ARCHITECTURAL SUPPORT FOR ENHANCING SECURITY IN CLUSTERS

A Dissertation

by

MAN HEE LEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2008

Major Subject: Computer Engineering

ARCHITECTURAL SUPPORT FOR ENHANCING SECURITY IN CLUSTERS

A Dissertation

by

MAN HEE LEE

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Eun Jung Kim
Committee Members,	Valerie E. Taylor
	Riccardo Bettati
	A. L. Narasimha Reddy
Head of Department,	Valerie E. Taylor

August 2008

Major Subject: Computer Engineering

ABSTRACT

Architectural Support for Enhancing Security in Clusters. (August 2008)

Man Hee Lee, B.E., Kyungpook National University;

M.E., Kyungpook National University

Chair of Advisory Committee: Dr. Eun Jung Kim

Cluster computing has emerged as a common approach for providing more computing and data resources in industry as well as in academia. However, since cluster computer developers have paid more attention to performance and cost efficiency than to security, numerous security loopholes in cluster servers come to the forefront. Clusters usually rely on firewalls for their security, but the firewalls cannot prevent all security attacks; therefore, cluster systems should be designed to be robust to security attacks intrinsically.

In this research, we propose architectural supports for enhancing security of cluster systems with marginal performance overhead. This research proceeds in a bottom-up fashion starting from enforcing each cluster component's security to building an integrated secure cluster. First, we propose secure cluster interconnects providing confidentiality, authentication, and availability. Second, a security accelerating network interface card architecture is proposed to enable low performance overhead encryption and authentication. Third, to enhance security in an individual cluster node, we propose a secure design for shared-memory multiprocessors (SMP) architecture, which is deployed in many clusters. The secure SMP architecture will provide confidential communication between processors. This will remove the vulnerability of eavesdropping attacks in a cluster node. Finally, to put all proposed schemes together, we propose a security/performance trade-off model which can precisely predict perfor-

mance of an integrated secure cluster.

To my wife, Eunyoung, and my daughters, Grace and Hannah, and my mother

ACKNOWLEDGMENTS

I would like to earnestly thank my advisor, Dr. Eun Jung Kim, for her guidance, encourgement, patience, and her willingness to discuss almost everything during my graduate studies. Especially in a series of rejections from various conferences, she kept encouraging me by saying that she trusts me. Additionally, I highly appreciate her sincere effort to have a close friendship with her students; the numerous coffee breaks and lunches that she provided invigorated me to resume my research at full throttle. I also wish to thank Dr. Valerie E. Taylor, Dr. Riccardo Bettati, and Dr. A. L. Narasimha Reddy for their continual support and constructive comments in the preliminary and final examinations. I am also grateful to Dr. Ki Hwan Yum, assistant professor in the Department of Computer Science at the University of Texas at San Antonio and Dr. Kim's husband, for carefully reading and commenting on so many revisions of all submissions to conferences and journals.

I would like to thank all of the members in Dr. Kim's High Performance Computing Laboratory including Hogil Kim, Yuho Jin, Heungki Lee, Inchoon Yeo, Lei Wang, Chih-chun Liu, and Sungho Park for their friendship and for useful discussions and questions in our office and lab meetings. Especially, I am extremely grateful for the critical assistance and help I received from Minseon Ahn and Baiksong An while I worked on the second and third research topic, respectively.

I am also very grateful for a close friendship with Dr. George C. Davis while he was a professor in the Department of Agricultural Economics at Texas A&M University. He volunteered to be a conversation partner helping international students improve their verbal skills in English, and his church, Grace Bible Church, assigned him to me by His Grace. The four-year long conversation meetings with him helped me significantly in English; he often corrected my pronunciation word by word. More

importantly, he soon became a great mentor to me who gave me precious advice on diverse dimensions of my life as Ph.D. student, as a husband, as a Christian, and as a professor hopefully in the future.

Acknowledgements are also extended to all my fellow Christians in Korean Church of A&M. With their support and prayer, I lived a fulfilling life and found His purpose of my life. Furthermore, I am so blessed to have close fellowship with Pastor Sunyeop Lee, Soolyeon Cho, Gieseung Lee, Kwanghyun Song, Jongmin Oh, and Byunghak Kim. They touched my heart so deeply that they changed my life and thought greatly. Especially, my cousin, Heejin Lim, and her husband, Sangeun Kim, showed me a great example of a sincere Christian and faithful deacon in a church. With their precious four children, they were a pure God's gift to my family.

I would like to thank my family. First of all, none of this would have been possible without the love, patience, encouragement, and support of my wife, Eunyoung Park. Quitting her job, raising two kids in a foreign country, and spending countless lonely nights must be a great challenge for her, but she sacrificed herself a lot, understood my situation, and raised two daughters gracefully as well as managed her time wisely to go to a college for a degree. I deeply appreciate all her works done for our family. I would like to thank my two lovely daughters, Grace and Hannah. Even though they cannot realize how critical they are to my study, they are a real source of true happiness and strength to me. Even when I was completely exhausted, the two girls' smile and sweet kisses recharged me with full energy like magic. I honestly feel privileged to be a father of such adorable daughters, God's pure blessing. My thanks also go to my mother, Leeja Ahn, who showed me true love and sacrifice as well as to my late father, Dongyoung Lee, who loved us so much with his humble mind. I am also grateful to my sisters and brother, Sunhee, Kyunghee, and Moohee, who always took care of and backed me in many ways from when I was born as the last child.

I give special thanks to Eunyoung's parents, Daeyong Park and Eunsook Jung, who allowed me to marry their wonderful daughter and keep praying for us everyday.

Lastly and most importantly, I give all my thanks to God. He saved me through Jesus Christ and gave me a vision to study abroad. In addition to making it possible, He was so careful in seeing us that He provided wisdom, ideas, people, and money necessary at every corner of this journey, so nothing fell short. He miraculously turned all my triumphant and sad moments into good things in the end, fulfilling "In all things God works for the good of those who love him, who have been called according to his purpose." (the Apostle Paul's letter to the Romans, chapter 8, verse 28, New International Version of the Bible). I thank and praise God again because He will love, take care of, and keep an eye on us in the rest of my life as always.

My study was partially funded by the Ministry of Information and Communication, Republic of Korea, through IT Student Scholarship from 2003 and 2004, the Department of Computer Science, Texas A&M University, through graduate teaching, non-teaching, and research assistantship from 2005, and through industrial affiliates program (IAP) scholarships in 2003 and 2007.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Widespread use of cluster systems in a diverse set of applications has spurred significant interest in designing such servers, considering performance, scalability, and Quality-of-Service. However, cluster computer developers have paid more attention to performance and cost efficiency than to security. As a result, numerous security loopholes in cluster servers come to the forefront and consequently the design of secure clusters has recently surfaced as a critical issue. Generally, any computer security needs to offer three basic services: confidentiality, authentication, and availability. Confidentiality refers to restricting access to data sent by a sender only to a designated receiver. Authentication allows the receiver to make sure that the received message is authentic, not modified or forged. Availability refers to providing the timely and reliable accesses to data and information services for authorized users [1]. Clusters usually rely on *firewalls* to defend against security attacks from the outside, not providing any security measures within clusters. However, such protecting systems themselves are vulnerable to security attacks [2, 3] and, more realistically, weak legitimate user passwords can be an open door to cluster systems at any time [4]. Therefore, it is desirable to design secure clusters which are robust to security attacks intrinsically.

The easiest way to block hackers is to isolate clusters physically as some government and military agencies do but this cannot be a general solution. Enforcing functionalities of firewalls can be a common method but it is out of scope of this research. Security of cluster systems can be enhanced in many other ways. For exam-

---

The journal model is *IEEE Transactions on Automatic Control.*

ple, using secure OS like SELinux, removing buffer overflows of server applications, or partitioning cluster nodes into several groups logically with different root privileges can improve cluster security. Among many possible approaches, we mainly focus on encrypting and/or authenticating communicating data. This approach is helpful especially when cluster interconnects which are previously used to connect enclosed cluster nodes are extended to connect external systems located outside a cluster as demontrated in Supercomputing Conference 2005 where multiple clusters and storage systems were connected through a cluster interconnect [5]. In this environment, the cluster nodes are liable to security attacks through the extended cluster interconnects. Encryption and authentication will prevent eavesdropping of data and passwords and block unauthorized access to cluster nodes which are exposed to outside of a cluster.

Its major challenge is to minimize performance degradation while providing security transparency at the same time. Software or OS based security enhancements cannot accomplish this goal because such schemes will deteriorate the overall cluster performance by taking up a great amount of host processors' computing resources. Furthermore, if OS or a user application has to be involved in security provisioning deeply, significant code modification should be made, especially causing great difficulties to legacy applications. Therefore, our main focus lies in designing secure cluster communication to be as transparent to applications as possible by integrating additional hardware and by amending cluster interconnect protocols. The more transparent security will help our schemes to be adopted to clusters more seamlessly. In addition, hardware-supported security measures can incur less performance overhead since most security operations will be done in hardware, not consuming host processors' computing power.

Major components we will consider are cluster interconnect protocols, cluster interconnects cards (CICs), and computing nodes with multiple processors. This

proposed research proceeds in a top-down fashion starting from proposing a secure cluster design to enforcing each component's security.

First, for secure cluster protocols, a handful of research groups have tried to enhance cluster security in [6, 7, 8, 9, 10, 11]. However, a comprehensive approach covering all three security services was not proposed. Especially, to the best of our knowledge, there has been no previous studies investigating a comprehensive framework for security enhancement in InfiniBand Architecture (IBA), a promising I/O communication standard positioned for building clusters and System Area Networks (SANs) [12]. In this research, we will point out security vulnerabilities of IBA and its potential threats. Then we will present an authentication and encryption method to remove those vulnerabilities with marginal performance degradation. Furthermore, we will propose a mechanism to block denial of service (DoS) attacks in IBA.

Second, in order to provide high speed security services, it is necessary to implement security functionalities in CIC. Moreover, recent clusters are capable of user-level communication (ULC), allowing user applications to bypass OS for communications, so most cluster communications are already being processed by CIC. This makes it almost impossible for host processor(s) to encrypt/decrypt or authenticate all cluster communications without substantial performance degradation. To solve this problem, we propose a CIC architecture enabling high performance secure clusters by means of security hardware in CIC.

Third, as shared memory multiprocessor (SMP) systems are expected to be more common in cluster systems, the communication between processors also needs to be secure for a cluster node's high-level security. A small body of literature already investigated secure multiprocessor designs [13, 14] focusing on bus-based SMP systems; however, there has been no encompassing study taking into account other types of SMPs with different network topologies. Our research proposes an Interconnect-

Independent Security Enhanced Shared Memory Multiprocessor Systems (I^2SEMS) for inter-processor communication security. By integrating an additional cache in each processor and a system-wide encryption/decryption controller, I^2SEMS will keep all data outside processors encrypted. Since we decouple the design of I^2SEMS from processor interconnecting topology, I^2SEMS can be applied to any types of SMP systems.

CHAPTER II

SECURITY BACKGROUND

Since we will address classical security services, in this section we briefly introduce the security services and common approaches related to our ideas.

A. Confidentiality

Confidentiality refers to restricting access to the data sent by a sender only to the designated receiver. Encryption and decryption will be needed to enforce confidentiality. Encryption scrambles the original plaintext into ciphertext using an encryption key; while decryption recovers the original plaintext from the ciphertext using a decryption key. If the two keys are identical, this is called a symmetric or secret key mechanism. Otherwise, it is called an asymmetric or public key mechanism, where anyone can encrypt a message using a public key, but only the person with the private key can decrypt it [15].

Block ciphers and stream ciphers are two basic symmetric encryption schemes. Among them we chose to use the block cipher because block ciphers usually operate on large blocks of data while stream ciphers do it on individual plaintext digits with time-varying transformation.

In our research, we use Advanced Encryption Standard (AES) as a basic block cipher due to its security strength and recent research on fast hardware implementation [16]. Block ciphers can operate in several modes. The most basic mode is Electronic Code Book (ECB). In ECB, a plaintext is split into several blocks and each block is encrypted separately. With a fixed key and n-bit block cipher, this mode is a sort of code book to map an n-bit string to another n-bit string. ECB mode using AES is decribed in the following equation when AES and AES^{-1} are

encryption and decryption functions, respectively.

$$\begin{aligned} ciphertext &= AES(plaintext, Key) \\ plaintext &= AES^{-1}(ciphertext, Key) \end{aligned}$$

Since it does not need any initial vector, it has no initialization overhead. In addition, it is parallelizable in that each encryption solely depends on the input plaintext and the key. However, ECB is a little vulnerable because it does not hide traffic patterns. Even though encrypted data is still secure, the patterns may be further exploited by a hacker. To solve this problem, feedback or chaining modes like Cipher Block Chaining (CBC), Output Feed-Back (OFB), and Cipher Feed-Back (CFB) are used. The result of the previous block is fed-back or chained into the next block. While it has the initialization overhead and the parallelization restriction, it can successfully hide patterns in the original plaintexts.

Note that a main limitation of the above methods especially like ECB is that the whole AES function is in the critical path because the function can start only after the plaintext or the ciphertext is available.

Counter (CTR) mode can solve the problem since the input of encryption functions is not a plaintext nor a ciphertext, but just a counter. An encryption function encrypts this counter to generate an encrypted counter, often called a *keystream*[*]. This keystream is XORed with a plaintext to make a ciphertext. Decryption should use the same counter to generate the same keystream. This keystream is XORed with the ciphertext to recover the original plaintext. Since CTR mode does not have any dependence, parallelism or pipelining can improve cipher operation throughput. Its

[*]Also known as one-time-pad (OTP) in previous literature.

operation is decribed in the following equation.

$$\begin{aligned} chipertext &= plaintext \oplus AES(counter, Key) \\ plaintext &= chipertext \oplus AES(counter, Key) \end{aligned}$$

The counter does not need to be secret because, even though an attacker gets both a ciphertext and its counter, it is practically impossible to recover its original plaintext without the secret key. The biggest benefit of this scheme is that AES latency can be out of the critical path because it is possible to precompute keystreams even when plaintexts or ciphertexts are not available. However, it is imperative to use distinct counters for different plaintexts to prevent security vulnerabilities. Suppose E_1 and E_2 are two different ciphertexts using the same counter C. If a hacker gets these two ciphertexts and XOR them, he will get partial information of two original plaintexts by clearing out the keystream, as described in the following.

$$\begin{aligned} E_1 &= D_1 \oplus AES(C, Key) \\ E_2 &= D_2 \oplus AES(C, Key) \\ E_1 \oplus E_2 &= D_1 \oplus D_2 \end{aligned}$$

B. Authentication

Authentication allows two parties to agree that a received message is authentic, not modified or forged. Message Authentication Code (MAC) is commonly used. Since MAC is created and verified with the same secret key, communication parties should share the key beforehand. Among several MAC schemes, keyed hash functions and

block cipher-based MACs are widely used. In the keyed hash MAC, a hash function maps a variable size message into a fixed length of digest. By digesting the message along with a secret key, the hash function makes an authentication tag. If the receiver has the same secret key, the receiver can make the same authentication tag. In the block cipher-based MAC, the authentication tag is the output of the last cipher block after encrypting the whole message. Well-known keyed hash MACs are HMAC-MD5 and HMAC-SHA1 [17, 18]. CBC-MAC and PMAC are examples of block cipher-based MAC [19, 20, 21, 22].

GCM is a block cipher mode of operation that encrypts and authenticates messages at the same time, and also called authenticated encryption [23]. Detailed application of GCM in our research are in Chapter V.

C. Availability

Availability refers to the "timely and reliable access to data and information services for authorized users" [1]. Availability has become more important in the Internet because of the increase in DoS attacks. Early DoS attacks dumped huge numbers of packets on a specific target system greatly slowing down the system and its network [24]. While recent DoS attacks staged by worms or viruses often do not target a specific system, they are still tying up infected or attacked systems and network resources [25, 26, 27]. The defense against DoS attacks is processed as follows: First a monitoring function detects a DoS attack; Second, when source addresses are spoofed, a source identification function, if available, tracebacks true attacking systems; Third, an access control is then applied to block the DoS attacks. In our research, we will propose an efficient blocking scheme to block a type of DoS attacks in IBA.

CHAPTER III

SECURE INFINIBAND CLUSTER*

A. Introduction

Computer clustering is a popular trend in academia as well as in the industry for high performance and high availability computing. Widespread use of cluster systems in a diverse set of applications has spurred significant interest in designing such servers, considering performance, scalability, and Quality-of-Service. However, cluster computer developers have paid more attention to performance and cost efficiency than to security. As a result, numerous security loopholes of cluster servers have been revealed and consequently the design of secure clusters has recently surfaced as a critical issue.

The easiest way to protect clusters is to isolate them physically, like some government and military agencies do, but this cannot be a general solution. Instead, reinforcing firewalls can be a common method, but it is well known that firewalls cannot prevent all possible threats. First of all, firewalls are useless to prevent inside attacks [28]. Also, firewalls scan only certain layers; therefore, there always exist potential attacks using upper layers. In addition, firewalls themselves have security vulnerabilities or can be mis-configured to operate improperly. The followings are such examples. Geer reported that several products of well-known security companies such as Check Point Software Technologies, Symantec, and Zone Labs had potentially dangerous flaws that could allow hackers to gain control of systems, dis-

*Reprinted with permission from "A Comprehensive Framework for Enhancing Security in InfiniBand Architecture" by Manhee Lee and Eun Jung Kim, 2007. IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 18, No. 10, pp. 1393-1406, Copyright 2007 IEEE.

able computers, or cause other problems [2]. Wool quantified configuration errors of firewalls installed on several sites. He found that many sites had serious configuration errors and, for example, almost 80 percent of firewalls allowed *any* service and insecure access to firewalls, which are not desirable for high security [3].

Besides improving the strength of firewalls, security of cluster systems can be enhanced in many other ways. For example, using secure operating systems (OS) like SELinux, preventing buffer overflows of server applications, or partitioning cluster nodes into several logical groups with different root privileges can improve cluster security. Among many possible approaches, we focus on secure communication by encrypting/authenticating communication data. Protecting communication data prevents potential eavesdropping on data and passwords and unauthorized accesses to cluster nodes. This approach will help to build secure cluster systems especially when cluster interconnects, which are previously used to connect enclosed cluster nodes, are being extended to connect external systems located outside the cluster, as demonstrated in Supercomputing Conference 2005 where multiple clusters and storage systems were connected through a cluster interconnect [5]. In this environment the cluster nodes are liable to security attacks through the extended cluster interconnects.

We can provide secure communication in two ways: software-based or hardware-based. However, software-based security enhancements cannot be proper solutions in cluster systems where high performance is a primary goal because such schemes will deteriorate the overall cluster performance by taking up a great amount of computing resources of host processors. Furthermore, if the OS or a user application needs to be deeply involved in the security, significant code modifications are necessary, especially posing great difficulties to legacy applications. However, hardware-supported security measures imposes little performance overhead since most security operations will be

done in hardware, not consuming host processors' computing power. In addition, the secure cluster communication provided in the cluster interconnect protocol level will be transparent to applications, resulting in seamless adoption of secure communication. Therefore, our challenge is to minimize performance degradation and maximize security transparency while providing secure communication in cluster.

There are several widely used cluster interconnects. Myrinet, proposed in 1995, is a switching network with low latency cut-through switches [29]. As of November 2006, 15.8% of the Top 500 supercomputers use Myrinet. Quadrics has been the interconnect of choice for high-end supercomputers. As of November 2006, 2.8% of supercomputers are using products from Quadrics. Following the success of Ethernet as a local area network, Gigabit Ethernet is making another success in high performance computing area; currently, 42.6% of supercomputers are using Gigabit Ethernet. InfiniBand Trade Association, an industry consortium, is leading the specification of IBA [12]. IBA is a promising I/O communication standard positioned for building system area networks that are used in clusters, multiprocessor systems, and storage area networks. User-level communication mechanisms in IBA improve overall system performance significantly by reducing the communication overhead of the operating system in message transfer. Among these cluster interconnects, we investigate security of IBA since it has potential security vulnerabilities due to lack of security considerations in the current specifications. In addition, an increasing number of sites are constructing IBA cluster systems. As highlighted in the recent Top 500 supercomputer list, the number of IBA cluster has been doubled from 36 to 78 in six months. This trend is expected to continue for IBA to overtake Myrinet.

The main contribution of this study is to investigate the following security issues for providing a comprehensive framework for security enhancement in IBA clusters. First, we point out security vulnerabilities of IBA and its potential threats. By sim-

ulating a DoS attack inside an IBA cluster, we show that DoS attacks can degrade clusters' communication performance by up to two orders of magnitude. Second, we present secret key management schemes which can be tightly coupled with the existing IBA key management and describe how to distribute the keys in IBA securely. Third, we introduce the scalable adoption of an authenticated encryption scheme, Galois/Counter Mode (GCM), into IBA with only minor modifications to the IBA specifications [23]. Finally, for better availability, we propose a stateful ingress filtering to block one kind of DoS attack with invalid IBA Keys and scalable packet marking algorithms to identify the location of the attackers. Since our scheme filters ports related to only active DoS attacks instead of all ports, it incurs no performance overhead in a normal situation. This dynamic filtering scheme is further enhanced by making it robust to spoofing attacks with fake source addresses. Our scalable packet marking algorithms can trace real attackers regardless of routing algorithms in huge regular networks.

We use a comprehensive cycle accurate simulation testbed for IBA [30]. Simulation results of a 16-node IBA network show that security performance overhead due to encryption/authentication on network latency is 12.4% for 64 byte long packets. When the packet length increases, the overhead decreases to as low as 0.7% with 1024 bytes long packets.

The remainder of the chapter is organized as follows: We briefly introduce related work and IBA in section B and C. In section D, we elaborate on security vulnerabilities of the current IBA after introducing security background. Section E presents a comprehensive framework to enhance IBA security, and its security and performance are analyzed in section F and G, followed by the concluding remarks of this study in section H.

B. Related Work

There has been some research to improve security inside clusters. Yurcik *et al.* introduced a new concept called emergent security properties to identify security characteristics unique to clusters, which can be used to develop a unified monitoring tool for clusters [6]. Pourzandi *et al.* proposed a new security model called the distributed security infrastructure [8]. It supports a fine-grained cluster-wide security enforcement on distributed applications by providing a process-level resource and access control [7]. Foster *et al.* proposed a communication library allowing programmers to communicate securely in geographically distributed computing environments [9], which suitably provides well-organized security in Grid. Connelly and Chien focused on incorporating confidentiality in remote procedure calls in tightly coupled applications. They applied traditional security functions such as transposition, substitution, and data padding on the marshalling layer [10]. Their performance analysis showed that encryption can be used in clusters with a minimal performance impact. Dimitrov and Gleeson presented security enhancement methods at three levels: network host interfaces, system area networks, and communication protocols [11]. Their approach is systematic enough to be considered as a guideline for enhancing security of Myrinet- or VIA-based clusters, but it is not compatible with the most recent IBA specification. Overview of various cluster interconnects and their security issues are well described in [31].

C. InfiniBand Architecture

In this section we introduce some features of InfiniBand architecture relevant to our security enhancement in IBA. IBA is a high-speed switched interconnect connecting end nodes such as processor nodes, I/O units, or routers to build clusters and system

area networks. An IBA consists of one or more subnets and each subnet is a set of end nodes, switches, and its common subnet manager (SM). A channel adapter (CA), similar to a network interface card, connecting a processor node or an I/O node to the IBA network is called a host channel adapter or a target channel adapter. One or more ports in a CA can be connected to the IBA fabric through serial links. IBA links provide various speeds from 2.5 Gbps up to 120 Gbps.

Queue-Pair (QP) is a pair of work queues, a send queue and a receive queue, for a consumer (or a process). When a consumer wants to send data to another consumer, a send request is queued up to the send queue and its CA executes the request by getting data from the user memory and sending them to the other consumer's receive queue. According to connection and acknowledgment modes, a QP is categorized as one of five types: reliable connection, unreliable connection, reliable datagram, unreliable datagram, and raw datagram. While a connection-oriented QP can communicate with its connected QP only, a datagram QP can communicate with multiple datagram QPs. A reliable QP provides the acknowledgment service guaranteeing the in-order communication without error or duplication. Since the raw datagram is designed to interact with non-IBA components, it is not our focus.

Among many features of IBA, *Management*, *Partition*, and *Key* are closely related to our design for security enhancement in IBA. First, an SM manages a subnet by configuring and managing routers, switches, and CAs in the subnet. Each node should have a subnet management agent to communicate with the SM. Second, IBA Keys are used to provide isolation and protection. Currently five types of IBA Keys are specified as follows: Management Key (M_Key), Baseboard Management Key (B_Key), Partition Key (P_Key), Queue Key (Q_Key), and Memory Key (L_Key and R_Key). The usage of each Key is summarized in [32].

A packet carries one or more IBA Keys and the receiving CA compares the

delivered Keys with the stored Keys in the CA. If the two are identical, the packet can access the node's resource, but if not, the packet is discarded. Third, a partition is defined as a collection of CA ports that can communicate with each other to provide the exclusive resource sharing. Partition is managed by a partition manager, but since the partition manager is usually a function of SM, in this paper we consider that the SM manages partitioning.

D. Security Threat in IBA

In this section, we describe assumptions of our research and point out IBA security vulnerabilities and their potential attacks which motivated our research.

1. Basic Assumptions

Before identifying and solving security vulnerabilities of IBA, we assume the following constraints:

- CA/switch/router can be compromised.
- each of those devices has a tamper-resistant storage, the contents of which cannot be read or modified from outside the device. The contents are only accessible by its subnet manager with a legitimate key.
- attacking traffic can have spoofed source addresses.

2. Vulnerabilities and Threats

a. Confidentiality

Since IBA does not provide any encryption and decryption method, eavesdropping attacks with help from compromised switches and CAs will succeed in acquiring

all communicating data. We call this *Type I attack*. This vulnerability is critical especially when a cluster processes classified data.

b. Authentication

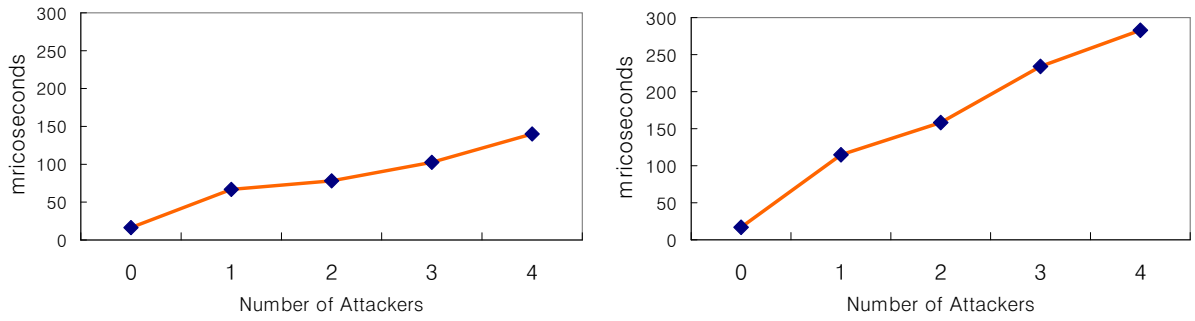
A resource to be protected has its own Key such as P_Key or L_Key and R_Key. Any captured packets which carry legitimate Keys will expose these plaintext Keys and allow a hacker to generate illegal traffic with the Keys, referred to as *Type II attack*. If this illegal traffic is using fake source addresses, it will be hard to find which port the attacker is attached to. We classify this attack as *Type III attack*.

c. Availability

An attacker who does not even know any legitimate Key can stage a DoS attack by sending tremendous traffic to a victim node. Although the traffic will be discarded at the victim nodes because of the illegal Key, the traffic can affect other normal traffic when it passes through the network. We define this attack as *Type IV attack*.

