

# Trusted Passages: Managing Distributed Trust to Meet the Needs of Emerging Applications

**Proposed Work.** The inherent complexity of applications, technologies, and platforms in today’s large scale distributed systems makes it extremely challenging for open systems to provide trustworthy services to end-users. In this research project, we propose an approach that integrates modern system virtualization techniques as well as new methods for runtime trust monitoring and assessment. This approach dynamically creates and maintains what we term *trusted passages* across distributed and potentially untrusted execution platforms. Thus, even with an insecure Internet, our goal is to continually assess the distributed computational platforms being used by applications, and based on that information, provide trusted services to the applications.

Our goal is to permit even critical applications to use open systems, enabling them to produce, process, and distribute information in a timely fashion. Applications that can benefit by our work range from remote surveillance, to the operational information systems used in government and commercial settings, to those that run daily e-commerce web services and similar information processing tasks. Attacks on such applications and their infrastructure addressed by our work include compromised systems through rootkits, viruses, or other malicious tampering, and degradation caused by temporary system overloads or failures.

**Solution Approach.** The trusted passage abstraction leverages new capabilities soon to be part of most, if not all, computational and network platforms. These capabilities allow one to fully virtualize and isolate the guest operating system on each platform, shielding it from external attacks and from other domains on the same platform. As a result, trust management software can run in isolated *trust controllers*, which continually carry out the monitoring and assessment tasks needed to detect attacks and/or compromised systems, and which can repair or work around them upon detection. Multiple trust controllers located on different machines cooperate to determine and maintain sets of trusted machines and operating systems for use by distributed applications. *Trusted passages* denote these actively maintained sets of trusted applications running on virtual and physical machines. Applications can take advantage of trusted passages through advanced middleware like the platform-aware overlays developed in our own research and/or through explicit management APIs provided by trust controllers. In either case, by actively engaging with the trusted passage abstraction, applications can use their own methods to manage compromised components, systems, or machines.

**Intellectual Merit.** By continually assessing the trust placed on applications and the distributed systems on which they run, we make it possible to create trusted passages across potentially untrusted sets of machines. New online monitoring techniques run by trust controllers can operate without requiring applications or operating systems to be instrumented. New methods for dynamic trust management utilize novel trust models for distributed systems and applications. System level support leverages both the virtualization and isolation capabilities of modern platforms, strongly integrating trust and trust management into the basic infrastructure used by applications and systems. System-level support also provides “safe” areas from which recovery actions can commence, or where normal system operation can continue despite abnormal behavior of some components.

**Broader Impact** This research would not only foster close collaboration among security and systems research, but it would strengthen our ongoing interactions with multiple industry research partners, including Internet Security Systems (ISS), HP, IBM, and Intel. Our students are currently working with companies that need to use open systems for critical infrastructure applications, including through funded research and as interns at Delta Air Lines and Worldspan Corporation in Atlanta. Our innovative educational programs, including the Masters in Information Security (see <http://www.cc.gatech.edu/content/view/181/133/>) degree, our active summer program for undergraduate students and minority students, and our participation in CRA’s women in Computer Science program complemented by Georgia Tech’s Women Resource Center (see <http://www.womenscenter.gatech.edu/>), will allow us to engage a diverse set of students in this research project.

