ES-MPICH2: A Message Passing Interface with Enhanced Security

Xiaojun Ruan, *Member*, *IEEE*, Qing Yang, *Member*, *IEEE*, Mohammed I. Alghamdi, Shu Yin, *Student Member*, *IEEE*, and Xiao Qin, *Senior Member*, *IEEE*

Abstract—An increasing number of commodity clusters are connected to each other by public networks, which have become a potential threat to security sensitive parallel applications running on the clusters. To address this security issue, we developed a Message Passing Interface (MPI) implementation to preserve confidentiality of messages communicated among nodes of clusters in an unsecured network. We focus on MPI rather than other protocols, because MPI is one of the most popular communication protocols for parallel computing on clusters. Our MPI implementation—called ES-MPICH2—was built based on MPICH2 developed by the Argonne National Laboratory. Like MPICH2, ES-MPICH2 aims at supporting a large variety of computation and communication platforms like commodity clusters and high-speed networks. We integrated encryption and decryption algorithms into the MPICH2 library with the standard MPI interface and; thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. MPI-application programmers can fully configure any confidentiality services in MPICHI2, because a secured configuration file in ES-MPICH2 offers the programmers flexibility in choosing any cryptographic schemes and keys seamlessly incorporated in ES-MPICH2. We used the Sandia Micro Benchmark and Intel MPI Benchmark suites to evaluate and compare the performance of ES-MPICH2 with the original MPICH2 version. Our experiments show that overhead incurred by the confidentiality services in ES-MPICH2 is marginal for small messages. The security overhead in ES-MPICH2 becomes more pronounced with larger messages. Our results also show that security overhead can be significantly reduced in ES-MPICH2 by high-performance clusters. The executable binaries and source code of the ES-MPICH2 implementation are freely available at http://www.eng.auburn.edu/~xqin/software/es-mpich2/.

Index Terms—Parallel computing, computer security, message passing interface, encryption.

1 Introduction

1.1 Motivation

Due to the fast development of the internet, an increasing number of universities and companies are connecting their cluster computing systems to public networks to provide high accessibility. Those clusters connecting to the internet can be accessed by anyone from anywhere. For example, computing nodes in a distributed cluster system proposed by Sun Microsystems are geographically deployed in various computing sites. Information processed in a distributed cluster is shared among a group of distributed tasks or users by the virtue of message passing protocols (e.g., message passing interface—MPI) or confidential data transmitted to and from cluster computing nodes.

Preserving data confidentiality in a message passing environment over an untrusted network is critical for a wide spectrum of security-aware MPI applications, because unauthorized access to the security-sensitive messages by untrusted processes can lead to serious security breaches. Hence, it is imperative to protect confidentiality of messages exchanged among a group of trusted processes.

It is a nontrivial and challenging problem to offer confidentiality services for large-scale distributed clusters, because there is an open accessible nature of the open networks. To address this issue, we enhanced the security of the MPI protocol by encrypting and decrypting messages sent and received among computing nodes.

In this study, we focus on MPI rather than other protocols, because MPI is one of the most popular communication protocols for cluster computing environments. Numerous scientific and commercial applications running on clusters were developed using the MPI protocol. Among a variety of MPI implementations, we picked MPICH2 developed by the Argonne National Laboratory. The design goal of MPICH2—a widely used MPI implementation—is to combine portability with high performance [14]. We integrated encryption algorithms into the MPICH2 library. Thus, data confidentiality of MPI applications can be readily preserved without a need to change the source codes of the MPI applications. Data communications of a conventional MPI program can be secured without converting the program into the corresponding secure version, since we provide a security enhanced MPI-library with the standard MPI interface.

1.2 Possible Approaches

There are three possible approaches to improving security of MPI applications. In first approach, application programmers

- *X. Ruan is with the Department of Computer Science, West Chester University of Pennsylvania, PA 19383. E-mail: xruan@wcupa.edu.*
- *Q. Yang is with the Department of Computer Science, Montana State University, MT 59717. E-mail: qing.yang@cs.montana.edu.*
- *M.I. Alghamdi is with the Department of Computer Science, Al-Baha University, Al-Baha City, Kingdom of Saudi Arabia. E-mail: mialmushilah@bu.edu.sa.*
- *S. Yin and X. Qin are with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347. E-mail: {szy0004, xqin}@auburn.edu.*

Manuscript received 27 Sept. 2010; revised 22 Aug. 2011; accepted 14 Dec. 2011; published online 11 Jan. 2012.
Recommended for acceptance by M. Singhal.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-2010-09-0167.
Digital Object Identifier no. 10.1109/TDSC.2012.9.

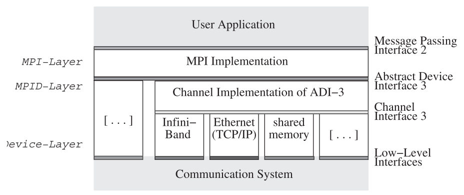


Fig. 1. Hierarchical Structure of MPICH2 [13].

can add source code to address the issue of message confidentiality. For example, the programmers may rely on external libraries (e.g., SEAL [26] and Nexus [11]) to implement secure applications. Such an application-level security approach not only makes the MPI applications error-prone, but also reduces the portability and flexibility of the MPI applications. In the second approach, the MPI interface can be extended in the way that new security-aware APIs are designed and implemented (see, for example, MPISec I/O [22]). This MPI-interface-level solution enables programmers to write secure MPI applications with minimal changes to the interface. Although the second approach is better than the first one, this MPI-interface-level solution typically requires an extra code to deal with data confidentiality. The third approach—a channel-level solution—is proposed in this study to address the drawbacks of the above two approaches. Our channel-level solution aims at providing message confidentiality in a communication channel that implements the Channel Interface 3 (CH3) in MPICH2 (see Fig. 1).

1.3 Contributions

In what follows, we summarize the four major contributions of this study.

- We implemented a standard MPI mechanism called ES-MPICH2 to offer data confidentiality for secure network communications in message passing environments. Our proposed security technique incorporated in the MPICH2 library can be very useful for protecting data transmitted in open networks like the Internet.
- The ES-MPICH2 mechanism allows application programmers to easily write secure MPI applications without additional code for data-confidentiality protection. We seek a channel-level solution in which encryption and decryption functions are included into the MPICH2 library. Our ES-MPICH2 maintains a standard MPI interface to exchange messages while preserving data confidentiality.
- The implemented ES-MPICH2 framework provides a secured configuration file that enables application programmers to selectively choose any cryptographic algorithm and symmetric-key in ES-MPICH2. This feature makes it possible for programmers to easily and fully control the security services incorporated in the MPICHI2 library. To demonstrate this feature, we implemented the AES and 3DES algorithms in ES-MPICH2. We also show in this paper how to add other cryptographic algorithms into the ES-MPICH2 framework.
- We have used ES-MPICH2 to perform a detailed case study using the Sandia Micro Benchmarks and the Intel MPI benchmarks. We focus on runtime performance overhead introduced by the cryptographic algorithms.

1.4 Roadmap

The paper is organized as follows: Section 2 demonstrates the vulnerabilities of existing MPI implementations by describing a security threat model for clusters connected by public networks. Section 3 not only provides a reason for focusing on the confidentiality issue of MPICH2 rather than other MPI implementations, but also gives an overview of the MPICH2 implementation. Section 4 presents the motivation of this work by showing why secured MPI is an important issue and also outlines the design of ES-MPICH2—the message passing interface with enhanced security. Section 5 describes the corresponding implementation details of ES-MPICH2. Section 6 discusses some experimental results and compares the performance of ES-MPICH2 with that of MPICH2. Section 7 presents previous research related to our project. Finally, Section 8 states the conclusions and future work of this study.

2 Threat Model

A geographically distributed cluster system is one in which computing components at local cluster computing platforms communicate and coordinate their actions by passing messages through public networks like the Internet. To improve the security of clusters connected to the public networks, one may build a private network to connect an array of local clusters to form a large scale cluster. Building a private network, however, is not a cost-effective way to secure distributed clusters. The Internet—a very large distributed system—can be used to support large-scale cluster computing. Being a public network, the internet becomes a potential threat to distributed cluster computing environments.

We first describe the confidentiality aspect of security in clusters followed by three specific attack instances. We believe new attacks are likely to emerge, but the confidentiality aspect will remain unchanged. Confidentiality attacks attempts to expose messages being transmitted among a set of collaborating processes in a cluster. For example, if attackers gain network administrator privilege, they can intercept messages and export the messages to a database file for further analysis. Even without legitimate privilege, an attacker still can sniff and intercept all messages in a cluster on the public network. Such attacks result in the information leakage of messages passed among computing nodes in geographically distributed clusters. Cryptography and access control are widely applied to computer systems to safeguard against confidentiality attacks.

