Time Division Multiplexing of Network Access by Security Groups in High Performance Computing Environments

by

Joshua Ferguson

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved March 2013 by the
Graduate Supervisory Committee:

Sandeep Gupta, Chair
George Ball
Georgios Varsamopoulos

ARIZONA STATE UNIVERSITY

May 2013

ABSTRACT

It is commonly known that High Performance Computing (HPC) systems are most frequently used by multiple users for batch job, parallel computations. Less well known, however, are the numerous HPC systems servicing data so sensitive that administrators enforce either *a*) sequential job processing - only one job at a time on the entire system, or *b*) physical separation - devoting an entire HPC system to a single project until recommissioned. The driving forces behind this type of security are numerous but share the common origin of data so sensitive that measures above and beyond industry standard are used to ensure information security. This paper presents a network security solution that provides information security above and beyond industry standard, yet still enabling multi-user computations on the system. This paper's main contribution is a mechanism designed to enforce high level time division multiplexing of network access (Time Division Multiple Access, or TDMA) according to security groups. By dividing network access into time windows, interactions between applications over the network can be prevented in an easily verifiable way.

ACKNOWLEDGEMENTS

I would like to acknowledge and thank Dr. Gupta for taking a chance and inviting me to work in Impact Lab, Dr. Varsamopoulos for his immense technical guidance (especially regarding Linux), and Dr. Ball for his earnest support of this thesis and its goals. I am intellectually and personally indebted to the members of Impact Lab for their help with the myriad of tasks that arose during my time with the lab. Special mention must go to Dr. Tridib Mukherjee, Dr. Ayan Bannerjee, and (soon to be Dr.) Zahra Abbasi for helping me with their seemingly boundless knowledge of the research we did, as well as Robin Gilbert for being a true friend and great colleague. I thank Raytheon for their capital support of our research. Finally, I thank my parents and brother for their love and support.

To my wife, Sara, for her unwavering support

Contents

CHAPTER Page

List of Figures

Figure Page

Chapter 1

Introduction

High Performance Computing (HPC) systems consist of numerous individual computing systems networked and administrated together such that they can be used as a single system. Examples of these systems from popular culture include custom made models such as the Cray I (historically one of the first systems deemed HPC) and the modern IBM Blue Gene [19]. More common examples are simple Computer Clusters such as Beowulf clusters in which Commercial Off The Shelf (COTS) equipment is utilized [7]. These latter systems are simple enough that they are frequently implemented by single users within hobbyists' homes [6]. Figures 1.1 , 1.2, and 1.3 show examples of these systems.

Security Concerns

Application developers for these systems span a broad spectrum, ranging from undergraduate students learning concurrent programming to defense contractors executing classified simulations. Key characteristics of this spectrum are shown in Figure 1.4. Moving towards the most demanding end of the spectrum, security concerns among application-side stakeholders increase substantially and additional

Figure 1.1: A beowulf cluster. [30]

Figure 1.2: The Cray I. [27]

Figure 1.3: IBM's Blue Gene. [18]

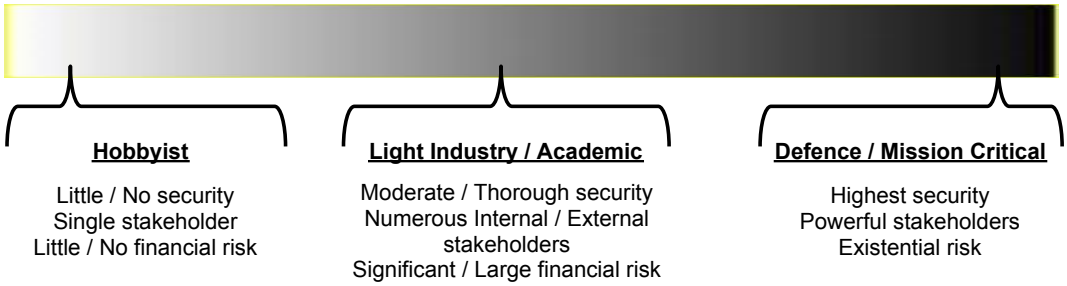

Figure 1.4: The spectrum of environmental security requirements based on uses and stakeholders.

methods are employed to enforce information security. At some point along this spectrum stakeholders demand physical separation of the system from other users during operation to satisfy security concerns. The reasons behind this can be numerous, but stem from two major goals: simplicity of implementation and verification; and risk aversion/management. In the defense industry particularly, information security breaches can threaten the existence of entire programs due to certification revocation from agencies such as the Department of Defense (DoD) [24], the DoD's Defense Security Service (DSS) agency [25], and the National Institute of Standards and Technology (NIST). Such risk reasonably implies physical separation of systems under operation from other users as well as numerous other physical security requirements.

It is undeniable that physical separation provides a level of information security that is difficult to replicate through the use of software, however the financial costs are significant - devoting entire HPC systems to a single project, or running jobs sequentially with downtime for data cleansing between [25].

Time Division Multiple Access Scheme

This paper presents a Time Division Multiple Access (TDMA) scheme of network resources as a viable alternative to physical separation. By modulating network

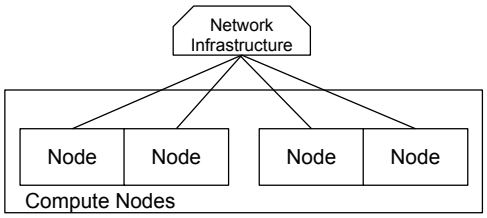

Figure 1.5: Unmodified computing nodes.

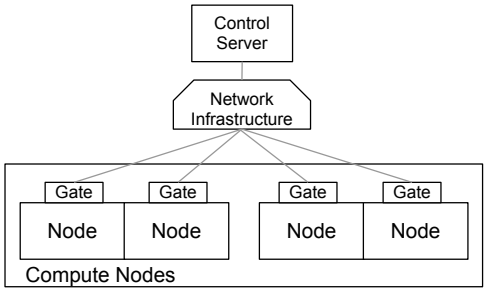

Figure 1.6: TDMA overlaid onto Figure 1.5

access between application security groups we can provide an intuitive security mechanism, verifiable in real-time, capable of mimicking aspects of the security provided by physical separation. Furthermore by implementing this mechanism at the operating system level it becomes transparent to user applications, meaning no modification to existing application code is necessary. Providing a mechanism for operating multiple user applications on a single HPC system securely with the support of stakeholders can provide substantial monetary savings and efficiency gains over physical separation.

The scheme works by inserting gates to network access at each computing node (computing devices devoted to executing user applications) within the system. These gates are modulated open and closed by a central administrative program, denoted as the *control server*, with knowledge of users and the data they own, denoted as *security groups*. The key operation of the scheme is the control server modulating network access of individual computing nodes such that systems executing application(s) from one security group never have access at the same time as systems containing data from a different security group.

Chapter 6 provides details on the implementation of this scheme, and Section 5 formally defines the scheme's operations.

Chapter 2

Related Work

The problem statement and proposed solution represent the intersection of two somewhat disparate fields - Time Division Multiple Access and High Performance Computing Security. Related works are therefore divided between the two.