# Trusted Passages: Managing Distributed Trust to Meet the Needs of Emerging Applications

Mustaque Ahamad, Wenke Lee, Karsten Schwan

## Contents

<b>1</b>	<b>Executive Summary</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Approach . . . . .	2
1.2.1	Trusted Passages . . . . .	2
1.2.2	Using Trusted Passages – An Example . . . . .	2
1.2.3	Architectural Overview. . . . .	3
1.2.4	Localized Trust . . . . .	4
1.2.5	Distributed Trust . . . . .	5
1.3	Expected Outcomes and Deliverables . . . . .	6
<b>2</b>	<b>Selected Technical Details</b>	<b>6</b>
2.1	Platform Capabilities . . . . .	6
2.1.1	Preliminary Study . . . . .	6
2.1.2	Research Tasks . . . . .	7
2.2	Trust Controllers . . . . .	8
2.2.1	Preliminary Study . . . . .	9
2.2.2	Research Tasks . . . . .	10
2.3	Distributed Dynamic Trust Management . . . . .	10
2.3.1	Preliminaries . . . . .	11
2.3.2	Research Tasks . . . . .	11
2.4	Evaluation . . . . .	12
<b>3</b>	<b>Education and Outreach</b>	<b>14</b>
<b>4</b>	<b>Results from Previous NSF Grants</b>	<b>14</b>
4.1	System Area Networks and Cluster Computing . . . . .	14
4.2	Network-Aware Middleware and Online Quality Management . . . . .	14
4.3	Morphable Software Services: Self-modifying Programs for Distributed Embedded Systems . . . . .	15
4.4	Agile Security for Storing Sensitive and Critical Information . . . . .	15
4.5	Intrusion Detection . . . . .	15

# Trusted Passages: Managing Distributed Trust to Meet the Needs of Emerging Applications

## 1 Executive Summary

### 1.1 Motivation

Distributed systems and applications are so complex that it is becoming increasingly difficult for end users to understand or control (1) where their data will be accessed and stored, and (2) where their processing will be performed. This is because: modern information processing infrastructures routinely cache data, intermediate results and parameters; they routinely integrate data, mine it, or operate on it on dynamically selected application servers; and they are now beginning to extend these host-level actions to also make use of underlying platform elements. An example is the placement of certain Web services into Akamai's Internet cache engines. At the same time, businesses must use the service architectures and infrastructures provided by industry, to control costs, to be able to interoperate with their partners, and more generally, to carry out the distributed IT processes that have now become routine. One of our industry research partners, Delta Air Lines, for instance, interoperates with Worldspan (another one of our research partners) to search for available Delta flights for customers using Delta.com, using the aforementioned Web service infrastructures. Additional interactions with external partners concern processing ticket payments, revenue management, catering functions, and many others.

We ask the following questions about the distributed systems and environments in which applications critical to an enterprise's operational capabilities are run. First, to what extent can one trust the open service-based infrastructures companies must use to contain costs and to gain interoperability with external partners? Second, can open systems like these be used to construct distributed applications that deliver information critical to an enterprise's ability to function, in a timely fashion and with trustworthy results? Third, is it possible to use the cost-effective shared Internet-based infrastructures for critical information processing and delivery in place of expensive enterprise-specific or point-to-point solutions (e.g., Delta Air Lines leasing capacity from Sprint to connect its airports to its datacenter)?

Unfortunately, the answer to the questions posed above is that today's security technologies are insufficient to provide this type of trust for large, distributed applications. To understand why, consider how these applications operate by looking at a specific example, Rubis [8], the eBay-like open source three tier Web service. Instead of the traditional "one Web server, one client" model, this application relies on many application and backend server machines that respond to different parts of each request. Data is passed between databases, data processing applications, data format applications, and data serving applications. Moreover, request parameters and intermediate results are cached in various locations between clients, applications, and backends. Traditional security technologies, therefore, are unable to effectively monitor all of these interactions and make autonomous trust decisions for the user. For example, a VPN could secure the data traveling between the client and the server, but it cannot make guarantees about the processing that happens within the distributed system 'behind' the server. Should the client trust the data produced by the server?