We identify the following three confidentiality attacks on MPI programs running on distributed clusters:

- *Sniffing message traffic.* Message traffic of an MPI program can be sniffed. For example, when MPCH2 is deployed in a cluster connected by a Gigabit Ethernet network, attackers can sniff plaintext messages transmitted through the TCP socket. Message sniffing can reveal security-sensitive data, metadata, and information.
- *Snooping on message buffer.* In an MPI program, buffers are employed to send and receive messages. Regardless of specific MPI implementations, message buffers are created before the send and receive primitives are

invoked. Attackers who snoop into the message buffers in memory can access data and information without being given specific access privileges.

- *Message traffic profiling*. Message traffic profiling attacks seek to use message type, time stamps, message size, and other metadata to analyze message exchange patterns and types of protocols being used in message transmissions. For example, an attacker can monitor the network connection of a cluster running an MPI program. If a message has been regularly transmitted, the attacker can speculate the importance of the message and intercept the content of the message.

Confidentiality services can effectively counter the aforementioned threats in MPI applications running on clusters connected by a public network. In this research, we encode messages using the Advanced Encryption Standard (AES) and the Triple Data Encryption Standard (Triple-DES or 3DES). It is worth nothing that 3DES is not considered for any modern applications, we investigate 3DES in this study because of two reasons. First, 3DES is still used in many legacy application systems in the industry. Second, 3DES is a good baseline solution used for the purpose of comparison. In the case that attackers intercept messages in an MPI program, they are unable to transform the ciphertext into the original plaintext due to the lack of data encipherment keys (DEK).

3 MPICH2 Overview

MPICH—one of the most popular MPI implementations—-were developed at the Argonne National Laboratory [14]. The early MPICH version supports the MPI-1 standard. MPICH2—a successor of MPICH—not only provides support for the MPI-1 standard, but also facilitates the new MPI-2 standard, which specifies functionalities like one-sided communication, dynamic process management, and MPI I/O [13]. Compared with the implementation of MPICH, MPICH2 was completely redesigned and developed to achieve high performance, maximum flexibility, and good portability.

Fig. 1 shows the hierarchical structure of the MPICH2 implementation, where there are four distinct layers of interfaces to make the MPICH2 design portable and flexible. The four layers, from top to bottom, are the message passing interface 2 (MPI-2), the abstract device interface (ADI3), the CH3, and the low-level interface. ADI3—the third generation of the abstract device interface—in the hierarchical structure (see Fig. 1) allows MPICH2 to be easily ported from one platform to another. Since it is nontrivial to implement ADI3 as a full-featured abstract device interface with many functions, the CH3 layer simply implements a dozen functions in ADI3 [18].

As shown in Fig. 1, the TCP socket Channel, the shared memory access (SHMEM) channel, and the remote direct memory access (RDMA) channel are all implemented in the layer of CH3 to facilitate the ease of porting MPICH2 on various platforms. Note that each one of the aforementioned channels implements the CH3 interface for a corresponding communication architecture like TCP sockets, SHMEM, and RDMA. Unlike an ADI3 device, a channel is easy to implement since one only has to implement a dozen functions relevant for with the channel interface.

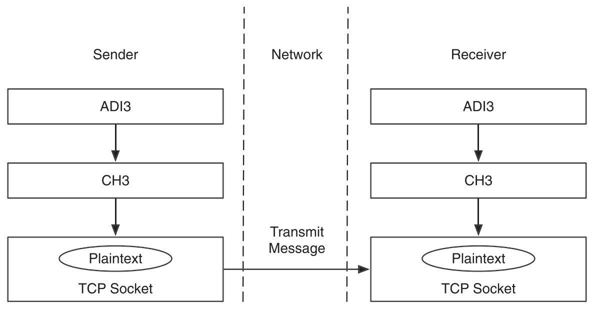


Fig. 2. Message passing implementation structure in MPICH2.

To address the issues of message snooping in the message passing environments on clusters, we seek to implement a standard MPI mechanism with confidentiality services to counter snooping threats in MPI programs running on a cluster connected an unsecured network. More specifically, we aim to implement cryptographic algorithms in the TCP socket channel in the CH3 layer of MPICH2 (see Fig. 2 and Section 5 for details of how to construct a cryptosystem in the channel layer).

4 The Design of ES-MPICH2

4.1 Scope of ES-MPICH2

Confidentiality, integrity, availability, and authentication are four important security issues to be addressed in clusters connected by an unsecured public network. Rather than addressing all the security aspects, we pay particular attention to confidentiality services for messages passed among computing nodes in an unsecured cluster.

Although preserving confidentiality is our primary concern, an integrity checking service can be readily incorporated into our security framework by applying a public-key cryptography scheme. In an MPI framework equipped with the public-key scheme, sending nodes can encode messages using their private keys. In the message receiving procedure, any nodes can use public keys corresponding to the private keys to decode messages. If one alters the messages, the ciphertext cannot be deciphered correctly using public keys corresponding to the private keys. Thus, the receiving nodes can perform message integrity check without the secure exchange of secret keys. Please refer to Section 5.6 for details of how to add integrity checking services in our MPI framework called ES-MPICH2.

4.2 Design Issues

The goal of the development of the ES-MPICH2 mechanism is to enable application programmers to easily implement secure enhanced MPI applications without additional code for data-confidentiality protection. With ES-MPICH2 in place, secure MPI application programmers are able to flexibly choose a cryptographic algorithm, key size, and data block size for each MPI application that needs data confidentiality protection.

ES-MPICH2 offers message confidentiality in an MPI programming environment by incorporating MPICH2 with encryption and decryption algorithms. In the process of designing ES-MPICH2, we integrated the AES and 3DES algorithms into the MPICH2 library.

The ES-MPICH2 implementation has the following four design goals:

- **Message confidentiality:** ES-MPICH2 aims to preserve message confidentiality from unauthorized accesses by untrusted processes. We leverage the AES to protect the confidentiality of messages, because AES is an encryption standard adopted by the US government. For comparison purpose, we also consider 3DES in the design of ES-MPICH2. AES with 128-bit keys can provide adequate protection for classified messages up to the SECRET level. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use [1]. In this study, we integrated data confidentiality services with MPICH2 by implementing the cryptographic algorithms in a CH3 channel.
- **Complete transparency:** Preserving message confidentiality in MPICH2 is entirely transparent to application programmers. Such confidentiality transparency is feasible and the reason is two-fold. First, the encryption and decryption processes can be built in the MPICH2 library at the channel transmission layer. Second, we maintain the same interface as the APIs of the MPICH2 implementation. Therefore, it is not necessary to modify MPI programs to adapt ES-MPICH2.
- **Compatibility and portability:** Ideally, ES-MPICH2 needs to be easily ported from one platform to another with no addition to the application source code. ES-MPICH2 is an extension of MPICH2 and; thus, ES-MPICH2 should have the same level of portability as MPICH2. However, it is challenging to achieve high portability in ES-MPICH2, because we have to implement a cryptographic subsystem in each channel in the CH3 layer in MPICH2.
- **Extensibility:** ES-MPICH2 must allow application programmers to selectively choose any cipher techniques and keys incorporated in MPICH2. This design goal makes it possible for programmers to flexibly select any cryptographic algorithm implemented in ES-MPICHI2. Although we implemented AES and 3DES in the channel layer of MPICH2, we will show in the next section how to add other cryptographic algorithms (e.g., Elliptic Curve Cryptography (ECC), [2]) to the ES-MPICH2 environment.

5 Implementation Details

During the implementation of ES-MPICH2, we addressed the following five development questions:

1. Among the multiple layers in the hierarchical structure of MPICH2, in which layer should we implement cryptographic algorithms?
2. Which cryptosystem should we choose to implement?
3. How to implement secure key management?
4. How to use the implemented ES-MPICH2?
5. How to add integrity checking services to ES-MPICH2?

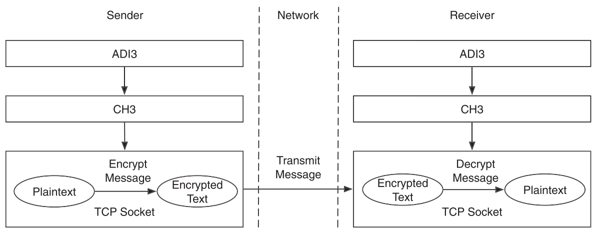


Fig. 3. Message passing implementation structure in ES-MPICH2 with encryption and decryption processes. A cryptosystem is implemented in the TCP socket layer to achieve the design goal of complete transparency.

5.1 Ciphers in the Channel Layer

Fig. 2 outlines the message passing implementation structure in the original version of MPICH2. In such a hierarchical structure of MPICH2, messages are passed from a sending process to a receiving process through the abstract device interface, the CH3, and the TCP socket channel. Cryptographic subsystems may be implemented in one of the three layers (i.e., ADI3, CH3, or the TCP socket channel). To achieve the design goal of a complete transparency, we chose to implement cryptographic algorithms in the TCP socket channel. Compared with ADI3 and CH3, the TCP socket channel is the lowest layer of the MPICH2 hierarchy. Implementing cryptosystems in the lowest layer can preserve message confidentiality in any conventional MPI program without adding extra code to protect messages. Fig. 3 depicts the implementation structure of ES-MPICH2, where a cryptosystem is implemented in the TCP socket layer. Thus, messages are encrypted and decrypted in the TCP socket channel rather than the ADI3 and CH3 layers.