High Performance Computing Security

The size and cost of HPC environments dictates that each system is somewhat unique. The security solutions implemented within each are similarly unique. Sandholm et al. [29] make an attempt at rectifying this larger problem by creating a framework that automates user access permissions and resource allocation using "XACML (eXtensible Access Control Markup Language)". They further extend their solution by tying it in to existing job submission tools (Globus Toolkit [23] and NorduGrid [26]).

Allcock et al. [2] developed a high-speed data transport protocol, GridFTP, as well as a corresponding administrative service providing for the creation, registration, and secure transportation of scientific computing datasets. For efficient execution, HPC applications must carefully consider characteristics of the data set under operation such as file size statistics, data creation/consumption rates, and logical distribution [8]. GridFTP implements management of these characteristics while maintaining customizable security using the authentication mechanisms defined in RFC 2228 "FTP Security Extensions" [12]. This solution, while useful in most scientific computing setting, still allows for application data, albeit encrypted, to be visible over the network to other user applications. This visibility renders it insufficient for the requirements of customers with the most stringent data security

needs.

Time Division Multiple Access

Mages and Feng [22] patented a similar control scheme of computing resources via a centralized controller over the network. Their scheme, however, specifies only local media resources of the node as under the control of the central administrative node. Furthermore, their patent is intended for a much wider distributed use as digital rights management and security in consumer media devices, rather than our work on security in HPC environments.

Chapter 3

Problem Definition

We begin by defining an abstract HPC environment through which the general case of our security challenge is shown. In this section we provide brief descriptions of the major resources common to most HPC systems. Furthermore, to design our mechanism, certain assumptions must be made on how each resources is operated.

Assumptions on the Computing Environment

There are four basic resources in most HPC systems represented in Figure 3.1 as *a*) compute nodes, *b*) persistent storage, *c*) administrative nodes, and *d*) network infrastructure. Worth consideration also is the process of job allocation and the execution of jobs.

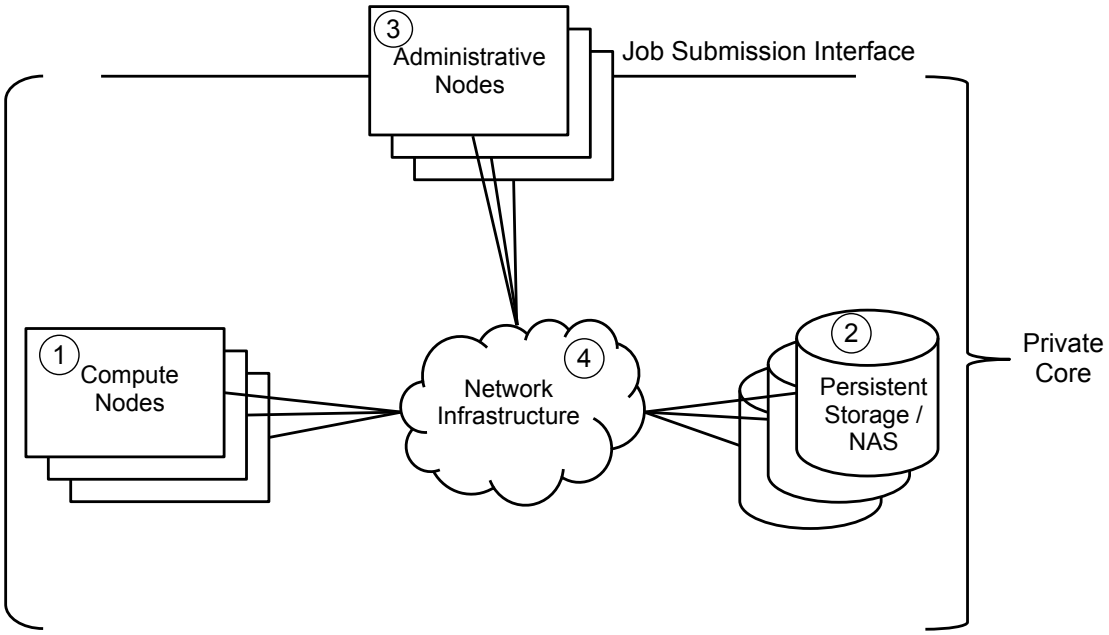

Figure 3.1: An abstract HPC environment.

Figure 3.2: The $KG-200$ Inline Media Encryptor, certified by the NSA for use in securing persistent storage [1].

Compute Nodes

Compute nodes are independent computing devices designated to run user submitted applications. These devices are capable of storing temporary data locally. They send and receive data across network infrastructure for three main purposes:*a*) storing or accessing data on the persistent storage devices; *b*) relaying data between other compute nodes working in tandem on the same user application; *c*) and sending or receiving commands (or reports, as the case may be) from the administrative nodes, through which users interact.

It is assumed that these compute nodes do not co-locate user applications (e.g., two different user applications are running in the same system memory) and that user applications are not given administrative access at this level. No assumption is made about the use of virtual machines on compute nodes.

Persistent Storage

Persistent storage as Network-Attached Storage (NAS) devices are capable of storing large quantities of user application data, and are usually of much higher capacity than the compute nodes. These devices commonly use RAID (redundant array of inexpensive disks) technology [15] for higher storage efficiency and redundancy.

It is assumed that Inline Media Encryptors (IMEs) and POSIX permissions are

used to enforce data access rules within persistent storage [9]. IMEs have been certified for use in classified networks by the U.S. National Security Administration since 2006 [1].

Administrative Nodes

Administrative nodes are computing devices where *a*) both administrators and users interact with the system, common tasks of which include issuing job or system commands, accessing reports and results, and performing maintenance; *b*) resource management software is centrally located and executed [16], common examples include IBM's Tivoli Workload Scheduler and the MOAB Cluster Suite by Adaptive Computing [3][13].

It is assumed that the scheduler located here is capable of providing access to the list of current running applications and the hardware resources devoted to them.

Network Infrastructure

Network infrastructure devices facilitate the transmission of data between nodes within the HPC system. Mediums vary widely and include copper, optical, and wireless. The most common technologies used in HPC environments are Ethernet and InfiniBand [5][21].

It is assumed that the network infrastructure uses Internet Protocol to communicate among nodes.

Job Execution

Best practices for developing jobs run on HPC systems dictates the minimization of I/O, both to disk and over the network [32]. This I/O minimization is due to the dramatic increase in access time as data moves further away from the CPU and main memory. It's over 50 times more costly to access 1MB of data from the network

Action	Time to Complete
L1 cache reference	0.5 ns
L2 cache reference	7 ns
Main memory reference	100 ns
Read 1 MB sequentially from memory	250,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns

Table 3.1: Access time examples showing the magnitude of difference between data over I/O and data locally stored [10].