This type of problem is not limited to online commerce and Web applications. Consider, for example, the critical nature of the aforementioned airline applications, where the company relies on a constant flow of information to keep its flights running smoothly. Planes are tracked in flight, people are tracked as they make reservations and check-in, and even food is tracked to ensure that each flight has the proper mix of meals for its passengers. These data are processed and used by airline employees, airport displays, partnering companies, and outside agencies such as the Federal Aviation Administration (FAA) and the Transportation Security Administration (TSA). All of this information passes through a large-scale distributed system that can store, process, create, and forward these types of events. In today's information-rich society, this system is a critical part of the airline business model. But how can users of the system know that the information they are receiving was produced by trusted components? How can the system know when to trust data input into the system? Again, traditional security solutions, which focus on secure storage and transmission of data,

are not adequate when data is processed and stored at multiple points that change over time. New security solutions are needed to ensure that all applications that produce and process the data remain trustworthy.

## 1.2 Approach

### 1.2.1 Trusted Passages

*Trusted passages provide the framework necessary to address the problem of securing applications that run on large-scale distributed systems.* Specifically, our research will explore dynamically managing trust for applications that execute on open distributed systems. From the client's perspective, any single component in a distributed system can be trusted only if it satisfies certain properties. These properties can vary based on the client's needs. Some examples include safety (e.g., correct execution of requested operations), proper handling of information, and acceptable response times. Likewise, the entire distributed application can only be trusted if each component that affects the application satisfies these properties. This last point is critical to understanding trusted passages. Information traveling through the components of a distributed application creates a virtual passageway. Therefore, the primary challenge for a client is to ensure that each part of the passage is trusted.

Dynamic trust management starts by continuously measuring the trust level of each component in the distributed system. We define a platform as *trusted* if it processes data without any tampering to the data or the processing platform. A trusted passage builds on this to also prevent manipulation of data in transit or the sending of corrupted data. While this definition, based on monitoring of components, is somewhat weaker than that of a system that is guaranteed to remain secure at all times, it is more practical for complex systems and is sufficient for the correct operation of important applications (i.e., to perform the steps originally programmed into the application without undetected modifications due to malicious attacks). A trusted passage accounts for all aspects of data processing, storage, and transport within the passage.

The trusted passage framework is composed of several components. A local *trust controller* monitors its host's activities using virtual machine introspection and innovative intrusion detection techniques. Multiple trust controllers are connected to create a distributed system that can use the local information from each host to provide dynamic management of trust and to construct trusted passages that meet the client's needs. New system-level abstractions support efficient trust controller operation, imposing only small overheads on application and system execution. The remainder of this proposal builds on these ideas to describe the research needed to understand and construct each piece of this system.

### 1.2.2 Using Trusted Passages – An Example

Government and industry are increasingly relying on complex distributed systems to form their core computing infrastructure. For example, companies like Google, Amazon, and eBay use tens of thousands of computers to support unique Web service applications. The Federal Bureau of Investigation, along with various state agencies, maintains a distributed database for crime-related information called the National Crime Information Center. Delta Air Lines, one of our partners in the CERCS (Center for Experimental Research in Computer Systems) research center, maintains a critical distributed system responsible for processing flight and passenger information. Each of these systems has unique requirements with regards to uptime, performance, storage, bandwidth, etc. However, they all utilize complex methods for processing and sharing data: (1) these demanding, distributed applications are considerably more complex than the traditional client-server model; and (2) data is passed throughout the system in complex paths, and it is stored, processed, and forwarded at many points between its source and destination.

To illustrate how trusted passages address the emerging security issues in these applications, consider the operational information system (OIS) used by Delta Air Lines. As shown in Figure 1, this massively distributed system combines transactional processing, with push-based event delivery and manipulation, with client-server actions at end points. Its purpose is to continually provide the company with up-to-date information about all of its flight operations, including data events about passenger boarding, flight arrivals and departures, flight positions, and baggage. Event generation, transport, processing, and output use a wide-area distributed network of end systems, servers, and networking equipment that connects them.