Fig. 4 shows that the encryption and decryption functions in ES-MPICH2 interact with the TCP socket to provide message confidentiality protection in the TCP socket layer. Before a message is delivered through the TCP socket channel, data contained in the message are encrypted by a certain cryptographic algorithm like AES and 3DES. Upon the arrival of an encrypted message in a receiving node, the node invokes the corresponding decryption function to decrypt the message. Fig. 4 demonstrates that ES-MPICH2 maintains the same application programming interface or API as that of MPICH2 by implementing the encryption and decryption algorithms in the TCP socket level. The confidentiality services of ES-MPICH2 were implemented in the MPICH2 libraries, thereby being totally transparent to MPI application programmers.

5.2 Block Ciphers

We have no intention of reinventing a cryptographic library, because it is very costly to guarantee that the security of your own implementation is higher than that of existing tested and audited security libraries. In the ES-MPICH2 framework, we adopted the implementation of the AES and 3DES cryptographic algorithms offered by the PolarSSL library in MPICH2 version 1.0.7. PolarSSL is an open-source cryptographic library written in C. We focus on block ciphers in the implementation of ES-MPICH2, because a block cipher transforms a fixed-length block of plaintext into a block of ciphertext of the same length. If the

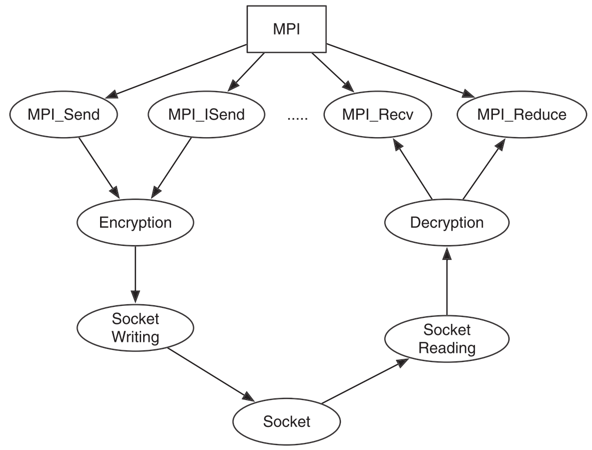


Fig. 4. The interface between the encryption/decryption processes and the TCP socket. ES-MPICH2 maintains the same API as that of MPICH2.

case where ciphertext and plaintext are different in length, MPI applications have to be aware of such a difference in order to correctly decode ciphers. Keeping in mind that securely passing messages should be transparent to MPI application programmers, we advocate the use of block ciphers rather than non-block-ciphers that force programmers to be aware of the lengths of plaintext and ciphertext.

5.3 Key Management

The goal of key management is to dynamically establish secure message-passing channels by distributing cryptographic keys. ES-MPICH2 maintains two types of keys—data encipherment keys (a.k.a., session keys) and interchange keys. In the MPI initialization phase of each MPI application, a data encipherment key is created and shared among all the communicating processes.

Fig. 5 presents the key management infrastructure in ES-MPICH2. Public key cryptography employed in ES-MPICH2 relies on interchange keys (i.e., public and private keys) to securely exchange session keys in an unsecured network. More specifically, when a master node attempts to share new session keys with other slave nodes, the master node uses the slave nodes' public keys to encrypt the session keys. The slave nodes make use of their private keys to decipher messages containing the session keys. Then, the master node and slave nodes can securely communicate using the MPI framework. Interchange keys also can be stored on physical devices like smart card and ROM [7], [9], [20].

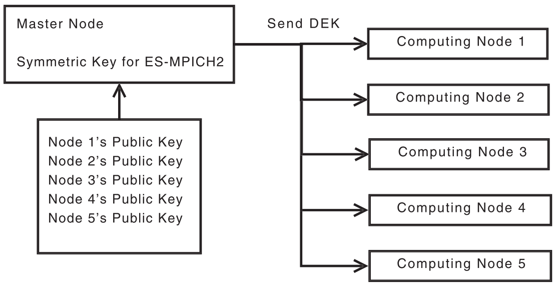


Fig. 5. Key management in ES-MPICH2. Public key cryptography employed in ES-MPICH2 relies on interchange keys (i.e., public and private keys) to exchange data encipherment keys in a secure way.

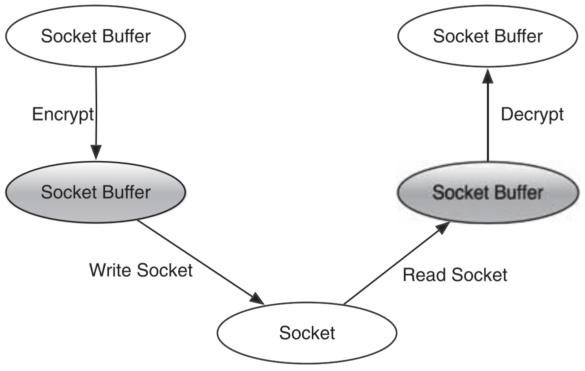


Fig. 6. ES-MPICH2 socket details.

5.4 Socket Programming

In socket programming, there is a buffer containing data sent and received through the TCP socket channel. Fig. 6 demonstrates the encryption and decryption process in ES-MPICH2. ES-MPICH2 encrypt the plaintext in the buffer of the sending node then decrypt it on the receiving node. Because the plaintext and ciphertext are identical in length in block cipher algorithms, the sizes of the buffers in both the sending and receiving nodes remain unchanged after the encryption and decryption processes.

5.5 Usage

The security features of ES-MPICH2 can be configured without modifying any MPI application source code. To securely pass messages using ES-MPCH2, the following configurations must be set before an MPI initialization. First, a security option should be enabled or disabled. Second, one has to select a specific cryptographic algorithm implemented in ES-MPICH2. Third, exchange keys must be securely stored in a configuration file or a physical device in each node (see Section 5.3 for details on the key management issue.) Then, users can run their MPI programs in the same way as they should run the programs in MPICH2. Thus, if an MPI program can be executed in MPICH2, one can also run the MPI program in ES-MPICH2 without modifying the source code of the program.

5.6 Incorporating Integrity Services in ES-MPICH2

In addition to confidentiality services, integrity checking services can be seamlessly incorporated into the ES-MPICH2 framework. In what follows, we address the implementation issue of how to integrate integrity checking services in ES-MPICH2.

Spreading feature of block encryption algorithms. Block encryption algorithms have a spreading feature which means if even 1 bit is changed in ciphertext, the decrypted text will be completely different from the original plaintext. Altered messages causing fatal errors cannot be interpreted. Although using the spreading feature is not a reliable solution, the spreading feature does provide an integrity checking method. Since both AES and 3DES are block encryption algorithms, ES-MPICH2 may rely on the spreading feature to perform integrity checking.

Public key. An integrity service tailed for small messages can be added into ES-MPICH2 using a public-key encryption scheme, in which sending nodes encode messages using

TABLE 1
The Configuration of a 6-Node Cluster of Intel Celeron Processors

	Node ×6
CPU	Intel Celeron 450 2.2GHz
Memory	2GB
OS	Ubuntu 9.04 Jaunty Jackalope
Kernel version	2.6.28-15-generic
Network	1000Mbps

TABLE 2
Performance Metrics Used in the Sandia Micro Benchmark Suite

Metric	Explanation
iter_t	total amount of time for the loop to complete
work_t	for each iteration of the post-work-wait loop
	the amount of work performed
overhead_t	the length of time that a processor is engaged in the
	transmission or reception of each message
base_t	message transfer time calculation threshold

private keys whereas receiving nodes use the corresponding public keys to decipher the ciphertext. Before a node delivers an encrypted message in ES-MPICH2, the node encrypts the message using the private key of the sending node. To check the integrity of the cipher message, a receiving node simply needs to decode the cipher message by applying the public key of the sending node.

Hash functions. When it comes to large messages, hash functions are widely used in integrity checking and digital signatures. MD5, SHA-1, and SHA-2 can be implemented to check the integrity of encrypted messages in ES-MPICH2. The hash is a cryptographic checksum or message integrity code that sending and receiving nodes must compute to verify messages. Specifically, a sending node uses a hash function to compute a checksum for a large message. The checksum is shorter than the original message. Then, the sending node signs the hash value with a shared key. In doing so, the integrity of the large message can be checked in an efficient way.

People use hash and then sign the hash with the key, the size of hash being much shorter than the message.