3. A Simulation of a DoS Attack

To show the impact of the DoS attack, we simulate a DoS attack on an IBA testbed. We partition 16 nodes of the IBA network into four partitions and choose attacker nodes randomly. The attacker nodes randomly select victim nodes and generate illegal traffic at their full speed (2.5 Gbps, 1X IBA link). Other non-attacker nodes communicate with each other in the same partition. Our simulation uses two kinds of traffic: real-time and best-effort. Real-time traffic is injected at a fixed rate and has a higher priority than best-effort traffic in intermediate switches. By using best-effort attacking traffic, we can see its effect on real-time traffic and on other best-effort traffic.



(a) Real-time traffic (b) Best-effort traffic

Fig. 1. Average end-to-end latency under DoS attacks. We simulate DoS attacks on our IBA testbed to estimate their effect on the overall network performance. The latencies of real-time and best-effort traffic are increased by 6 and 18 times, respectively. This shows that the number of attackers increases, the average network latencies will increase significantly.

Fig. 1(a) shows how real-time traffic is affected by illegal best-effort traffic. With no attacker, the average end-to-end latency is around 20 μs. The figure clearly shows that the latency could increase by as much as 50 μs when even one attacker dumps traffic. As more attackers are added, the average latency increases up to 120 μs. Fig. 1(b) shows how illegal best-effort traffic degrades other legal best-effort traffic. Its average latency increases more dramatically than that of real-time traffic. This is because real-time traffic has a higher priority than best-effort traffic.

E. Security Enhancement in IBA

Security vulnerabilities are inevitable as long as IBA packets continue to carry plaintext IBA Keys for their access control. To remove such vulnerabilities, we propose that IBA use traditional cryptographic methods. Our main goal is to integrate encryption and authentication into IBA not only with minor modifications to the IBA specification but also with only marginal performance overhead. For this, new cryptographic keys in addition to IBA Keys are necessary. Therefore, we first propose new

key management schemes to manage and distribute the keys. Based on these key management schemes, we describe how GCM can be integrated into IBA and then propose an efficient mechanism to block DoS attacks and a scalable packet marking scheme to trace back DoS attacks which use fake source addresses.

1. Secret Key Management in IBA

To provide any security service, it is mandatory to have an efficient and secure key management scheme. Since the existing key management of the current IBA is not designed for security, we propose two new key management schemes: partition-level and QP-level key managements. The partition-level key management enforces that all communications inside a partition use the same secret key. This scheme ensures that data communications are secure against attacks by a hacker located in a different partition. This is simple to implement because each CA only has to maintain the same number of secret keys as partition keys. However, this scheme is vulnerable to security attacks originating from the same partition. To remove this problem, we propose to use the QP-level key management for finer-grained security. Since QP is the smallest communication entities, the QP-level key management will guarantee confidentiality and integrity for all QP communications. Both management schemes can be adopted into IBA without significant changes in the specification. We choose the symmetric key mechanism for better performance.

a. Partition-Level Key Management

In this key management, a secret key will be created and assigned to each partition. All components in each partition like switches and CAs should have its secret key. The secret key is used to make an MAC and encrypt every packet transferred within the partition. Even though P_Keys are still available in packets, the exposure of

the P_Keys does not pose any further threats since the encryption/authentication relies on its secret key, not on the P_Keys. Therefore, even if an attacker captures IBA packets, he cannot make legitimate packets with the captured P_Keys because the attacker does not know their secret keys. This partition-level key management removes the aforementioned Type I and II attacks coming from different partitions.

There are several advantages of this scheme. First, every IBA packet should carry a partition key while other IBA Keys may not be carried, so it is easy to enforce security throughout the network. Second, since the number of partitions are often small, the number of secret keys will not be so large, thus requiring a small amount of space for the key management. Last, this key management makes use of the well-defined key management mechanisms of IBA by adding one more column into the existing Key tables. The current IBA key management, therefore, can successfully manage new secret keys along with its IBA Keys. However, this key management has a disadvantage in that it cannot block attacks originating from the same partition. This is because CAs automatically decrypt and authenticate incoming packets as long as the packets have the same P_Key as the CA.

b. Queue Pair-Level Key Management

In order to remove aforementioned threats inside the partition, we alternatively propose Queue Pair (QP)-level key management scheme; two connection-oriented QPs share a temporary secret key, referred to as a *session key*, and a connectionless QP has its own secret key that can be sent to other connectionless QPs. There are two benefits by implementing the QP-level key management. One is that even in the same partition Type I and II attacks are not viable without a legitimate QP secret key. The other benefit is that it also removes additional threats resulting from the exposure of other IBA Keys. For example, even if an R_Key is exposed to a hacker, he cannot

access the corresponding remote memory without the QP's secret key. Therefore, QP-level key management provides fine-grained key management enough to secure all communications in IBA.

c. Initial Key Distribution

To use the aforementioned key management schemes, it is necessary to distribute secret keys securely. We first define two basic keys, a CA secret key (CA_SK) and an SM secret key (SM_SK), on which subsequent key distributions depend. A CA_SK is a unique secret key assigned to each CA by its manufacturer and stored in each CA's tamper-resistant storage.† A manufacturer passes this key offline, assumed to be secure, to an IBA cluster administrator. Therefore, since the administrator has all secret keys of CAs in the cluster, he can access any CA using the CA's secret key. The other basic key, SM_SK, is assigned to each subnet by the administrator and stored in SM's CA. Then, the SM distributes its secret key to all CAs in its subnet as follows:

(1) SM sends SM_SK to all CA_i in the subnet:

$E_{CA\_SK_i}(SM\_SK)$

(2) CA_i decrypts SM_SK:

$SM\_SK = D_{CA\_SK_i}(E_{CA\_SK_i}(SM\_SK))$

Note that MAC is not described for simplicity and $E_K(T)$ and $D_K(T)$ represent that a plaintext, T, is encrypted and decrypted by a secret key, K, respectively. Since the SM_SK is encrypted using the receiving CA's secret key, the distribution is secure. The decrypted SM_SK is stored in each CA. One concern in storing the SM_SK in

†All the following secret keys will be stored in this storage, too.

CAs is that a non-SM node might use the SM_SK inappropriately. Therefore, each CA needs to be hardwired so as not to decrypt or authenticate normal packets using the SM_SK. If there are multiple subnets in an IBA network, each subnet has its own SM_SK, and SMs should communicate securely with each other. For this purpose, the administrator defines a shared SM secret key, SSM_SK, to be shared by all SMs.

d. Partition-Level Key Distribution

In the partition-level security, a partition secret key, P_SK, is created and distributed as follows:

(1) SM sends P_SK to all CAs in the subnet:

$E_{SM\_SK}(P\_SK)$

(2) CA_i decrypts P_SK:

$P\_SK = D_{SM\_SK}(E_{SM\_SK}(P\_SK))$

When a partition is created across several subnets, one SM transfers the P_SK to other SMs to distribute it inside their own subnets.

(1) SM sends P_SK to all SMs in other subnets:

$E_{SSM\_SK}(P\_SK)$

(2) SM_j decrypts P_SK:

$P\_SK = D_{SSM\_SK}(E_{SSM\_SK}(P\_SK))$

(3) SM_j sends P_SK to CA_i in the subnet:

$E_{SM_j\_SK}(P\_SK)$

(4) CA_i decrypts P_SK:

$P\_SK = D_{SM_j\_SK}(E_{SM_j\_SK}(P\_SK))$

From now on all communications inside the partition can be encrypted and authenticated by P_SK.

e. Queue Pair-Level Key Distribution

For QP-level security, there are two secret key setup processes depending on the type of communication. When a consumer wants a connection-oriented communication, the source CA creates a session key randomly and then sends a request message (REQ) containing the session key to a destination CA. If the destination CA accepts the request, the CA creates a connection-oriented QP and decrypts the session key and then replies with a reply message (REP) containing the destination QP's information. The session key is used to encrypt and authenticate all the following messages between the two QPs.

To protect the session key distribution, we assume every pair of CAs in a partition has a unique secret key. Let CA_SK_{ij} be a secret key for communication between CA_i and CA_j and $\text{CA\_SK}_{ij} = \text{CA\_SK}_{ji}$. Therefore, an REQ from CA_i to CA_j is encrypted using CA_SK_{ij}. The SM generates all n-to-n keys and distributes the keys to all CAs beforehand. Based on this secret key, the session key distribution is described as follows:

(1) CA_i sends an REQ message to CA_j containing a session key, SK_l:

$\text{REQ}(\ .\ .\ .\ , \text{E}_{CA\_SK_{ij}}(\text{SK}_l))$

(2) CA_j decrypts SK_l:

$\text{SK}_l = \text{D}_{CA\_SK_{ji}}(\text{E}_{CA\_SK_{ij}}(\text{SK}_l))$

(3) CA_i sends data to CA_j: $\text{E}_{SK_l}(\text{Data})$

(3)' CA_j sends data to CA_i: $\text{E}_{SK_l}(\text{Data})$

After step (2), CA_i and CA_j can send data using SK_l at any time in step (3) and (3)'.

In contrast to a connection-oriented QP, a datagram QP can communicate with more than one QP. To control accesses to this QP, each datagram QP carries the destination QP's Q_Key and an access to this QP is allowed only if the Q_Key in the requesting packet is the same as the one in the receiving QP. Therefore, before two datagram QPs send data, they need to exchange packets to know each other's Q_Key. As noted earlier, the Q_Key in the packet is also plaintext, so it is vulnerable to the capturing attack. To prevent this problem, we propose a secret key be assigned to each Q_Key and exchanged. The following explains a reliable datagram QP's communication establishment steps. In an unreliable datagram communication, SIDR_REQ and SIDR_REP are used instead of REQ and REP.

(1) CA_i sends an REQ message to CA_j containing a session key, SK_l:

REQ(..., Q_Key_{local}, $E_{CA\_SK_{ij}}(SK_l)$)

(2) CA_j decrypts SK_l:

$SK_l = D_{CA\_SK_{ji}}(E_{CA\_SK_{ij}}(SK_l))$

(3) CA_j sends an REP message to CA_i containing another session key, SK_r:

REP(..., Q_Key_{remote}, $E_{CA\_SK_{ji}}(SK_r)$)

(4) CA_i decrypts SK_r:

$SK_r = D_{CA\_SK_{ij}}(E_{CA\_SK_{ji}}(SK_r))$

(5) CA_i sends data to CA_j:

E_{SK_r}(Data), Q_Key_{remote}

(5)' CA_j sends data to CA_i:

E_{SK_l}(Data),Q_Key$_{local}$

The space to store a 128 bits session key in request and reply packets is available in the PrivateData field in the packet formats. The space overhead for storing 1-to-n secret keys, CA_SK_{ij}, in each CA is 128 bits times the number of CAs in its partition. If a partition consists of 2^{10} CAs, the total space is 20K bytes including 32 bits MAC for each secret key. If the tamper-resistant storage cannot hold all secret keys, they can be stored in normal memory in encrypted form.

The initial overhead for generating and distributing all CA_SKs is analyzed as follows. Those secret keys are random numbers and they can be generated by the AES function in CTR mode [33]. In a partition with 2^{10} CAs, the total number of SA_SKs is 2^{18}. Since a recent hardware implementation of AES can encrypt at 30$\sim$70 Gbps, it can generate all the secret keys within a second, assuming that the length of a secret key is 128 bits long. An SM needs to distribute the all secret keys sequentially. If an IBA cluster has 2.5 Gbps link, assuming a 1024 bytes long packet is carrying a secret key, it will take also less than one second to send out all the packets. Considering fairly long operation time of cluster systems, our approach is scalable to even larger networks since such a small initial overhead will be easily amortized.

2. Message Encryption and Authentication in IBA

Now, we will explain where an MAC is to be located in each packet and show an example of how an authenticated encryption algorithm, GCM, is integrated into IBA.

Cyclic Redundancy-Check (CRC) codes are widely used for error detection on data communication. IBA defines three types of CRC: Invariant CRC (ICRC), Variant CRC (VCRC), and Link Packet CRC (LPCRC) as shown in Fig. 2 (Only ICRC and VCRC are depicted). ICRC covers all invariant fields from Local Route Header

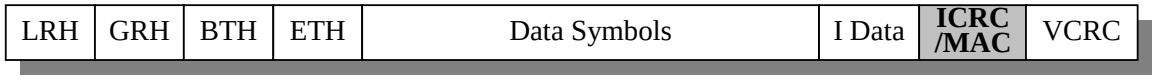


Fig. 2. IBA packet format with Message Authentication Code. ICRC covers all invariant fields from LRH to I Data to detect errors on the data communication. Since ICRC does not change end-to-end, we propose to use this field as the MAC location that will carry MAC in the security enhanced IBA (LRH: Local Route Header, GRH: Gobal Route Header, BTH: Base Transport Header, ETH: Extended Transport Header, ICRC: Invariant CRC, VCRC: Variant CRC).

(LRH) to I Data, not covering variant fields that switches or routers can change. In contrast, VCRC covers the LRH up to the last byte before the VCRC. If some fields are changed in a switch or a router, the VCRC is calculated again. LPCRC is present at the end of all Link packets. The only Link packet in the current IBA specification is the flow control packet, so we do not consider LPCRC.

Our main idea is to use the ICRC field as the MAC location depicted in Fig. 2. We can gain the following two advantages in using the ICRC field. First, ICRC does not change from end to end. It only covers static fields, so it can be considered as a transport-level CRC. Since the authentication is an end-to-end transport feature, the ICRC field is a nice fit for the MAC location. The second advantage is that we do not have to change the IBA packet format, which is extremely important. By having ICRC as a default and MAC as an option, the original drivers that use ICRC do not have to be changed. Therefore, our idea will become fully compatible with the current IBA.

To accommodate various authentication algorithms, *Reserved* field of Base Transport Header (BTH) is used to identify authentication algorithms. BTH is a basic header that every transport packet carries. If the value is zero, the packet is using the original ICRC. Non-zero value indicates that an authentication algorithm is in use and a MAC is stored in the ICRC field. This can be exploited to provide an on-

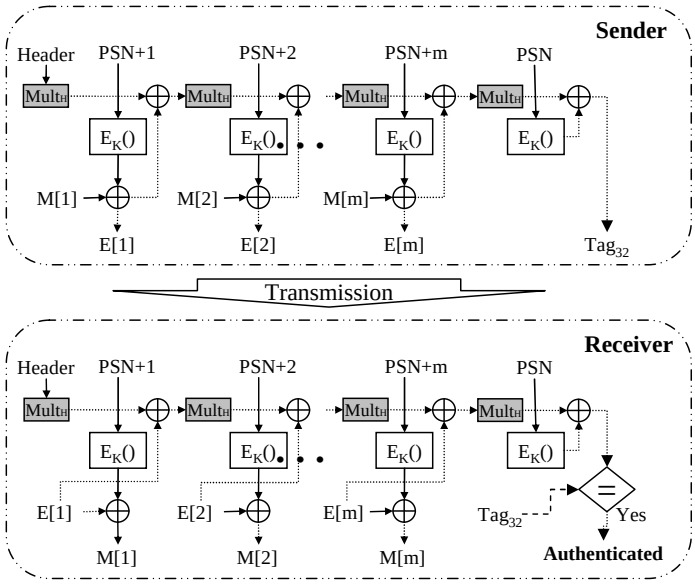
demand authentication service. For instance, suppose in a partition a very important job is running. The administrator can enable authentication for that partition by using MAC instead of ICRC. Since the authentication can be disabled and enabled at any time, our mechanism provides the flexible authentication service.

Although any encryption/authentication algorithm can be used for secure IBA, it will be very helpful to take one algorithm and show its performance because we can estimate the performance impact of our approach in real systems. We choose GCM since it has two main advantages in terms of security and speed. Security strength of GCM is the same as the strength of its block cipher [34, 23]. Since we use AES in this research and it is considered to be secure without any serious weakness until now, the confidentiality of each IBA packet can be improved greatly. With duplicate hardware, the whole authentication can be done in parallel in several additional cycles after AES computation [23].

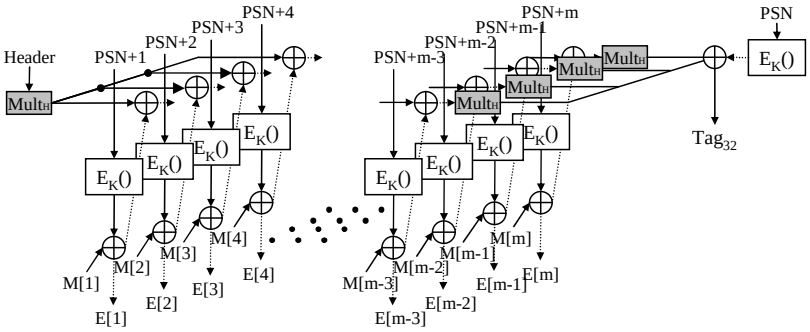
Fig. 3 depicts the architecture of GCM to encrypt and authenticate packets using AES. Packet Sequence Number (PSN) and its sequential numbers are encrypted and XORed with plaintexts to generate ciphertexts. The header of a packet and ciphertexts are multiplied in GF (Galois Filed) to generate an MAC. This architecture can fully utilize the pipelined AES as shown in Fig. 3(a). The parallelism can be further exploited by the use of the multiple encryption units depicted in Fig. 3(b). 12X IBA network can be extended in the same way. Note that the only Tag calculation requires the previous results. The performance overhead coming from this delay will be analyzed in Section G.

3. Stateful Ingress Filtering

Until now, we described how to provide confidentiality and authentication services in IBA. In this section we explain how to mitigate the DoS (Type IV) attack problems



(*a*) 1X GCM



(*b*) 4X GCM

Fig. 3. GCM architecture using AES. In the sender side, an original data, M, is divided into multiple 128 bits long messages, $M[1]$, $M[2]$,..,$M[m]$. This substring is encrypted by XORing the result of encryption of PSN+i. The first input of $Mult_H$ is the header of the packet and subsequent inputs are encrypted substrings, $E[i]$. Note that due to the pipelining capability of recent AES and $Mult_H$ hardware, adjacent substrings are encrypted and authenticated in parallel with a small pipelining delay. Receiver's architecture is almost similar except that $E[i]$ and $M[i]$ are switched. 4X GCM utilizes four AES hardware to keep up with 4X InfiniBand speed. Since there is no dependence between AES blocks, it can get nearly linear speed-up.

Table I. Partition enforcement overhead

	DT	IF	SIF
one switch mem	$r \cdot p \cdot k$	$\frac{n}{s} \cdot p \cdot k$	$\frac{n}{s} \cdot Min(p', p) \cdot k$
all switches mem	$r \cdot p \cdot s \cdot k$	$n \cdot p \cdot k$	$n \cdot Min(p', p) \cdot k$
Look-up/pkt	$h \cdot f(p)$	$f(p)$	$Pr(n) \cdot f(Min(p', p))$

for better availability.

This vulnerability stems from the fact that IBA Key checking is done in destination nodes. A straightforward remedy is to make all switch ports filter out packets carrying illegal Keys. We call this method Duplicate Table (DT) scheme. However, this is very inefficient in terms of memory usage and network performance because every switch port has to maintain a partition table and all packets should be checked at every hop. For example, when a network consists of n nodes and s switches with r ports and all nodes join the same number of p partitions and k is the length of a partition key in bytes, each switch will need $r \cdot p \cdot k$ bytes to store partition tables, which means $r \cdot p \cdot s \cdot k$ bytes will be used in total for the whole network as shown in Table I. Let $f(i)$ be the time that a table look-up function, f, takes to search an entry from a table having i entries. DT will take $f(p)$ at every hop, thus requiring $h \cdot f(p)$ to route a packet to a destination with h, an average of hop counts.

To remove high overhead of DT, Ingress Filtering (IF) commonly used on the Internet can be adopted in IBA. In IF, the packet filtering is applied to ingress ports that are directly connected to nodes. If all invalid P_Key packets are filtered out at ingress ports, it is not necessary to check at intermediate hops. Therefore, IF can increase the memory and network efficiency by using less memory and taking shorter time as shown in Table I where $\frac{n}{s}$ is the average number of nodes directly connected

to per switch.

Still, however, there are redundant operations in IF because it is necessary to look up P_Key tables in all ingress ports regardless of whether a DoS attack is occurring or not. To remove such redundancy, we propose Stateful Ingress Filtering (SIF) that filters attacking traffic only when attacks are active. To decide when and where to enable filtering, we propose to use a *trap* message. In IBA, when an incoming packet's Key does not match the receiver's Key, the receiver optionally sends a trap message to its SM. We suggest this trap message convey the invalid Keys and the packet's source address. When the SM receives the trap message, it will find the switch port connected to the attacking node and enables the filtering of the port.

Note that this filtering needs invalid Keys, not valid Keys. Since the current IBA switches have a table of valid P_Keys designed for the partition enforcement, we introduce a table of invalid P_Keys, referred to as *Invalid_P_Key_Table*. When the SM receives a trap message, it registers the invalid P_Key to its Invalid_P_Key_Table. To disable this filtering when attacks end, we define *Ingress P_Key Violation Counter* in switch ports. It counts the number of invalid P_Keys sent from the filtered node. If the Ingress P_Key Violation Counter does not increase for some time, the contents of Invalid_P_Key_Table will be flushed and the ingress filtering will be disabled. Consequently, since SIF will be activated only when a node is injecting packets with invalid P_Keys and only where attacking traffic is coming from, it will eliminate all redundant filtering operations of DT and IF.

One concern is that the Invalid_P_Key_Table might grow bigger than the valid P_Key_Table as an attacker uses many invalid P_Keys. In this case, to prevent the long table look-up time, the ingress filtering needs to look up P_Key_Table to pass legitimate traffic with valid P_Keys. The Ingress P_Key Violation Counter still needs to be counted because if it does not change for some time, the switch's port has

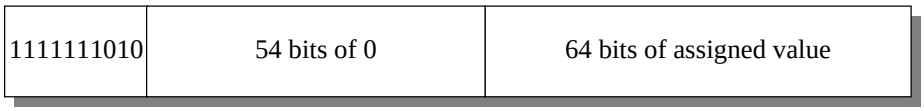
to return to the normal operation. When disabling filtering, it is necessary to flush Invalid_P_Key_Table and reset Ingress P_Key Violation Counter. Due to this, any attempts to stage a DoS attack on the SM by triggering multiple trap messages will also be blocked soon. This is because once the number of invalid P_Keys becomes larger than the valid P_Key_Table, packets only with valid P_Keys will be passed.

SIF's overhead depends on how often DoS attacks are occurring and how many invalid P_Keys are used for the attacks. Let $Pr(n)$ be the probability that one node joins a P_Key attack and let p' be the number of invalid P_Keys used in the P_Key attacks. The memory overhead for one switch is $\frac{n}{s} \cdot Min(p', p) \cdot k$ bytes and the table look-up operation will be $Pr(n) \cdot f(Min(p', p))$ where $Min(p', p)$ is the minimum between p' and p. By limiting the maximum p', the memory overhead will be smaller than or equal to that of DT or IF. Furthermore, considering that DoS attacks are not occurring often, the low $Pr(n)$ will make the table look-up overhead of SIF negligible.

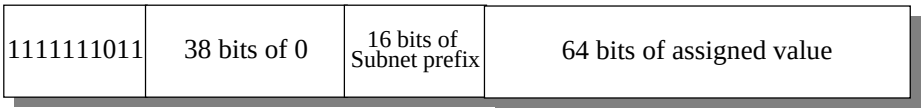
4. Source Identification Scheme in IBA

One more concern to implement the SIF is that a compromised node may use fake source addresses, referred to as *spoofing*. In this case, the SIF would block wrong ports. Since IBA allows software to choose which source address will be used in an outgoing packet, spoofing is possible if a hacker successfully manipulates the address information in a CA.

To remove this problem, a source identification scheme capable of tracing back the real attackers is necessary. In [35], Aljifri categorized traceback methods into four categories: link testing, logging, ICMP-based traceback, and packet marking. Since link testing and ICMP-based traceback should generate additional traffic, they will degrade the overall performance of cluster. Logging requires additional storage and computing power in routers/switches, which are not desirable due to additional costs.



(a) Link-local address format



(b) Site-local address format

Fig. 4. GRH address format. GRH can be in use both within a subnet and between subnets by using link-local prefix, 1111111010, and site-local prefix, 1111111011, respecively. 64 bits are used for addressing inside a subnet and an additional subnet address (16 bits) is necessary for addressing between subnets. The commonly unused 38 bits of two formats can be used as MF.