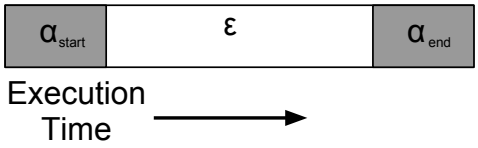

Figure 3.3: The assumed model of application execution in an HPC environment. α_{start} and α_{end} are periods where execution is I/O bound, and ε is the prominent period where execution is CPU bound. This structure adheres to research showing batched I/O minimizes the I/O cost in terms of time.

than it is from main memory [10]. This overhead increases to almost a factor of 100 if that data is initially read from disk then sent over the network [10].

In the effort to minimize the cost of I/O transactions, previous researchers have shown that batching I/O into larger transactions can reduce overhead [31]. The difference between sequentially reading 1K files from network disks and reading 256MB from network disks shows a factor of 1700 improved performance by reading in larger batches [32]. The batching of I/O, especially the most costly forms (disk and network) is therefore considered best practice when possible [4]. It is therefore assumed that job developers will attempt to maximize I/O batching, the optimal case of which would have an I/O transaction history similar to that shown in Figure 3.3.

		Solutions		
Hardware Location	**Security Challenge**	Board Separation	IME	Posix Permissions
Compute Nodes	Local Data Storage	✓		✓
	Local Data Processing	✓		✓
Persistent Storage	Co-location of user data		✓	✓
Administrative Node	Accept and Schedule User Jobs			✓
Network Infrastructure	Transmit intra-application data between compute nodes			
	Transmit data to/from compute nodes and persistent storage			
	Transmit commands from scheduler to compute nodes			

Table 3.2: Security challenges and technology used to solve them

Security Challenge

The security fear of users with extremely sensitive data is that a different user could, through chance or intention, acquire or manipulate their data. The four basic resources described above represent the resources across which data may be exposed. Table 3.2 organizes the aforementioned ways in which these resources are secured [7].

So far we have described ways in which three of the four shared resources are secured by technology that is either certified by national defense agencies (as in the case of persistent storage) or intuitive and easily verifiable (as in the case of compute node board separation and administrative POSIX permissions). This leaves the shared networking resources as a point of data sharing.

Insufficient Solutions

There exist current solutions for the strict problem of preventing plaintext data sharing, but these solutions lack certain qualities that make them sufficient for the narrow solution we're seeking when trying to assure users assuming significant risk. These existing solutions fail through their lack of thoroughness, lack of intuitiveness, inability to be simply verified, and even through established security flaws.

Encryption

Socket to socket encryption is a common solution to preventing data theft over a network, though somewhat out of place in an HPC environment due to their overhead. This solution falls short in its inability to prevent data sharing. Though the data is encrypted, the value of the plaintext (unencrypted data) is commonly so high that copying, storing and later decrypting the network traffic a risk to be prevented.

Virtual Local Area Networks (VLANs)

VLANs, standardized in IEEE 802.1Q [20], are a common solution to preventing data sharing specifically within large computing infrastructures such as data centers and mainframes, though sometimes used in HPC environments. VLANs work by tagging traffic on network switches and routers according to configurable tables of LAN membership matched with physical interface, with the intention of mimicking the configurability and security of LANs. Within the VLAN specification there lie two inherent flaw with VLANs to verifiability that meets our envisioned users' needs, as well as a few unresolved security flaws.

VLAN solutions are difficult to verify in two ways: *a*) the logic of VLAN technology is hidden within firmware which is expensive to analyze. As VLAN technology improves and becomes more complicated, this problem will only increase

in future versions of the systems [11][17]. *b*) VLAN hardware is commonly manufactured by international firms that may have pressure from outside governments to include secret backdoors in the firmware, further exacerbating the previous verification difficulty. Examples of this uncertainty can be seen in a special report by the U.S. House of Representatives from October 8th, 2012 [28].

Furthermore, due to backwards compatibility standards outlined in IEEE 802.1Q, the tagging mechanism of VLANs can be abused via a ”double-encapsulated 802.1Q / Nested VLAN” attack which works by placing two VLAN tags on a packet. By doing so, during certain situations it is possible for packets to ”escape” their VLAN designation and convey messages to hosts outside their configured VLAN [33].

Chapter 4

Design Goals

Chapter 3's discussion on security flaws within HPC environments shows that there exists an untouched niche for a software solution that replicates the security of physical user separation. Here we describe the requirements and goals in designing our solution to this problem.

A More Thorough and Intuitive Network Security

The gap between the security mechanisms discussed in Section 3 and the current approach of physical user separation is large. From manufacturing and political policy problems [28] to simple security flaws [33], current solutions are insufficient. For users in this domain to accept a software approach to network security in HPC environments, the proposed solution must be thorough, simple, intuitive and easily verifiable. The thoroughness of physical user separation is inherent in that it operates at the very lowest level of network operations, the physical layer. The closer our mechanism approaches this layer, the more thorough it will be considered.

Dynamic Control

Within HPC environments the type, number, and scale of jobs assigned to any of the systems can be widely varied. To handle this, the solution must be capable of receiving and modifying policy at the start of each new job. Furthermore, to manage performance tradeoffs a fine-grained control of the mechanism at higher resolution than job submission rate is desired.

Network Fabric Agnostic

Two major technologies are used to network HPC systems: Ethernet and InfiniBand. Any tool for improving security across the broad spectrum of HPC systems must be capable of operating in each. Further, numerous network topologies exist within these technologies; switched fabric and tree structures are the most common for InfiniBand and Ethernet, respectively. For our purposes we define this solution to be network fabric agnostic if it is conceptually capable of being implemented in either Ethernet or InfiniBand networks.

User Application Transparent

A fundamental requirement of the solution is that it be transparent to user applications. Applications written for HPC environments are often quite complex and it is likely that customers would be reluctant to make even minor modifications, especially to programs written in the past that are under re-use. For our purposes we define user application transparency as the ability to run an application without modification to successful end on an HPC system using our solution, given that it can also do so on a system not using our solution.

Chapter 5

Time Division Multiple Access of Shared Network Resources

Formal Definition

Before discussing our implementation of the solution, we first define an abstract definition that describes how the mechanism works beyond any specific implementation. The best way to describe this mechanism is through a language that represents the mechanism in operation. This language is defined formally by stating the grammar that generates it.

Suppose S is a finite set of security groups, s.t. each security group $s \in S$ is made up of a number of compute nodes. Given S, the language our mechanism operates on can be generated by the following grammar. Because the language is dependent on the security groups S, this grammar must be generated based on it. This is done in two steps:

First, we define the base grammar:

$$
\begin{aligned}
&G^1 = (V^1, \Sigma^1, R^1, \mathscr{A}), \text{ where} \\
&V^1 = \{\mathscr{A}, W\} && \text{non-terminal symbols} \\
&\Sigma^1 = \{\emptyset\} && \text{terminal symbols} \\
&R^1 = \{ \quad \mathscr{A} \to \varepsilon, && \text{rules of production} \\
&\qquad\quad \mathscr{A} \to W\mathscr{A} | W\}
\end{aligned}
$$

This base grammar, through the non-terminal symbols and production rules, establishes a means of generating the base language form of unordered windows ($W \in V^1$) in an arbitrary length such as WW or $WWWWW$.