6 Experimental Evaluation

To evaluate the features and performance of ES-MPICH2, we implemented ES-MPICH2 and deployed it on two clusters with different configurations. The first cluster has six nodes of 2.2 GHz Intel Celeron processors with 2 GB memory. The second cluster contains 10 nodes. The master node has a 3.0 GHz Intel Pentium Core 2 Duo processor with 1 GB memory, whereas the nine slave nodes have 333 MHz Intel Pentium II processors with 64 MB memory. The six nodes in the first cluster are connected by a 1 Gbps Ethernet LAN. The 10 nodes in the second cluster are connected by a 100 Mbps Ethernet LAN. Apparently, the overall performance of the first cluster is higher than that of the second cluster.

We use a fast cluster (i.e., the first one) and a slow (i.e., the second one) cluster to conduct experiments, because one of the goals is to illustrate the impact of computing capacity of clusters on the performance of ES-MPICH2.

6.1 A 6-Node Cluster of Intel Celeron Processors

6.1.1 Experimental Testbed

Let us first evaluate the performance of both MPICH2 and ES-MPICH2 on a 6-node cluster. Table 1 reports the configuration of the first cluster with six identical computing nodes of Intel Celeron processors. The operating system used in the six nodes is Ubuntu 9.04 Jaunty Jackalope. The computing nodes are connected by a 1 Gbps network. All the slave nodes share a disk on the master node through the network file system (NFS) [25]. The MPI library used in the 6-node cluster is MPICH2 version 1.0.7. We run the Sandia Micro Benchmarks and the Intel MPI Benchmarks to evaluate and compare the performance of MPICH2 and ES-MPICH2. When we test ES-MPICH2 in each experiment, we set the cryptographic service to AES and 3DES, respectively. The length of data encipherment keys generated and distributed in ES-MPICH2 is 192-bit.

6.1.2 SMB: Sandia Micro Benchmark

The Sandia National Laboratory developed the Sandia Micro Benchmark Suite (a.k.a., SMB) to evaluate and test high-performance network interfaces and protocols. Table 2 described the four performance metrics used in the SMB benchmark suite. These metrics include total execution time (i.e., iter_t), CPU execution time for iterations (i.e., work_t), message passing overhead (i.e., overhead_t), and message transfer time calculation threshold (i.e., threshold or base_t). The detailed information on these metrics can be found at http://www.cs.sandia.gov/smb. Please note that the message passing overhead can be derived by subtracting the CPU execution time from the total execution time. Each benchmark has 1,000 iterations.

Fig. 7 shows the total execution time of the SMB benchmark running on the original MPI implementation (i.e., MPICH2) as well as AES-based ES-MIPCH2 and 3DES-based ES-MPICH2. We observe from this figure that when the message size is small (e.g., 1 KB), the performance of ES-MPICH2 is very close to that of MPICH2. For example, the encryption modules in ES-MPICH2 only modestly increase the execution time by less than two percent. These results indicate that ES-MPICH2 can preserve confidentiality of small messages with negligible overhead.

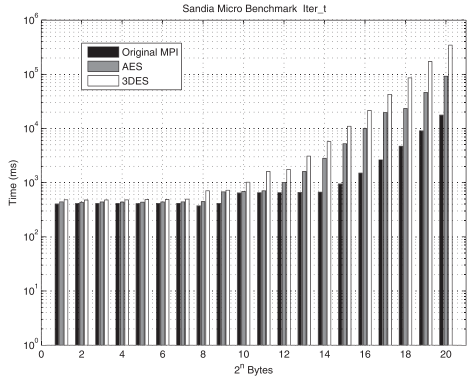
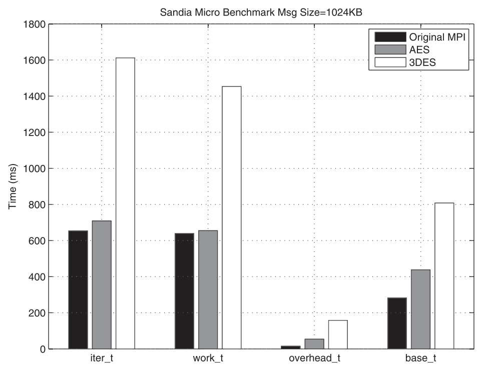
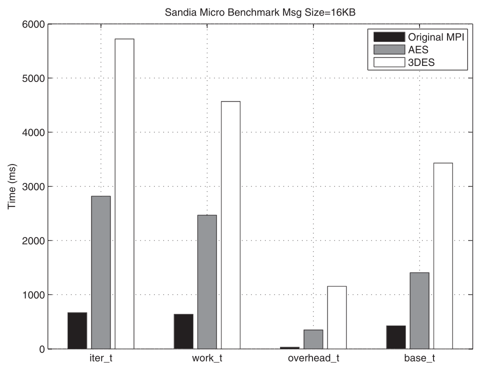


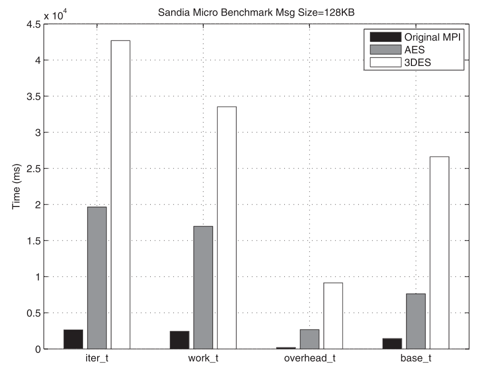
Fig. 7. Sandia Micro Benchmark iter_time.



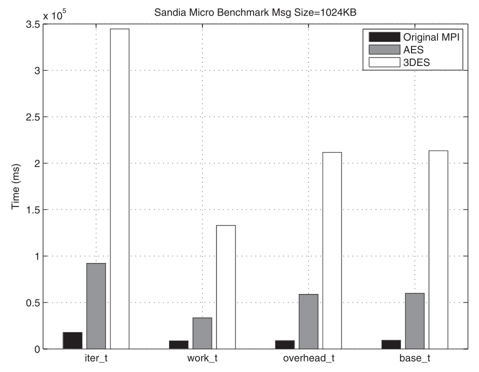
(a) Message Size is 2KB



(b) Message Size is 16KB



(c) Message Size is 128KB



(d) Message Size is 1024KB

Fig. 8. Sandia Micro Benchmark, Iteration Time, Work Time, Overhead Time, and Base Time on different message size from 6-node cluster of Intel Celeron.

Figs. 8a, 8b, 8c, and 8d show the total execution time, CPU time, overhead, and threshold of MPICH2 and ES-MPICH2 when the message size is set to 2, 16, 128, 512, and 1,024 KB, respectively. The results plotted in Fig. 8a show that AES-based ES-MPICH2 and MPICH2 have similar performance in the case of small messages. For example, the AES module in ES-MPICH2 increases the execution times of $iter\_t$ and $work\_t$ by 5.9 and 1.2 percent, respectively. However, when 3DES is employed in ES-MPICH2, the security overhead of ES-MPICH2 becomes noticeable even for small messages. For example, let us consider a case where the message size is 2 KB. Compare with the 3DES module, the AES module can reduce the execution times of $iter\_t$ and $base\_t$ by approximately 56 and 45 percent, respectively. Figs. 8b, 8c, and 8d illustrate that both AES and 3DES in ES-MPICH2 introduce much overhead that makes ES-MPICH2 performs worse than MPICH2—the original MPI implementation. Security overhead in ES-MPICH2 becomes more pronounced with increasing message size. Since AES has better performance than 3DES, AES-based ES-MPICH2 is superior to 3DES-based ES-MPICH2. We recommend the following two approaches to lowering overhead caused by encryption and decryption modules in ES-MPICH2. First, one can reduce the security overhead in ES-MPICH2 by enhancing the performance of block cipher algorithms. Second, multicore processors can boost efficiency of the encryption and decryption modules, thereby benefiting the performance of ES-MPICH2.

6.1.3 *IMB: Intel MPI Benchmarks*

The Intel MPI benchmark suite or IMB was developed for testing and evaluating implementations of both MPI-1 [8] and MPI-2 [12] standards. IMB contains approximately 10,000 lines of code to measure the performance of important MPI functions [5], [24]. We have evaluated the performance of ES-MPICH2 and the original MPICH2 by running the benchmarks on the 6-node cluster. Table 3 lists all the Intel benchmarks used to measure the performance of ES-MPI2 and MPICH2. The benchmarks in IMB-MPI1 can be categorized in three groups: single transfer, parallel transfer, and collective benchmarks. Single transfer benchmarks are focusing on a single message transferred between two communicating processes. Unlike single transfer benchmarks, parallel transfer benchmarks aim at testing patterns and activities in a group of communicating processes with concurrent actions. Collective benchmarks are implemented to test higher level collective functions, which involve processors within a defined communicator group. Please refer to http://software.intel.com/en-us/articles/intel-mpi-benchmarks for more information concerning IMB.

