 13 - 15, pp 36 - 42, 2003 Reno, Nevada, USA,
- [3] D. Abramson, R. Sosic, J. Giddy, and B. Hall. "Nimrod: A tool for performing parametrised simulations using distributed workstations." In *Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing*, Virginia, August 1995.
- [4] R. Buyya, D. Abramson, and J. Giddy. "Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid." In *Proceedings of HPC Asia 2000*, Beijing, China, May 14-17 2000.
- [5] National Computer Security Center. "A guide to understanding covert channel analysis of trusted systems." Technical Report NCSC-TG-030, National Computer Security Center, Fort Meade, Maryland, November 1993. Available: <http://www.radium.ncsc.mil/tpep/library/rainbow/NCSC-TG-030.html>.
- [6] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional Computing Series. Addison-Wesley, second edition, 2003.
- [7] The condor project homepage. <http://www.cs.wisc.edu/condor>.
- [8] Inferno Netverk A/S. "Dante: A free SOCKS implementation." URL: <http://www.inet.no/dante/>.
- [9] R. Elfving, U. Paulsson and L. Lundberg. "Performance of SOAP in web service environment compared to CORBA." In *Proc. IEEE 9th Asia-Pacific Software Engineering Conference (APSEC'02)*, pages 84-93, Dec. 2002.
- [10] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. "A security architecture for computational grids." In *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pages 83-92, 1998.
- [11] I. Foster, C. Kesselman, and S. Tuecke. "The anatomy of the Grid: enabling scalable virtual organizations." *International Journal of Supercomputer Applications*, 15(3), 2001.
- [12] Ian Foster and Carl Kesselman. "Globus: A metacomputing infrastructure toolkit." *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115-128, Summer 1997.
- [13] Overview of the grid security infrastructure (GSI). Available: URL: <http://www.globus.org/security/overview.html>.
- [14] The Globus Alliance website: <http://www.globus.org>.
- [15] S. Graupner and C. Reimann. "Globus grid and firewalls: Issues and solutions in a utility data center environment." Technical Report HPL-2002-278, HP Laboratories Palo Alto, 2002.
- [16] Ying-Da Lee. "SOCKS: A protocol for TCP proxy access across firewalls." Available:<http://archive.socks.permeo.com/protocol/socks4.protocol>
- [17] P. McMahon. "GSS-API authentication method for SOCKS version 5. RFC 1961." June 1996.
- [18] J.M. Schopf and B. Nitzberg. "Grids: Top ten questions." *Scientific Programming, special issue on Grid Computing*, 10(2):103-111, August 2002.
- [19] Y. Tanaka, M. Sato, M. Hirano, H. Nakada and S. Sekiguchi. "Performance Evaluation of a Firewall-Compliant Globus-Based Wide-Area Cluster System," *hpdc*, vol. 00, no. , p. 121, 2000.
- [20] Web services architecture. Available: <http://www.w3.org/TR/ws-arch>.
- [21] V. Welch. "Globus toolkit firewall requirements." Available: <http://www-unix.globus.org/toolkit/security/firewalls/>.