Next, we generate the S specific definitions. To do so it is first necessary to

define notation for two special terminal symbols and three special sets:

$o_{s,i}$ - an open command issued to node i within security group s,

$a_{s,i}$ - an acknowledgement received from node i within security group s,

θ_s - the set of all $o_{s,i}$ terminals for security group s,

α_s - the set of all $a_{s,i}$ terminals for security group s, and

$\pi(A)$ - the set of all permutations of the set A.

These definitions allow us to define a final, special set:

$$\Lambda_s = \pi(\theta_s) \times \pi(\alpha_s)$$

Intuitively, Λ_s is a set of ordered sets expressing each permutation of θ_s matched with each permutation of α_s. For example, given a security group s made up of two elements s.t. $s = \{1,2\}$, Λ_s is defined:

$$\begin{aligned}\Lambda_s = \{&(o_{s,1}, o_{s,2}, a_{s,1}, a_{s,2}), \quad (o_{s,2}, o_{s,1}, a_{s,1}, a_{s,2}),\\ &(o_{s,1}, o_{s,2}, a_{s,2}, a_{s,1}), \quad (o_{s,2}, o_{s,1}, a_{s,2}, a_{s,1})\}\end{aligned}$$

The sets within Λ_s represent all legitimate command sequences within a window (W) for security group s. The key property of the sets within Λ_s is that each node within the security group is issued an open command, in any order, followed by acknowledgements from each node within the security group, once again in any order.

With these definitions established we can now formally define an S specific grammar:

$G^2 = (V^2, \Sigma^2, R^2, \emptyset)$, where

$V^2 = \{W\}$

$\Sigma^2 = \{[o_{s,i}, a_{s,i}] : \forall i \in \forall s \in S\}$

$R^2 = \{[W \rightarrow \lambda] : \forall \lambda \in \Lambda_s : \forall s \in S\}$

These definitions add new terminal symbols and the necessary production rules to generate them. Note the use of Λ_s in the production rules. These rules provide every possible command sequence possible for any window W s.t. every node issued an open command is required to report back with an acknowledgement before continuation onto another window.

Finally, the language our mechanism accepts for security group S can be formed using the union of the previous two grammars:

$G = (V, \Sigma, R, \mathscr{A})$, where

$V = V^1 \cup V^2$

$\Sigma = \Sigma^1 \cup \Sigma^2$

$R = R^1 \cup R^2$

Chapter 6

Implementation

As a proof of concept we have implemented a version of the tool for the Linux operating system using C++11. In this section we will describe the tool's architecture, operation, and how it adheres to the design goals from the previous section.

Overview

Algorithm 1 Window Controller opening and closing network access windows.

```
1:  function Open_Windows(Scheduler)
2:      Scheduler.initialize();
3:      while End_Command_Not_Received do
4:          Security_Group ← Scheduler.get_next_group();
5:          Security_Group.state ← STATE.OPEN;
6:          for each node ∈ Security_Group do
7:              send(node.address,
                    Security_Group.crypto_sign(COMMAND.OPEN));
8:              node.state ← STATE.OPEN;
9:          while Security_Group.state == STATE.OPEN do
10:             node_response ← block_on_receive_message();
11:             if node_response.state == STATE.CLOSED then
12:                 node.state ← STATE.CLOSED;
13:             else
14:                 throw ERROR.UNCLOSED_NODE;
15:             Security_Group.state ← STATE.CLOSED;
16:             for each node ∈ Security_Group do
17:                 if node.state == STATE.OPEN then
18:                     Security_Group.state ← STATE.OPEN;
```

The tool is composed of four major components: the window scheduler, ingress controller, egress controller, and the state controller. The window scheduler can be located on any administrative node within the system, preferably co-located with the system job scheduler. The remaining controllers are located throughout the

Algorithm 2 Node Control Mechanism opening and closing access to the network.

```
1:  function Node_Control_Mechanism
2:      Queue ← initialize_queue;
3:      while exit_command_not_received do
4:          state ← Close_Network_Access(Queue);
5:          while open_command_not_received do
6:              message ← block_on_receive_message();
7:          state ← Open_Network_Access(Queue);
8:          sleep(message.time);
9:          state ← Close_Network_Access(Queue);
10:         send_acknowledgement(state);

11: function Close_Network_Access(Queue)
12:     state.egress ← Network_Egress.enqueue(Queue);
13:     state.ingress ← Network_Ingress.drop_packets();
14:     return state

15: function Open_Network_Access(Queue)
16:     state.ingress ← Network_Ingress.accept_packets();
17:     state.egress ← Queue.process_packets_to_network();
18:     return state
```

HPC environment, with a copy on each compute node that is designated to execute user applications.

The system provides enhanced security by time dividing access to the network according to security groups. As jobs are scheduled on the system, the window scheduler must be informed of the intended location and their assigned security group. As jobs are run on compute nodes throughout the system the window scheduler communicates with the state controller on each node to designate time windows. During any individual time window only one security group has authorization to access the network. For any given time window in which a security group does not have access to the network, outgoing network packets are stored in a local queue while incoming packets are just ignored and deleted. The window scheduler

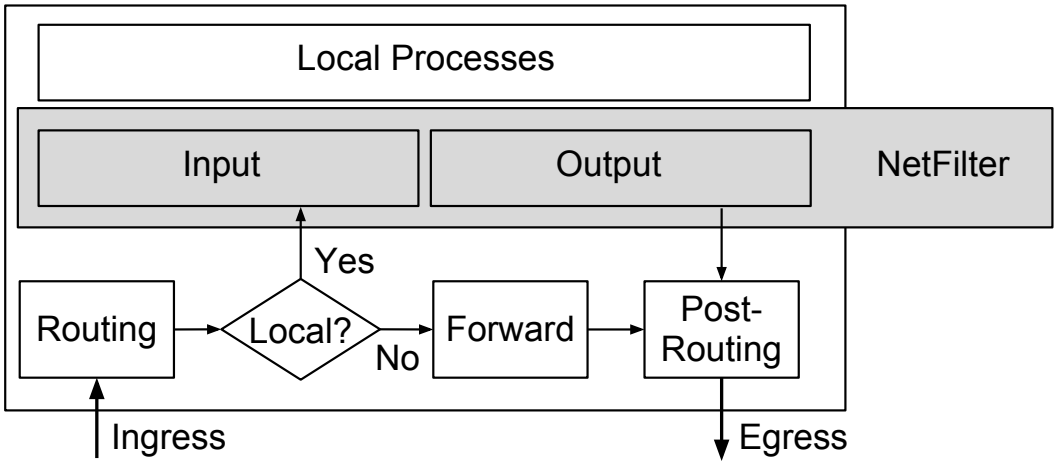

Figure 6.1: Data flow architecture of *iptables*, the packet filtering firewall with *NetFilter* located within the Linux kernel. The input and output ”chains” within *NetFilter* provide an interface for administrators to control and filter packets sent into user space.

is tasked with alternating time window authorization between security groups.

The following subsections describe each components operations in further detail.