In contrast, marking schemes proposed in [36, 37, 38] simply mark (or write) some information in packets while routing the packets to help receiving nodes investigate where the attacking traffic comes from. Since marking schemes do not cause additional overhead except additional marking time in each switch, we choose to develop a marking scheme in this study.

a. Marking Field in IBA

The first requirement to use any marking scheme is an additional space in an IBA packet to mark on, referred to as *Marking Field (MF)*. An IBA packet uses different headers relying on where a destination node is located. If a destination and a source nodes are located in the same subnet, LRH is enough to switch inside a subnet. But IBA specifies that a GRH can be used within a subnet by setting the link-local prefix to 1111111010 as depicted in Fig. 4(a). If the two communicating nodes are located in different subnets, the GRH is necessary to route these packets properly. For this, the destination address should set the site-local prefix to 1111111011 described in

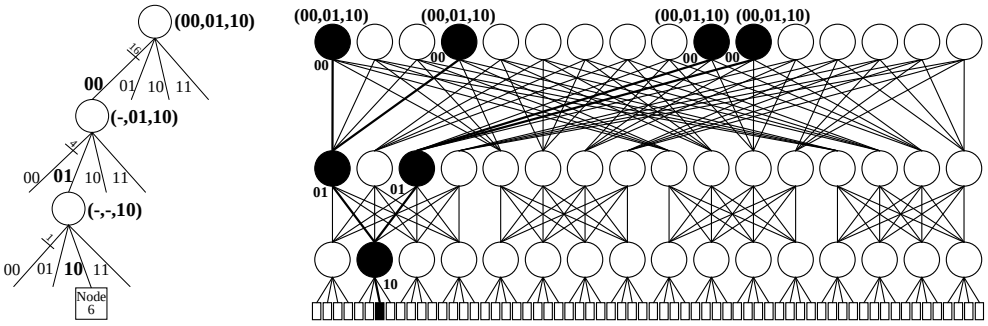
Fig. 4(b). Therefore, the 10 bits prefix indicates whether a packet's destination node is inside the site. If a packet is leaving a site, we do not mark the packet. Otherwise, 54 and 38 bits of a destination address are set to zeroes. We suggest using the common 38 bits as the MF.

b. Deterministic Packet Marking

In a deterministic marking algorithm, each switch (or router) writes its own information such as switch indexes in every packet's MF. When a cluster is not so big, end nodes and switches are usually connected in a tree structure. For example, the lowest level switches equipped with tens of ports are connecting end nodes and those switches are connected to upper level switches hierarchically. In this environment, all hops can be most likely recorded in the MF. If there are n switches in a cluster, $\lfloor 38/\log n \rfloor$ hops can be recorded in the 38 bits marking field. Alternatively, each switch's port index can be recorded instead of the switch index. In this method with the 38 bits MF, $\lfloor 38/\log p \rfloor$ hops will be traced back where p is the maximum number of ports in switches.

We show an example of a deterministic packet marking on a fat tree which is the most popular network topology in IBA clusters. Fig. 5(a) depicts a 3-level fat tree in a logical view. As the level goes up, the bandwidth of each link is squared to provide non-blocking switching. Before marking packets, each port is first numbered in its switch using $\log p$ bits where p is the number of ports in the switch. In Fig. 5(a), two-bit port numbers are shown next to links. Accordingly, an address of a leaf node is the concatenation of port numbers from the top level switch to the lowest level switch. For example, the address of node 6 is (00,01,10).

To identify a source node, when a packet traverses to upper levels, each switch writes down the index of the port that the packet comes through. For example, the



(a) logical topology (b) an implementation using 8-port basic switches

Fig. 5. Deterministic packet marking on 3-level fat tree. As a packet goes up, each switch marks an incoming port's number in the packet's MF. Marking completes when a packet reaches the top level switch. This marking scheme works for a fat tree built by a large number of minimal basic switches since this scheme marks only port numbers not switch indexes.

lowest switch marks 10 on packets coming from node 6, so their MF will be (-,-,10). Then, 01 and 00 are written at the upward switches. As a result, the MF field will have (00,01,10) at the top level switch which is exactly same as node 6's address. A real implementation of the fat tree using a simple small-size switch is depicted in Fig. 5(b). Even though packets coming from one leaf node can take different upward paths, the recorded MF values are always the same because each switch records a port number, not a switch index. As described in Fig. 5(b), although packets from node 6 take four different paths, all the recorded values are (00,01,10). When the maximum number of ports is p, a 38 bits long MF will record at most $\lfloor 38/\log p \rfloor$ hops. Assuming all intermediate nodes are switches and leaf nodes are computing nodes or storage nodes, the maximum size of cluster which the 38 bits MF supports is $p^{\lfloor 38/\log p \rfloor}$ as summarized in Table II.

However, when a cluster consists of tens or hundreds of thousands of nodes with regular interconnects such as mesh and hypercube, the average number of hops becomes so large that deterministic packet marking schemes cannot work. For example,

Table II. Scalability of Deterministic Packet Marking scheme

Topology	Max ports per switch	Max Cluster Size
fat-tree	p	$p^{\lfloor 38/\log p \rfloor}$

the diameter of a 20×20 mesh network is 39 where *diameter* is the longest hop in the shortest paths between any pair of nodes. Furthermore, when the routing is not static but adaptive, recording all hops becomes impossible because a routing path can be much longer than the shortest path depending on the current network condition. Therefore, the deterministic marking schemes will be inapplicable to large networks with regular topologies or adaptive routing schemes.

c. Deterministic Distance Packet Marking

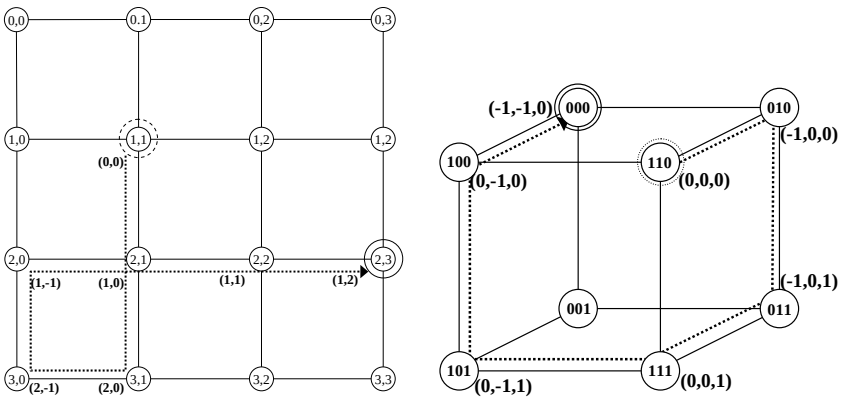
To alleviate this problem, we propose the Deterministic Distance Packet Marking (DDPM) scheme. This scheme records only relative distance from the current position of a packet to the source node without keeping track of intermediate paths. For example, in an n-dimensional mesh, we locate a node, X by using its coordinate, $(x_0, x_1, ..., x_{n-1})$. A relative distance between two nodes, X $(x_0, x_1, ..., x_{n-1})$ and Y $(y_0, y_1, ..., y_{n-1})$, can be defined as a vector V $(v_0, v_1, ..., v_{n-1})$ where $v_i = y_i$ - x_i. This vector is the coordinate difference. There is a unique distance vector between any two nodes in the network. That is, with a distance vector V, node Y identifies a unique node, X.

Inputs:	Destination D $(d_0, d_1, ..., d_{n-1})$
	Current X $(x_0, x_1, ..., x_{n-1})$
Output:	Next Y $(y_0, y_1, ..., y_{n-1})$
	Source S $(s_0, s_1, ..., s_{n-1})$
Variables:	Distance V $(v_0, v_1, ..., v_{n-1})$
	New distance $V\prime$ $(v_0\prime, v_1\prime, ..., v_{n-1}\prime)$
	Difference Δ $(\delta_0, \delta_1, ..., \delta_{n-1})$
Procedure:	
	if $X = D$ then
	V := Extract_MF();
	S := X - V;
	exit;
	endif
	Y := Routing(V);
	V:= Extract_MF();
	Δ := Y - X;
	$V\prime$:= V + Δ;
	Store_MF(V');
	exit;

Algorithm 1. Deterministic Distance Packet Marking

In each hop, each packet's distance vector is updated. V is set to a zero vector when a packet first enters a switch from a node, which is hardcoded in switch or CA. After the switch decides the next hop, it updates the distance vector. In regular networks, a packet moves by one hop in one dimension at each switching. Therefore, at each hop, one v_i is added or subtracted according to the routing direction. After a switch stores the new distance vector V in the MF, the switch transmits the packet to the next switch. The full algorithm is described in Algorithm 1. *Routing()* is a function that returns the next node coordinate. *Extract_MF()* returns a distance vector from the GRH header, and *Store_MF()* stores it in the header. Regardless of intermediate routing paths, the final distance vector V is the exact difference from the source to the destination node. Therefore, the destination can identify the source node instantly with only one packet.

Fig. 6(a) shows an example that a packet traverses a 2-D mesh adaptively from (1,1) and (2,3).



(a) 4×4 mesh (b) 3-hypercube

Fig. 6. An exmaple of deterministic distance packet marking. When a packet traverses through switches, each switch calculates the difference between its own coordinate and a neighboring switch passing the packet. The difference is added up to the current distance vector to update the MF. The receiving node easily identifies the source node by subtracting (or XORing) the distance vector from the receiving node's position. Note that this marking algorithm is robust to any adaptive routing algorithms which allow looping as shown in (a).

The distance vector changes as following: (0,0), (1,0), (2,0), (2,-1), (1,-1), (1,0), (1,1), and (1,2). When (2,3) node receives the distance vector (1,2), it can subtract (1,2) from (2,3) and quickly identify the source (1,1). In the hypercube in Fig. 6(b), the distance vector changes as following: (0,0,0), (1,0,0), (1,0,1), (0,0,1), (0,1,1), (0,1,0), and (1,1,0). (0,0,0) can identify the source (1,1,0) by XORing its coordinate (0,0,0) and the distance vector (1,1,0). Note that in hypercube XOR is used instead of subtract.

The DDPM can support a larger number of nodes compared to the deterministic marking algorithm regardless of routing policies. To support $n \times n$ 2-D mesh and torus, the half of the 38 bits MF contains the distance in one dimension. The distance can be negative, so $2^{18} \times 2^{18}$ 2-D mesh and torus networks can be marked by the DDPM. For a 3-D mesh and torus, the DDPM can mark $2^{11} \times 2^{11} \times 2^{11}$ networks by splitting

Table III. Scalability of Deterministic Distance Packet Marking scheme

Topology	Required Field	Max Cluster Size
$n \times n$ mesh, torus	2 $\log n$ - 2	$2^{18} \times 2^{18}$
$n \times n \times n$ mesh, torus	3 $\log n$ - 3	$2^{11} \times 2^{11} \times 2^{11}$
n-cube hypercube	$\log 2^n$	2^{38}

the MF into three parts (2^{33} nodes cluster). For the hypercube, the whole MF can be used for the distance vector, so the DDPM can mark 38-cube hypercube (2^{38} nodes cluster). Table III summarizes these facts.

F. Security Analysis

• **Defending Attacks**

Type I attack: As we showed an example in Section 2, GCM encrypts using AES with the CTR mode considering a PSN as a counter. Since AES is considered to be secure without any serious weakness until now, the brute-force attack is believed to be the most effective attack. When a 128 bits key is used, on average 2^{127} trials are needed to find the key, which is infeasible in the foreseeable future. Another concern is the multiple use of the same PSN since it will expose partial information of original plaintexts even though the amount of information is not much. Suppose E_1 and E_2 are two different ciphertexts using the same counter C and the same secret key, K. If a hacker gets these two ciphertexts and XOR them, he will get an XOR of two original plaintexts by clearing out the keystream.

In the QP-level key management, since a new session key is assigned to a new communication session, the multiple use of the same PSN will not cause this problem. However, it does in the partition-level key management since multiple consumers will

use the same key. To avoid the problem, we propose a 128 bits random number be created and shared during a communication set-up phase. The use of a PSN along with a random number as an input to AES will remove the concern except when any two random numbers happen to be the same, called *collision*. The probability, $P_{collision}$, that any two random numbers out of given n integers with range [1,d] are same is approximated as in equation (3.1) [39].

$$P_{collision} = 1 - e^{-(n(n-1))/2d} \tag{3.1}$$

Suppose 2^{10} nodes keep sending packets at their full speed in 1X IBA (2.5 Gbps) for ten years and the average length of packets is 1024 bytes. During the time, around 2^{61} random numbers will be generated. Therefore,

$$P_{collision} \quad = \quad 1 - e^{-(2^{61}(2^{61}-1))/2^{129}} \approx 0.0078$$

Considering the normal amount of traffic, $P_{collision}$ will be far less than this approximation. Therefore, the proposed encryption scheme will be secure in the foreseeable future.

Type II attack: Any MAC algorithms using 32 bits MAC can be used to remove Type II attack. It is known that the security strength of GCM is the same as the strength of its block cipher, AES [34, 23]. As Hellekalek *et al.* concluded that pseudorandom numbers generated by AES are indistinguishable from real random numbers [33], a 32 bits tag generated by GCM using AES is a near random number. Therefore, on average after 2^{31} trials, a hacker can successfully make an authentic MAC without a legitimate key. In an IBA cluster with 2.5 Gbps link, assuming all packets are 1024 bytes long, 2^{18} packets can be generated in one second. This means it will take the hacker about less than three hours to spoof a packet. However, note that the attacker should send wrong MACs 2^{31} times on average before he succeeds. These

attacks can be detected by issuing an alert in a case of consecutive authentication errors.

Type III attack: The DDPM described in Section 4 can trace back the original location of an attacking node, which is robust to fake addresses.

Type IV attack: The SIF blocks DoS attacks which are using illegal Keys.

• **Replay Attacks**

In replay attacks, a hacker captures a packet carrying a legitimate MAC and keeps sending it to disrupt the network. Our authentication scheme is vulnerable to this attack because the proposed scheme is per packet authentication method. Therefore, repeated packets will be considered as authentic due to the legitimate MAC. Among possible approaches, timestamps can be deployed in IBA clusters [40]. In the timestamps scheme, a sending node inserts a timestamp into each packet and a receiving node checks whether this timestamp is sufficiently close to the local time. For sure, the timestamps should be included for generating MACs to prevent false timestamps. There are two basic requirements to use this scheme. One is that communication nodes should be synchronized and the other is that each packet should have a space to carry a timestamp. In a cluster system, the global time synchronization is rather easy because time synchronization packets can be broadcasted periodically with a very short delay. Concerning the space, as we mentioned in Section 4, the GRH can be used in every packet and some bits out of MF can used to store a timestamp as long as the rest of MF bits support the network. For example, 4K nodes mesh network requires 14 bits as a marking field. Then, 24 bits can be used as a timestamp space and it will wrap around in 194 days when the time granularity is second. If the network is huge and the administrator is more concerned about the replay attack than type III attack, the source identification scheme may be disabled and the whole

38 bits can be used for timestamps, which will make the wrap-around time thousands of years.

• **Tamper-Resistant Storage**

As we assumed before, each device has tamper-resistant storage. This storage is usually built using battery backed RAM (BBRAM) [41]. Its contents will be instantly zeroed by cutting off the power supply once a tamper is detected. FIPS 140-1 recommends BBRAM to store sensitive data [42]. The contents of BBRAM should be toggled periodically because if the same data is stored for a long time the contents can be imprinted into RAM. In addition, each device should be hardwired to keep any secret information on BBRAM only accessible through its subnet manager. Finally, all secret keys stored in devices' BBRAM will be protected against any software and physical attacks.

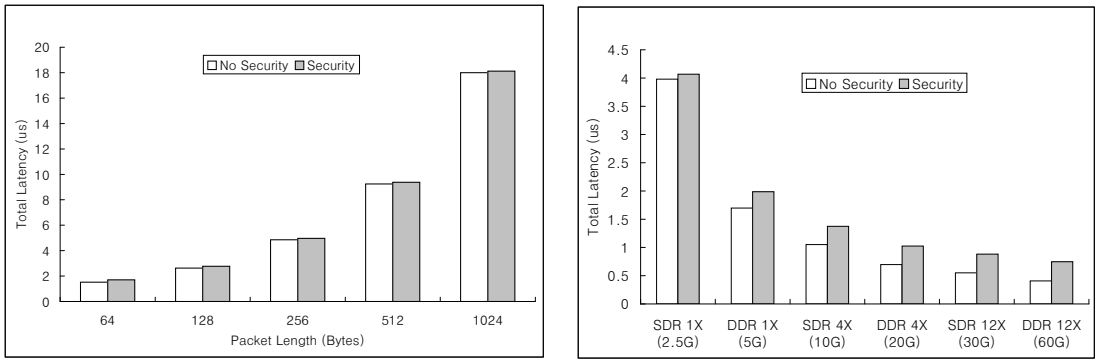
G. Performance Analysis

1. Simulation Testbed

Our IBA testbed includes packet-level switches and host channel adapters compliant with the IBA specification [30]. For our experiments, we simulated a 16-node network designed using 5-port switches. To simulate the Single Data Rate (SDR) and Double Data Rate (DDR), we use high bandwidth physical links up to 60 Gbps (DDR 12X). For performance analysis, we use best-effort traffic only.

2. Performance Slowdown by Encryption & Authentication

To see the impact of encryption/authentication scheme, we compare end-to-end latency. All delays of security operations are added up to the total latency. A recent



(a) 2.5 Gbps IBA (b) 48 Gbps IBA, 256 Bytes packet

Fig. 7. Performance slowdown due to security operations. (a) Security configuration where both encryption and authentication are enabled incurs relatively small overhead ranging from 0.7% to 12.4%, compared to No Security configuration. (b) As IBA network speed increases, the performance overhead of Security configuration also increases proportionally. However, since the additional overhead is still nanoseconds scale, Security configuration will be practical.

design of AES can encrypt 30∼70 Gbps using 0.18μm CMOS with 80 *ns* of initial latency [43, 13]. We call this initial latency *AES latency*. This means that at the sender in Fig. 3(a) it takes one AES latency to get $E[1]$ at the first encryption. The subsequent encryptions till $E[m]$ are pipelined with no delay. According to [23], a parallel implementation of $mult_H$ can keep up with any pipelined implementation of AES. Calculating Tag requires one more $mult_H$ latency because it should wait until the completion of $E[m]$ in our simulation. So we set the total additional overhead to 320 *ns* per packet. We assume that the table look-up time for partition- or QP-level secret key is negligible. If there are a huge number of QP-level secret keys, it will be necessary to take the look-up time into account.

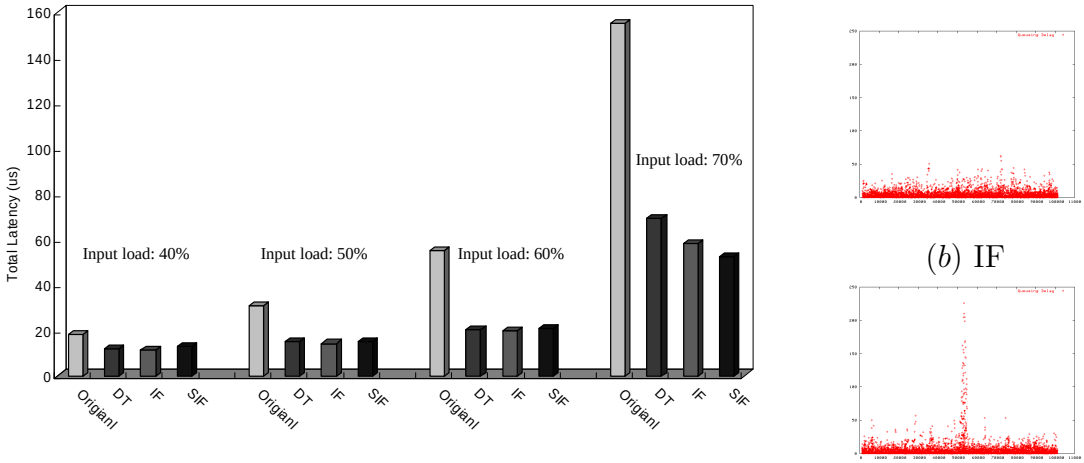
Fig. 7(a) shows the overall network performance slowdown incurred by encryption and authentication. *No Security* represents the original IBA configuration without any security function enabled. In the *Security* configuration, all encryption/authentication

are enabled. As the packet length increases from 64 bytes to 1024 bytes, the average latency decreases proportionally from 12.4% to 5.5%, 2.6%, 1.2%, and 0.7% since the additional delay is constant to the packet length.

To see the effect of our scheme on very high speed networks, we simulate up to 60 Gbps (DDR 12X IBA). To rule out network congestion effect, we scaled down the injection rate in high speed networks. The packet length is set to 256 bytes as default. As shown in Fig. 7(b), the proportion of security overhead is increasing because the 320 *ns* AES latency is not changing while the average network latency decreases. Even though our scheme incurs more performance overhead proportionally in high speed IBA networks, the total additional latency is shorter than one microsecond, hopefully making our scheme still practical.

3. Stateful Ingress Filter Simulation

To investigate the effectiveness of the SIF, we compare the SIF with the DT and the IF. When DoS attacks are very rare or not happening, the SIF is definitely the most efficient because it induces no delay while the DT and the IF require constant additional delay regardless of the occurrence of DoS attacks. In the DT, all switch ports compare each packet's Key with valid Keys stored in switches while in the IF only ingress switch ports do so. According to the IBA specification, each port can have at most 32768 P_Keys, so the maximum size of memory to store all the P_Keys is 64K bytes because one P_Key is 16 bits long. Considering that the QP-level key management can be enabled and the number of QPs is much larger than that of P_Keys, we assume the size of storage for holding all Keys is 1024K bytes in our simulation. According to a cache access model [44], 1024K bytes SRAM memory can be accessed within 5 *ns*. Therefore, the DT has 5 *ns* additional delay at every switching port while the IF has the delay only at ingress ports.



(b) IF

(a) Performance comparison among Original, DPT, IF, and SIF (c) SIF

Fig. 8. Effect of SIF. (a) DT, IF, and SIF block DoS attacks successfully and show little difference in terms of total latency except at the input load 70% where SIF shows the best performance. Note that since SIF is enabled only when there are active DoS attacks, SIF will show the best performance in normal situation. (b) and (c) shows the distribution of end-to-end latency. Since IF blocks all DoS traffic from coming into IBA network, there is no difference in end-to-end latency as shown in (b). However, SIF allows DoS traffic for short time. Due to this, there are high spikes of end-to-end latency in (c). All static methods, DPT and IF, have to incur constant overhead regardless of the occurrence of DoS attacks.

We simulate a DoS attack in the middle of the simulation and measure the average end-to-end delay in four configurations: Original, DT, IF, and SIF shown in Fig. 8(a). The simulation consists of four sets of simulations varying the input load from 40% to 70%. The first four bars in the figure are tested with input load 40% and so forth. The first bar of each set shows the average end-to-end latency under a DoS attack in the original configuration. As the input load increases, the network performance is further deteriorating. This is because the DoS attack saturates the already heavily loaded network by injecting more traffic, thus resulting in significant delays of normal traffic. The second and third bars of each set show the average delay of the DT

and the IF. Since they block all illegal traffic, their latencies are not affected by the attack. However, redundant P_Key table look-up operations incur some amount of delay. Especially, under the heavy traffic, their filtering overhead results in longer delay than the SIF. The fourth bar is showing the performance of SIF. Note that the SIF enables port filtering only when a DoS attack is active. Therefore, with SIF there is a delay to register an illegal Key in appropriate ingress ports after detecting the illegal key. During the time, the attack affects other normal traffic even though its overhead is small. Fig. 8(c) shows that the SIF blocks the traffic instantly but the delay results in high spikes of end-to-end latency in the middle the simulation. In contrast, the IF in Fig. 8(b) does not have such variations. This explains that the average latencies of the SIF for the input load with 50% and 60% are slightly longer than those of the DT and the IF

However, even with this effect the SIF in the input load 70% shows the best performance. This implies that the ingress filtering in the DT and the IF can make a tangible effect on heavily loaded networks. Standard deviations of the SIF at 40% and 50% input load are around 14 and 11, standing higher than those of the DT and the IF, around 5 and 8, due to the short-lived DoS attack. But at the high load up to 70%, standard deviations for all methods become much bigger because of high traffic.

H. Conclusion

This paper addresses a security provisioning framework for secure IBA. We have argued that the security vulnerability inside IBA can be a serious problem and proposed a comprehensive framework to enhance IBA security in view of confidentiality, authentication, and availability. For confidentiality and authentication, we proposed partition-level and QP-level secret key management schemes and showed how IBA

accommodates GCM with minor modification to the IBA specification. For better availability, we proposed a stateful ingress filtering scheme which is enabled only when there is a DoS attack using invalid IBA Keys. The packet marking algorithms further improve the availability by identifying source nodes instantly.

The important conclusions of this work are the following: First, we elaborated on the IBA Key exposure problem and simulated possible DoS attacks inside IBA cluster systems. Simulation results showed that overall performance can be affected significantly. Second, we also showed that the overall performance overhead by adopting an encryption and authentication algorithm into IBA clusters can be as low as 0.7%. Considering the security strength of GCM with AES, our scheme improves the security of IBA significantly with marginal performance degradation. Third, our stateful ingress filtering incurs a low performance overhead by being active only when necessary. Furthermore, our source identification algorithm successfully identifies real attackers in large irregular networks and huge regular networks having 2^{36} nodes with 2-D mesh and torus topology or 2^{38} nodes with hypercube topology.

We are currently examining a number of possible extensions of this work. First, we would like to implement our ideas and build a real testbed. Second, we will extend our research to other cluster interconnects like Quadrics, Myrinet, and Gigabit Ethernet. Since they have common contraints such as the speed limitation on security operations, the hardware approach used in this research will be applicable to other interconnects. Third, in a very large size IBA cluster, the number of QP-level secret keys become so big that its look-up time is expected to become longer. For this, we are investigating a fast secret key management with architectural support. Last, since the OS can get much benefit from the proposed secure IBA for better cluster security, we plan to investigate this possibility.

CHAPTER IV

A SESSION KEY CACHING AND PREFETCHING SCHEME FOR SECURE COMMUNICATION IN CLUSTER SYSTEMS

A. Introduction

Cluster systems have emerged as the most cost-effective solution for many kinds of applications and services. As many institutes like banks, military, and government agents that can be targets of intensified terrors adopt cluster systems, the importance of cluster security increases significantly. Besides, ordinary hackers are also attracted to cluster systems because they can stage a massive security attack by utilizing abundant resources such as huge computation power, large disk space, and high speed networks. Unfortunately, traditional security countermeasures like firewalls and intrusion detection systems (IDS) are not sufficient to provide complete security for cluster systems, as shown in the past successful attacks on cluster systems [45].

Among many possible attacks, we focus on physical and software attacks on cluster communications*. Physical attacks on cluster communications capture or modify data from cluster interconnects through a snooping device attachable to the interconnects. Since all cluster communication messages are plaintext, one successful physical attack will leak a significant amount of information from cluster systems. In a recent survey questioned to supercomputer administrators [46], 8 percent of respondents reported that there were unlawful physical approaches to their systems, and another 8 percent told that someone had tried to bribe inside personnel to help them infiltrate cluster systems. Considering tremendous aftereffects of physical attacks, the percentages do not seem negligible at all. Moreover, as shown in [5], cluster interconnects can

*Cluster communication in this paper is equivalent to intra-cluster communication.

be used to connect multiple clusters and storage systems, inevitably having its cable vulnerable to possible physical attacks. Therefore, we believe that the possibility of physical attacks cannot be overlooked any more, so the attacks should be prevented with great care.

Meanwhile, software attacks on cluster communication are done by a hacker who acquired a cluster account illegitimately and logged into a cluster node as a normal cluster user. If the hacker injects or sniffs packets without constraints, the cluster system will be compromised seriously. Note that preventing software attacks on cluster communication can be very effective in bolstering cluster security because the hacker may prefer such attacks to other software attacks. Moreover, the attacks on cluster communications may be the last option for the hacker; thanks to enhanced cluster security by using partitioning, sandboxing, multi-level security, or other confinement schemes [47, 48, 49, 50], other possible software attacks beyond predefined resources can be prevented. Consequently, we believe that cluster communications need to be protected effectively against software attacks as well as physical attacks.