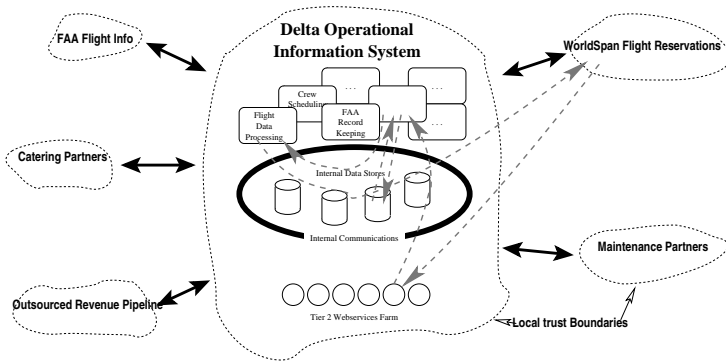


Figure 1: Delta Air Lines OIS.

Business logic applied to data creates meaningful information and generates the additional events used for tasks ranging from the update of airport terminal displays to notifications sent to caterers of passengers’ food preferences. This logic is run on multiple, high end server systems that continuously process input streams comprised of FAA flight position updates and Delta-specific flight information. These server systems form an Internal Event System (IES) that interacts with clients, both by generating continuously derived system-state updates and/or by responding to explicit external requests

for information. The large number of clients, the complexity of the business logic being applied and its working set size of hundreds of gigabytes, and a 24/7 uptime requirement dictate that business logic is implemented by multiple subsystems, some of which may be replicated across multiple nodes (and locations, for disaster recovery). Requirements on a critical system like this include high performance, high reliability and availability, and the ability to maintain constant service levels (as perceived by system clients), even under ‘unusual’ operating conditions.

The Delta OIS system exhibits many of the architectural features that make these distributed applications challenging to secure. *Trusted passages* are designed to provide security to an architecture with these key features:

- **Distributed Data Processing.** Data is generated, processed, and inspected at multiple locations between its origination and destination, across machines internal to a company and with a variety of external partners. In the Delta OIS example, this can be seen as inputs or data feeds (e.g., FAA data), business rules (e.g., internal flight processing), and transaction processing throughout the system.
- **Distributed Data Storage.** Data is stored at multiple locations. This can be for redundancy, locality, or other architectural reasons. This is seen as multiple databases in the Delta OIS example.
- **Architectural Redundancy.** Critical architectural components are replicated in order to provide the reliability and uptime that these applications demand.

Together, these features describe the framework of a complex distributed system. Trusted passages go beyond existing security techniques to actively ensure trustworthy operation of these complex systems.

### 1.2.3 Architectural Overview.

*Isolated Trust Controllers.* Figure 2 shows a trusted passage view based on different subsystems and machines jointly providing services to external clients. This is derived from a concrete example of the Rubis [8] three tier service architecture. The figure specifically shows multiple machines implementing different functionality like `http` content caching occurring between clients and Rubis Apache web server front end, parameters and intermediate result caching between front end, application servers, and backends, and the request- and load-dependent paths requests follow through the application. Most relevant to this proposal, however, is the additional presence of trust controllers running on each machine used by the Rubis application processes, with a vertical line between trust controllers and application processes indicating the strong, platform-enforced isolation between both. In addition to the the solid lines indicating application-level communications, dotted lines indicate communications between cooperating trust controllers. Our initial architecture implementation will assume that application and controller communications share a common network medium, potentially causing problems like network DoS attacks. We note, however, that as with ongoing efforts in platform