State Controller

The state controller has three major tasks:

1. Securely send and receive communication with the window scheduler for the system
2. Transit both the ingress and egress controllers between states of network access and denial
3. Collect and store performance data on the egress queue's memory usage

Ingress Controller

The ingress controller is a firewall of incoming network packets and has two states:

1. Open access of network packets to applications on the node

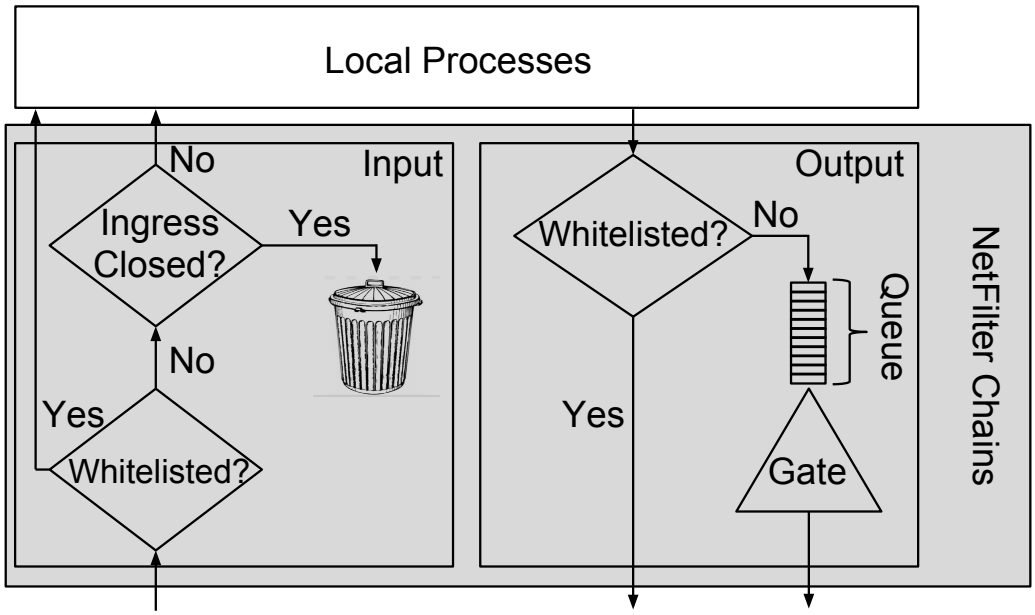

Figure 6.2: A detailed look at the logic within the *NetFilter* chains that makeup the Ingress and Egress controllers on compute nodes.

2. Closed to network packets except for those from explicitly allowed sources (a whitelist style of firewall)

During the open state incoming packets are processed normally. During the ingress closure, incoming packets, except those allowed by the whitelist, are dropped. The whitelist is designated to allow only necessary infrastructure communications such as Network Time Protocol (NTP), performance measurements, and especially packets from the window scheduler.

Egress Controller

The egress controller, similar to the ingress controller, has two major states:

1. Open flow of packets onto the network
2. Diversion of outgoing packets into a blocked queue

Window Scheduler

The window scheduler has three major tasks:

Determine system network access states for the next time window Validate the closure of the previous time window Communicate the next time window states to compute nodes To determine the network access states for the next time window the scheduler must run a scheduling algorithm on a few historical inputs. The base case scheduling algorithm is round robin (i.e., equal window size for each security group). To improve performance, a number of heuristics have been considered for the creation of a dynamic priority scheduling algorithm: the egress controller memory usage of compute nodes, number of TCP timeouts, and externally imposed priorities.

Chapter 7

Performance

Expected Values

MAKE SURE TO CITE NETPERF [14] Leading systems in high performance computing can serve as indicators of where commercial HPC systems will be in the upcoming years. The Titan system at Oak Ridge National Laboratories (ORNL) is one such system, the technical details of which are displayed in Table 7.1.

Here we attempt to quantify how the mechanism affects memory usage at the compute node level and effective bandwidth.

$T_{on,n}$ Time, within a window, where compute node n has access to the network.

T_{window} Length in time of a single window.

$\Pi_{app,n}$ Speed at which the application on compute node n generates network traffic.

$\Pi_{NIC,n}$ Speed at which the NIC on compute node n can transmit traffic onto the network.

Case Study

To verify and test the mechanism a test bed was created out of five Dell 1955 servers seen in Figure 7.7. These servers run Ubuntu server version 12.04 and had their network interfaces configured and connected according to Figure 7.6.

Technical Specifications	
CPUs	16 cores @ 2.2GHz
Main Memory	32GB @ DDR3

Table 7.1: Technical specifications of compute nodes within the Titan at ORNL.

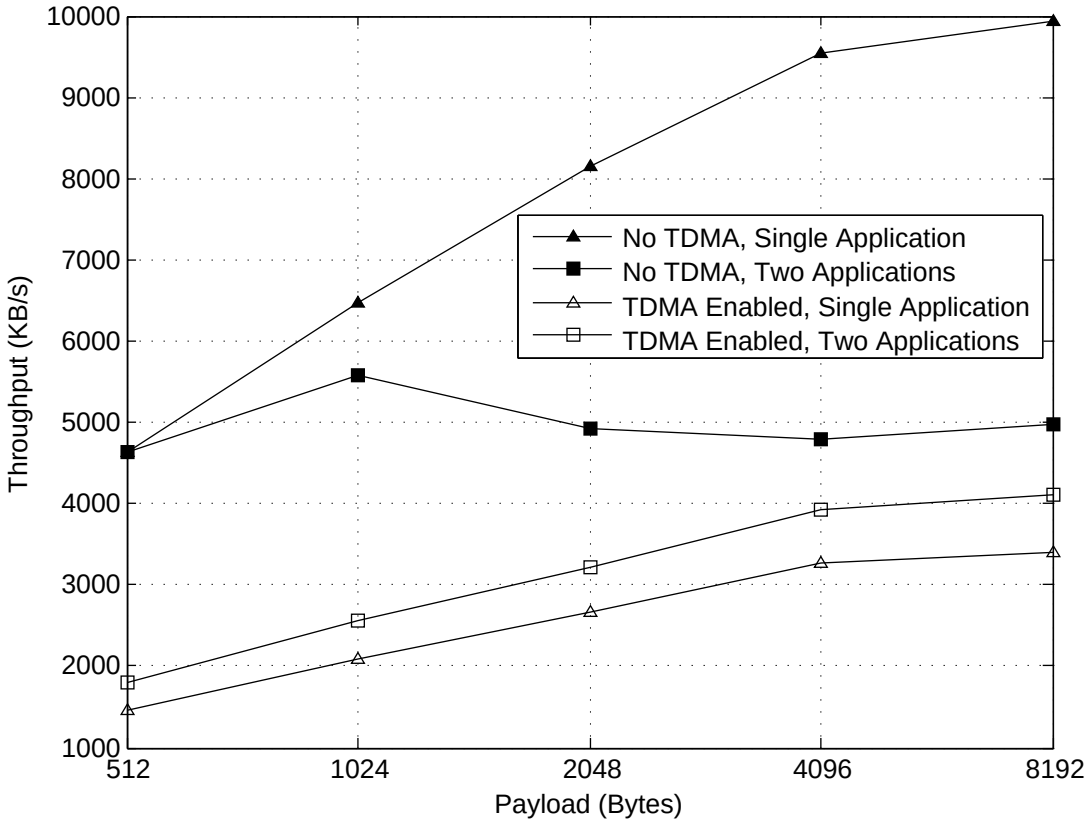

Figure 7.1: TDMA effect on TCP application performance and demonstration of payload size effect on network throughput.