To prevent these attacks, in the previous chapter, we proposed a comprehensive security framework for InfiniBand cluster systems. The framework uses a fine-grained security scheme where any two communicating processes or queue-pairs in InfiniBand architecture (IBA) share a session key dynamically to encrypt/authenticate all the communications. Since in their framework the security function resides in cluster interconnect cards (CICs), not in host CPUs, it causes no additional performance overhead to host CPUs. However, we assumed that session keys stored in the CIC are accessible for cryptographic operations without any delay. This assumption of zero delay in session key access time is unrealistic for the following reasons. If the session keys are stored in an off-chip memory in the CIC, every packet arrival incurs an additional memory access, resulting in a very long packet processing time. Even

if a data cache in an embedded processor in the CIC can be used to store the session keys, a cryptographic hardware that is usually implemented as a separate hardware module cannot share the data cache with the embedded processor easily. As a remedy for this, the hardware cryptographic unit can have its own cache to store recently used session keys, referred to as *session key cache (SKC)*. In order for the SKC to be used in the CIC, a comprehensive study is necessary to answer the following important questions:

- How will the CIC look like? Where are the hardware cryptographic unit and SKC located? And what are their hardware costs in terms of area and power consumption? If physical attacks are possible, the CIC can be vulnerable to the attack, too. Then, what kind of security technology should be applied to the CIC to protect the CIC from the physical attacks?

- Is a small SKC scalable to large-scale cluster systems? Will there exist any analytic model to estimate a proper size of the SKC? What is the relationship between the size and the hit rate of the SKC?

- If there are multiple applications (or processes) running on a node, does this affect hit rates of the SKC? If so, is there a solution to maintain a high SKC hit rate? Does the solution need to closely work with the OS's schedulers? Will the solution work well consistently with various cluster schedulers?

We attempt to answer these questions by doing rigorous research on cluster traffic simulation and analysis. We first provide a detailed design of the CIC. Second, to estimate the behavior of the SKC in large-scale clusters, we develop an analytical model based on an observation that cluster traffic patterns are well fit to exponential distributions [51]. We use this model to find out the relationship between SKC sizes

and hit rates of the SKC. Third, to maintain a short session key access time even in multitasking environment we propose to incorporate a prefetch buffer in the CIC to fetch session keys ahead of time by predicting the next scheduling decision. Finally, to evaluate the new CIC design, we developed a trace-driven simulator by modifying a cycle-accurate cluster network simulator that was used in [52, 30, 53]. We captured real traces from an SGI Altix 3700 supercomputer by modifying source codes of NAS parallel benchmarks. We also developed five schedulers: Linux local, spin-block (SB) [54], dynamic coscheduling with spin-block (DCS-SB) [55, 56], periodic boosting (PB) [55], and gang scheduler [57, 58].

Simulation results show that an SKC reduces security overhead on network latency by 50 percent on average, compared to non-SKC configurations. In multitasking environments, the session key prefetching scheme is helpful in reducing network latency by 5 percent more on average. An analytic cache simulation estimates that a 16-Kbyte SKC can support up to thousands of session keys with high SKC hit rates.

The remainder of this chapter is organized as follows: Section B explains a threat model of our research, and describes major challenges in providing high performance cluster security. In Section C, we propose the design of an SKC and a prefetch buffer, an analytical model for NAS parallel benchmarks, and a prefetching scheme. Simulation results and their analysis are presented in Section D, followed by related work and concluding remarks of this chapter in Sections E and F.

B. High Performance Cluster Security

In this section, we describe the threat model and the architectural environment of our research, and then we advocate the CIC approach for high-performance cluster security by comparing three possible scenarios: host CPU, security coprocessor, and the CIC.

1. Threat Model

At first, we would like to begin with defining a threat model in order to clarify what kinds of threats we want to defend against. As for physical attacks, we assume the following constraints. We assume that an intruder can access a cluster system physically and he wants to be as unobtrusive as possible in order to go unnoticed. So he prefers installing a snooping device in the system to accessing a console connected to the cluster system to get a root privilege or copy data from the console so that he can steal data from the system secretly for a long time [59, 60, 61, 62, 63, 64, 65, 66, 14, 13, 67, 68]. Many previous studies have investigated physical attack prevention mechanisms for a single system or each cluster node [59, 61, 62, 64, 65, 66]. So we assume it is impossible to capture or modify data from processor, memory system, memory bus, or hard drives. In addition, based on the studies of security coprocessor and smartcard [69, 70], we also assume that the CIC has tamper-resistant mechanism so that some part of the CIC can be physical attack-proof. As shown in Figure 9, we assume that the tamper-resistant mechanism is applied to the CIC controller containing an embedded processor and caches, while other components like the CIC memory and its memory bus are insecure, which is a common assumption in hardware security research. Other cluster components including switch/router and cables are also assumed to be vulnerable to physical attacks.

As for software attacks, we assume that a hacker compromised a cluster node, so he has a full access to cluster interconnects through the node's CIC. We assume that a security compromise in one node does not mean that the whole cluster system is compromised. This assumption is based on that system administrators can apply appropriate confinement schemes such as logical partitioning, sandboxing, or multi-level security to confine security attacks to predefined resources [47, 48, 49, 50]. We also assume that switch/router are vulnerable to software attacks, so they can be configured to mirror all traffic to some ports.

Interestingly, under the above assumptions, a physical intruder and a software hacker will do a similar thing; they capture and inject packets from and to the cluster interconnects. Through capturing attacks, they can steal data. If a cluster system deals with confidential or classified data, data theft itself is a serious problem. Besides, captured plaintext packets may reveal information of a global root privilege. Through injecting attacks, they can inject new packets or modify in-transit packets. Without a proper authentication scheme, any illegal modification on confidential data could go unnoticed. In addition, the hacker can stage a spoofing attack using fake source addresses so that it reaches other nodes beyond confined resources.

2. Solutions to Secure User-Level Communications

Before we propose a solution to the above mentioned security problems, we provide the rationale behind the current design using CIC, not using host CPU or security co-processor. First of all, it is very important to understand the main difference between traditional communication and high-performance cluster communication. Traditional data communication involves considerable OS interventions that need multiple memory copies, with centralized communication stack processing. A long latency with a large overhead of a host CPU has been a big obstacle to high performance computing.

To solve this problem, most of high-end cluster systems have used the user-level communication (ULC) including Myrinet and InfiniBand [29, 71, 72, 73, 74, 75, 12, 76], which allows applications to bypass the OS to access network adapters directly. An instance of the ULC usually consists of two stages: setup and data transfer. In the setup stage, an application requests the OS for an access to its network adapter. The OS then checks the request to guarantee the protection of the memory and communication among multiple applications, and approves the request by giving it a handler to access the network adapter. In the data transfer stage, the application can send and receive data directly using the handler. As a result, memory copies are reduced dramatically because data are transferred between the application's memory area and the network adapter without any interventions by the OS.

An easy solution to secure ULC is the host CPU approach; before an application sends a message, a host CPU executes a security function to encrypt it, and then a receiving application calls another security function to decrypt the message. However, it is well known that such cryptographic operations done by the host CPU incurs significant performance overhead by consuming a large amount of CPU time [77, 78, 79, 80]. For example, a recent study on performance analysis of transport layer security (TLS) web servers [78] shows in an experiment that the encryption and authentication operations was almost the same as the original web server's execution time, meaning that the cryptographic operations incur 100% performance overhead. So, it is quite obvious that such overhead will be intolerable for most cluster systems because one of main purposes of the cluster systems is to provide high performance computing power.

Another possible solution is the security coprocessor approach; security coprocessors can take over the host CPU's cryptographic operations. In fact, this approach was thoroughly investigated in [81]. It found out in real experiments of various secu-

rity coprocessors that the use of the coprocessor needs additional PCI transactions for memory copies to/from the coprocessor and they become a limiting factor of system performance.

By agreeing with [81]'s conclusion that the cryptographic support for secure communication needs to be done in other places other than the security coprocessor, we advocate that a CIC augmented with security hardware should implement secure ULC. This CIC approach will not need the additional PCI transactions for memory copies. Applications can simply copy data to/from the CIC as in normal ULC operations so that the CIC takes care of all cryptographic operations by using the security hardware in it. Therefore, we believe that the CIC approach completely satisfies ULC's requirements for low latency and less OS involvement as well as meets new demands for secure cluster systems.

C. Architectural Support for Secure Cluster Communication

In this section, we first explain the reason why we propose an SKC in the CIC for secure cluster communication, and then we present a detailed architecture for an SKC. Then, we develop an analytical model in order to find the right size of the SKC. For further enhancement, we suggest a coscheduling-aware prefetching scheme.

1. Session Key Cache Architecture

In the fine-grained security scheme where a pair of communicating processes (or queue-pairs in InfiniBand) share a unique session key as proposed in [48], each packet needs to carry a session key identifier so that the receiving CIC can look up a corresponding session key. If all the session keys are stored in an off-chip external memory, every packet transmission requires additional delay for accessing the session keys. To

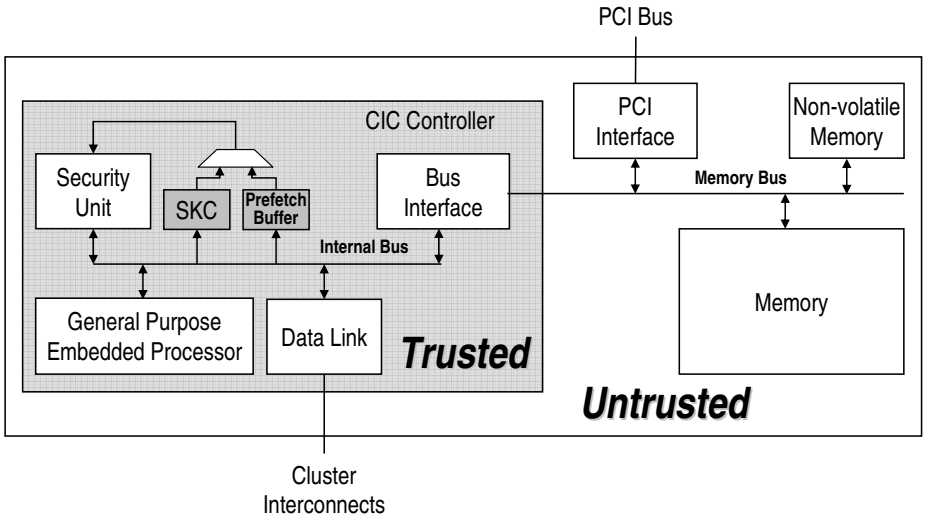


Fig. 9. SKC, prefetch buffer, and security unit in a CIC.

minimize average session key access time, we propose the CIC should have an on-chip SRAM, referred to as *session key cache (SKC)*, for storing recently used session keys,.

Figure 9 depicts a general CIC design. The CIC controller is assumed to be tamper-resistant while other components are not. The embedded processor is in charge of all communication processing including packetization, connection management, and protection, etc. Data Link unit represents any data link layer module that manages data send/receive across physical links by doing error detection, flow control, and buffering. Security Unit controlled by the embedded processor is a generic module for implementation of any security algorithms.

For security management, each embedded processor needs to be fabricated to carry at least one unique secret key or a private key for protecting an initial security setup such as distributing a system-wide secret key. To avoid the long system re-initialization at every system rebooting, it is better for the CIC to store the system-wide secret key and related secret information in non-volatile memory. Note that it is necessary to encrypt/authenticate all data available in the untrusted area.

The SKC stores recently accessed session keys. Upon a session key cache miss, the SKC fetches the session key from the external memory. Least Recently Used (LRU) is used as a replacement policy to utilize temporal locality. For now, the SKC is assumed to be a fully associative cache in which a session key can be stored at any locations. Each cache entry should be large enough to hold one session key and its identifier. To further increase the session key hit rate, we propose to add a small buffer next to the SKC, called a *Prefetch Buffer*, that will be explained in Section 3.

To estimate our scheme's impact on power and space of the CIC as well as on the overall performance, we choose Galois/Counter Mode (GCM) [23] because its authenticated encryption is enough to prevent eavesdropping and authentication attacks described in our threat model. Its space and energy consumption are analyzed as follows: a recently designed GCM's throughput is about 35 Gbps at 271 MHz clock rate, consuming around 500,000 gates [82]. According to [83], AMIS can make an ASIC with high density (100,000 Gates / mm^2) and low power (30nW/MHz/Gate @ 1.8V) using 0.18 μm technology. By using this technology, one GCM block will consume 5 mm^2 space and 4 Watt with 271MHz clock speed. Meanwhile, we can estimate the space and power overhead of the SKC and the prefetch buffer by using CACTI model. An 16-Kbyte SKC with an entry size of 32 bytes needs 36.8 mm^2 space and 0.347 Watt, and a 16-entry prefetch buffer needs 2.9 mm^2 space and 0.132 Watt. Therefore, the total space and power for a GCM, an SKC, and a prefetch buffer is 44.7 mm^2 and 4.479 Watt, respectively.

2. Size of the SKC

A bigger SKC will increase the SKC hit rate, but its size will soon begin to matter with the big space overhead and long access time as well as hardware cost. In this section, we develop an analytical method to estimate a proper cache size by modeling

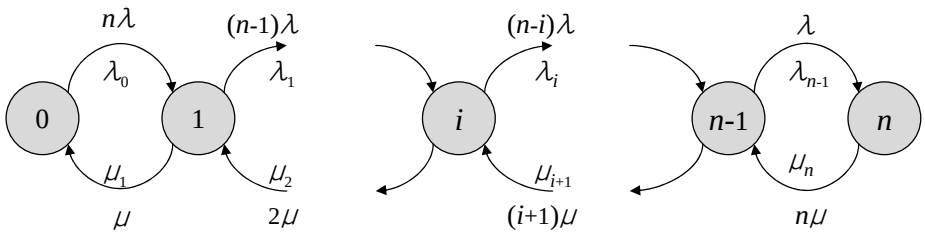


Fig. 10. Markov chain for traffic aggregation of n processes.

process-to-process communications in cluster traffic.

One study [51] modeled and analyzed the workload characteristics of NAS parallel benchmarks. It showed that most benchmarks follow a traditional ON-OFF traffic pattern. That is, each process will have a communication period *ON*, followed by a computation period *OFF* that has few communications. The most interesting observation in the study is that the lengths of ON and OFF are well fit to exponential distributions. Owing to the distribution's memoryless characteristic, we can analytically model how many processes will be in ON stage concurrently. n processes with the ON/OFF traffic pattern can be represented as a Markov chain with state space $\{0, 1, . . . , n\}$, where n is the number of processors in a cluster system, assuming that one process is running on each processor. A transition happens only between adjacent states as depicted in Figure 10. State i means that the number of processes in ON stage is i. Let λ and μ be the arrival rates of OFF and ON stage of each process, respectively. An arrival of OFF stage initiates a start of ON stage. Since all processes begin at OFF stage, the transition rate from state 0 to state 1, λ_0, is $n\lambda$. More formally, the transition rate from state i to state i+1 is $a_{i,i+1} = \lambda_i = (n - i)\lambda > 0$, and likewise $a_{i+1,i} = \mu_{i+1} = (i + 1)\mu > 0$ for $i = 0, 1, . . . , n$. We obtain the probability of being in state i, P_i, by solving the balance equation $\lambda_i P_i =$

$\mu_{i+1}P_{i+1}$ as shown in Equation 4.1.

$$P_i = P_0 \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i \tag{4.1}$$

With Equation 4.1 and $\sum_{i=0}^{n} P_i = 1$, the probability of zero active process P_0 can be expressed as

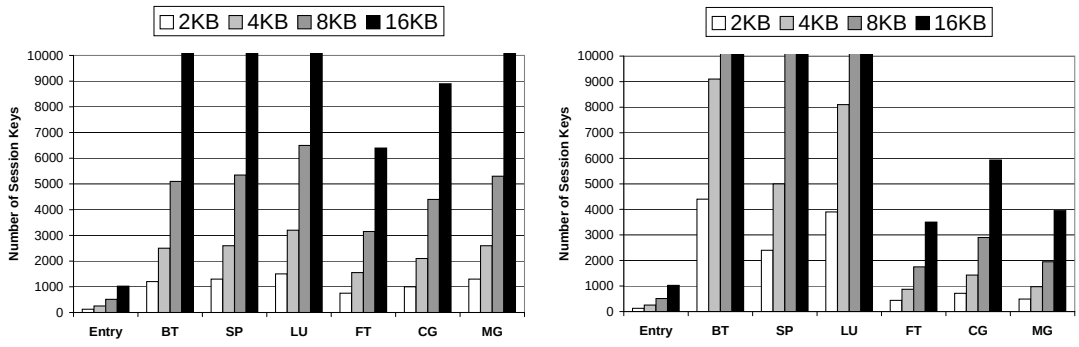
$$P_0 = 1/\sum_{i=0}^{n} \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i \tag{4.2}$$

Using these equations, we can get a cumulative distribution function $F_n(x)$ shown in Equation 4.3. For any integer x ($0 \leq x \leq n$), $F_n(x)$ represents the probability that the random variable X is less than or equal to x.

$$F_n(x) = P_n(X \leq x) = \sum_{i=0}^{x} P_i = P_0 \sum_{i=0}^{x} \binom{n}{i} \left(\frac{\lambda}{\mu}\right)^i \tag{4.3}$$

We can use this equation in two ways. First, we can estimate an SKC hit rate p when a cache size x and the number of processes n are given. For example, when $\lambda = \mu$, $x = 3$, and $n = 4$, $P_0 = P_4 = \frac{1}{16}$, $P_1 = P_3 = \frac{1}{4}$, and $P_2 = \frac{3}{8}$. We expect that p will be at least $\frac{15}{16}$ because $\sum_{i=0}^{3} P_i = \frac{15}{16}$. Second, we can also estimate n that a cache of size x supports with a target p. n is referred to as *cache capacity*. Similar to the previous example, if a target p is equal to 0.9, its cache capacity is 4 because $\sum_{i=0}^{3} P_i = \frac{15}{16} > 0.9$, and $\sum_{i=0}^{3} P_i = \frac{13}{16} < 0.9$ when $\lambda = \mu$ and $n = 5$. To CIC architects, the latter usage will be more interesting because they can project the size of a cluster system that a cache size under consideration can support with a target SKC hit rate.

Note that in this model we implicitly assumed that a process communicates with all other processes with an equal probability. According to [84], most processes in many parallel applications including NAS benchmarks communicate with several neighbors more frequently than the others, but there are also some processes that have



(a) SGI Origin 2000 (b) IBM SP-2

Fig. 11. Cache capacities of SKC for NAS benchmarks for target hit rate 0.9.

uniformly distributed destinations. Furthermore, [85] reported that there exist some parallel applications whose processes communicate with all other processes equally.

To estimate cache capacities of real cluster systems, we use an analysis of NAS parallel benchmarks running on an SGI Origin 2000 and an IBM SP-2 systems [51]. The analysis presented each benchmark's average ON/OFF length (a reciprocal of the arrival rate). We set p to 0.9 and x to 2, 4, 8, and 16-Kbyte caches. Since any two communicating processes share a session key, a cache capacity directly represents the total number of session keys that an SKC can support. Figure 11 shows cache capacities for those benchmarks. In most cases, the cache capacities are much larger than the number of cache entries represented as *Entry*. For example, in the SGI Origin 2000, the cache capacity of 2KB SKC in BT benchmark is about 1000 while its Entry is 128. This means that in BT benchmark SKC's hit rate will be higher than 90% until there are 1000 session keys. The capacity is quite surprising because the 2KB SKC can hold up to 128 session keys. The reason for this is that the OFF stage duration is at least several times longer than the ON stage duration. That is, the number of processes that are simultaneously active is relatively small. Therefore, we can expect that even a small SKC can support the communication with a fairly

large number of processes with a high SKC hit rate. Some might think that there will be few cluster systems that need thousands of sessions keys. However, according to the supercomputer Top 500 list released in November 2007, the percentage of supercomputers that have more than one thousand processors is 91%. Considering that its percentage in November 2005 list was only 29%, the number of processors in cluster systems will continue to increase quickly. Therefore, the large cache capacities supported by a small SKC will be practical.

3. Coscheduling-Aware Prefetching Scheme

In order to maximize the cluster system's overall throughput, researchers have proposed coschedulers that can schedule physically distributed processes of a job to run as concurrently as possible [54, 86, 55, 56]. If these coschedulers are used in cluster systems, multiple processes are running on each cluster node and there will be context switches inevitably. The problem of context switches is that SKC hit rates will decrease because a newly scheduled process needs to use a completely different set of session keys for communicating with whole different nodes. As a result, many subsequent session key accesses for the new process will miss until its SKC warms up. The best solution to this problem would be to synchronize the contents of SKC according to scheduling decisions of the coscheduler. That is, when a coscheduler picks a process to run next, the OS provides the CIC with the decision so that the CIC can prefetch session keys for that process. But it will cause OS interventions and additional communications between the OS and the CIC. Moreover, it will increase the complexity of the CIC design for the synchronization. To solve this problem, we propose a simple prefetching scheme that can synchronize SKC and coscheduling decisions without OS interventions.

Before describing the prefetching scheme, we would like to begin with explaining

why coschedulers are proposed and how they work. A drawback of batch scheduler that assigns a number of processors to an application exclusively is the low CPU utilization because a CPU often waits for I/O operations or messages. A simple remedy is to assign multiple applications to each node by using a local scheduler. However, since it does not consider the behavior of other processes running on other nodes, one process would have to wait until the other process wakes up, resulting in the low CPU utilization. In contrast, gang scheduler can explicitly run all processes of a job at the same time [57, 58], but unfortunately it becomes quite complex in large-scale cluster systems. To remove the above mentioned problems, researchers proposed several communication-driven coscheduling schemes such as SB, DCS-SB, and PB. These scheduling schemes are generally called *implicit coscheduling* because a message arrival from a process in a remote node implicitly notices that the process is running actively on the node. Its main benefit is that it still produces better CPU utilization and throughput than the local scheduler while it does not need a complex global control mechanism as in the gang scheduler [54, 55, 56]. In the SB [54], after a process sends a message, it spins (or busy wait) for a fixed time before blocking itself, hoping that a reply message arrives within the spinning time. The DCS-SB [55, 56] acts like the SB except that, when a message arrives at a blocked process, it increases the priority of the process. The PB [55] periodically boosts the priority of processes that have unconsumed messages in a round-robin fashion.

Our prefetching scheme takes advantage of the fact that the aforementioned implicit coschedulers use message arrivals as a hint for coscheduling. In other words, if a message arrives at a process and the process is not occupying its CPU currently, the CIC presumes that implicit schedulers will try to schedule the process soon. Based on this, the CIC loads the process's frequently used session keys to a prefetch buffer. When a new message arrives, its session key can be found in the SKC if the message

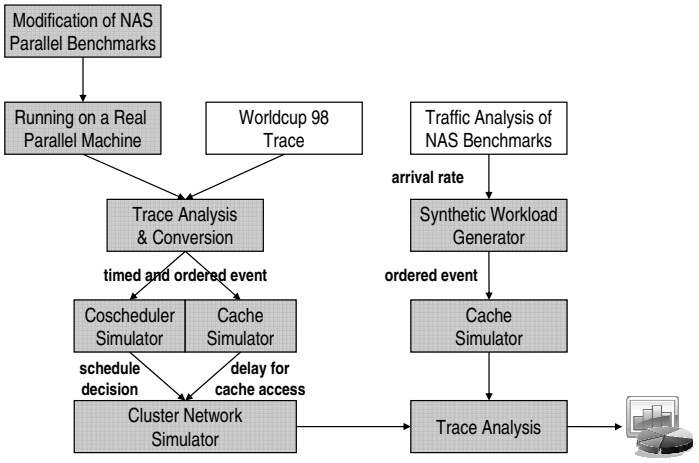


Fig. 12. Coscheduled cluster communication simulation platform.

belongs to the current process, or can be found in the prefetch buffer if the message is for the new process. For this, a session key should be looked up in the SKC and the prefetch buffer in parallel. If its session is found in the prefetch buffer, the session key is moved to the SKC and one session key needs to be evicted from the SKC. At every session key eviction, the CIC updates the list of frequently accessed session keys for the process that the evicted session key belongs to. This operation can be done by the embedded processor later, so the operation is not in the critical path.

D. Performance Evaluation

1. Simulation Platform

Figure 12 depicts the simulation platform for the coscheduled cluster communications. Shaded rectangles represent what we developed, modified, or executed in this study. We use NAS parallel benchmarks [87] that are widely used for measuring the system performance in cluster/supercomputing environments. We first modified source codes of the benchmarks to print all packets' source and destination with a timestamp. We ran the modified benchmarks on an SGI Altix 3700 consisting of 128 Itanium-2

processors and 256-Gbyte main memory with a batch scheduler where each processor is assigned to only one program at a time. Due to our network simulator's cycle-accuracy, we used the small problem size (Class A) of the benchmarks using 16 processors. In this problem size, IS and EP benchmarks produced too few traces to be used in our simulator. We also dropped FT because it had a compiling problem in our system environment. Therefore, we used the other five benchmarks: CG, SP, MG, BT, and LU. Then, we converted the captured traces into a different format so that our cluster network simulator can use the format as input.

Our cluster network simulator used in [52, 30, 53] includes packet-level switches and CICs that are compliant with InfiniBand, one of the promising cluster interconnects. To simulate more realistic cluster communications, we made three major changes to the simulator: timed simulation, coscheduling simulation, and cache simulation. First, we developed a trace-driven cluster simulator by modifying the existing cluster network simulator in which all packets were generated randomly with a certain distribution. An important requirement of the new simulator was that packet injection time needs to be adjusted dynamically. This is because a packet may come across a shorter or longer network path during simulation due to differences between the simulated networks and the real networks. To apply these dynamics to simulation, our simulator provides the relative-timed injection by maintaining the actual time distance between two consecutive packets' timestamps. In other words, let us suppose that in a real execution a node receives and sends two packets at t and $t+i$, respectively. If the first event (packet arrival) occurs at t' during simulation, then the second event (packet injection) should be scheduled at $t'+i$. This timed injection prevents unrealistically early or late packet injections. Because of this effect, the simulated execution time of the benchmarks is different from the actual execution time in the real system as shown in Table IV.

Table IV. Original and simulated execution time of NAS benchmarks

Time (sec)	CG	MG	LU	SP	BT
Original	3.72	11.86	268.61	244.12	291.73
Simulation	3.85	12.80	257.85	233.90	278.40

Second, to simulate a cluster system in which multiple applications are competing for network and computing resources, we implemented a Linux local scheduler and four coschedulers: SB, DCS-SB, PB, and gang. The Linux local scheduler is the same as the one in the Linux kernel 2.6. The simulation parameters of each coscheduler are the followings: the spinning time for SB and DCS-SB is 750 μs, a regular priority boost in PB occurs every 2 ms, and the interval of the gang scheduling is 20 ms. All the parameters are optimized through multiple simulations, and other implementation issues are described in [86, 55].