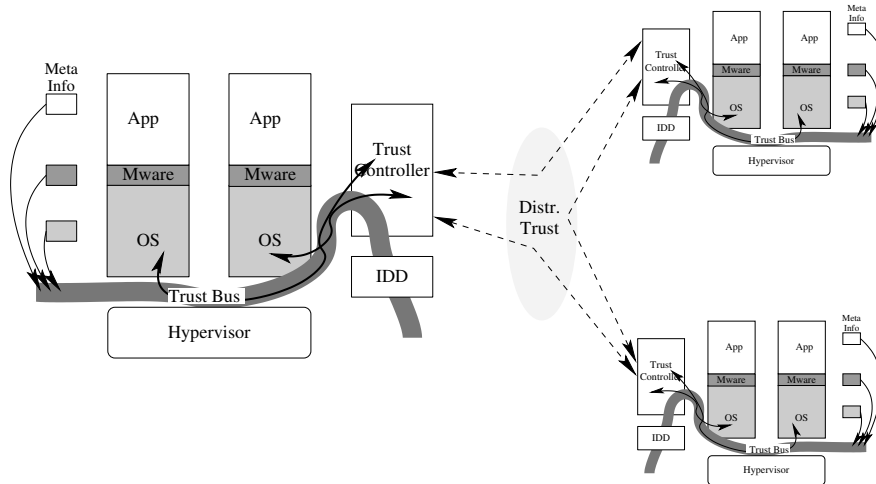


Figure 2: Trusted Passages Architecture.

virtualization, there are now intensive industry [10, 23] and academic [24] efforts to also add virtualization and isolation support to networks. Trusted passages can operate without such support, but their performance and ease of implementation will be improved by such future functionality.

Not explicitly shown in the figure is how trust controllers detect problems and make decisions about what machines and software systems to trust. This will be discussed in more detail in Sections 2.2 and 2.3. Here, we simply note that such decisions will be based on (1) *localized trust* – per platform monitoring of applications’ behavior, including their communication actions, and (2) *distributed trust* – information exchanged between multiple trust controllers. Trust decisions are made for each application, and the trust controllers to which an application has subscribed will endeavour to maintain some viable trusted passage that enables it to carry out its distributed processing tasks.

In our ongoing middleware research [7], we use a combination of active and passive standby nodes to attain reliable operation in the presence of nodes that can no longer be trusted. In the Agile Store project [30], we combine replication with secret sharing techniques to deal with dynamically detected attacks. In research conducted in peer-to-peer settings, we have used a combination of timeouts and result comparison to detect compromised subsystems, then react with runtime request re-routing and re-replication [46]. The important insight from this research is that trusted passages do not themselves assure applications the ability to smoothly resume operations when compromised subsystems or machines are detected. Instead, we envision a cooperative strategy, where a trusted passage is able to create a new, uncompromised domain on a machine used by the application, and where the application uses its own methods for re-joining the computations and data exchanges being performed. Conversely, trusted passages will attempt to detect compromised subsystems and machines under attack, but application specific handling may be necessary to deal with detected problems.

#### 1.2.4 Localized Trust

Trusted Passages demand that each participating platform actively monitors and manages its activities to ensure that certain trust properties are met. At a local level, this is handled by the trust controller. The trust controller has three primary responsibilities:

- *Cooperate with remote hosts* to support dynamic, distributed trust management.
- *Monitor the local host*, collect this information, and make a local trust decision.

- *Interact with local host applications*, to give applications access to trust information and therefore, the ability to deal with trust gain or loss.

In order to properly perform these tasks, it is critical that the trust controller be inherently trusted. Figure 3 demonstrates how this trust is provided using virtual machines to form a distinct boundary between the trust controller and the monitored platform. Leveraging existing work, the proposed architecture uses the Xen hypervisor [4] as a virtualization platform. The trust controller can execute in a privileged domain, and the monitored platform can execute in an unprivileged or user domain. This separation isolates the trust controller from traditional attacks. In addition to the isolation properties, each trust controller will operate in a protected environment complete with a hardened operating system and an intrusion detection system. Combining this with the isolation provided by the Xen hypervisor, the trust controller is able to operate at a significantly higher level of assurance than the monitored operating system. Finally, in order to ensure trustworthy communication between trust controllers, information must be securely transmitted. We plan to leverage existing work here and use techniques seen in virtual private networks (VPNs) such as encryption, authentication, and integrity checking using certificates.