To simulate job execution each node ran a program that sent ICMP (ping) commands at stochastic intervals and speeds to the other member other member of the security group. The mechanism, started and controlled at the control server (see Figure 7.7) alternated between network access for each security group at a time interval of one second of network access per group. A Wireshark (REF PACKET SNIFFING) trace was ran at the control server and the results of which can be seen in Figure 7.8 showing a history of packets sent through the network. The black entries represent communications originating from the control server, while the red and blue entries represent packets originating from nodes within the security group. [REMOVE MAKE IMAGE OF RANDOMIZED PINGING AND COM-

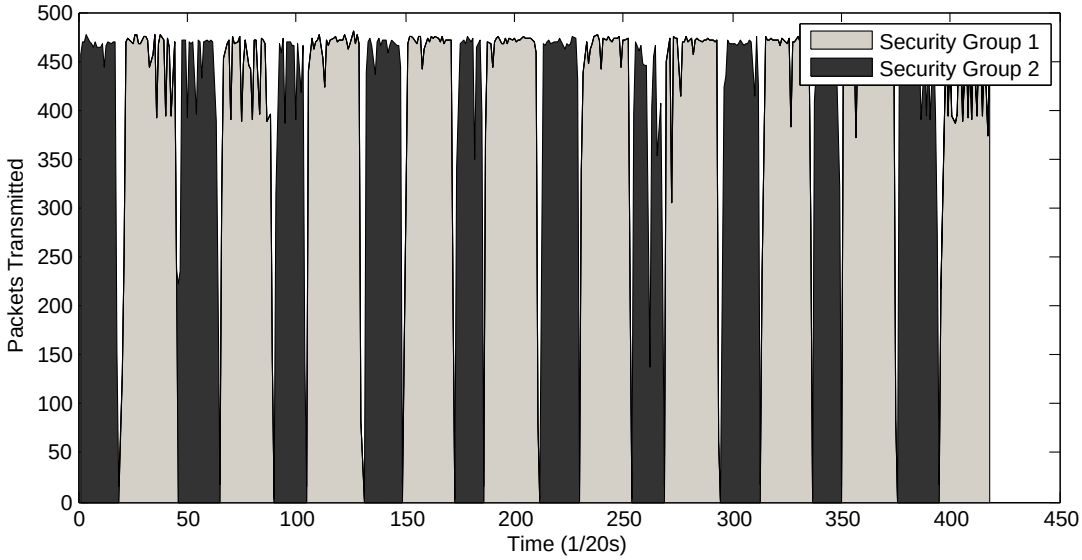

Figure 7.2: A trace of network traffic under performance testing while TDMA controls access.

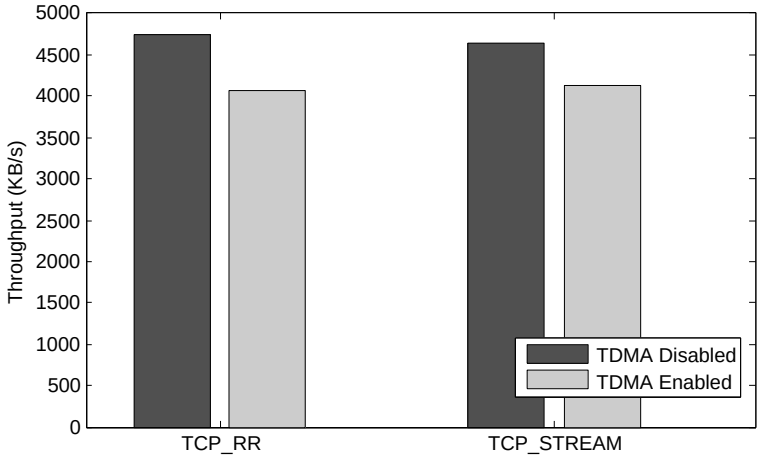

Figure 7.3: The impact of TDMA on TCP performance under two different 'netperf' tests.

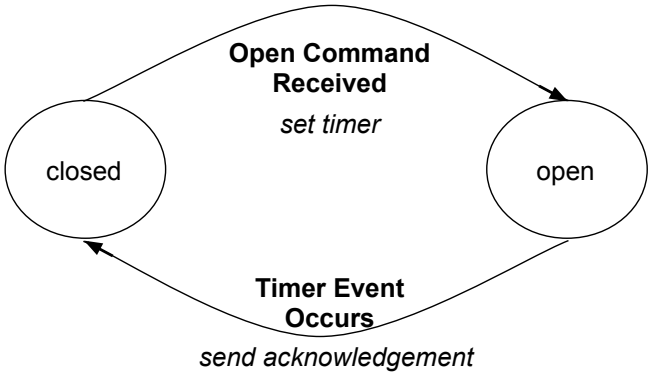

Figure 7.4: State diagram of a compute node.

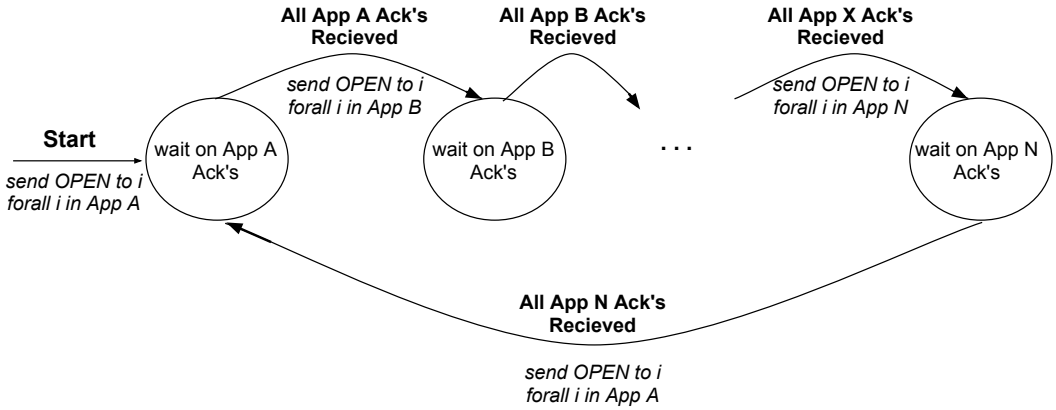

Figure 7.5: State diagram of an example window controller.

PARE THE BEFORE AFTER - MENTION THIS IS EGRESS INFORMATION, FIND WAY TO SHOW INGRESS.]