Figs. 9a and 9b show the performance of PingPong and PingPing—two single transfer benchmarks in IMB. Since single transfer benchmarks are used to test a pair of two active processes, we run PingPong and PingPing on two nodes of the 6-node cluster. The total execution times of PingPong and PingPing go up when the message size

TABLE 3
Intel MPI Benchmarks

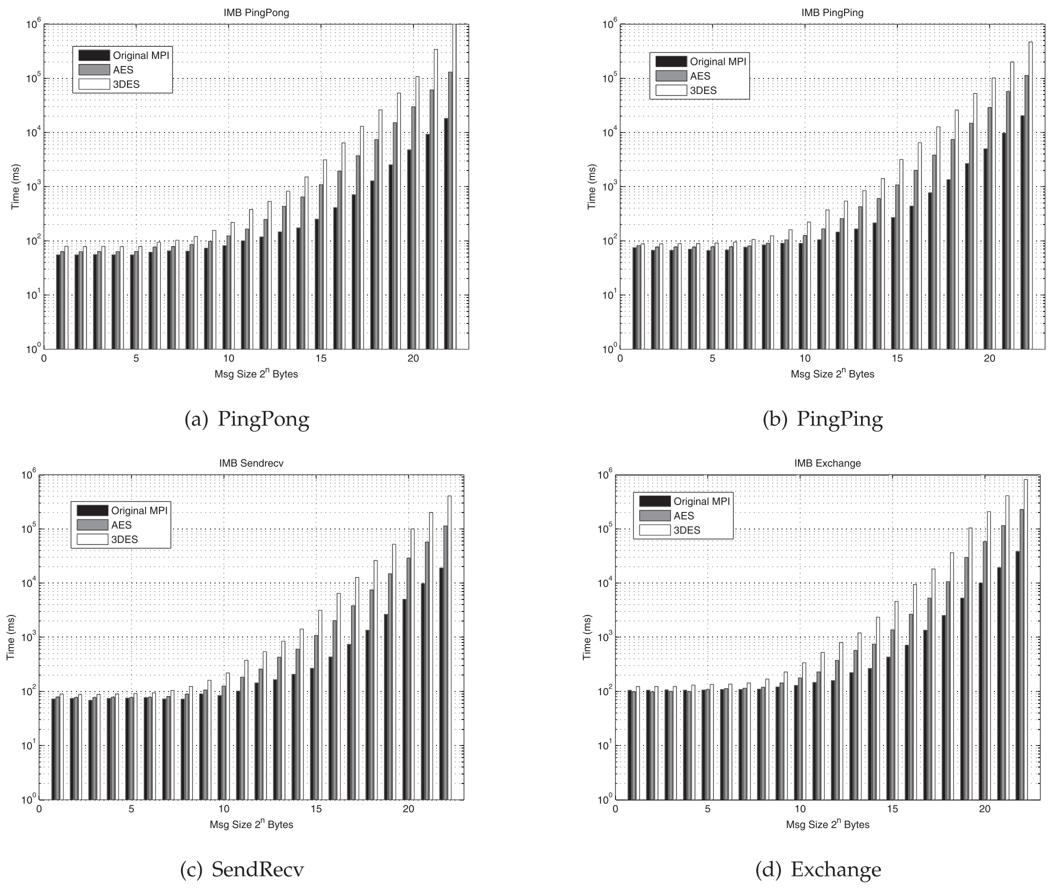
Benchmarks	Classification	Semantics
PingPong	Single Transfer	MPI-1
PingPing	Single Transfer	MPI-1
Sendrecv	Parallel Transfer	MPI-1
Exchange	Parallel Transfer	MPI-1
Bcast	Collective	MPI-1
Allgather	Collective	MPI-1
Allgatherv	Collective	MPI-1
Scatter	Collective	MPI-1
Scatterv	Collective	MPI-1
Gather	Collective	MPI-1
Gatherv	Collective	MPI-1
Alltoall	Collective	MPI-1
Alltoallv	Collective	MPI-1
Reduce	Collective	MPI-1
Reduce_Scatter	Collective	MPI-1
Allreduce	Collective	MPI-1
Window	Other	MPI-2

increases because larger messages give rise to higher encryption and decryption overheads. Compared with MPICH2, the execution times of AES-based and 3DES-based ES-MPICH2 are more sensitive to message size.

Now we analyze the performance of Sendrecv and Exchange—two parallel transfer benchmarks in IMB—running on ES-MPICH2 and MPICH2 on the 6-node cluster. Sendrecv, in which the main purpose is to test the MPI_Sendrecv function, consists of processes forming a periodic communication chain. Similarly, Exchange is a benchmark focusing on the evaluation of the MPI_ISend, MPI_Waitall, and MPI_Recv functions. Unlike the aforementioned single transfer benchmarks, message passing operations in these two parallel benchmarks are performed in parallel.

Fig. 9c plots the performance results of the SendRecv benchmark on the cluster, where each node receives data from its left neighbor and then sends data to its right neighbor. The total execution time of the SendRecv benchmark does not noticeably change when we vary the number of computing nodes in the cluster. We attribute this trend to the factor that message passing in multiple nodes are carried out in parallel rather than serially. Thus, increasing the number of nodes does not affect SendRecv's total execution time. With respect to parallel transfers, the performance of AES-based and 3DES-based MPICH2 is close to that of the original version of MIPCH2 when message size is relatively small. When it comes to large messages, AES-based ES-MPICH2 has better parallel transfer performance than 3DES-based MPICH2.

Fig. 9d depicts the total execution time of the Exchange benchmark. Comparing Fig. 9d with Fig. 9c, we realize that regardless of the MPI implementations, the execution time of the Exchange benchmark is longer than that of the SendRecv benchmark under the condition of same message size. This is mainly because in Exchange each node transfer data to both left and right neighbors in the communication



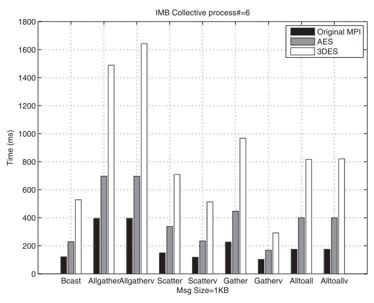
(a) PingPong

(b) PingPing

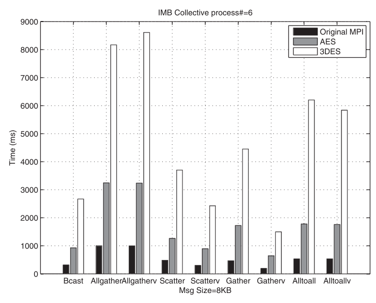
(c) SendRecv

(d) Exchange

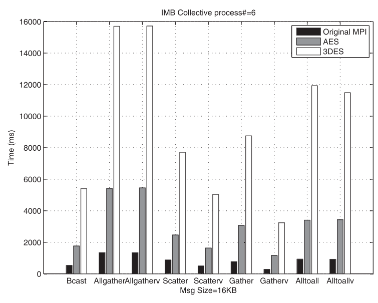
Fig. 9. Intel MPI Benchmarks, PingPong, and PingPing are Single Transfer Benchmarks, SendRecv and Exchange are Parallel Benchmarks on 6-node cluster of Intel Celeron.



(a) Message Size is 1KB

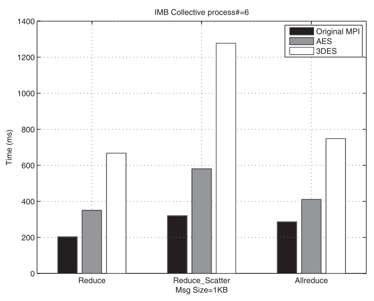


(b) Message Size is 8KB

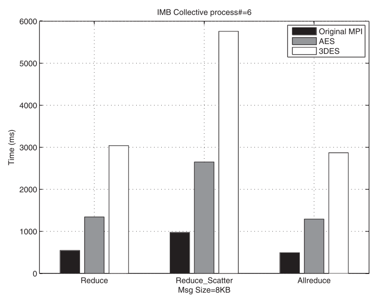


(c) Message Size is 16KB

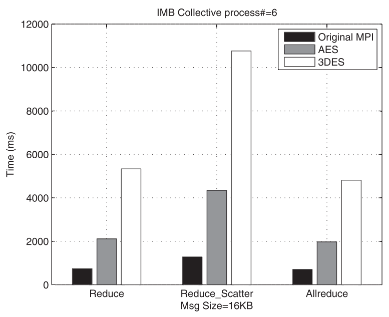
Fig. 10. Intel MPI Benchmarks, Collective Benchmarks, Group A, a group of benchmarks from Collective on 6-node Cluster of Intel Celeron.



(a) Message Size is 1KB



(b) Message Size is 8KB



(c) Message Size is 16KB

Fig. 11. Intel MPI Benchmarks, Collective Benchmarks, Group B, a group of benchmarks from Collective on 6-node Cluster of Intel Celeron.

chain. Thus, communication time in Exchange is larger than that in SendRecv. As a result, the total execution time of Exchange is approximately two times higher than that of Sendrecv when message size is large.