Third, to simulate the SKC, we also implemented a cache simulator in the CIC. Before sending and after receiving a packet, the CIC checks its SKC to see whether the necessary session key is cached. If found, it takes only a cache access time to read the session key, but, if not, a memory access time will be consumed. In Figure 12, we draw the three simulators (coscheduler simulator, cache simulator, and cluster network simulator) separately for better understanding, but in fact they are integrated into the cluster network simulator.

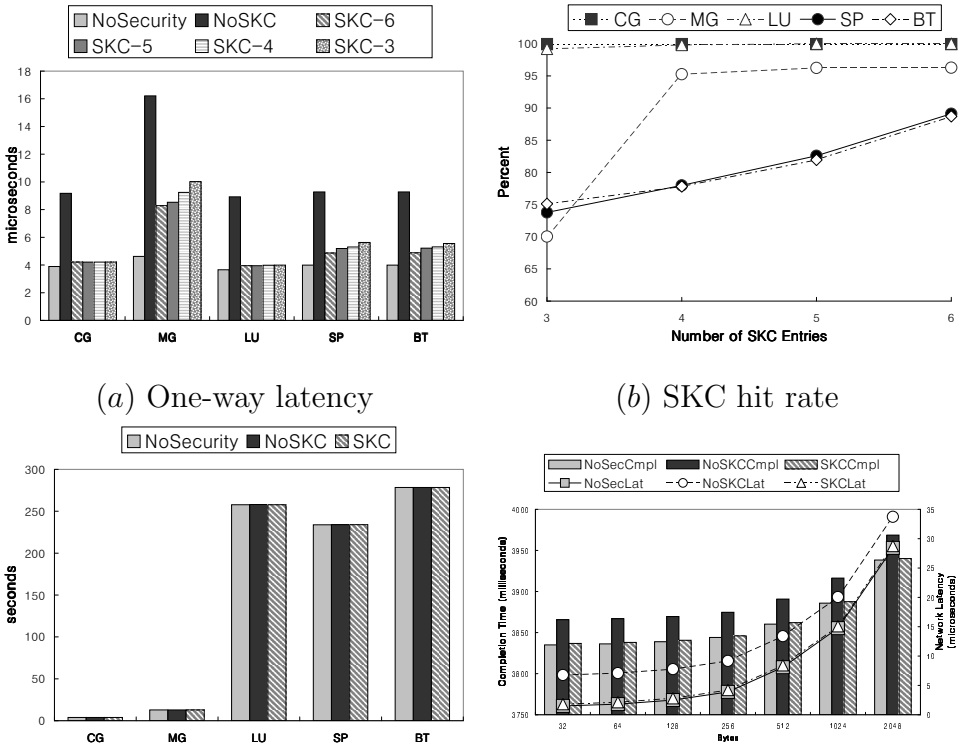
Additionally, we developed a synthetic packet generator and a cache simulator to simulate the analytical ON/OFF traffic model of NAS parallel benchmarks explained in Section 2. Note that, since we are interested in the SKC hit rate, we use only the sequence of packet arrivals, not the interval between packet arrivals. The benefit of this simulation is to overcome the time limitation in simulating a large number of long-running processes in a short simulation time, which would be impossible in a

cycle-accurate simulator.

Finally, there are several important simulation parameters. A GCM implementation presented in [82] requires 12 cycles to encrypt/decrypt the first 128-bit block and one cycle for each additional block with last two cycles to generate an authentication tag. If a 256-byte packet is in use, the additional delay is 30 cycles in sending or receiving a packet. Since this delay needs to occur both in a sender and in a receiver, the total additional one-way delay is 60 cycles that are converted to approximately 220 ns with 270MHz. According to the CACTI model, the cache access time of a 16-Kbyte fully associative cache is 3 ns. An access time to an external memory in a CIC is set to 2.5 μs, as measured in [88].

2. Effectiveness of SKC

Figure 13(a) compares the average network latencies of several configurations of each NAS benchmark. The reason why we compare the network latencies is that, since all security operations are done in the CIC, the security overhead will be reflected in the network latency. *NoSecurity* represents the original configuration without a security function enabled. *NoSKC* represents the configuration where encryption/decryption and authentication are enabled but the SKC does not exist. *SKC-n* configurations have n entries in the SKC. Compared with *NoSecurity*, *NoSKC* has around 160 percent overhead on the network latency because each packet requires an external memory access for retrieving a session key. In contrast, *SKC-6* reduces the overhead by 50 percent on average from *NoSKC*, adding 29 percent overhead to *NoSecurity* configuration. The SKC hit rates of CG and LU shown in Figure 13(b) are almost 100 percent, showing little difference in the network latencies of *SKC-n* configurations in Figure 13(a). However, other benchmarks have relatively low hit rates in small SKCs, resulting in longer network latency. Therefore, we can say that the higher SKC



(*a*) One-way latency (*b*) SKC hit rate

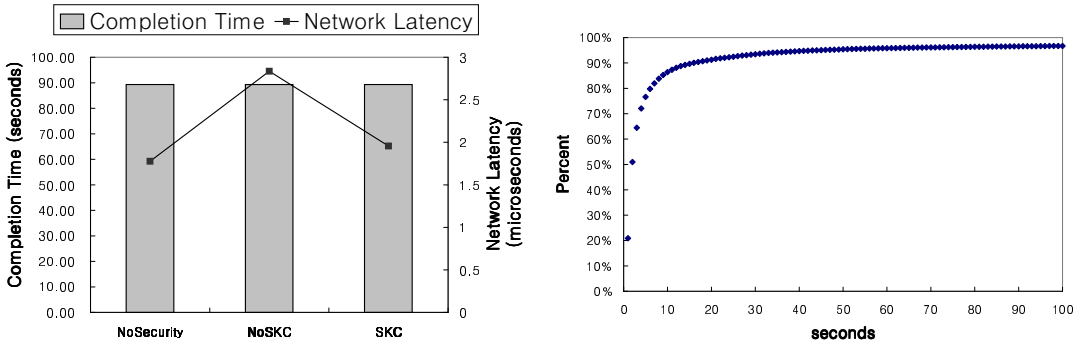
(*c*) Completion time (*d*) CG completion time

Fig. 13. Performance comparisons of network latency of single application execution.

hit rate is positively related to the shorter network latency.

Figure 13(c) compares the benchmark completion time of the above configurations. In contrast to our expectation, it does not seem that the big difference in the network latency has a big impact on the overall application completion time. The similar trend was reported by [89], explaining that many cluster applications are insensitive to the network latency, even though some are sensitive. This implies that NAS parallel benchmarks are not much sensitive to the network latency. However, we believe that there are some applications that can get a great benefit from the short network latency. For example, real-time multimedia applications used in a remote surgery need quick responses as well as strong security on each packet. Considering that cluster systems are being deployed for diverse applications, it will be useful to minimize the additional network latency while providing fine-grained security. Figure 13(d) shows the completion time and the network latency of CG with various packet lengths. Since the latency is measured till the last flit of a packet arrives at its destination, the average latency also increases as the length of a packet increases. Note that the network latency of *SKC* is similar to that of *NoSecurity*, but shorter than that of *NoSKC*.

We also investigate the effectiveness of the SKC in providing cluster systems with Internet security services such as TLS. Although those services can be done by firewall, front-end routers, or host CPUs, the CIC with security hardware can provide or at least help the services more efficiently, minimizing possible performance overhead to other cluster components. For this, we fed the Worldcup 98 Web traffic into our simulator. We choose the traces of June 10, 1998, the day of the final match of the Worldcup, and randomly select an around 90-second-long trace for simulation. The number of packets is about 20,000. Node 0 acts as a distributor to

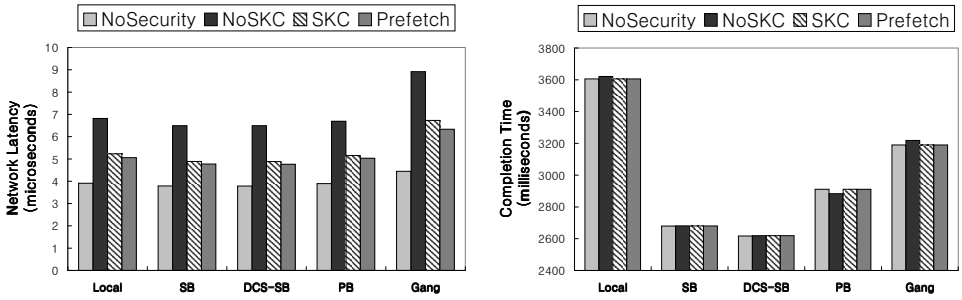


(a) Completion time & latency (b) Interarrival distribution

Fig. 14. Worldcup 98 Web traffic simulation.

forward web requests to other nodes according to their source IP addresses[†]. A simple modulo operation is used for deciding a destination node so that packets having the same IP address are forwarded to the same node for high SKC hit rate. When a destination node receives a packet, it sends a reply message to Node 0 with a static web request processing delay that is set to 1.3 ms, which is empirically measured in one of our department's web servers. Figure 14(a) shows a similar result as those of NAS benchmarks; the completion times are almost similar, but the SKC minimizes the additional network latency. The average SKC hit rate is 77.2 percent with a four-entry SKC. Considering that the number of distinct IP addresses is around 1,200 in the trace, the small SKC yields quite a high SKC hit rate. To understand this effect, we analyze inter-arrival time of the trace, the time duration between two consecutive packets originated from the same IP address. Figure 14(b) shows that most of consecutive packets from the same node arrive within a very short time period. This high temporal locality is the reason for the high SKC hit rate. Therefore, even a small SKC will have a high hit rate for any cluster applications whose traffic patterns

[†]Due to privacy concerns, real IP addresses are not available in the trace. However, a client ID that the authors use as a requester's identifier is mapped into a unique IP address. Therefore, an IP address and its client ID can be used interchangeably.



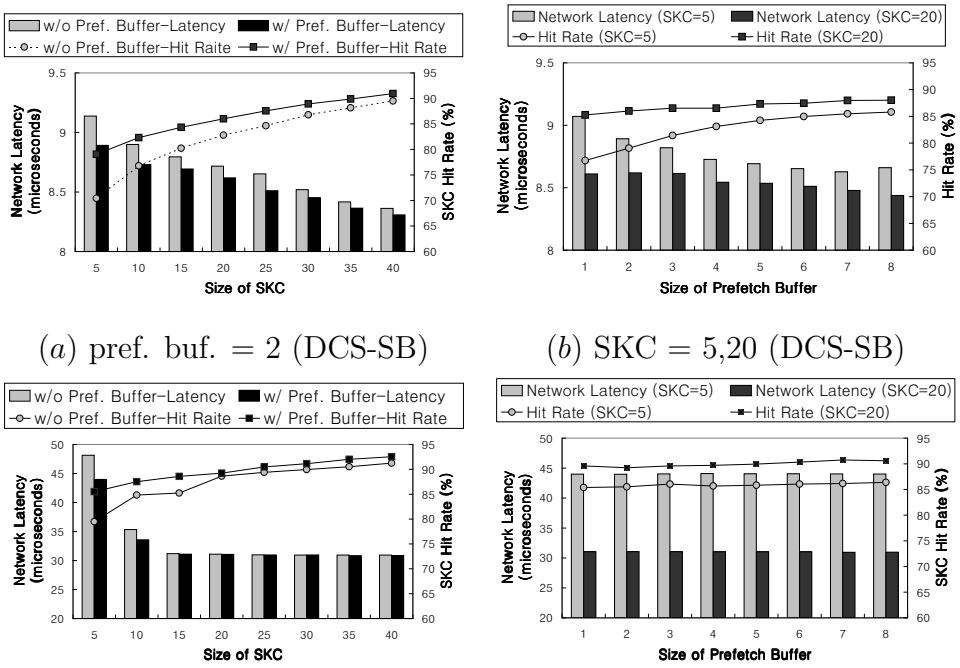
(a) Network latency (b) Completion time

Fig. 15. Performance gain from prefetch buffer on various coschedulers.

have a high temporal locality.

3. Effectiveness of Prefetching

To make workloads for the coscheduling environment, we assign a mix of several NAS parallel benchmark traces to a node so that a scheduler of the node coschedules the traces. Figure 15 shows the overall performance of all coschedulers in terms of network latency and completion time. Each workload uses 15 NAS benchmarks traces: three instances per each of the five benchmarks. The number of entries in the SKC is 10, and the prefetch buffer can hold eight session keys. In Figure 15(a), *Prefetch* reduces network latency by 5 percent on average. This performance gain mainly comes from around 13 percent hit rate increase in *Prefetch*. However, it is still hard to tell that the shorter latency always enhances the performance in terms of total completion time based on our simulation results shown in Figure 15(b). Among coschedulers, DCS-SB and SB have the shortest completion time, and Local the longest, which is similar to the results in other literature [86, 55]. In the same coscheduler, most completion times are similar except for PB; contrary to our expectation, *NoSKC*'s completion time is even shorter than completion times of other configurations. The main reason for this is that even a subtle difference in the network latency changes the scheduling



(a) pref. buf. = 2 (DCS-SB) (b) SKC = 5,20 (DCS-SB)

(c) pref. buf. = 2, (Gang) (d) SKC = 5,20 (Gang)

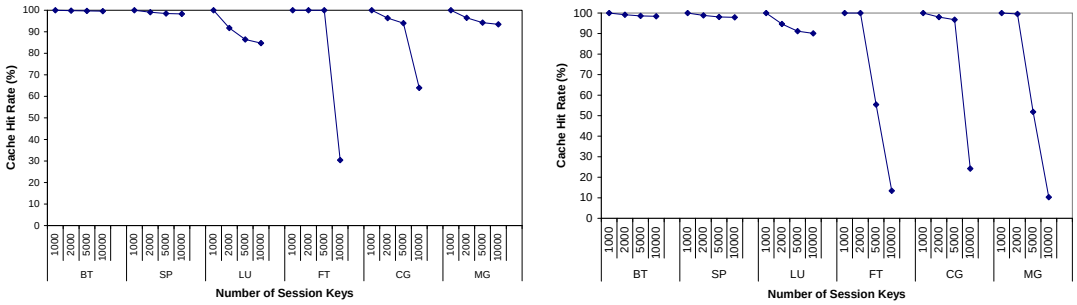
Fig. 16. Performance comparison on varying size of SKC and prefetch buffer.

order of a node, and this change has a chain effect, thus resulting in a completely different scheduling order in the whole cluster system. In our simulation environment, the variability of scheduling orders seems bigger than that of the performance gain from the prefetching scheme, especially in terms of completion time. However, we believe that the shorter network latency will help improve the overall performance in the long term.

Figure 16 describes how the network latency and the SKC hit rate are affected by the changes in the size of the SKC and the prefetch buffer. The SKC hit rate encompasses all hits both on the SKC and on the prefetch buffer. In Figure 16(a), we fix the size of the prefetch buffer to two, and increase the size of SKC. DCS-SB is used as the coscheduler in this simulation. We compare two cases: one with a prefetch buffer (/w) and the other without it (w/o). As expected, the latency

decreases and the SKC hit rate increases with the increasing size of SKC. Note that, with a bigger size of the SKC, the additional gain from the prefetch buffer is slightly decreased. This is because more session keys residing in a big SKC yields more hits on the SKC, consequently reducing hits on the prefetch buffer. Therefore, the prefetch buffer is more useful when the size of SKC is small. Figure 16(b) depicts how the SKC performance varies as the size of the prefetch buffer increases, with two different sizes of SKC, 5 and 20. In case of a 20-entry SKC, although the size of the prefetch buffer becomes big enough to contain more sets of prefetched sessions keys of previous SKC misses, the effect of old prefetched session keys is diminishing gradually. In the figure, although the prefetch buffer size increases by eight times from 1 to 8, there is a small performance gain, 0.17 μs in the latency and 2.8 percent in the SKC hit rate. All other schedulers except the gang scheduler showed the similar trend, so they are omitted in this paper.

Figure 16(c) and Figure 16(d) show the gang scheduler's results. To our surprise, as the size of the SKC reaches around 15, neither a prefetch buffer nor a larger SKC improves network latency. This is related to how the gang scheduler coschedules multiple applications. The gang scheduler usually makes all processes of an application in multiple nodes run concurrently, and then it moves to the next application after a fixed time-slice. Since all applications are executed in order, the session keys used by the previous application will not be used for a long time. In other words, once the size of the SKC becomes large enough to hold all session keys of an application, its hit rate will not increase much. This is why the increase in the SKC hit rate in Figure 16(c) is minimal. For the same reason, larger prefetch buffers make little difference in the network latency and the SKC hit rate as shown in Figure 16(d). From the above results, we conclude that the effect of the prefetch buffer varies according to the size of SKC and coscheduling policy.



(a) SGI Origin 2000 (b) IBM SP-2

Fig. 17. Cache hit rate of NAS benchmarks on a 16-KByte SKC.

4. SKC Size

To simulate the SKC behavior on a large-scale cluster system, the synthetic packet generator increases the total number of session keys in one CIC from 1,000 to 10,000. Although it is quite improbable for one process to communicate with 10,000 processes at the same time, this simulation can show how a small SKC withstands such extreme loads. The communication pattern between two processes follows the ON/OFF model as described in [51]. In Figure 17(a) and Figure 17(b), the 16-Kbyte SKC shows relatively high hit rates in most benchmarks, except for FT in SGI Origin 2000, and FT, CG, and MG in IBM SP-2. The reason for the exceptions is that the cache capacities of the benchmarks are less than 10,000 session keys as shown in Figure 11. It is noteworthy to point out that LU has a little lower hit rates, even with a large cache capacity in Figure 11. The reason for this is that LU has 10∼100 times shorter ON and OFF durations than those of other benchmarks, which means that communicating processes are alternating frequently, thus resulting in lower SKC hit rate. Still, we can say that a high cache capacity is strongly and positively related to a high SKC hit rate. If a large fully associative cache causes space and speed problems, we suggest that the CIC has a 16-way or 32-way SKC instead of a fully associative cache because they show similar results to that of the fully associative

SKC.

E. Related Work

There has been some research to improve security inside clusters. An early study [11] presented security enhancement methods for system area networks. [10] showed that encryption can be used in clusters with a minimal performance impact. [8] proposed the distributed security infrastructure (DSI) to supports a fine-grained cluster-wide security enforcement by providing a process-level resource and access control. [6] identified security characteristics unique to clusters. NVisionCC [90] was presented to enhance the security inside cluster systems by monitoring processes across cluster nodes. [91] proposed an instant attack stopper (IAS) scheme to instantly block inside attackers in InfiniBand cluster systems. Recently, [48] pointed out security vulnerabilities of InfiniBand and proposed a security framework that can be easily integrated into its specifications.

Some recent work has focused on the performance gain by use of the security coprocessor on real systems. [80] analyzed the relative cost of network security protocols, and advocated that the dedicated cryptographic hardware can make any security protocols practically viable. [78] investigated the performance overhead of TLS web servers using the host processor and the security coprocessor approaches. Due to the limited performance gain by the security coprocessor, they concluded that an additional host processor would be more beneficial than the security coprocessor. [81] presented an OpenBSD cryptographic framework. It contains a cryptographic API to provide a uniform access to various cryptographic hardware, thus enabling the high utilization of the hardware. [92] found that cryptographic hardware attached to I/O bus would not improve security performance due to the severe bus contention,

and then the authors proposed a scheme similar to the CIC approach, but they left out detailed performance evaluation. Several network adapters equipped with cryptographic hardware are available in the market [93, 94, 95]. However, since they were not developed for the ULC of cluster systems, and their detailed architecture is not publicly available, it is difficult to directly compare them with our approach.

F. Conclusions

This study proposed an architectural support to enable fine-grained secure communications in coscheduled cluster systems with low performance overhead. The important conclusions of this study are the followings. First, we advocated the CIC approach because it incurs virtually no performance overhead to the host CPU and no additional PCI transactions, while retaining the performance benefit of the ULC through bypassing the host CPU.

Second, to support fine-grained security, we proposed an SKC inside the CIC to store recently accessed session keys. If the communication pattern of a cluster application shows a high temporal locality, most of session key accesses will be found in the SKC, thus incurring little performance overhead in managing the fine-grained session keys. Simulation results showed that network latency is reduced by 50 percent on average, compared with non-SKC configurations.

Third, to further increase the SKC hit rate, we proposed a prefetch buffer to fetch session keys from memory ahead of time by predicting the job scheduler's decision. We implemented the Linux local scheduler and four coschedulers including SB, DCS-SB, PB, and gang scheduler. Simulation results showed that the prefetch buffer reduces network latency by 5 percent on average from that of SKC-only configurations.

Fourth, we constructed an analytical model to estimate the number of concurrent

processes that can be supported by a given size of SKC. The model showed that even a 16-Kbyte SKC yields a high hit rate even when managing thousands of session keys.

We are currently examining a number of possible extensions to this study. First, we are investigating the reason for the insensitive completion time. Second, we plan to implement the proposed architecture on a reconfigurable NIC. Finally, we are exploring other design alternatives to incorporate more security features in the CIC such as fast attack detection and response.

CHAPTER V

DESIGN OF SECURE SHARED MULTIPROCESSOR SYSTEM*

A. Introduction

Recent malicious attacks to many research/commercial servers have made protection and security essential requirements in all computer systems. Especially, server systems operated by institutes dealing with highly confidential data such as banks, military, or government agents need absolute security because even one incident of information leakage could result in very serious problems. Therefore, those institutes should seek effective schemes to prevent/block any possible security attacks on their computer systems. Considering that most of recent server systems have multiple processors and share single or distributed memory, we believe that there should be appropriate protections on the memory and communications betweeen processors. Moreover, since performance compromise should be the last resort in the server systems, any security schemes free of performance overhead are much desirable despite additional hardware costs.

There are two types of possible attacks: software and physical attacks. Software attacks exploit software vulnerabilities of the operating system (OS) and server applications such as web and database services. Buffer overflow attack is a good example [96]. Physical attacks that are our focus capture or modify data from the system memory or system bus through external devices physically attached to the system. Although high-end servers are often assumed to be secure against such physical at-

*Reprinted with permission from "I^{2}SEMS: Interconnects-Independent Security Enhanced Shared Memory Multiprocessor Systems" by Manhee Lee, Minseon Ahn, and Eun Jung Kim, 2007. Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT), pp. 94-103, Copyright 2007 IEEE.

tacks, that assumption cannot be sustained in many situations; it is possible that inside personnel turn against their organization to steal data, and, moreover, there exist real snooping devices that are attachable to system buses [97, 98]. To mitigate the physical attacks, memory encryption and authentication have been widely investigated for uniprocessor systems [59, 60, 61, 62, 63, 64, 65, 66]. However, the uniprocessor security models are not sufficient to design secure multiprocessor systems due to the absence of cache-to-cache communication protection.

When multiple processors share data, a cache coherence protocol guarantees data consistency among caches. Without protecting cache-to-cache communication, memory protection would be useless. A few studies raised this question and provided fast encryption and authentication techniques that can be deployed on bus-based shared memory multiprocessors [14, 13]. One common feature in the studies is that a bus sequence number that counts every message on a shared bus is used as a counter to generate an encrypted counter, called *keystream* [99]. Since all communications on the shared bus are broadcasted to all processors, they can predict the next bus sequence number to generate the same keystream in advance, thus turning a complex encryption/decryption operation into a simple XOR (eXclusive OR) operation. However, since the above schemes are dependent on the shared bus, they cannot be easily extended to general multiprocessor systems having different underlying networks or distributed shared memory [100, 101].

A recent study by Rogers, *et al.* proposed a novel mechanism for protecting cache-to-cache communication in distributed shared memory (DSM) multiprocessors as a remedy for this problem [67]. However, since their design focused on the directory-based cache coherence protocol used for the point-to-point communication, their idea cannot be directly applied to the systems with other cache coherence protocols used for multicasting/broadcasting communication such as the token coherence protocol

[102, 103]. For example, even in the directory-based cache coherence protocol, a multicast network is reported to improve system performance by up to 18 percent because the invalidation time is lessened by sending a multicast message to invalidate multiple cache blocks instead of sending multiple messages sequentially [104, 105, 106].

To provide diverse multiprocessor architectures with a more general security model, we propose an interconnect independent and cache coherence protocol independent security model for shared memory multiprocessor systems, referred to as Interconnects-Independent Security Enhanced Shared Memory Multiprocessor System (I^2SEMS). To our best knowledge, I^2SEMS is the first attempt to support the protection of shared memory multiprocessor systems without any restrictions on communication types and cache coherence protocols.

Our main idea is that a *Global Counter Controller (GCC)* assigns counters upon a request from a processor and broadcasts the assignment to the other processors. When receiving the counter assignment, processors generate the counters' keystreams and store them in a *keystream queue* or a *keystream pool*, depending on whether the keystream will be used for encryption or decryption. To minimize the use of new counters, a *keystream cache* stores both counters and keystreams so that it can re-use them only when the re-use is safe; the same counter and its keystream can be used for the same data block securely. With this architectural support, I^2SEMS can minimize enryption/decryption delay by providing timely available keystreams as well as protect communications by guaranteeing that different blocks never use the same counter.

We implemented I^2SEMS by using Simics along with Wisconsin multifacet General Execution-driven Multiprocessor Simulator (GEMS) in a hierarchical switched network [107, 108]. GEMS is a set of modules for Virtutech Simics to enable detailed cache, memory, and interconnection simulations. We use SPLASH-2 benchmarks

and one SPEC OMP benchmark for workloads on up to 16-processor shared memory multiprocessors [109, 110]. Simulation results show that the overall performance slowdown is 4 percent on average, and the keystream pool hit rate is as high as 78 percent, meaning that 78 percent of incoming messages are decrypted without delay. Furthermore, we observed that, due to the re-use of keystreams through the keystream cache, the increase of counter management related messages seems almost static compared to that of total messages especially in 16-processor multiprocessor systems. Based on this observation, we believe that I^{2}SEMS will be scalable to much larger multiprocessor systems.

The remainder of the chapter is organized as follows. Section B explains Galois/Counter mode and describes how the previous studies provided secure computing models in uni/multiprocessor systems. In Section C, we explain our security model and its design in detail. Security analysis and simulation results are presented in Sections D and E, followed by the concluding remarks in Section F.

B. Secure Computing Models

In this section, we describe our threat model, and explain how the previous works provided secure computing in uni/multiprocessor systems.

1. Threat Model

Before enhancing security of a system, it is necessary to classify secure and insecure components in the system. Processing core, registers, caches, and control and data paths in a processor are considered to be secure. We assume that memory and I/O controllers are secure, while other off-chip components including memory, memory bus, and I/O devices are insecure. We also assume that interconnection networks

that connect multiple processors and memory banks are insecure.