Each trust controller will be responsible for monitoring any other domains running on the same hypervisor. The process of monitoring between virtual machines is known as virtual machine introspection (VMI) [18, 35, 25]. VMI allows one domain to monitor the current state of other domains including all physical memory, the CPU, device I/O, and any other data that passes between the hypervisor and domain. In order to facilitate interactions such as monitoring and response, we will create the XenAccess Library to provide the trust controller with a high-level view of each domain. This higher level abstraction will facilitate rapid development and exploration of new ways to leverage the powerful technique of virtual machine introspection.

Using the XenAccess Library, we propose to develop innovative ways to monitor a domain. First, trust controllers will monitor program execution and compare the results with execution of the same request on other nodes. This technique is related to the behavior distance work by Gao et al. [17]. However, in contrast to this work, we will explore the use of input beyond system calls. The XenAccess Library will allow behavior distances to be computed using anything from the raw memory in a process image to user-level API calls. We will research different distance metrics to understand which input provides the most useful measurement for trusted passages.

In addition to the behavior distance work, we propose using the XenAccess Library to provide input for anomaly detection of local program executions [14, 38, 15, 21, 45]. As described above, the XenAccess Library provides an opportunity to experiment with new types of system information as well as new abstractions. Our work will start with a traditional anomaly detection approach, and then determine which new data sources (e.g., resource utilization by the application) to add into the training set. The end result will provide an additional tool that will detect deviation from a pre-defined normal behavior.

Combining these two techniques, the trust controller will have a powerful view into the state of the monitored domain. If the anomaly detector indicates a problem, the trust level of that domain and its host can immediately go down. However, if the domain passes the anomaly detection test, then the behavior distance approach will provide a more fine-grained view into its operations. These two approaches complement each other in such a way that it would be much more difficult for an attacker to avoid detection while still carrying out a malicious task.

This section has provided a brief overview of the trust controller and its capabilities. For more in-depth technical information about the trust controller and VMI, refer to Section 2.2.

### 1.2.5 Distributed Trust

Trust controllers at each managed platform collect information about actions of domains that perform processing and communication for a trusted passage. We also require that multiple trust controllers executing at different machines coordinate and compare their results. A compromised domain’s observations are likely to differ from others and based on such comparisons, a trust value is associated with the domain. We plan to use models where trust values, that are meaningful at the application level, dynamically change in the range from 0 and 1 based on observed behavior of nodes. These values are used to represent the level of

trust a controller associates with a domain and the platform where it runs. Higher trust values indicate a more trusted platform that meets the needs of a trusted passage and lower values indicate that the resources at the platform cannot be trusted to support the passage. A trust value degrades rapidly when the trust controller suspects that its observations indicate anomalous behavior or when they differ from observations of other controllers. We want the trust value of a platform that effectively supports a trusted passage to gradually increase and get close to 1 with time. Notice that we establish a distributed network of trusted hypervisors and controllers, and the trust value of a platform is used to determine if the platform can meet the safety and performance needs of the trusted passage. In this sense, our network of trust controllers acts as a trust management infrastructure for a trusted passage. Our research will explore models for dynamically evolving trust values based on trust controller observations. We will also investigate how to effectively utilize dynamically computed trust values of multiple platforms to make resource management decisions that ensure that a trusted passage can be supported effectively.

### 1.3 Expected Outcomes and Deliverables

Our research will address multiple challenges to effectively support the trusted passage abstraction. Our primary thrust will be on building trust controllers that are run on a distributed set of platforms to manage the resources that support a trusted passage. To achieve this goal, we will monitor platform execution using virtual machine introspection and other performance metrics relevant to the trusted passage. The trust controllers at different nodes will share their local information and coordinate their actions to ensure that the entire passage fulfills our definition of trust.

This research will yield several advancements in the areas of establishing and utilizing dynamic trust and in host-based monitoring. Our primary focus will be centered on the following challenges:

- Discover new ways to monitor and manage trust in multiple, distributed platforms such that application relevant metrics are captured by the trust associated with the platforms that execute the application.
- Research the communication needs demanded by online monitoring and active trust management. Understand and evaluate these needs in order to provide an appropriate blend of performance and security within the trusted passage.
- Develop new software mechanisms that complement the hardware level management functions to support a trust management model that dynamically adapts to the observed behavior of platforms.