Let us vary message size and evaluate the performance of collective benchmarks. We run the benchmark 10 times on each MPI implementation and report the average execution times. Figs. 10a and 10b show the performance of the first group of nine collective benchmarks. We observe from these figures that the total execution time of each collective benchmark continually increases with increasing message size. MPICH2 has better performance than AES-based and 3DES-based ES-MPICH2 across all the collective benchmarks, because the confidentiality is preserved at the cost of message passing performance. Figs. 11a and 11b plot the execution times of the second group of three benchmarks. The performance results of the second benchmark group are consistent with those of the first benchmark group reported in Figs. 10a and 10b.

Fig. 12 shows the results of the Window benchmark, which aims to test MPI-2 functions like MPI_Win_create, MPI_Win_fence, and MPI_Win_free. In this benchmark, a window size message is transferred to each node, which in turn creates a window using a specified size. Fig. 12 indicates that the execution time of the benchmark is not sensitive to message size. The results confirm that AES-based ES-MPICH2 improves the security of the Window benchmark on MPICH2 with marginal overhead.

6.2 A 10-Node Cluster of Intel Pentium II Processors

6.2.1 Experimental Testbed

Now we evaluate the performance of MPICH2 and ES-MPICH2 on a 10-node cluster of Intel Pentium II processors. The cluster configuration is summarized in Table 4. The operating system running on this cluster is Fedora Core release 4 (Stentz). Although the processors of the nine slave nodes are 333 MHz Intel Pentium II, the master node contains a 3.0 GHz Intel Pentium Core 2 Duo processor, which is almost 10 times faster than the processors in the slave nodes. Each slave node has only 64 MB memory, whereas the master node has 1 GB memory. All the 10 nodes are connected by a 100 Mbps Ethernet network. Like the first cluster, all nodes in the 10-node cluster share disk space on the master node through the network file system.

6.2.2 SMB: Sandia Micro Benchmark

Figs. 13a, 13b, and 13c reveal the total execution time, CPU time, overhead, and threshold of MPICH2 and ES-MPICH2 when the message size is set to 1, 16, and 32 KB, respectively. The results show that the performance of

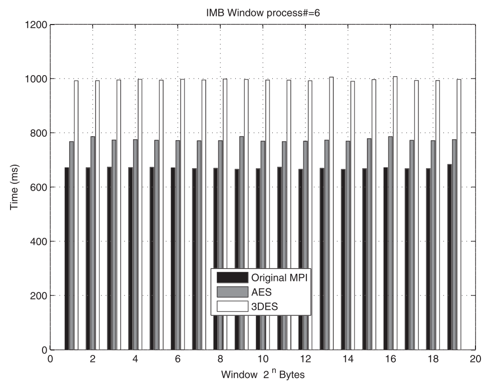


Fig. 12. Intel Micro Benchmark Window six nodes.

TABLE 4
The Configuration of a 10-Node Cluster of Intel Pentium II Processors

	Master ×1	Slaves ×9
CPU	Pentium Core 2 Duo 3.00GHz	Pentium II 333MHz
Memory	1GB	64MB
OS	Fedora Core release 4	Fedora Core release 4
Kernel	2.6.12 − 1.1456 FC4smp	2.6.17 − 1.2142 FC4
Network Adapter	100Mbps	100Mbps

AES-based and 3DES-based ES-MPICH2 is noticeably worse than that of MPICH2, because the encryption and decryption modules in ES-MPICH2 spend significant amount of time in encrypting and decrypting messages issued from the benchmarks. This trend is true even when messages are small (e.g., see Fig. 13a where message size is 1 KB).

Comparing Fig. 13a and Fig. 8a, we draw the following three observations. First, the first 6-node cluster is significantly faster than the second 10-node cluster. Although the 10-node cluster has more computing nodes than the second one, the nodes of the 10-node cluster have lower computing capacity than those in the 6-node cluster. This is because the hardware configuration of the 10-node cluster is worse than that of the 6-node cluster. In other words, the processors in the 10-node cluster are Pentium II CPUs; the 6-node cluster relies on Intel Celeron 450 to run the benchmarks. Second, compared with the 10-node slow cluster, the 6-node fast cluster allows the iter_t and the work_t benchmarks to spend smaller time periods dealing with the security modules in MPICH2. For example, on the 6-node fast cluster, iter_t and work_t spend approximately 7.1 and 3.2 percent of the benchmarks' total execution times in the AES-based security modules (See Fig. 8a). On the 10-node slow cluster, the AES modules account for about 35.5 and 39.3 percent of these two benchmarks' total execution times(See Fig. 13a). Third, the performance of AES-based MPICH2 on the first cluster is very close to that of MPICH2 when message size is smaller than 2 KB.

The above observations indicate that given message-intensive MPI applications, one can improve the processor computing capacity of a cluster to substantially reduce the portion of time (out of the applications' total execution time) spent in processing security modules in ES-MPICH2.

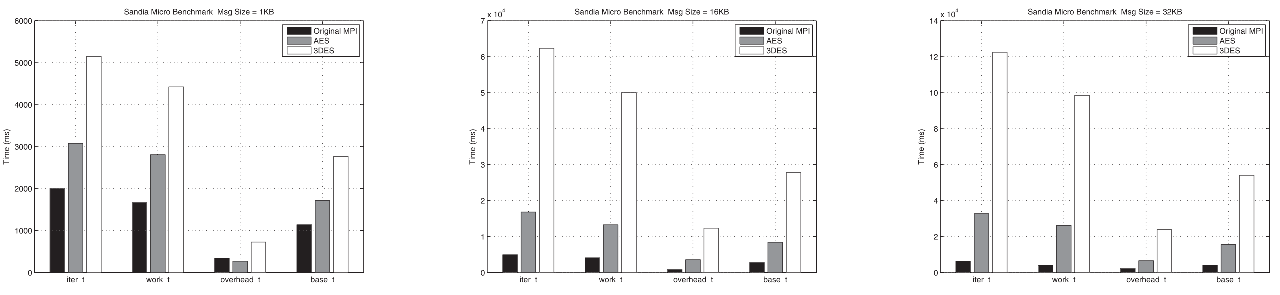
6.2.3 IMB: Intel MPI Benchmarks

Figs. 14a, 14b, 14c, and 14d depict the performance of the PingPong, PingPing, SendRecv, and Exchange benchmarks in IMB. The total execution times of the four IMB benchmarks increases with increasing message size. Compared with MPICH2, the execution time of ES-MPICH2 is more sensitive to message size. More importantly, Figs. 14a, 14b, 14c, and 14d demonstrate that when ES-MPICH2 is deployed on a slow cluster, ES-MPICH2 preserves message confidentiality by substantially degrading the performance of the original MPICH2. By comparing the Intel benchmark performance on both the 6-node cluster (see Figs. 9a, 9b, 9c, and 9d and 10-node clusters (see Figs. 14a, 14b, 14c, and 14d)), we observe that the performance gap between MPICH2 and ES-MPICH2 on the fast cluster is much smaller than the performance gap on the slow cluster. An implication of this observation is that security overhead in ES-MPICH2 can be significantly reduced by deploying ES-MPICH2 in a high-end cluster.

7 Related Work

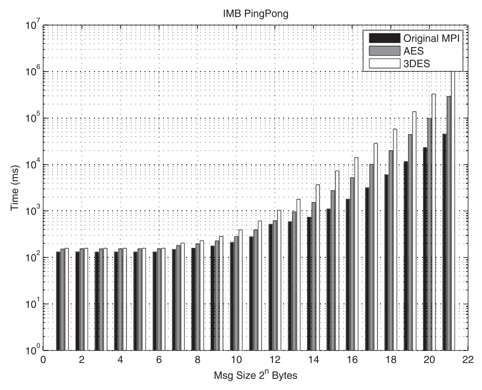
Message passing interface. The Message Passing Interface standard (MPI) is a message passing library standard used for the development of message-passing parallel programs [14]. The goal of MPI is to facilitate an efficient, portable, and flexible standard for parallel programs using message passing. MPICH2—developed by the Argonne National Laboratory—is one of the most popular and widely deployed MPI implementations in cluster computing environments. MPICH2 provides an implementation of the MPI standard while supporting a large variety of computation and communication platforms like commodity clusters, high-performance computing systems, and high-speed networks [13].