Through these insecure components, the following physical attacks are possible. First, any wiretapping device attached to memory bus or interconnection network can get information about memory transactions and processor-to-processor communications. Since in many cases cables of the interconnection network are exposed to the outside of the system to connect multiple nodes packaged in one or several racks, they are vulnerable to eavesdropping attacks. To make servers resistant to this attack, it is necessary to provide the confidentiality service where all communications from processors are encrypted. Second, assuming that attackers are much more determined and well equipped, they can further inject or modify messages to the systems. This attack includes injecting new data messages, modifying data in messages in transit, and replaying old messages. To prevent this attack, the authentication service needs to make sure that all data messages are genuine, not spoofed, modified, or replayed by illegal devices. Third, in some cases attackers may intend to undermine the availability of the systems by keeping injecting garbage messages. These messages will be finally discarded since authentication information will not be correct. but they will have negative effects on the system performance because of possible congestion that would occur in interconnection network or controllers for cache and memory. This is similar to the denial of service attack in the Internet. However, this attack does not seem much attractive to attackers not only because attackers get little benefit from the attack, but also because a sudden performance drop or a system crash would lead to a thorough search for attached devices. Therefore, the availability service is not our focus. From now on, we will give an overview how the previous research provided the confidentiality and authentication services in uniprocessor and multiprocessor systems.

2. Uniprocessor Secure Model

Several uniprocessor memory authentication schemes were proposed in [59, 61, 64, 65]. XOM (eXecute Only Memory) uses MAC to verify each memory block's integrity [61]. MAC for each block cannot defend against replay attacks where a hacker gets a valid memory block and keeps resending it. Gassend, *et al.* proposed a hash tree to guarantee the integrity of the whole contents of memory [59]. While it removes the replay attacks, a hash-tree has relatively high run-time overhead because it should check memory integrity for every memory access with a logarithmic number of integrity checks. To overcome this performance overhead, Suh, *et al.* suggested that MAC verifies a series of memory operations using multiset hashing functions, resulting in only 5 percent performance slowdown [64]. Moreover, Yan, *et al.*, who first introduced GCM to this research area, minimized authentication overhead by authenticating all necessary levels of a hash tree in parallel [65].

For uniprocessor memory encryption, early studies used ECB mode [61, 63], but later Counter mode was adopted for performance reasons [62, 64, 65, 66, 111]. In the uniprocessor secure computing model, decryption speed is more critical than encryption speed because decryption should be performed quickly so that the processor can use it for execution as early as possible. In contrast, the encryption delay of evicted data is not time-critical because the data is first written in a write buffer and later stored back to memory.

To speed up decryption, Yang, *et al.* suggested using an additional cache to store counters [64, 66]. While a processor is waiting for a reply after sending a request to memory, it can precompute a keystream by using a stored counter. To alleviate this cache space overhead, a prediction scheme proposed by Shi, *et al.* precomputes incremental counters by utilizing the principle of locality in memory access patterns

[62]. With an optimization scheme, they can predict keystreams with up to 99 percent accuracy. Yan, *et al.* combines encryption and authentication using GCM, resulting in 5 percent performance overhead [65].

3. Multiprocessor Secure Model

For multiprocessor shared-memory protection, it is possible to apply uniprocessor security schemes, but cache-to-cache communications need a different protection scheme. Unlike uniprocessor secure computing models, encryption and generation of MAC in multiprocessor systems become time-critical because a receiving processor may stall to wait for a reply. As for authentication of cache-to-cache communications, Shi, *et al.* proposed an *authentication speculation execution* to remove MAC latency from the critical path [14]. In this scheme, while the receiver verifies an incoming message, it continues to execute speculatively by using un-authenticated data. Those executions are committed only after all operands become authenticated. This scheme reduces performance overhead by overlapping authentication and CPU execution. However, each processor needs a complex speculation circuit, and this scheme is still vulnerable to replay attacks. Zhang, *et al.* used Cipher Block Chaining (CBC) mode in which the previous MAC is used to make the next MAC, thus preventing replay attacks [13].

Rogers, *et al.* pointed out the limitation of above schemes on DSM systems and proposed an efficient data protection design [67]. By focusing on point-to-point communications of the directory-based cache coherence protocol, they were able to utilize DSM systems' temporal locality of communications, which means a processor communicates with a relatively small number of neighboring processors in a short period of time. Such locality makes it possible for each processor to have a small table to hold counters, resulting in good scalability.

Please note that, in multiprocessor shared-memory protection, all processors and related components like the memory controller need to share the same secret key. This key can be fabricated inside processors and a memory controller from factory, or a runtime distribution method is available as described in [112]. Since we assume that processors and the memory controller are secure, capturing, snooping, or predicting the secret key is impossible. Furthermore, even if an ASIC or FPGA is hooked up to the system and pretends to be a peer processor in the multiprocessor systems, it cannot break the privacy and integrity of the system since it is practically impossible for the illegal device to have the same secret key.

C. Architectural Design of I^{2}SEMS

In this section, we focus on the architectural design of I^{2}SEMS providing contiguous counters so that it is easy to predict the next counters. First, we discuss design considerations of I^{2}SEMS. Then, we present the design overview of I^{2}SEMS, followed by detailed explanation on each component.

1. Design Considerations for I^{2}SEMS

In this section, we provide the rationale behind the current I^{2}SEMS design. The first design consideration is to decide whether messages should carry counters. In previous studies on secure bus-based systems [14, 13], cache-to-cache messages do not carry counters because receiving processors can always correctly predict the next counter. However, such strict global synchronization is almost impossible in general shared memory systems. Therefore, although it will incur additional overhead, in our scheme each message carries its own counter. The overhead will be different depending on implementation of interconnection networks. For example, if a wide

channel bus is used, an extra channel is necessary. In a network using serial links, the message serialization latency will be increased by the time to serialize the counter.

The second design consideration is to decide whether to maintain multiple local counters or one global counter. In the local counter management, the whole counter range is divided into exclusive subranges that are assigned to the processors so that each processor manages its own local counter, without worrying about the duplicate use of the same counter. A disadvantage of this management is that adjacent blocks would have discontinuous counters, when multiple processors share and update those blocks. This discontinuity will result in slow decryption since predicting the next counter, based on the current memory block's counter, is often incorrect. In the global counter management, in contrast, a central counter controller assigns counters upon a counter request, thus resulting in relatively contiguous counters. To improve both counter continuity and prediction correctness, the global counter management is desirable, and thus we propose GCC. Since a request to the GCC should be made early enough to finish generating new keystreams before their usage, a keystream queue is necessary to hold the precomputed keystreams. A completely different design alternative without using counters is possible by encrypting and authenticating messages in all point-to-point links. However, this approach incurs additional encryption/authentication delay per hop, thus increasing the total network delay significantly in large networks. Power/energy consumption will also increase linearly depending on the number of hop counts. Therefore, this approach will not be suitable for the multiprocessor environment.

The third design consideration is to decide whether to use counters persistently or compulsorily. In the persistent scheme, each processor always uses new counters to encrypt cache blocks regardless of their cache status. Most of previous studies consume counters in this way. However, we can make use of the fact that it is not

necessary to use new counters all the time because a previously used counter can be re-used securely if a cache block has not been changed since the first use of the counter. We refer to this as compulsory scheme. The benefit of this scheme is that it will consume less number of counters than the persistent scheme, thus reducing the number of counter request messages. This is why I^{2}SEMS stores used counters and keystreams in a keystream cache.

The last consideration is to decide when and how to start counter prediction: responsive or broadcast counter prediction. In the responsive prediction, a receiving processor starts to predict subsequent counters only after the processor receives a message carrying a counter. A similar approach was introduced in the uniprocessor prediction scheme [62]. The prediction correctness drops inevitably in data sharing intensive applications because it is hard to predict which counter will be used next. However, since we use the global counter management, a counter request to the GCC hints that the requesting processor is actively consuming new counters and likely to use the newly assigned counters shortly. In the broadcast counter prediction, the GCC broadcasts the global counter to all other processors, and the processors precompute keystreams based on the global counter. Although there is no guarantee that messages carrying the new counters will arrive at the processors, if so, the prediction correctness will increase. Since the responsive and broadcast schemes are orthogonal to each other, we choose both of them to increase a prediction hit rate. To store these precomputed keystreams, we use a keystream pool in each processor.

2. Design Overview of I^{2}SEMS

Figure 18 shows the architecture of I^{2}SEMS and its data flow. The system-wide GCC

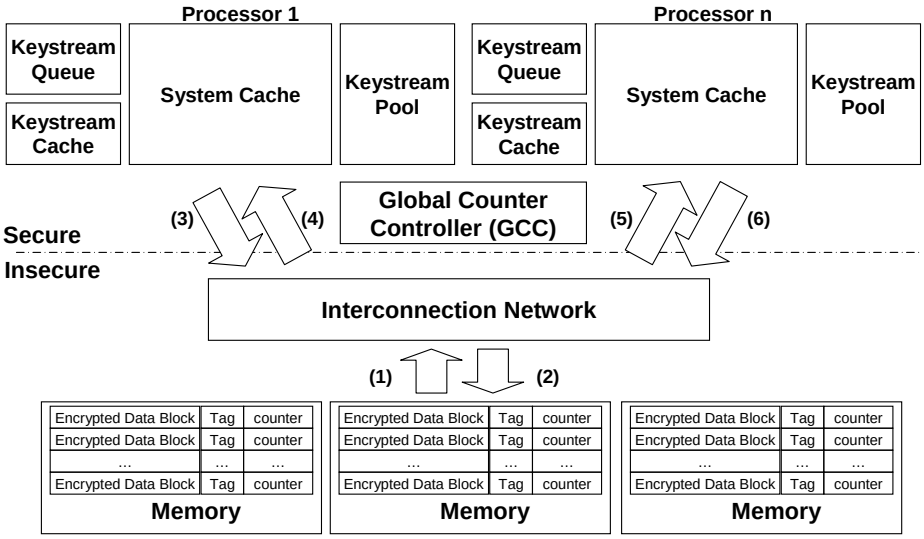


Fig. 18. I^{2}SEMS security model.

is located in the memory controller[†]. In addition to a unified L2 cache, called System Cache, each processor has a keystream queue, a keystream cache, and a keystream pool. The GCC assigns counters upon a request from a processor and broadcasts the assignment to the other processors. Keystreams in the keystream queue and the keystream cache are used to encrypt outgoing messages, while those in the keystream pool to decrypt incoming messages. Explanations on data flow between components are in the followings.

When a processor transmits a data block to other caches or memory as shown in Figure 18 (3) and (6), it needs to encrypt the block. If the cache block is modified or exclusively owned by the processor, a new counter and its keystream are popped

[†]Depending on the location of the memory controller, the GCC can reside off or on chip. When a processor has an off-chip memory controller like most Intel processors, a multiprocessor system using such processors will have the GCC in its off-chip memory controller usually located in the north bridge (or Memory Controller Hub). However, some recent processors such as IBM's POWER5 and AMD's Athlon 64 and Opteron processors have the on-chip memory controller for faster memory access. In this case, by enabling one processor's GCC, a single GCC will be located in an integrated on-chip memory controller.

from the keystream queue to encrypt the cache block. In many cases, the cache status changes to *Owned*, which means that this processor is in charge of replying requests that other processors send to get this block's copy. In this status, the previously used counter and keystream can be re-used because the data did not change. For this purpose, after using the new counter and its keystream, we store them in the keystream cache. Therefore, when a cache block that will be transmitted has the Owned status, the processor looks up the keystream cache. If not found, a new counter and its keystream need to be popped from the keystream queue. This operation is described in Algorithm 1 (3) and (6).

```
(1)
(EncryptedDataBlock, counter) = Memory.read(address);
Send(address, EncryptedDataBlock, counter);

(2)
(address, EncryptedDataBlock, counter) = Receive();
Memory.write(address, EncryptedDataBlock, counter);

(3,6)
(DataBlock, counter, Keystream, status) = SystemCache.Read(address);
if (isNewKeystreamNecessary(status)) {
        (counter, Keystream) = KeystreamQueue.pop();
        Send(address, DataBlock ^ Keystream, counter);
        KeystreamCache.store(counter, Keystream);}
else if (KeystreamCace.IsKeystreamPresent(counter)) {
        (counter, Keystream) = KeystreamCache.read(counter);
        Send(address, DataBlock ^ Keystream, counter);}
else   {(counter, Keystream) = KeystreamQueue.pop();
        Send(address, DataBlock ^ Keystream, counter);
        KeystreamCache.store(counter, Keystream);}

(4,5)
if (Data block arrived) {
[t]     (address, EncryptedDataBlock, counter) = Receive();
        PARALLEL {
                // Generate p Keystreams from counter to counter + p ^ 1
                counter[0,...,p-1] = {counter,..., counter+p-1};
                Keystream[0,...,p-1] = AES(counter[0,...,p-1], secretkey);
                KeystreamPool.store(counter[1,...,p-1], Keystream[1,...,p-1]);}
        PARALLEL {
                if (KeystreamPool.IsKeystreamPresent(counter)) {
                        Keystream = KeystreamPool.read(counter);
                        DataBlock = EncryptedDataBlock ^ Keystream;
                        SystemCache.store(address, DataBlock);
                        KeystreamPool.delete(counter);}
                else { //wait until for an AES latency
                        DataBlock = EncryptedDataBlock ^ Keystream[0];
                        SystemCache.store(address, DataBlock);} } }
else if (GCC reply arrived) {
        (counter) = Receive();
        counter[0,...,CR-1] = {counter-CR+1,..., counter};
        Keystream[0,...,CR-1] = AES(counter[0,...,CR-1], secretkey);
        KeystreamQueue.enqueue(counter[0,...,CR-1], Keystream[0,...,CR-1]);}
else if (GCC broadcast arrived) {
        (counter) = Receive();
        counter[0,...,CR-1] = {counter-CR+1,..., counter};
        Keystream[0,...,CR-1] = AES(counter[0,...,CR-1], secretkey);
        KeystreamPool.store(counter[0,...,CR-1], Keystream[0,...,CR-1]); }
```

Algorithm 1. Pseudocode for I^{2}SEMS security model

When a processor receives a data block as shown in Figure 18 (4) and (5), it

tries to find a keystream in the keystream pool. Simultaneously it begins to generate p keystreams by using the pipelining ability of AES logic. If found in the keystream pool, referred to as a *keystream hit*, the counter's keystream is used to decrypt the arrived message. If not found, called a *keystream miss*, the keystream will be available in an AES latency because the keystream generation already started regardless of keystream hit or miss. All generated keystreams except the first one are stored in the keystream pool, hoping that subsequent counters may be used in the near future. p is called *prediction depth* and set to five as default. The impact of deep prediction depth on performance is analyzed in Section 5.

When the remaining number of keystreams in the keystream queue becomes less than *counter reserve* (CR), it sends a request to the GCC to get new counters. CR is the number of counters that the GCC assigns at a time. More details on CR are described in Section 3. Upon arrival, the GCC assigns and sends new counters to the requesting processor. The GCC also broadcasts the assignment to the other processors for them to precompute the counters' keystreams. When a GCC reply or a broadcast message arrives at a processor in (4) and (5), the processor generates the CR number of keystreams and store them in the keystream queue or the keystream pool, respectively. This operation is summarized in Algorithm 1 (4) and (5). All other operations shown in Figure 18 are also described in Algorithm 1.

3. GCC and Keystream Queue

Since the GCC and the keystream queue work interactively to enable the efficient counter assignment, we explain two components together. Figure 19 shows the detail architecture of keystream queue, keystream cache, and keystream pool in I^{2}SEMS. When a data block is to be transmitted, the cache status bits select one keystream from the keystream queue or the keystream cache. Since the system cache and the

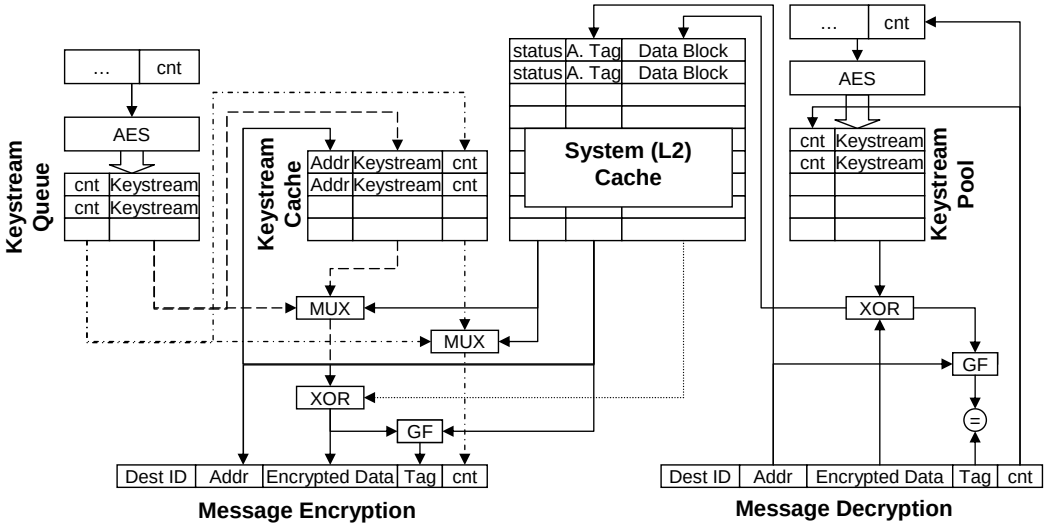


Fig. 19. Architecture of keystream queue, keystream cache, and keystream pool.

keystream cache are accessed in parallel, the access time to the keystream cache is not in the critical path. Note that I^{2}SEMS checks the cache block status only, which means I^{2}SEMS works independently of any cache coherence protocol as long as the protocol changes cache block status correctly.

To prevent a situation where the keystream queue has no available keystreams, it sends a request to the GCC early enough to guarantee the keystreams' availability all the time. Appropriate counter request timing is closely related to the number of messages that the processor can send in a round trip time to the GCC. Equation 5.1 defines CR formally where R is the round trip time to the GCC, O is the keystream generation delay, M is the cache block size, and B is the network bandwidth. If the number of remaining keystreams become less than CR, the keystream queue sends a request to the GCC. To decrease the number of requests, the GCC assigns a block of counters per request instead of one counter. We suggest that the size of the block be also CR because the keystream queue can send only one request when the number of remaining counters becomes less than CR. Thus, the size of the keystream queue

only needs to be large enough to hold $2 * CR$ keystreams and counters.

$$\begin{aligned} R + O &\leq \frac{CR * M}{B} \\ CR &\geq \frac{B}{M} * (R + O) \end{aligned} \tag{5.1}$$

Possible security problems on messages to and from the GCC will be discussed in Section D. Besides security problems of the GCC, its scalability could be a problem since I^{2}SEMS has only one GCC. There are three possible problems; long latency between processors and the GCC, bottleneck of the GCC, and high traffic overhead of broadcasting of counters. The long latency problem can be solved by adjusting *CR*. In Equation 5.1, the increased R will change *CR* so that it will guarantee the availability of keystreams in the keystream queue. As for the bottleneck problem of the GCC, since the GCC performs simple operations such as managing a counter and replying to processors, the GCC will not be a bottleneck in communications. If the underlying network does not support broadcasting or multicasting, the GCC should generate and send all messages one by one, possibly causing some delays in broadcasting. However, this delay will not be much critical to performance because the newly assigned counters are not needed until the requesting processor uses up all (*CR*-1) keystreams. Moreover, as we will show in Section 6, the total number of requests to the GCC and its broadcasting messages is very small, as compared to the total number of transmitted messages by processors. Therefore, we believe that I^{2}SEMS is scalable to large systems.

4. Architecture of Keystream Pool and Keystream Cache

Both the keystream pool and the keystream cache have the cache architecture. The keystream pool size will affect the system performance. If its size is too small, the

keystream pool hit rate will decrease due to capacity misses, but, if the size is too big, the long keystream pool access time increases the total decryption time. To minimize performance slowdown and get a high hit rate at the same time, we will use the largest possible cache architecture as long as its access time is hidden from the system cache access. According to CACTI model [44], one cache access consists of several circuit level delays. The following equation shows that a cache data access time can be divided into address decoding, wordline, bitline, and sense amplifier time, which occur sequentially.

$$T_{data} = T_{decode} + T_{wline} + T_{bline} + T_{sense} \tag{5.2}$$

T_{decode} is the address decoding time to access a specific cache set in the cache. This stage requires the address only, so we can utilize T_{decode} to parallelize accesses to the system cache and the keystream pool. In other words, if the keystream pool access time, $T_{access(KeystreamPool)}$, is equal to or smaller than the system cache's address decoding time, $T_{decode(SystemCache)}$ as in Equation 5.3, we can hide the keystream pool access time. In our experiment, we used a 1M bytes 4-way system cache with $6ns$ access time. Since its T_{decode} is $3ns$, we use a 512K bytes 4-way keystream pool with $3ns$ access time for the keystream pool, which meets the requirement of Equation 5.3. The effect of set associativity on the system performance is analyzed in Section **??**. Note that the six clock cycle authentication delay is not considered here. That is because the authentication is not in the critical path for program execution. Therefore, I^{2}SEMS issues a lazy authentication fail alert.

$$T_{access(KeystreamPool)} \leq T_{decode(SystemCache)} \tag{5.3}$$

The high hit rate of the keystream cache will reduce the counter usage effectively. Thus, the design of the keystream cache is tightly related to the scalability

of I^{2}SEMS. Different from the keystream pool, the keystream cache can utilize the temporal locality of memory accesses. When a cache block is being shared by multiple processors actively, they are more likely to access the block within a short time than later. In our simulation, we observed that, as the size of cache increases, the keystream cache hit rate also increases, but the effect of larger caches is gradually diminishing. Therefore, we believe in most cases even a small keystream cache can yield a relatively high hit rate.

D. Secure Communications of I^{2}SEMS

Until now, we proposed an architectural design of I^{2}SEMS and showed how it can predict and precompute next counters. In this section, we explain how I^{2}SEMS provides secure communications. First, we discuss how data messages are protected in detail. Then, we describe methods to protect other messages such as GCC messages and control messages.

1. Protection on Data Messages

By architectural support of I^{2}SEMS, we can assume that a sender and its receiver share the same counter. Even if the counter is not correctly predicted in the receiving node, it will be available soon when the message arrives. Figure 20 shows how to encrypt a data packet carrying a 32-byte plaintext and generate a message authentication codes (MAC) using GCM. Decryption has a similar design. Since the block size of AES is 128 bits, the 32-byte plaintext is divided into two sub-plaintexts, Plaintext_1 and Plaintext_2. Larger cache blocks will be divided into more number of sub-plaintexts. The two plaintexts are encrypted in parallel by XORing with two keystreams, making two ciphertexts, Ciphertext_1 and Ciphertext_2. The ciphertexts

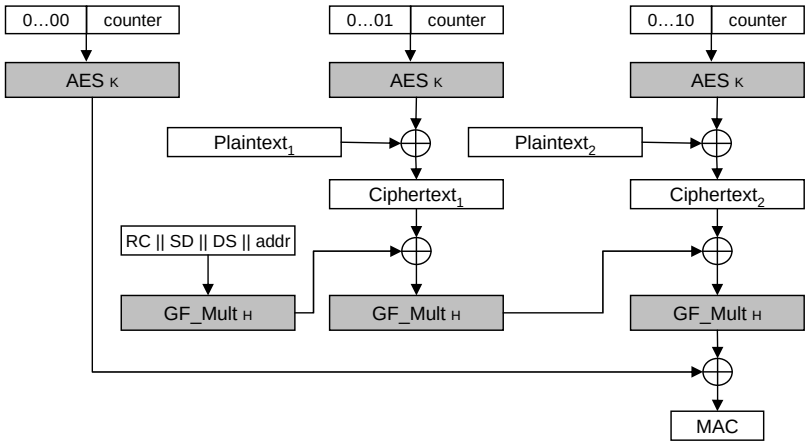


Fig. 20. Galois/counter mode

are concatenated to make 32 bytes encrypted payload. GF_Mult_H denotes multiplication in $GF(2^{128})$ using a hash key H, which is the encrypted zero counter, $AES(0^{128}, K)$, with a secret key K.

• Encryption of Data Message

Since AES is considered to be secure without any serious weakness until now, the plaintext will be kept secret as long as all input counters of AES are unique for different data. AES_K is hardware implementation of AES algorithm with a secket key, K. Note that in Figure 20 one 64-bit global counter is used to compose three 128-bit counters to generate three different keystreams. In this example, I^{2}SEMS appends three different prefixes: $0^{62}00$, $0^{62}01$, and $0^{62}10$, by using the 62-bit common prefix. If the 64-bit counter is globally unique and all components including processors and the memory share the same 62-bit common prefix, every counter used as an input of AES is unique in the whole system. With regard to the uniqueness of the 64-bit global counter, we already described how I^{2}SEMS can accomplish it in the previous section. Secure sharing of secrets like the common prefix was fully investigated in [112], so we can assume that I^{2}SEMS can share the common prefixes securely. To prevent

the global counter from starting with the same number at every booting stage, the common prefix will be increased by one, and distributed securely to all processors and the memory.

The last concern of confidentiality of data messages is counter-wrap-around. If the 64-bit global counter wraps around, two different data blocks will use the same keystream. We estimate the expected wrap-around time by considering the maximum counter usage rate of a system. Let's suppose a system has 2^6 processors using a 3.2G bytes/sec network and a 32 byte-long message and counter is 64 bits long. When one half of the processors keep sending modified cache blocks at its full speed and the other half are receiving them, this system will consume 2^{32} keystreams per second, thus taking about 2^7 years to wrap around the counter in the worst case. Considering that I^{2}SEMS assigns new counters only when necessary, we expect the actual counter wrap around time will be far longer than the estimation. Therefore, the 64-bit counter will be big enough to support system lifetime of fairly large multiprocessor systems. Even if a huge multiprocessor system runs for exceptionally long period of time enough to reach its theoretical wrap-around-time, the system can be simply. Therefore, I^{2}SEMS can protect data messages from eavesdropping attacks trying to recover all or partial information of plaintexts.

• Authentication of Data Message

Since the security strength of GCM is the same as the strength of its block cipher [23], the authentication strength is greatly improved. Owing to this, any modification and spoofing attack will not be successful because the hacker does not know the secret key K. Additionally, the source node (SN) and destination node (DN) addresses are input of GF_Mult_H, so the redirection attack is not possible, either.