Trust controllers, which will support these ideas, will enable us to provide the rich distributed processing and communication abstraction that we call a trusted passage. We will demonstrate the usefulness of the trusted passage abstraction and our approach for implementing them by experimenting with lab-scale versions of applications like the Delta OIS and by working with open source three tier Web service applications like Rubis.

## 2 Selected Technical Details

Below, we describe some of the key technical problems and the approaches we propose for their solution.

### 2.1 Platform Capabilities

#### 2.1.1 Preliminary Study

In the extended summary of this proposal, we explained how the virtualization capabilities of modern platforms permit us to implement trust controllers in their own domains, isolated from guest operating systems and applications. This section explains in more detail how dynamic trust management can be efficiently achieved on future virtualized platforms. Three facts about future platforms enable the innovative methods we plan to explore for managing trust dynamically:

1. *From system-level observability to localized trust.* On virtualized platforms, all interactions of guest operating systems with devices pass through isolated external agents, termed *Service VMs* or *Device Driver Partitions*. Since all guest OS or application actions with external parties also involve interactions with communication devices, they are therefore, observable without requiring modifications to guest operating systems, to applications, or to the middleware they are using. There are limits to guest OS observability, however. System calls, for example, cannot be observed without additional techniques. This is addressed by the novel introspection techniques described in Section 2.2.
2. *TrustBus – deriving semantic information through runtime interpretation and analysis.* Hypervisors and Service VMs observe the actions of virtual machines at a low level of abstraction, involving disk blocks, Ethernet frames, etc. It is difficult, therefore, to identify specific behaviors (e.g., high external request rates) as problematic, to attribute observed problems to specific components, or to predict how behavior changes may affect other parties or machines. We will raise the level at which VM actions can be observed. This will involve generalizing the existing “Xenbus” abstraction for block-level interactions across virtual machines to become what may best be termed a “TrustBus”. The idea is to automatically derive higher level semantic information with current VM actions, using available computational cores. An example of such information is timing data about VM actions, which can be used to cluster multiple actions into single higher level actions and then used to detect correlations across multiple higher level actions (e.g., identifying them as request/response actions). Our initial thoughts about the TrustBus and its implementation appear later in this section. Figure 2 indicates the way in which the TrustBus extends natural VM-Service and VM interactions and the use of metadata in the trust controller’s execution. In particular, data provided by the TrustBus will be used by the trust controller for monitoring the local host.
3. *Concurrency, asynchrony, and distribution.* Since trust controllers run in their own virtual machines, their execution is concurrent with that of guest OSs and applications. Particularly for next generation multi-core hardware, this implies that a trust controller can continuously monitor the operation and behavior of the other virtual machines running on the platform and that a TrustBus can continually refine its interpretations of current VM behavior. It also means, however, that the event detection and repair actions implemented by trust controllers will typically execute asynchronously with guest actions. As a result, detected errors or unusual behaviors may spread to other parts of guests’ virtual machines or to other virtual machines before repair actions or defenses are put in place. The need for extending trust management across multiple machines is apparent, indicated as interactions between multiple trust controllers in the figure. Distributed trust control is described in more detail in Section 2.3.

### 2.1.2 Research Tasks

We propose to extend the XenBus abstraction implemented in the open source Xen hypervisor with additional functionality.