As early as 1997, Brightwell et al. from the Sandia National Laboratory insightfully pointed out barriers to creating a secure MPI framework [3]. The barriers include control and data in addition to cryptographic issues. In a secure MPI, both control and data messages must be protected from unauthorized access of attackers and malicious users. Although there is a wide range of implementations of the MPI and MPI-2 standards (e.g., MPICH and MPICH2 are two freely available implementations from the Argonne National Laboratory), there is a lack

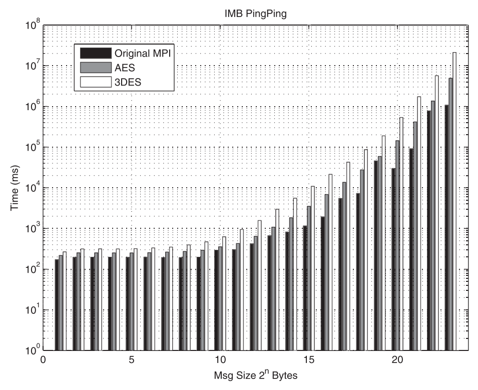


(a) Message Size is 1KB (b) Message Size is 16KB (c) Message Size is 32KB

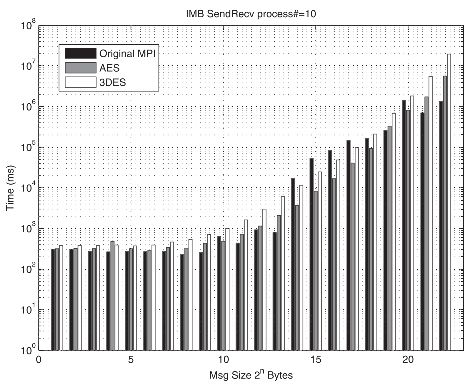
Fig. 13. Sandia Micro Benchmarks on the 10-node cluster of Intel Pentium II.



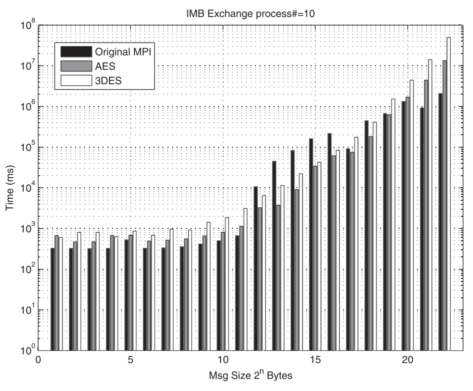
(a) PingPong



(b) PingPing



(c) SendRecv



(d) Exchange

Fig. 14. Intel MPI Benchmarks, PingPong, and PingPing are Single Transfer Benchmarks, SendRecv and Exchange are Parallel Benchmarks on 10-node Cluster of Pentium II.

of secure MPI frameworks developed for large-scale clusters distributed across wide area networks.

Data confidentiality in MPI-I/O. Prabhakar et al. designed and implemented a secure interface called MPISec I/O for the MPI-I/O framework [22]. MPISec I/O preserves the advantages of both parallel I/O and data confidentiality without significantly impacting performance of MPI applications. It is flexible for MPI programmers to manually set encryption rules in MPISec I/O. Data can be encrypted and written onto disks in MPISec I/O, then encrypted data can be read from the disks before being decrypted. There are two interesting features of MPISec I/O. First, MPISec I/O programmers need to carefully set up encryption and decryption rules in their MPI programs. Otherwise, some data may be either stored on disks without encryption or read without decryption and as a result, the MPI programs are unable to function properly until the rules are set in a correct way. Second, MPISec is not completely compatible with nonsecure MPI libraries. In other words, preserving data confidentiality in MPISec I/O is not transparent to MPI application programmers. One has to modify the source code of conventional MPI programs to improve security of the MPI programs. Apart from updating the source code of the MPI programs before MPISec I/O can be used properly, disk-resident data must be marked as encrypted or unencrypted.

Block ciphers. The Data Encryption Standard (DES) provides a relatively simple method of encryption. 3DES encrypts data three times instead of one using the DES standard [4]. 3DES is a block and symmetric cipher chosen by the US National Bureau of Standards as an official Federal Information Processing Standard in 1976. 3DES increases the key size of DES to protect against brute force attacks without relying on any new block cipher algorithm. A hardware implementation of 3DES is significantly faster than the best software implementations of 3DES [10] [15]. A software implementation of 3DES was integrated in ES-MPICH2. A hardware 3DES can substantially improve performance of 3DES-based ES-MPICH2.

In November 2001, the symmetric block cipher Rijndael was standardized by the National Institute of Standards and Technology as the Advanced Encryption Standard [6]. AES—the successor of the Data Encryption Standard—has been widely employed to prevent confidential data from being disclosed by unauthorized attackers. AES can be used in high-performance servers as well as small and mobile consumer products. AES is the preferred cryptographic algorithm to be implemented in ES-MPICH2, which was built based on symmetric block ciphers. Although AES introduces overhead due to additional security operations in ES-MPICH2, the overhead caused by AES in ES-MPICH2 can be significantly reduced by AES hardware architectures (see [19] for details of a highly regular and scalable AES hardware architecture).

Security enhancement in clusters. There are several research works focusing on the security enhancement in commodity clusters. For example, Lee and Kim developed a security framework in the InfiniBand architecture (IBA) [17]. For confidentiality and authentication, Lee and Kim proposed the partition-level and QP-level secret key management schemes. The security in IBA is improved with minor modifications to the IBA specification. Ramsurrun and Soyjaudah constructed a highly available transparent Linux cluster security model, which offers a new approach to enhancing cluster security [23]. Koenig et al. implemented a tool that monitors processes across computing nodes in a cluster [16]. The tool delivers real-time alerts when there are immediate threats. Similarly, Pourzandi et al. investigated the security issues of detecting threats and hazards in distributed clusters [21]. The aforementioned security solutions developed for clusters are inadequate to directly support security-sensitive MPI programs, because the existing security solutions generally require application developers to implement security functionality in their MPI programs.

8 Conclusions and Future Work

To address the issue of providing confidentiality services for large-scale clusters connected by an open unsecured network, we aim at improving the security of the message passing interface protocol by encrypting and decrypting messages communicated among computing nodes. In this study, we implemented the ES-MPICH2 framework, which is based on MPICH2. ES-MPICH2 is a secure, compatible, and portable implementation of the message passing interface standard. Compared with the original version of MPICH2, ES-MPICH2 preserves message confidentiality in MPI applications by integrating encryption techniques like AES and 3DES into the MPICH2 library.

In light of ES-MPICH2, programmers can easily write secure MPI applications without an additional source code for data-confidentiality protection in open public networks. The security feature of ES-MPICH2 is entirely transparent to MPI programmers because encryption and decryption functions are implemented at the channel-level in the MPICH2 library. MPI-application programmers can fully configure any confidentiality services in MPICHI2, because a secured configuration file in ES-MPICH2 offers the programmers flexibility in choosing any cryptographic schemes and keys seamlessly incorporated in ES-MPICH2. Besides the implementation of AES and 3DES in ES-MPICH2, other cryptographic algorithms can be readily integrated in the ES-MPICH2 framework. We used the Sandia Micro Benchmarks and the Intel MPI benchmarks to evaluate and analyze the performance of MPICH2.

Confidentiality services in ES-MPICH2 do introduce additional overhead because of security operations. In the case of small messages, the overhead incurred by the security services is marginal. The security overhead caused by AES and 3DES becomes more pronounced in ES-MPICH2 with larger messages (e.g., the message size is larger than 256 KB). Our experimental results show that the security overhead in ES-MPICH2 can be significantly reduced by high-performance clusters. For example, the overhead added by AES in ES-MPICH2 is reduced by more than half when the 6-node cluster of Intel Celeron is used instead of the 10-node cluster of Intel Pentium II. In addition to high-end clusters, the following two solutions can be applied to further reduced overhead caused by confidentiality services in ES-MPICH2. First, AES/3DES hardware implementations can lower security overhead in ES-MPICH2. Second, security coprocessors can hide the overhead by allowing the encryption and decryption processes to be executed in parallel with the message passing processes.

We are currently investigating varies means of reducing security overhead in ES-MPICH2. For example, we plan to study if multicore processors can substantially lower the overhead of confidentiality services in ES-MPICH2.

Another interesting direction for future work is to consider several strong and efficient cryptographic algorithms like the Elliptic Curve Cryptography in ES-MPICH2. Since ECC is an efficient and fast cryptographic solution, both the performance and the security of ES-MPICH2 are likely to be improved by incorporating ECC.

A third promising direction for further work is to integrate encryption and decryption algorithms in other communication channels like SHMEM and InfiniBand in MPICH2 because an increasing number of commodity clusters are built using standalone and advanced networks such as Infiniband and Myrinet.

The current version of ES-MPICH2 is focused on securing the transmission control protocol (TCP) connections on the internet, because we addressed the data confidentiality issues on geographically distributed cluster computing systems. In addition to the MPI library, other parallel programming libraries will be investigated. Candidate libraries include the shared memory access library and the remote direct memory access library. We plan to provide confidentiality services in the SHMEM and RDMA libraries.

Last but not least, we will quantitatively evaluate the performance of integrity checking services that are incorporated into the ES-MPICH2 framework. The goal of developing the integrity checking services is to reduce overhead of the services in ES-MPICH2.