Security Validation

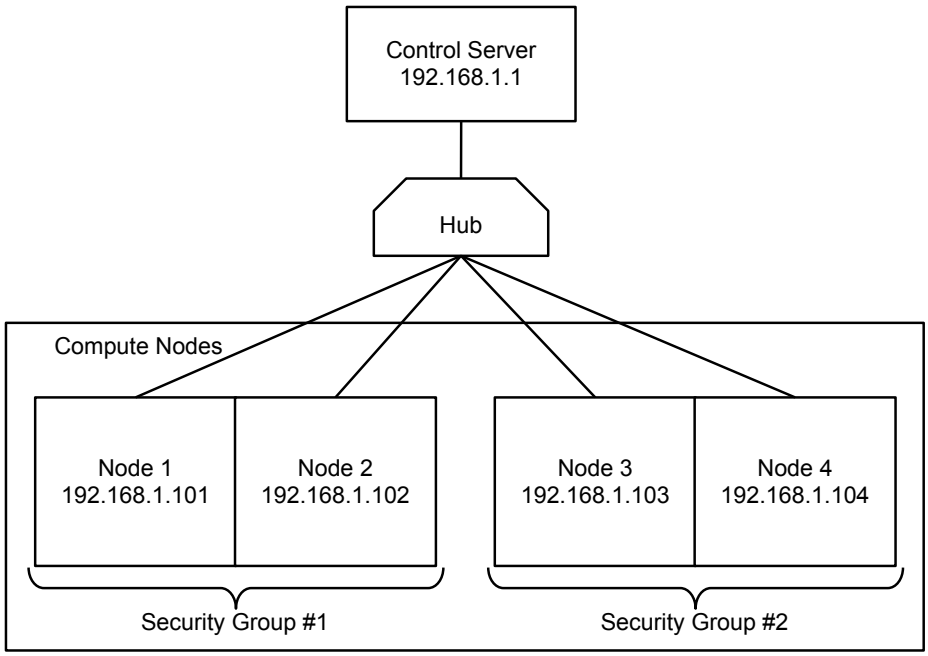

Figure 7.6: The network architecture of our demonstration test bed.

Figure 7.7: The TDMA test bed located in Impact Lab at Arizona State University.

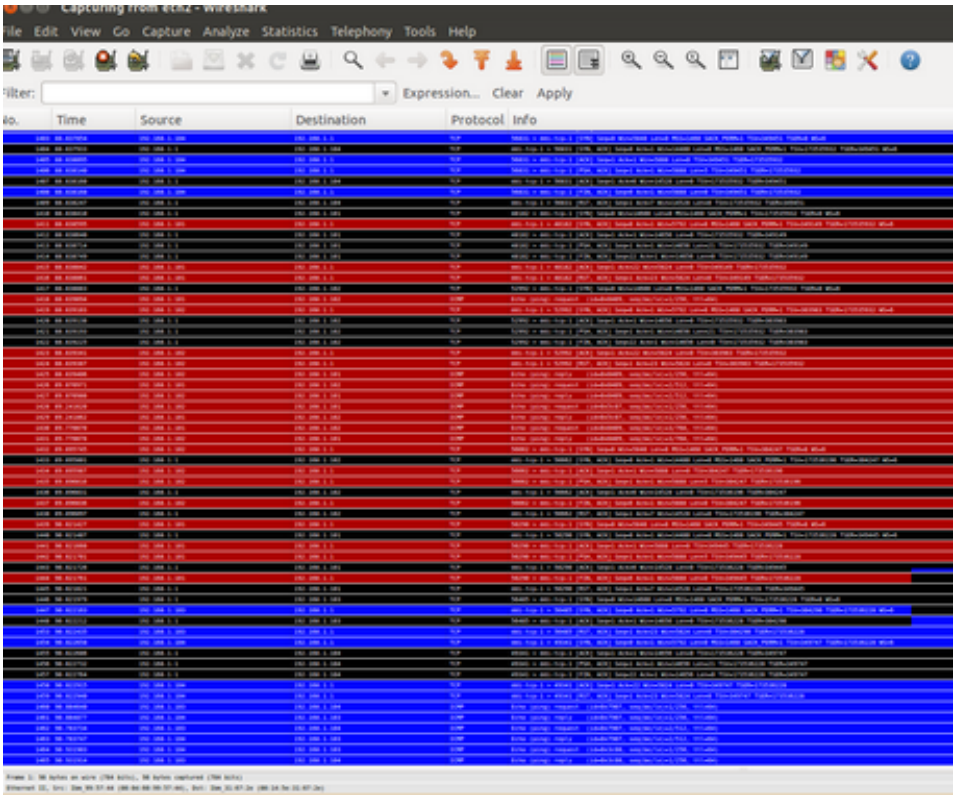

Figure 7.8: An example trace of the mechanism's time division property captured using the packet capturing application Wireshark. The colored records represent traffic based from compute nodes of two separate security groups - denoted as red and blue.

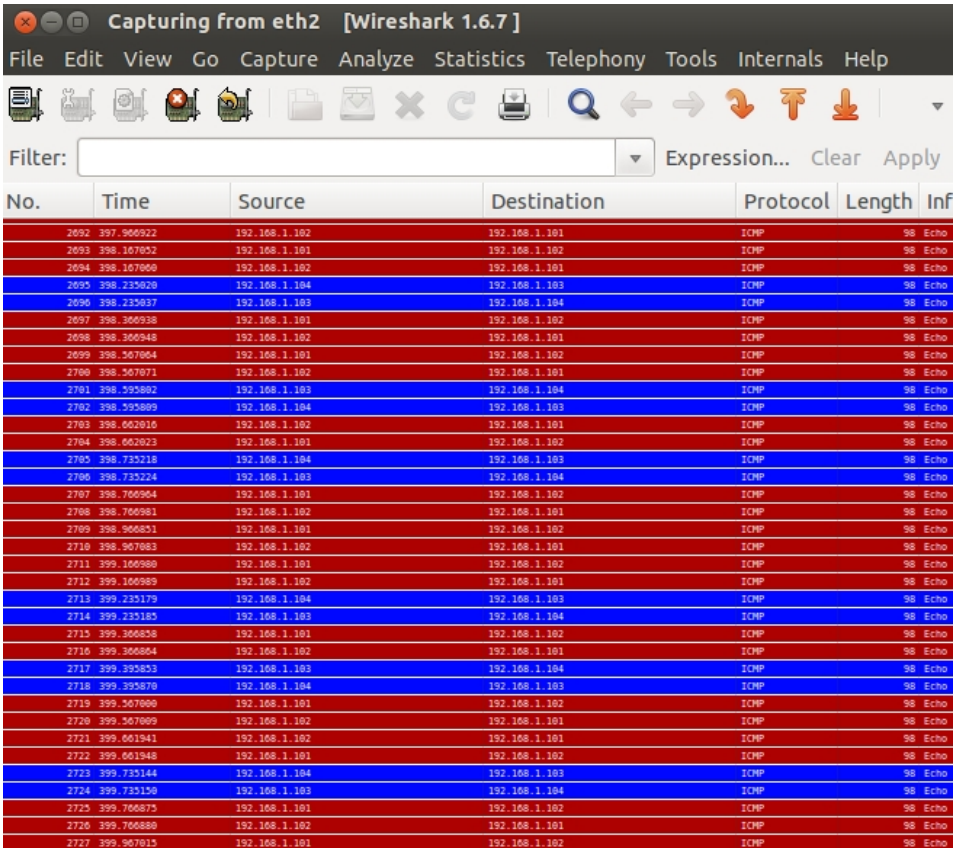

Figure 7.9: An example trace of applications operating without TDMA.Note the interweaving of traffic from the different security groups - denoted as red and blue.