The last authentication concern is replay attacks on data messages. According to why a data message transfer is necessary, there are different replay attack scenarios

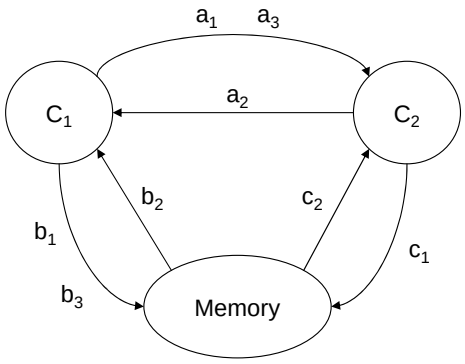


Fig. 21. Data messages transfer

as shown in Figure 21. The first relay attack point is on data messages between two L2 caches, which are numbered as a_1,a_2, and a_3. When an L2 cache read miss occurs in C_1 and its authoritative copy is stored in other L2 cache, C_2, a read request (a_1) is delivered to C_2. C_2 sends the requested block (a_2) to C_1, and then C_1 sends an acknowledgement message (a_3) to C_2. A possible replay attack is as follows. Suppose a hacker captures two messages a_1 and a_2 for a specific address. Later, if C_1 evicts the cache block and needs it again, C_1 will send a new request a_1' to C_2 with the same address. The hacker can drop a_1', replay a_2 to C_1, and drop a new acknowledgement a_3'. This attack will be successful because a_2 has a correct MAC with the same node addresses and memory address. To prevent this attack, we propose that every read request from an L2 cache should need to carry a unique sequential number, referred to as *Request Counter* (RC). Whenever a read request is sent out, the RC is increased and attached to the request message. In the above example, a_1 will carry RC_{C_1}. C_2 includes RC_{C_1} in generation of its MAC. This is why the first input of GF_Mult_H is $RC_{C_1} \parallel SN \parallel DS \parallel addr$. Let's look at the replay attack again. a_1' will carry a new RC, RC'_{C_1} that is different from RC_{C_1} used in a_1. The authentication of replayed a_2 will not be successful because C_1 expects an MAC generated with $RC'_{C_1} \parallel SN \parallel DS \parallel addr$ but the replayed a_2 carries an MAC generated with $RC_{C_1} \parallel SN \parallel DS$

$\| \ addr$. Since C_1 keeps increasing RC_{C_1} per memory request, no two RC_{C_1} will be the same within its wrap-around-time. To estimate the wrap-around-time of RC, we need to fix the number of nodes and the size of memory. Suppose there are 2^6 processors and the memory size is 2^8 GB. Since $\mid SN \mid$ is 6 bits, $\mid addr \mid$ is 38 bits, and the input width of GF_Mult_H is 128 bits, $\mid RC \mid$ can be 78 bits. Even if RC's width is 59 bits and a cache keeps sending 32-bits messages using a new RC in a 3.2 Gbytes/sec network, its wrap-around-time is approximately 2^7 years like that of the global counter. Thus, the first 19 bits can be used as a prefix that is incremented by one at the booting stage as we did for the global counter. After 2^{19} rebootings, we can see the same RC from the same cache, which will be almost impossible to happen in real systems. Therefore, our scheme can prevent replay attacks on data message between caches.

The second relay attack point is on data messages from the memory to an L2 cache, which are numbered as b_1,b_2, and b_3. When C_1 wants to access the memory for read, a memory request is delivered to the memory (b_1). The memory sends the requested block (b_2), and then C_1 sends an acknowledgement message to the memory (b_3). Since communication steps are exactly the same as those between two caches, replay attack scenario and its prevention are also the same.

The third replay attack point is on data messages from an L2 cache to the memory. The transfer happens when a modified block is evicted from the cache and written back to the memory. This replay attack is different from the two previous attacks because there will no read request such as a_1 and b_1. C_2 directly sends a write-back request with data (c_1) in Figure 21. A possible replay attack is as follows. Suppose a hacker captured c_1 for a specific address. Then, the hacker can recover the old data anytime by replaying c_1. Since c_2' will also be dropped, the attack will go unnoticed by cache coherence protocols. To prevent this replay attack, we insert the

RC in the request as well as use RC to make an MAC. That is, C_2 use $RC_{C_2} \parallel SN \parallel DS \parallel addr$ for its MAC, and c_1 carries RC_{C_2}. The memory should remember each cache's last RC so that it can raise a security alert when an arriving RC is smaller than or equal to the last RC. Therefore, I^{2}SEMS prevents the replay attacks on data messages.

For the complete protection of memory, we additionally adopt Merkle tree or a recently proposed Bonsai Merkle tree [65, 113]. Even though it is out of our research scope, Merkle tree can prevent memory authentication attacks like memory block replacements. Note that since I^{2}SEMS prevents replay attacks, we can ease a hard requirement of Merkle tree authentication that a root authentication code needs to be stored in processor/cache that is assumed to be secure. In our design, as long as the secure memory controller can store the root authentication code, making sure that the memory contents are not modified or replaced through physical attacks, I^{2}SEMS can guarantee secure communications between caches and memory. This will remove possible overhead for updating the root authentication codes in multiple processor systems at every memory writing.

Until now, we implicitly assumed that the underlying general interconnect provides in-order delivery and there is no packet loss because we simply compared the most recent RC and a newly arrived RC. In larger multiprocessor systems, however, out-of-order delivery and message loss are possible. For example, two invalidation requests from the same node to the same destination node can arrive out of order by taking different paths in adaptive routing schemes, meaning the smaller RC can arrive later. The strict requirement of monotonic increasing RC will not be working in out-of-order delivery interconnects. Note that, however, the replay attack will be only successful when a hacker sends exactly same copies of previous messages, which means that the replayed message's RC already arrived before. In other words, out-

of-order delivery will cause skipped numbers in received RC while replay attacks will result in duplication among received RCs. For this, each cache needs to maintain history information of other caches' RC. The maximum delay of out-of-order delivery and message retransimission time will decide the size of the storage. Since most multiprocessor systems provide highly reliable interconnects, it seems less likely to have a larger number of out-of-order or lost messages for long period of time. Therefore, the space overhead of the stoage will be small.

2. Protection on Control Messages

Control messages like invalidation or write requests need appropriate protection for the follwoing reason. In Figure 21, suppose that C_2 is an exclusive owner of a cache block of an address. A hacker spoofs a write request with C_1 as the source node and C_2 as the destination node. C_2 would invalidate its block by understanding that C_1 wants to have an owenership to overwrite the block. This attack results in cache incoherence because the cache coherence protocol considers C_2 as the owner of the address, but C_2 already invalidated the block. For this, we propose to authenticate control messages by using local counters. Each cache has its own 64-bit counter and increases it by one at every control message. This local counter instead of the global counter is used as an input of AES_K in Figure 20. Since there is no data to encrypt, Plantext_1 and Plantext_2 will be simply 0^{128}. Instead, two ciphertexts will be used only for generation of MAC. Still, the input of GF_Mult_H is $RC \parallel SN \parallel DS \parallel addr$, and the message needs to carry RC. Like the global counter, the local counter will be initialized at every booting stage to prevent every local counter from starting with the same number. Note that we do not consider performance implication of authentication of control messages. That is because we can presume this will not cause significant performance overhead by using the combination of local counter prediction

and speculative authentication. As we did in the global counter, each cache can predict the next local counter and precompute its keystream so that it can instantly generate incoming control message's MAC. According to [67], there is a temporal locality in cache coherence communications, meaning that a cache communicates with relatively small number of caches at a time. Therefore, the prediction hit rate is expected to be very high. Furthermore, in case of a local counter miss, which could incur an authentication delay, we use authentication speculation. In other words, when a keystream to authenticate a control message is not available, instead of waiting for the generation of the keystream, the receiving cache simply checks *RC* first, and then process the control message as requested. Considering that the authentication will be successful when there is no security attack, the authentication of control messages will not affect the overall performance.

3. Protection on Counter Messages

Since the whole security mechanism depends on the counters assigned by the GCC, the protection of the GCC and its counter distribution is critical. Therefore, the GCC itself needs to be tamper-resistant so that its operation and secret keys will be protected. So, we need to provide appropriate protection on the counter messages. to remove any possibility of modifying, forging, or replaying attacks,

First of all, counters carried on counter messages need authentication only, not encryption, because even if the counter is available to a hacker he cannot generate or predict any keystreams without knowing a secket key in use. For authenticating counter messages, we propose a new counter, *GCC Counter* (*GC*), similar to the local counter in protection of control messages. When a keystream queue needs more global counters, it sends a counter request with an *RC* to prevent a possible replay attack on counter request and reply messages. Upon a counter request, the GCC first

makes a reply message to the requester. The input of AES_K uses GC, Plaintext_1 is a new global counter, Plaintext_2 is zero, and the input of the input of GF_Mult_H is $RC \parallel GCC \parallel DS$. Then, the GCC makes a broadcast message to all keystream pools. The input of AES_K uses $GC + 1$, Plaintexts are the same, and the input of the input of GF_Mult_H is $GCC \parallel *$. * is a special character for a broadcasting address. The GCC increases GC by two for next counter requests. A replay attack on counter broadcast messages does not look interesting to a hacker since it will end up with broadcasting an old counter that can be simply dropped by comparing the arriving global counter and the most recent global counter. Out-of-order delivery interconnects may allow a broadcast message to arrive earlier than its reply message or its previous broadcast messages. In the former case, the cache keeps waiting for a reply message by discarding the broadcast message. In the latter case, if there is skipped global counters in the broadcasted messages, keystream pools assume that out-of-order delivery or packet loss happens, so it can precompute all the global counters' keystreams including the skipped counters. With regard to performance of authentication of GCC messages, keystream queues and keystream pools authenticate most GCC messages with little delay by correctly predicting GC because the GCC keeps broadcasting GC to all processors.

E. Performance Analysis

1. Simulation Framework

Simulator: We evaluate the performance of I^{2}SEMS by using the Simics full-system multiprocessor simulator developed by Virtutech AB [107]. To simulate general shared memory systems and various cache coherence protocols, we use GEMS as an extension of Simics [108]. We developed I^{2}SEMS in four cache coherence protocols:

Table V. Processor model parameters

Parameters	**Values**
CPU	1 GHz
L1 I-Cache	128K bytes, 4-way, 2ns latency
L1 D-Cache	128K bytes, 4-way, 2ns latency
L2 Cache	1M bytes, 4-way, 6ns latency
Cache Block Size	32 bytes
Keystream Pool	512K bytes, 4-way, 3ns latency
Keystream Cache	32-entry fully associative, 2ns latency
Memory	2G bytes, 80ns
network link bandwidth	3.2G bytes/sec
AES latency	80ns
AES throughput	3.2G bytes/sec

broadcasting, directory, hammer, and token. Broadcasting and directory coherence protocols are traditional protocols used in current shared memory multiprocessor systems. Hammer cache coherence protocol is an approximation of AMD's Hammer protocol used in AMD multiprocessor systems [114]. Token coherence protocol recently proposed by Martin, *et al.* is a low latency cache coherence protocol enabled by decoupling performance and correctness [103].

Configuration: We configured Simics as a Sun Fire server with UltraSPARCIII+ processors and Solaris 9 OS using the parameters shown in Table V. Since the decoding time of the unified L2 cache is approximately $3ns$, we use 512K bytes keystream pools with $3ns$ access time to parallelize a keystream pool access with an L2 cache access. The keystream cache is 32-entry fully associative cache with $2ns$ hit latency. Due to the recent development of AES implementation, a 128 bit AES unit can pro-

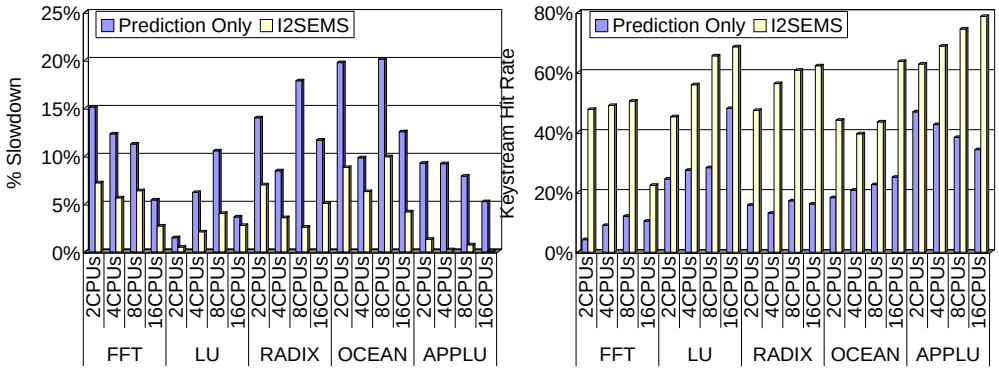
duce 30∼70 Gbps using 0.18μm CMOS technology. In our experiment, default AES latency is 80ns and throughput is 3.2G bytes/sec [62, 13, 43]. CR is calculated to be 32. We configured the network using hierarchical switches with fanout degree of 4, which is default in GEMS [108]. Since in our simulation the network latency to transmit a 32 bytes cache block is 10ns, the six clock cycle delay for GF multiplications discussed in Chapter II can be overlapped without causing an additional delay.

Benchmarks: We used four SPLASH-2 benchmarks; FFT, LU, RADIX, and OCEAN with their typical settings as described in [110] and one SpecOMP2001 benchmark, APPLU.

2. Overall Performance Slowdown

We ran five benchmark programs using two secure configurations and one non-secure configuration; one secure configuration is I^{2}SEMS and the other is *prediction only* scheme to compare with I^{2}SEMS. Similar to Counter mode proposed by Shi, *et al.* [62], the prediction only scheme has a keystream generator and a small number of keystream buffers. Regardless of a keystream pool hit or miss, the keystream generator keeps generating p keystreams by using the next counters. We assume that, in the prediction only scheme, each cache controller knows its own exclusive range of counters.

Figure 22 (a) shows the overall performance slowdown normalized to the execution time of the non-secure configuration. As compared to the baseline configuration with no security measures, I^{2}SEMS incurs around 4 percent performance slowdown on average. It is clear that I^{2}SEMS causes less performance slowdown than the prediction only scheme. However, it is hard to find a common trend in performance slowdown among varying number of processors. That is possibly because the execution time can be easily affected by other factors. Since we use the non-deterministic



(*a*) Performance Slowdown (*b*) Keystream Pool Hit Rate

Fig. 22. Overall performance.

full-system architectural simulator and multithreaded workloads, the execution path of each benchmark program is not always the same. Thus, a small difference in the selection of OS scheduling may cause large variability in the execution time. The similar effect was discussed by Zhang, *et al.* [13]. Instead, we use the keystream pool hit rate for performance comparison not only because it is less variable than execution time, but also because the high hit rate is positively correlated with performance.

Figure 22 (*b*) shows keystream pool hit rates of five benchmark programs. The hit rates vary from 22 percent to 78 percent, but note that in all benchmarks I^{2}SEMS has higher hit rates than the prediction only scheme. This is because the prediction and broadcast schemes adaptively yield a high keystream pool hit rate in both memory-read and memory-write dominant applications.

We would like to emphasize that the counter prediction of a uniprocessor system is completely different from that of multiprocessor systems. According to [62], uniprocessor's prediction scheme showed 82 percent keystream hit rate and optimization techniques can increase the rate even up to 99 percent. However, in our simulation, the prediction only scheme shows a lower keystream pool hit rate. This is because the counter in the uniprocessor system will be highly contiguous, so the sequential predic-

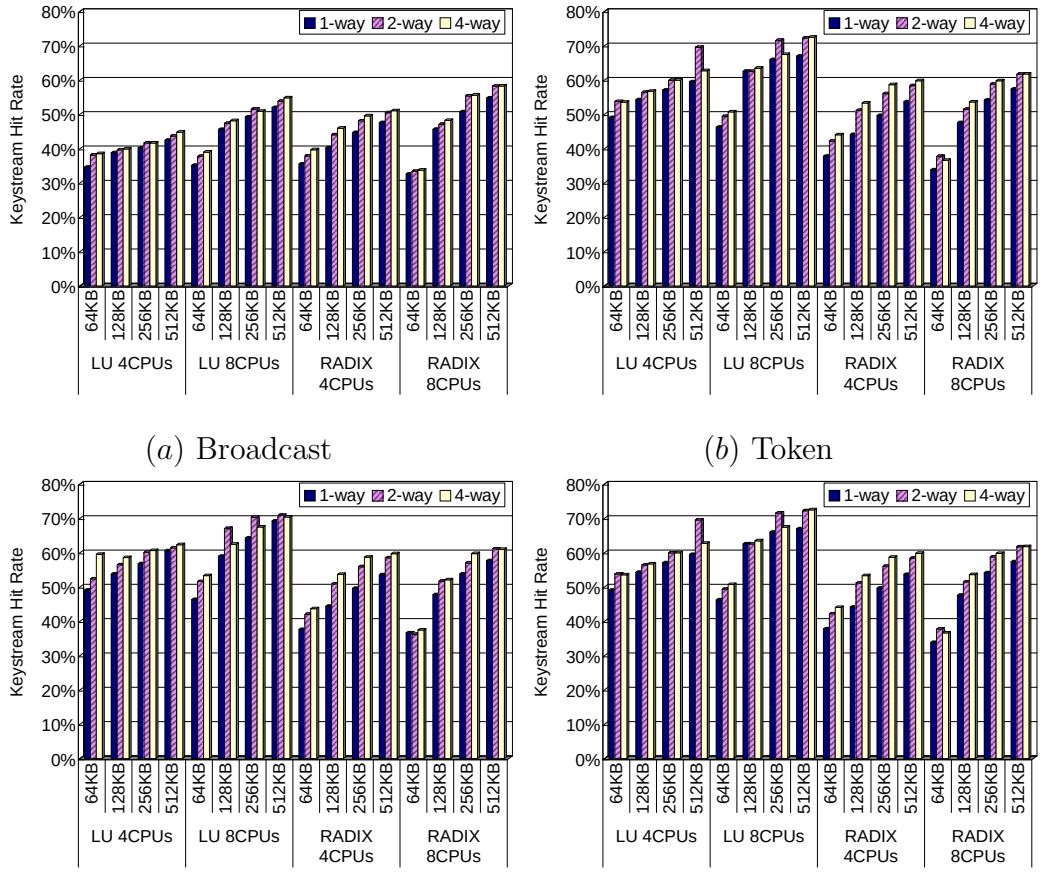
tion is very likely to be correct. However, in the prediction only scheme, the exclusive counters would result in counters' discontinuity in adjacent cache blocks when those blocks are modified by multiple processors having different counter ranges. Therefore, it is unavoidable to have a high miss rate, especially in data sharing intensive applications. In contrast, our scheme will show a high prediction hit rate in both high and low data sharing situations because the high hit rate of I^{2}SEMS comes from not only the counter prediction but also the counter broadcast.

3. Keystream Pool Size

To investigate the effect of the keystream pool size and associativity on the keystream pool hit rate, we varied the keystream pool size from 64K bytes to 512K bytes with direct-mapped, 2-way, and 4-way set associativities. In Figure 23, as the size increases, keystream pool hit rates also increase. It is intuitive that a large cache can hold more keystreams and consequently yield a higher keystream pool hit rate. Note that, even after the keystream pool size is increased exponentially, the keystream pool hit rate does not go up dramatically, ranging from 5 percent to 20 percent. The reason is that the recently assigned or predicted keystreams are more likely to be hit. Cache set associativity does not have a substantial effect on the keystream pool hit rate. A high set associative cache is useful only when many blocks are mapped to the same set. However, the counter itself is increasing monotonically, so cache contentions do not occur often. Therefore, we conclude that, in the presence of area and power constraints, the direct-mapped keystream pool is more desirable.

4. Cache Coherence Protocol

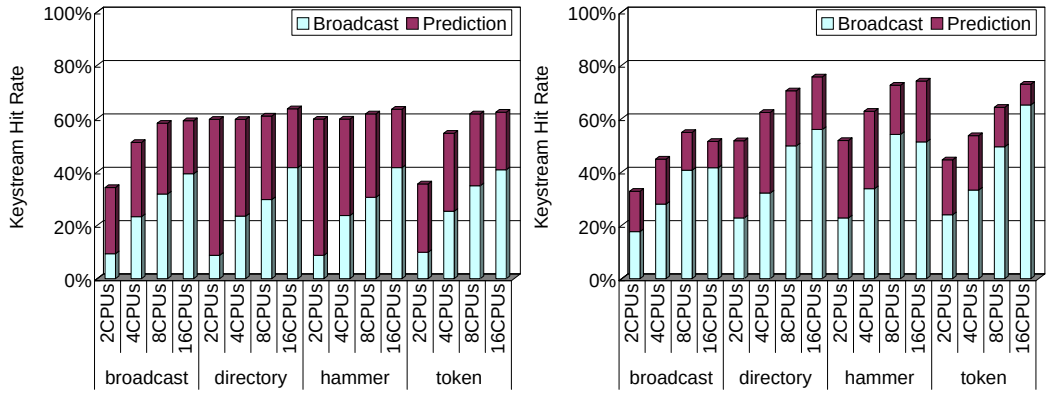
We investigate the relationship between the keystream pool hit rate and cache coherence protocols. In Figure 24, to illustrate individual contributions, we show the



(*a*) Broadcast (*b*) Token

(*c*) Directory (*d*) Hammer

Fig. 23. Hit rate vs. keystream pool size and associativity.



(*a*) RADIX (*b*) LU

Fig. 24. Keystream origination.

breakdown of keystream hit rate according to their origination: Broadcast and Prediction. We observe that, in both graphs, the proportion of hits on broadcasted counters is increasing as the number of processors is increasing. Therefore, we can conclude that a memory-writing dominant application will have a high keystream pool hit rate. However, even in a memory-reading dominant application, its keystream pool hit rate will not drop sharply because read-only memory accesses usually show high locality of memory accesses, and consequently the prediction scheme will contribute to a high keystream pool hit rate more than the broadcast scheme. Therefore, I^{2}SEMS will show good performance irrespective of memory access patterns in large multiprocessor systems.

5. Prediction Depth

The prediction depth is the number of keystreams that a keystream pool generates upon an arrival of a message. To investigate the impact of deep prediction depth on the keystream pool hit rate, we increased the prediction depth up to 160, testing on up to 16-processor systems as depicted in Figure 25. As the prediction depth increases, both the keystream pool hit rate and its prediction portion gradually increase while

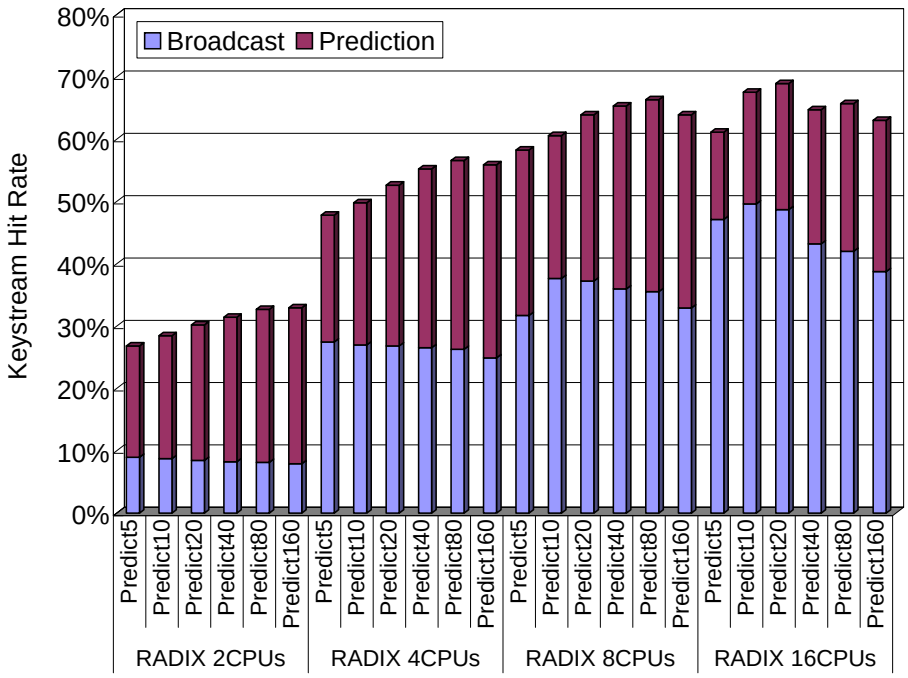
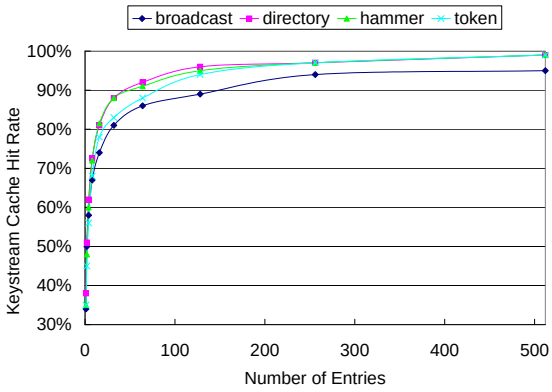


Fig. 25. Hit rate vs. prediction depth.

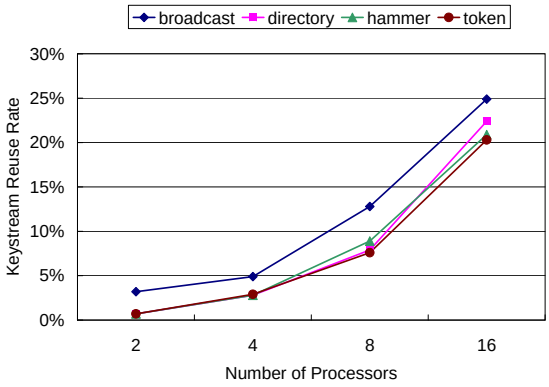
the broadcast portion slightly decreases. This is mainly because as the number of predicted keystreams increases, previously precomputed broadcast keystreams are evicted from the keystream pool. Note that, however, when the prediction depth reaches 160, the overall keystream hit rate begins to decrease. Therefore, this result shows that too deep prediction will have a negative impact on the performance from some points, and that, even with deep prediction, the broadcast scheme contributes substantially to the overall keystream pool hit rate.