9 Availability

The executable binaries and source code of ES-MPICH2 are freely available, along with documentation and benchmarks for experimentation, at http://www.eng.auburn.edu/~xqin/software/es-mpich2/.

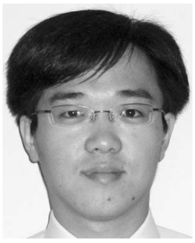
Acknowledgments

The work reported in this paper was supported by the US National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), as well as Auburn University under a startup grant and a gift (Number 2005-04-070) from the Intel Corporation. Mohammed Alghamdi's research was supported by the King Abdulaziz City for Science and Technology (KACST) and AL-Baha University.

REFERENCES

[1] Nat'l Security Agency. Nat'l Policy on the Use of the Advanced Encryption Standard (aes) to Protect Nat'l Security Systems and Nat'l Security Information cnss Policy no. 15 Fact Sheet no. 1, June 2003.
[2] I.F. Blake, G. Seroussi, and N.P. Smart, *Elliptic Curves in Cryptography.* Cambridge Univ. Press, 1999.
[3] R. Brightwell, D.S. Greenberg, B.J. Matt, and G.I. Davida Barriers to Creating a Secure mpi, 1997.
[4] D. Coppersmith, D.B. Johnson, S.M. Matyas, T.J. Watson, D.B. Johnson, and S.M. Matyas Triple des cipher Block Chaining with Output Feedback Masking, 1996.
[5] Intel Corporation. Intel mpi Benchmarks User Guide and Methodology Description, 2008.
[6] J. Daemen and V. Rijmen, *The Design of Rijndael.* Springer, 2002.
[7] D.E. Denning, "Secure Personal Computing in an Insecure Network," *Comm. ACM,* vol. 22, no. 8, pp. 476-482, 1979.
[8] J.J. Dongarra, S.W. Otto, M. Snir, and D. Walker, "An Introduction to the Mpi Standard," technical report, Knoxville, TN, 1995.
[9] W. Ehrsam, S. Matyas, C. Meyer, and W. Tuchman, "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard," *IBM Systems J.,* vol. 17, no. 2, pp. 106-125, 1978.
[10] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An fpga-Based Performance Evaluation of the aes Block Cipher Candidate Algorithm Finalists," *IEEE Trans. Very Large Scale Integration Systems,* vol. 9, no. 4, pp. 545-557, Aug. 2001.
[11] I. Foster, N.T. Karonis, C. Kesselman, G. Koenig, and S. Tuecke, "A Secure Communications Infrastructure for High-Performance Distributed Computing," *Proc. IEEE Sixth Symp. High Performance Distributed Computing,* pp. 125-136, 1996.
[12] A. Geist, W. Gropp, S. Huss-Lederman, A. Lumsdaine, E.L. Lusk, W. Saphir, T. Skjellum, and M. Snir, "Mpi-2: Extending the Message-Passing Interface," *Proc. Second Int'l Euro-Par Conf. Parallel Processing (Euro-Par '96),* pp. 128-135, 1996.
[13] R. Grabner, F. Mietke, and W. Rehm, "Implementing an mpich-2 Channel Device over Vapi on Infiniband," *Proc. 18th Int'l Parallel and Distributed Processing Symp.,* p. 184, Apr. 2004.
[14] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the Mpi Message Passing Interface Standard," *Parallel Computing,* vol. 22, no. 6, pp. 789-828, 1996.
[15] P. Hamalainen, M. Hannikainen, T. Hamalainen, and J. Saarinen, "Configurable Hardware Implementation of Triple-des Encryption Algorithm for Wireless Local Area Network," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing (ICASSP '01),* pp. 1221-1224, 2001.
[16] G.A. Koenig, X. Meng, A.J. Lee, M. Treaster, N. Kiyanclar, and W. Yurcik, "Cluster Security with Nvisioncc: Process Monitoring by Leveraging Emergent Properties," *Proc. IEEE Int'l Symp. Cluster Computing and Grid (CCGrid '05),* 2005.
[17] M. Lee and E.J. Kim, "A Comprehensive Framework for Enhancing Security in Infiniband Architecture," *IEEE Trans. Parallel Distributed Systems,* vol. 18, no. 10, pp. 1393-1406, Oct. 2007.
[18] J. Liu, W. Jiang, P. Wyckoff, D.K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen, "Design and Implementation of Mpich2 over Infiniband with rdma Support," *Proc. 18th Int'l Parallel and Distributed Processing Symp.,* p. 16, Apr. 2004.
[19] S. Mangard, M. Aigner, and S. Dominikus, "A Highly Regular and Scalable aes Hardware Architecture," *IEEE Trans. Computers,* vol. 52, no. 4, pp. 483-491, Apr. 2003.
[20] S. Matyas and C. Meyer, "Generation, Distribution, and Installation of Cryptographic Keys," *IBM Systems J.,* vol. 17, no. 2, pp. 126-137, 1978.
[21] M. Pourzandi, D. Gordon, W. Yurcik, and G.A. Koenig, "Clusters and Security: Distributed Security for Distributed Systems," *Proc. IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '05),* vol. 1, pp. 96-104, May 2005.
[22] R. Prabhakar, C. Patrick, and M. Kandemir, "MPISec I/O: Providing Data Confidentiality in MPI-I/O," *Proc. IEEE/ACM Ninth Int'l Symp. Cluster Computing and the Grid,* pp. 388-395, 2009.
[23] V. Ramsurrun and K.M.S. Soyjaudah, "A Highly Available Transparent Linux Cluster Security Model," *Proc. IEEE Int'l Performance, Computing and Comm. Conf. (IPCCC),* pp. 69-76, Dec. 2008.
[24] S. Saini, R. Ciotti, B.T.N. Gunney, T.E. Spelce, A. Koniges, D. Dossa, P. Adamidis, R. Rabenseifner, S.R. Tiyyagura, and M. Mueller, "Performance Evaluation of Supercomputers using hpcc and imb Benchmarks," *J. Computer and System Sciences,* vol. 74, no. 6, pp. 965-982, 2008.
[25] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler, and D. Noveck, "Network File System (nfs) Version 4 Protocol," 2003.
[26] D.S. Wong, H.H. Fuentes, and A.H. Chan, "The Performance Measurement of Cryptographic Primitives on Palm Devices," *Proc. 17th Ann. Computer Security Applications Conf. (ACSAC),* pp. 92-101, 2001.

Xiaojun Ruan (S'07-M'09-A'10) received the BS degree in computer science from Shandong University in 2005, PhD degree in computer science from Auburn University in 2011. He is an assistant professor of Computer Science at West Chester University of Pennsylvania. His research interests include energy efficient storage systems, parallel and distributed systems, computer architecture, operating systems, HDD and SSD technologies, and computer security. He is a member of the IEEE.



Qing Yang (S'08-M'11) received the BE and ME degrees in computer science and technology from Nankai University and Harbin Institute of Technology, China, in 2003 and 2005, respectively, and the PhD degree in computer science from Auburn University in 2011. He is currently a RightNow Technologies assistant professor of computer science at Montana State University. His research interests include wireless vehicular ad hoc networks (VANET), wireless sensor networks, network security and privacy, and distributed systems. He is a member of the IEEE.

Mohammed I. Alghamdi received the BS degree in computer science from King Saud University, Riyadh, Saudi Arabia in 1997, the MS degree in software engineering from Colorado Technical University, Colorado, the master's of science degree in information technology management from Colorado Technical University, Denver, Colorado, 2003 and the PhD degree in computer science from New Mexico Institute of Mining and Technology.

Shu Yin (S'09) received the BS degree in communication engineering from Wuhan University of Technology (WUT) in 2006, the MS degree in signal and information processing from WUT in 2008, and is currently, working toward the PhD degree in the Department of Computer Science and Software Engineering at Auburn University. His research interests include storage systems, reliability modeling, fault tolerance, energy-efficient computing, and wireless communications. He is a student member of the IEEE.

Xiao Qin (S'00-M'04-SM'09) received the BS and MS degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, and the PhD degree in computer science from the University of Nebraska-Lincoln, Lincoln, in 1992, 1999, and 2004, respectively. For three years, he was an assistant professor with the New Mexico Institute of Mining and Technology, Socorro. Currently, he is an associate professor with the Department of Computer Science and Software Engineering, Auburn University, Alabama. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. His research is supported by the US National Science Foundation (NSF), Auburn University, and Intel Corporation. He was a recipient of the US National Science Foundation Computing Processes and Artifacts Award, the NSF Computer System Research Award in 2007, and the NSF CAREER Award in 2009. He was a subject area editor of the *IEEE Distributed System Online* in 2000-2001. He has been on the program committees of various international conferences, including IEEE Cluster, IEEE International Performance, Computing, and Communications Conference, and International Conference on Parallel Processing. He is a senior member of the IEEE.