Chapter 8

Conclusion

The problem of enforcing authenticated access to resources in an HPC environment is a difficult one. Further, creating a solution that is intuitive and secure enough to convince highly sceptical stakeholders of its security narrows available options. This thesis presents a method of solving this problem that is both intuitive and verifiably secure, in real time. The worst case performance penalty of utilizing this mechanism is shown to be within bounds of reasonable expectations. Practical performance is expected to minimize the overhead of the mechanism.

Further Work

Implement dynamic scheduling algorithm on the window scheduler using memory usage statistics from compute nodes.

Bibliography

[1] United States National Security Administration. Inline media encryptor. `http://www.nsa.gov/ia/programs/inline_media_encryptor/index.shtml`, January 2009.

[2] Bill Allcock, Joe Bester, John Bresnahan, Ann L Chervenak, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, Steven Tuecke, and Ian Foster. Secure, efficient data transport and replica management for high-performance data-intensive computing. In *Mass Storage Systems and Technologies, 2001. MSS'01. Eighteenth IEEE Symposium on*, pages 13–13. IEEE, 2001.

[3] V.S. Arackal, B. Arunachalam, MB Bijoy, BB Prahlada Rao, B. Kalasagar, R. Sridharan, and S. Chattopadhyay. An access mechanism for grid garuda. In *Internet Multimedia Services Architecture and Applications (IMSAA), 2009 IEEE International Conference on*, pages 1–6. IEEE, 2009.

[4] Julian Borrill, Leonid Oliker, John Shalf, and Hongzhang Shan. Investigation of leading hpc i/o performance using a scientific-application derived benchmark. In *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–12. IEEE, 2007.

[5] M. Bozzo-Rey, M. Jeanson, M.N. Nguyen, C. Gauthier, M. Barrette, P. Vachon, K. Gaven-Venet, H.Z. Lu, S. Allen, and A. Veilleux. Design, deployment and bench of a large infiniband hpc cluster. In *High-Performance Com-*

puting in an Advanced Collaborative Environment, 2006. HPCS 2006. 20th International Symposium on, pages 8–8. IEEE, 2006.

[6] Robert G Brown. Engineering a beowulf-style compute cluster. *Duke University Physics Department*, 2004.

[7] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*, volume 1. Prentice Hall, Upper SaddleRiver, NJ, USA, 1999.

[8] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications*, 23(3):187–200, 2000.

[9] G.N. Cohen, B. Kamenel, and C.M. Kubic. Security for integrated ip-atm/tactical-strategic networks. In *Military Communications Conference, 1996. MILCOM '96, Conference Proceedings, IEEE*, volume 2, pages 456 –460 vol.2, oct 1996.

[10] J. Dean. Designs, lessons and advice from building large distributed systems. Presented at Large-Scale Distributed Systems and Middleware (LADIS), 2009.

[11] Jean-Yvesc Emery, Cédricc Vandenweghe, Brunoc Thery, et al. Security management process of at least one vlan of an ethernet network, April 13 2011. EP Patent 2,073,455.

[12] Marc Horowitz and Steve Lunt. Ftp security extensions. Technical report, RFC 2228, October, 1997.

[13] D.B. Jackson. On-demand access to compute resources, April 7 2006. US Patent App. 11/279,007.

[14] Rick Jones, Karen Choy, and David et al. Shield. netperf - a network performance benchmark. `http://www.netperf.org/netperf/`. Version 2.5.0.

[15] R.H. Katz, G.A. Gibson, and D.A. Patterson. Disk system architectures for high performance computing. *Proceedings of the IEEE*, 77(12):1842 –1858, 1989.

[16] A. Keller and A. Reinefeld. Anatomy of a resource management system for hpc clusters. *Annual Review of Scalable Computing*, 3(1):1–31, 2001.

[17] Raymond Kloth. Derived vlan mapping technique, March 27 2001. US Patent 6,208,649.

[18] Argonne National Laboratory. The ibm blue gene/p supercomputer installation at the argonne leadership computing facility. `http://en.wikipedia.org/wiki/File:IBM_Blue_Gene_P_supercomputer.jpg`, December 2007.

[19] N. Leavitt. Big iron moves toward exascale computing. *Computer*, 45(11):14–17, 2012.

[20] IEEE Local, Metropolitan Area Network Standards Committee, et al. Virtual bridged local area networks, 1998.

[21] B. Madai and R. Al-Shaikh. Performance modeling and mpi evaluation using westmere-based infiniband hpc cluster. In *Computer Modeling and Simulation (EMS), 2010 Fourth UKSim European Symposium on*, pages 363–368. IEEE, 2010.

[22] Kenneth G Mages and Jie Feng. Method of secure server control of local media via a trigger through a network for instant local access of encrypted data on local media, April 6 1999. US Patent 5,892,825.

[23] University of Chicago. Globus toolkit homepage. `http://www.globus.org/toolkit/`, 2013.

[24] Department of Defense. Dod manual 5200.01-v3 dod information security program: Protection of classified information, February 2012.

[25] Department of Defense Defense Security Service. Clearing and sanitization matrix as of: June 28, 2007, 2007.

[26] University of Oslo. Nordugrid homepage. `http://www.nordugrid.org/`, 2013.

[27] Clemens Pfeiffer. Ein cray-1, aufgenommen im deutschen museum, mnchen. `http://en.wikipedia.org/wiki/File:Cray-1-deutsches-museum.jpg`, November 2006.

[28] Mike Rogers and Dutch Ruppersberger. Investigative report on the u.s. national security issues posed by chinese telecommunications companies huawei and zte, October 2012.

[29] Thomas Sandholm, Peter Gardfjäll, Erik Elmroth, Lennart Johnsson, and Olle Mulmo. An ogsa-based accounting system for allocation enforcement across hpc centers. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 279–288. ACM, 2004.

[30] Alex Schenck. Picture of a beowulf cluster. `http://en.wikipedia.org/wiki/File:Beowulf.jpg`, January 2008.

[31] Hongzhang Shan, Katie Antypas, and John Shalf. Characterizing and predicting the i/o performance of hpc applications using a parameterized synthetic benchmark. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 42. IEEE Press, 2008.

[32] Hongzhang Shan and John Shalf. Using ior to analyze the i/o performance for hpc platforms. In *Cray Users Group Meeting (CUG)*, pages 7–10, 2007.

[33] Cisco Systems. Vlan security white paper - cisco catalyst 6500 series switches. `http://www.cisco.com/en/US/products/hw/switches/ps708/products_white_paper09186a008013159f.shtml`, 2002.