6. Scalability of Global Counter Controller

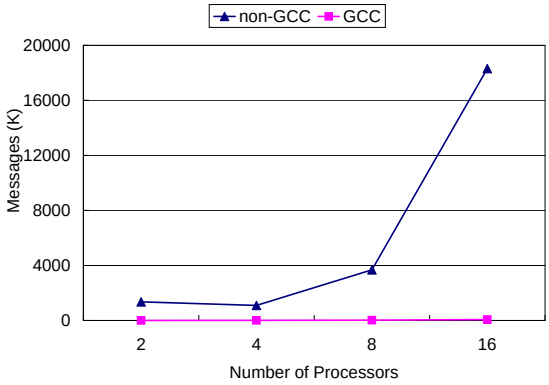
Figure 26 shows the simulation results of RADIX benchmark using up to 16 processors. Figure 26 (a) shows that the keystream cache hit rate will be relatively high even with the small keystream cache. This is due to the locality of memory accesses. Figure 26 (b) illustrates how much a 32-entry keystream cache reduces the counter



(*a*) Keystream Cache Hit Rate



(*b*) Keystream Reuse Rate



(*c*) Number of Messages

Fig. 26. Scalability of Global Counter Controller.

usage. When the number of processors is two, less than 3 percent of outgoing data blocks reuse keystreams, but in the 16 processors the rate reaches around 25 percent. Considering the rate is going up rapidly as the number of processors increases, it is expected that a significant portion of outgoing messages will reuse keystreams. In Figure 26 (*c*), while the number of normal messages increases dramatically at 16 processors, the number of GCC messages does not change proportionally, but even appears static. This is because our scheme successfully reduces the number of new counters through the keystream reuse by using both the keystream cache and the block assignment of new counters. Although simulation results are only available in up to 16 processors, the trend hints that the GCC will be scalable to larger shared memory multiprocessor systems.

F. Conclusions

In this study, we proposed I^{2}SEMS to guarantee confidentiality and integrity of shared memory and cache-to-cache communication in multiprocessor systems by incorporating a small amount of additional hardware components: GCC, keystream queue, keystream cache, and keystream pool. The GCC assigns globally unique counters for memory/cache-to-cache communication security. Keystream queues and keystream caches minimize encryption delay while reducing the counter usage rate. For fast decryption, keystream pools precompute/store keystreams at a counter broadcast by the GCC as well as at a message arrival. In addition, by separating security implementation and cache coherence verification, I^{2}SEMS can work on diverse cache coherence protocols.

The important conclusions of this work are the following: First, we provided confidentiality and integrity of shared memory and cache-to-cache communication in

multiprocessor systems with low performance overhead. We used GCM with AES for better security and performance. The performance overhead of I^{2}SEMS was 4 percent on average although execution time was too variable to find a common trend. The keystream pool hit rate was as high as 78 percent, meaning 78 percent of incoming messages were instantly decrypted and authenticated upon arrival. Second, simulation results showed that I^{2}SEMS will have good performance in large scale shared memory multiprocessor systems. Even though we tested up to 16-processors due to the current status of our simulator, its trend looks obvious. In addition, I^{2}SEMS works well with any applications. If an application is memory-read dominant, the prediction scheme where subsequent counters are predicted will contribute significantly to a high keystream pool hit rate because of the high locality of memory access. In memory-write dominant applications, the broadcast scheme where newly assigned counters are broadcasted will increase the keystream pool hit rate. Therefore, we conclude that I^{2}SEMS can support large scale shared memory multiprocessors with diverse memory access patterns. Note that, due to the limitation of the simulator, we could not simulate distributed shared memory systems. Nevertheless, we believe general trends will be similar not only because the number of GCC-related messages will be still very small, but also because both the prediction scheme and counter broadcast scheme will result in high keystream pool hit rates. Third, we found that relatively small keystream pools can support a large system although the larger keystream pools are beneficial to the high keystream pool hit rate. Set associativity does not have much impact on the keystream pool hit rate. Therefore, a simple but moderate sized keystream pool is desirable in I^{2}SEMS.

We are currently examining a number of possible expansions to this work. First, we were unable to analyze the realistic web-based or database servers. In-depth experiments with those server applications should fortify the I^{2}SEMS design. Next,

we would like to expand our research to much larger multiprocessor systems with DSM and to new multiprocessor architectures such as Chip Multiprocessor (CMP) systems.

CHAPTER VI

CONCLUSIONS

We have proposed three major topics in this dissertation. These are (*i*) study of a secure cluster design that is robust to confidentiality, authentication, and availability attacks, (*ii*) study of a security accelerating network interface card that can improve security performance significantly by offloading security operations from host processors, (*iii*) design of interconnect-independent secure shared-memory multiprocessor systems protecting the inter-processor communications from confidentiality attacks.

In the first study, we elaborated on the Key exposure problem of InfiniBand-based cluster systems and showed that possible DoS attacks can affect the overall performance significantly. Then, we detailed how to adopt an encryption and authentication algorithm into IBA clusters with as low as 0.7% performance overhead. We also proposed the stateful ingress filtering that can be active only when necessary. Furthermore, our source identification algorithm can trace back real attackers in large networks regardless of their topologies. In the second study, we focused on enabling fine-grained secure communications in coscheduled cluster systems with low performance overhead. For this, we proposed a small session key cache and a prefetch buffer inside cluster interconnect card and showed that SKC is very effective when communication patterns show high temporal locality. To estimate the size of SKC, we developed an analytical model. This model showed that a 16-Kbyte SKC can support large-scale cluster systems with a high hit rate. In the third study, we proposed a secure shared-memory multiprocessor systems called I^{2}SEMS. To provide confidentiality and integrity of shared memory and cache-to-cache communication, I^{2}SEMS has a small amount of additional hardware components: GCC, keystream queue, keystream cache, and keystream pool. Simulation results showed that I^{2}SEMS will

show good performance in providing large scale shared memory multiprocessor systems with better security using GCM and AES. Its performance overhead is around 4 percent with as high as 78 percent keystream hit rate.

We are currently examining a number of possible extensions to this work. They include secure cluster design for other cluster interconnects by extending ideas used in secure IBA cluster system. We plan to implement secure NIC design by using a reconfigurable NIC. By doing this, we can get more practical analysis of its performance impact on performance of cluster systems. Finally, we are exploring alternative secure multiprocessor designs in various multi-processor or -core architecture.

REFERENCES

[1] Committee on National Security Systems, *National Information Assurance Glossary*, http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf; accessed April 24, 2008.

[2] David Geer, "Just How Secure Are Security Products?," *Computer*, vol. 37, no. 6, pp. 14–16, 2004.

[3] Avishai Wool, "A Quantitative Study of Firewall Configuration Errors," *Computer*, vol. 37, no. 6, pp. 62–67, 2004.

[4] "Attackers Penetrate Supercomputing Networks," http://www.teragrid.org/news/apps/0404/sconline3.html; accessed April 24, 2008.

[5] HPC wire, "InfiniBand Cluster Deployed at SC05," http://news.taborcommunications.com/msgget.jsp?mid=506904; accessed April 24, 2008.

[6] William Yurcik, G.A.Koenig, X. Meng, and J. Greenseid, "Cluster Security as a Unique Problem with Emergent Properties: Issues and Techniques," in *Proceedings of Linux Revolution 2004*, 2004, (CDROM).

[7] "Distributed Security Infrastructure," http://disec.sourceforge.net/; accessed Arpil 23, 2008.

[8] M. Pourzandi, "A new Distributed Security Model for Linux Clusters," in *Proceedings of the USENIX 2004 Annual Technical Conference, Extreme Linux Special Interest Group*, 2004, pp. 231–236.

[9] Ian Foster, Nicholas T. Karonis, Carl Kesselman, and Steven Tuecke, "Managing security in high-performance distributed computations," *Cluster Computing*, vol. 1, no. 1, pp. 95–107, 1998.

[10] Kay Connelly and Andrew A. Chien, "Breaking the Barriers: High Performance Security for High Performance Computing," in *Proceedings of the 2002 Workshop on New Security Paradigms*, 2002, pp. 36–42.

[11] Rossen Dimitrov and Matthew Gleeson, "Challenges and New Technologies for Addressing Security in High Performance Distributed Environments," in *Proceedings of the 21st National Information Systems Security Conference*, 1998, pp. 457–468.

[12] InfiniBand Trade Association, "InfiniBand Architecture Specification," Volume 1, Release 1.1, 2002, http://www.infinibandta.org/specs/; accessed April 24, 2008.

[13] Youtao Zhang, Lan Gao, Jun Yang, Xiangyu Zhang, and Rajiv Gupta, "SENSS: Security Enhancement to Symmetric Shared Memory Multiprocessors," in *Proceedings of the 11th International Symposium on High Performance Computer Architecture*, 2005, pp. 352–362.

[14] Weidong Shi, Hsien-Hsin S. Lee, Mrinmoy Ghosh, and Chenghuai Lu, "Architectural Support for High Speed Protection of Memory Integrity and Confidentiality in Multiprocessor Systems," in *Proceedings of the 13th International Conference on Parallel Architecture and Compilation Techniques*, 2004, pp. 123–134.

[15] Kohnfelder, "Toward a Practical Public Key Cryptosystem," Bachelor's thesis, MIT, Cambrige, MA, 1978.

[16] National Institute of Science and Technology, "Advanced Encryption Standard (AES)," FIPS 197, 2001, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf; accessed April 24, 2008.

[17] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104 (Informational), Feb. 1997, http://www.ietf.org/rfc/rfc2104.txt; accessed April 24, 2008.

[18] Gene Tsudik, "Message Authentication with One-Way Hash Functions," in *Proceedings of IEEE INFOCOM*, 1992, pp. 2055–2059.

[19] American National Standards Institute, "Financial Institution Retail Message Authentication," ANSI X9.19, 1996.

[20] National Institute of Standards and Technology, "Computer Data Authentication," FIPS 113, 1994.

[21] International Organization for Standards and International Electrotechnical Commission, "Information Technology. Security Techniques. Data Integrity Mechanism Using a Cryptographic Check Function Employing a Block Cipher Algorithm," ISO/IEC 9797-1, 1999.

[22] John Black and Phillip Rogaway, "A Block-Cipher Mode of Operation for Parallelizable Message Authentication," in *EUROCRYPT '02: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*, 2002, pp. 384–397.

[23] D. McGrew and J. Viega, "The Galois/Counter Mode of Operation (GCM)," Submission to NIST Modes of Operation Process, 2004.

[24] Sven Dietrich, Neil Long, and David Dittrich, "Analyzing Distributed Denial of Service Tools: The Shaft Case," in *Proceedings of the 14th USENIX Conference on System Administration (LISA '00)*, 2000, pp. 329–340.

[25] A. Machie, J. Roculan, R. Russell, and M. V. Velzen, "Nimda Worm Analysis," Tech. Rep., Incident Analysis, SecurityFocus, 2001.

[26] R. Russell and A. Machie, "Code Red II Worm," Tech. Rep., Incident Analysis, SecurityFocus, 2001.

[27] D. Song, R. Malan, and R. Stone, "A Snapshot of Global Internet Worm Activity," Tech. Rep., Worm Activity, 2001.

[28] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin, *Firewalls and Internet Security; Repelling the Wily Hacker*, Second Edition, Reading, MA: Addison-Wesley, 2003.

[29] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su, "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995.

[30] Eun Jung Kim, Ki Hwan Yum, Chita R. Das, Mazin S. Yousif, and José Duato, "Performance Enhancement Techniques for InfiniBand Architecture," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA '03)*, 2003, pp. 253–262.

[31] Manhee Lee, Eun Jung Kim, Ki Hwan Yum, and Mazin Yousif, "An Overview of Security Issues in Cluster Interconnects," in *Proceedings of the Second International Workshop on Cluster Security (Cluster-Sec), CCGRID '06 Workshop*, 2006, p. 25.

[32] Manhee Lee, Eun Jung Kim, and Mazin Yousif, "Security Enhancement in InfiniBand Architecture," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005, p. 105.

[33] Peter Hellekalek and Stefan Wegenkittl, "Empirical Evidence Concerning AES," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 13, no. 4, pp. 322–333, 2003.

[34] D. McGrew and J. Viega, "Flexible and Efficient Message Authentication in Hardware and Software," Manuscript, 2003.

[35] Hassan Aljifri, "IP Traceback: A New Denial-of-Service Deterrent?," *IEEE Security and Privacy*, vol. 1, no. 3, pp. 24–31, 2003.

[36] Dawn Xiaodong Song and Adrian Perrig, "Advanced and Authenticated Marking Schemes for IP Traceback," in *Proceedings of IEEE INFOCOM*, 2001, pp. 878–886.

[37] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson, "Practical Network Support for IP Traceback," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '00)*, 2000, pp. 295–306.

[38] Abraham Yaar, Adrian Perrig, and Dawn Song, "Pi: A Path Identification Mechanism to Defend against DDoS Attacks," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, 2003, p. 93.

[39] William Feller, *An Introduction to Probability Theory and Its Applications*, Third Edition, Hoboken, NJ: John Wiley & Sons, 1968.

[40] Dorothy E. Denning and Giovanni Maria Sacco, "Timestamps in Key Distribution Protocols," *Communications of the ACM*, vol. 24, no. 8, pp. 533–536, 1981.

[41] Joan Dyer, Ron Perez, Sean Smith, and Mark Lindemann, "Application Support Architecture for a High-Performance, Programmable Secure Coprocessor," in *Proceedings of the 22nd National Information Systems Security Conference*, October 1999, (CDROM).

[42] National Institute of Standards and Technology, "Security Requirements for Cryptographic Modules," FIPS 140-1, 1994.

[43] A. Hodjat and I. Verbauwhede, "Minimum Area Cost for a 30 to 70 Gbits/s AES Processor," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2004, pp. 83–88.

[44] S. Wilton and N. Jouppi, "CACTI: An Enhanced Cache Access and Cycle Time Model," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.

[45] Jonathan Krim and Robert O'Harrow Jr., "Data Under Siege," The Washington Post, March 10, 2005, http://www.washingtonpost.com/wp-dyn/articles/A19982-2005Mar9.html; accessed April 24, 2008.

[46] George Markowsky and Linda Markowsky, "Survey of Supercomputer Cluster Security Issues," in *Proceedings of the 2007 International Conference on Security and Management*, 2007, pp. 474–480.

[47] Robert Fischer and Ming-Yang Kao, "Multi-Domain Sandboxing: An Overview," Harvard Technical Report Computer Science Group TR-05-00, Harvard, Cambridge, MA, 2000.

[48] Manhee Lee and Eun Jung Kim, "A Comprehensive Framework for Enhancing Security in InfiniBand Architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1393–1406, 2007.

[49] Xin Qi, G. Parmer, and R. West, "An Efficient End-host Architecture for Cluster Communication Services," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, 2004, pp. 83–92.

[50] Chris Runge, "The Path to Multi-level Security in Red Hat Enterprise Linux® and HP Industry Standard Servers," HP Whitepaper, 4AAO-4070ENUS, HP, Palo Alto, CA, 2006.

[51] A. Waheed and J. Yan, "Workload Characterization of CFD Applications Using Partial Differential Equation Solvers," in *Proceedings of Workshop on Workload Characterisatisation in High-Performance Computing Environments*, 1998.

[52] Eun Jung Kim, Ki Hwan Yum, and Chita R. Das, "Performance Analysis of a QoS Capable Cluster Interconnect," *Performanace Evaluation*, vol. 60, no. 1-4, pp. 275–302, 2005.

[53] Ki Hwan Yum, Eun Jung Kim, Chita R. Das, and Aniruddha S. Vaidya, "MediaWorm: A QoS Capable Router Architecture for Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 12, pp. 1261–1274, 2002.

[54] Andrea C. Arpaci-Dusseau, David E. Culler, and Alan M. Mainwaring, "Scheduling with Implicit Information in Distributed Systems," in *Proceedings of ACM SIGMETRICS '98/PERFORMANCE '98*, 1998, pp. 233–243.

[55] Shailabh Nagar, Ajit Banerjee, Anand Sivasubramaniam, and Chita R. Das, "Alternatives to Coscheduling a Network of Workstations," *Journal of Parallel*

and Distributed Computing, vol. 59, no. 2, pp. 302–327, 1999.

[56] Patrick Sobalvarro and William E. Weihl, "Demand-Based Coscheduling of Parallel Jobs on Multiprogrammed Multiprocessors," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '95)*, 1995, pp. 106–126.

[57] Y. Etsion and D. G. Feitelson, "User-Level Communication in a System with Gang Scheduling," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS '01)*, 2001, p. 58.

[58] Atsushi Hori, Hiroshi Tezuka, and Yutaka Ishikawa, "Highly Efficient Gang Scheduling Implementation," in *Proceedings of the 1998 ACM/IEEE conference on Supercomputing (Supercomputing '98)*, 1998, pp. 1–14.

[59] B. Gassend, G. E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and Hash Trees for Efficient Memory Integrity Verification," in *Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, 2003, p. 295.

[60] T. Gilmont, J.D. Legat, and J.J. Quisquater, "Enhancing the Security in the Memory Management Unit," in *Proceedings of the 25th EuroMicro Conference*, 1999, vol. 1, pp. 449–456.

[61] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, 2000, pp. 168–177.

[62] Weidong Shi, HsienHsin S. Lee, Mrinmoy Ghosh, Chenghuai Lu, and Alexandra Boldyreva, "High Efficiency Counter Mode Security Architecture via Prediction and Precomputation," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2005, pp. 14–24.

[63] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architecture for Tamper-Evident and Tamper-Resistant Processing," in *Proceedings of the 17th International Conference on Supercomputing (ICS)*, 2003, pp. 160–171.

[64] G. Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processors," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 339.

[65] Chenyu Yan, Brian Rogers, Daniel Englender, Yan Solihin, and Milos Prvulovic, "Improving Cost, Performance, and Security of Memory Encryption and Authentication," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, 2006, pp. 179–190.

[66] Jun Yang, Youtao Zhang, and Lan Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003, p. 351.

[67] Brian Rogers, Milos Prvulovic, and Yan Solihin, "Efficient Data Protection for Distributed Shared Memory Multiprocessors.," in *Proceedings of the 15th International Conference on Parallel Architecture and Compilation Techniques (PACT '06)*, 2006, pp. 84–94.

[68] Manhee Lee, Minseon Ahn, and Eun Jung Kim, "I²SEMS: Interconnects-Independent Security Enhanced Shared Memory Multiprocessor Systems," in *16th International Conference on Parallel Architecture and Compilation Techniques (PACT '07)*, 2007, pp. 94–103.

[69] Sean W. Smith and Steve Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks*, vol. 31, no. 9, pp. 831–860, 1999.

[70] Oliver Kömmerling and Markus G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," in *Proceedings of the USENIX Workshop on Smartcard Technology*, 1999, pp. 9–20.

[71] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," in *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA '92)*, 1992, pp. 256–266.

[72] Scott Pakin, Mario Lauria, and Andrew Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet," in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (Supercomputing '95)*, 1995, p. 55.

[73] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-Net: a User-Level Network Interface for Parallel and Distributed Computing," in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles (SOSP '95)*, 1995, pp. 40–53.

[74] Cezary Dubnicki, Angelos Bilas, and Kai Li, "Design and Implementation of

Virtual Memory-Mapped Communication on Myrinet," in *Proceedings of 11th International Parallel Processing Symposium (IPPS '97)*, 1997, p. 388.

[75] R. Dos Santos, Ricardo Bianchini, and Claudio L. Amorim, "A Survey Of Messaging Software Issues And Systems For Myrinet-Based Clusters," *Parallel Distributed Computer Practices, Special issue High-Performance Comput. Clusters*, vol. 2, no. 2, 1999.

[76] Quadrics, http://www.quadrics.com/; accessed April 24, 2008.

[77] George Apostolopoulos, Vinod Peris, and Debanjan Saha, "Transport Layer Security: How Much Does it Really Cost?," in *Proceedings of IEEE INFOCOM*, 1999, pp. 717–725.

[78] Cristian Coarfa, Peter Druschel, and Dan S. Wallach, "Performance Analysis of TLS Web Servers," *ACM Transactions on Computer Systems*, vol. 24, no. 1, pp. 39–69, 2006.

[79] A. Keromytis, J. Ioannidis, and J. Smith, "Implementing IPsec," in *Proceedings of Global Internet (GlobeCom) '97*, 1997, pp. 1948–1952.

[80] Stefan Miltchev, Sotiris Ioannidis, and Angelos D. Keromytis, "A Study of the Relative Costs of Network Security Protocols," in *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, 2002, pp. 41–48.

[81] Angelos D. Keromytis, Jason L. Wright, Theo De Raadt, and Matthew Burnside, "Cryptography As An Operating System Service: A Case Study," *ACM Transactions on Computer Systems*, vol. 24, no. 1, pp. 1–38, 2006.

[82] Bo Yang, Sambit Mishra, and Ramesh Karri, "A High Speed Architecture for Galois/Counter Mode of Operation (GCM)," Cryptology ePrint Archive,

Report 2005/146, 2005.

[83] AMIS, http://www.amis.com/asics/standard_cell.html; accessed April 24, 2008.

[84] Sucheta Chodnekar, Viji Srinivasan, Aniruddha S. Vaidya, Anand Sivasubramaniam, and Chita R. Das, "Towards a Communication Characterization Methodology for Parallel Applications," in *Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture (HPCA '97)*, 1997, p. 310.

[85] JunSeong Kim and David J. Lilja, "Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs," in *Proceedings of the Second International Workshop on Network-Based Parallel Computing (CANPC '98)*, 1998, pp. 202–216.

[86] Gyu Sang Choi, Jin-Ha Kim, Deniz Ersoz, Andy B. Yoo, and Chita R. Das, "Coscheduling in Clusters: Is It a Viable Alternative?," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC '04)*, 2004, p. 16.

[87] NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/; accessed April 24, 2008.

[88] Mellanox, "InfiniHost III Ex MemFree Mode Performance," http://www.mellanox.com/pdf/whitepapers/PCIxVsMemfree_WP_100.pdf; accessed April 24, 2008.

[89] Richard P. Martin, Amin M. Vahdat, David E. Culler, and Thomas E. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture," in *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA '97)*, 1997, pp. 85–97.

[90] G. A. Koenig, Xin Meng, A. J. Lee, M. Treaster, N. Kiyanclar, and W. Yurcik, "Cluster Security with NVisionCC: Process Monitoring by Leveraging Emergent Properties," in *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, 2005, vol. 1, pp. 121–132.

[91] Manhee Lee, Eun Jung Kim, Ki Hwan Yum, and Mazin S. Yousif, "Instant Attack Stopper in InfiniBand Architecture," in *Proceedings of the First International Workshop on Cluster Security (Cluster-Sec), CCGRID '05 Workshop*, 2005, pp. 105–110.

[92] J. Smith, C. Traw, and D. Farber, "Cryptographic Support for a Gigabit Network," in *Proceedings of INET*, 1992, pp. 229–237.

[93] 3Com, "3Com Secure NIC," http://www.3com.com/products/en_US/detail.jsp?tab=features&pathtype=purchase&sku=3CR990B-97; accessed April 24, 2008.

[94] Intel, "Intel Pro/100 S Server Adapter," http://www.intel.com/network/connectivity/products/pro100s_srvr_adapter.htm; accessed April 24, 2008.

[95] Broadcom, "Broadcom Gigabit Ethernet Controller," http://www.broadcom.com/products/Small-Medium-Business/Gigabit-Ethernet-Controllers/BCM5752M; accessed April 24, 2008.

[96] Aleph One, "Smashing The Stack For Fun And Profit," *Phrack*, vol. 7, no. 49, 1996.

[97] Andrew "bunnie" Huang, "The Trusted PC: Skin-Deep Security," *Computer*, vol. 35, no. 10, pp. 103–105, 2002.

[98] Andrew "bunnie" Huang, *Hacking the Xbox: An Introduction to Reverse Engineering*, No Starch Press, San Francisco, CA, 2003.

[99] H. Lipmaa, P. Rogaway, and D. Wagner, "CTR-Mode Encryption," in *Proceedings of NIST Workshop on Modes of Operation*, 2000.

[100] HP, "HP Superdome," http://www.hp.com/products1/servers/scalableservers/superdome/index.html; accessed April 24, 2008.

[101] SGI, "SGI Origin 3000," http://www.sgi.com/products/servers/origin/3000/; accessed April 24, 2008.

[102] E. Ender Bilir, Ross M. Dickson, Ying Hu, Manoj Plakal, Daniel J. Sorin, Mark D. Hill, and David A. Wood, "Multicast Snooping: A New Coherence Method Using a Multicast Address Network," in *Proceedings of 26th Annual International Symposium on Computer Architecture (ISCA '99)*, 1999, pp. 294–304.

[103] Milo M.K. Martin, Mark D. Hill, and David A. Wood, "Token Coherence: A New Framework For Shared-Memory Multiprocessors," *IEEE Micro*, vol. 23, no. 6, pp. 108–116, 2003.

[104] Hung-Chang Hsiao and Chung-Ta King, "An Application-Driven Study of Multicast Communication for Write Invalidation," *The Journal of Supercomputing*, vol. 18, no. 3, pp. 279–304, 2001.

[105] M. P. Malumbres, Jose Duato, and Joseph Torrellas, "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors," in *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96)*, 1996, p. 186.

[106] Dhabaleswar K. Panda, Sanjay Singal, and Pradeep Prabhakaran, "Multidestination Message Passing Mechanism Conforming to Base Wormhole Routing Scheme," in *Proceedings of the First International Workshop on Parallel Computer Routing and Communication (PCRCW '94)*, 1994, pp. 131–145.

[107] P.S.Magnusson and et al., "Simics: A Full System Simulation Platform," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[108] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[109] "SpecOMP2001 Benchmark Suite," http://www.spec.org/omp/; accessed April 24, 2008.

[110] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the 22nd International Symposium on Computer Architecture*, 1995, pp. 24–36.

[111] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas, "Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions," in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, 2005, pp. 25–36.

[112] Ruby B. Lee, Peter C. S. Kwan, John Patrick McGregor, Jeffrey Dwoskin, and Zhenghong Wang, "Architecture for Protecting Critical Secrets in Micro-

processors," in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA '05)*, 2005, pp. 2–13.

[113] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin, "Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly," in *Proceedings of the 40th annual IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 183–196.

[114] Milo M.K. Martin and et. al., "Protocol Specifications and Tables for Four Comparable MOESI Coherence Protocols: Token Coherence, Directory, Snooping, and Hammer," 2003, http://www.cs.wisc.edu/multifacet/theses/milo_martin_phd/; accessed April 24, 2008.

VITA

Name: Man Hee Lee

Address: Department of Computer Science
Texas A&M University
College Station, TX 77843-3112

Email Address: manheelee@gmail.com

Education: **Ph.D.** in Computer Engineering, Dept. of Computer Science, Texas A&M University, 2008

M.E, Dept. of Computer Engineering, Kyungpook National University, Korea, 1997

B.E, Dept. of Computer Engineering, Kyungpook National University, Korea, 1995

Experience: **Lecturer**, Dept. of Computer Science, Texas A&M University, Spring 2008, CPSC321 Computer Architecture

Intern, Midrange Routing Business Unit, Cisco Systems, Inc., May 2007-Aug. 2007

Research Assistant, Dept. of Computer Science, Texas A&M University, Spring & Fall 2007

Teaching Assistant, Dept. of Computer Science, Texas A&M University, Fall 2005-Fall 2006 Fall 2005, spring & summer 2006: CPSC321 Computer Architecture, Fall 2006: CPSC614 Advanced Computer Architecture

Network researcher, Supercomputing Center, KISTI, Korea, Dec. 1996-Jul. 2003