*TrustBus will apply analyses to block-based data streams* to automatically derive important metadata about the external interactions of a VM. We expect to be able to statistically determine, through clustering and correlation analysis, the following metadata about these data flows. (1) Timing analysis will identify typical separations in time between different input and output actions of VMs and by correlating that with current virtual machine speed, diagnose potentially unusual VM behavior. (2) Identification of request/response patterns will use an analysis that first clusters multiple data blocks into meaningful request or response messages and then correlates both. The timing model for typical request/response behaviors of virtual machines can then be used to differentiate its normal and abnormal behaviors. Externally injected metadata about what VMs are currently doing (e.g., are they running a Web server in a three tier web services infrastructure?), from applications or from middleware would be useful here, but such metadata inputs may themselves be compromised and produce suspect results that need to be compared with similar results elsewhere. (3) Deep packet inspection will copy and buffer packets over windows that capture entire requests/responses and then analyze them for validity. Our analysis approach leverages the fact that parameter and results caching have been shown highly useful for three tier Web service infrastructures. Valid behaviors of VMs, therefore, should exhibit reasonable cache hit and miss properties. The results of these (initial) analyses will be used by the

local trust controller for further analysis. For example, the local timing analysis result can be compared with the results of remote hosts to detect more subtle abnormal behavior.

*TrustBus will provide means for associating trust actions with external communications.* A variety of simple security actions are easily associated with the TrustBus data fast path. The packet copying and buffering actions needed for online analysis are an example. Other examples are simple packet filtering or elimination steps, such as forbidding a VM all access to some device (e.g., prohibit all external communications), or disable access to/from a USB device like a memory stick being plugged into a machine when such a device does not properly authenticate. Other simple actions are packet replication to remote machines for long term storage or extensive analysis. However, to filter application-level messages, for instance, would require first reassembling them with the protocol used for packet fragmentation, then inspecting selected packet content, then eliminating suspicious messages. While high end network devices have been shown capable of performing tasks like these at link speeds [19], such real-time analyses and actions may not be possible in trust controllers. On each single platform, therefore, we envision trust controllers as playing diagnostic roles and as carrying out high level repair actions like repairing or deleting and restarting untrusted VMs. Interesting actions concerning distributed trust and trust management carried out by multiple trust controllers are described in Section 2.3.

*TrustBus will have explicit support for dynamic trust management across distributed platforms.* Local trust controllers will determine the actual analysis and trust actions associated with TrustBus. TrustBus itself, however, will provide a basis for easily “splicing” together, in a distributed framework, multiple TrustBusses into an information delivery framework of use to distributed trust management. This will enable (1) direct communications of trust controllers with the remote controllers of whom they are aware, and (2) associating additional communication actions with a local TrustBus, thereby extending it to remote machines and their trust controllers. This research component will leverage our substantial ongoing research on efficient publish/subscribe communication middleware [12].

## 2.2 Trust Controllers

The primary duty of the trust controller is to monitor the local host and communicate the monitoring results with other trust controllers. The monitoring capabilities rely on information provided by the TrustBus (see Section 2.1) and virtual machine introspection (VMI) [18, 35, 25]. VMI will be built using the Xen hypervisor. Xen is an open source implementation of a hypervisor (i.e., virtual machine monitor) that was created out of a research project at the University of Cambridge [4]. In Xen, one physical computer runs multiple operating systems in isolated execution environments called domains. The first domain is started at system boot time and is referred to as domain 0, or simply dom0. The other domains are referred to as the user domains or domU. Dom0 is granted special privileges by the hypervisor including the ability to view and control each of the other domains. The VMI architecture will leverage Xen by placing the trust controller on dom0. The operating system monitored by the trust controller, hereafter referred to as the guest OS, will run in domU.

With this architecture, shown in Figure 3, the Xen hypervisor provides strong isolation between the trust controller running in dom0 and the guest OS. This isolation allows software run within dom0 to behave reliably even if the guest OS is under attack or suffering from a fatal software flaw (e.g., kernel panic or the infamous “blue screen of death”). In addition, the Xen architecture provides strong performance isolation which prevents any events on the guest OS from impacting the performance of VMI [4, 9].

To enable VMI, we will extend our XenAccess Library to satisfy the monitoring requirements. Xen only provides low level functions for access between domains. For example, memory access is done by requesting to see a specific physical page number. The XenAccess Library provides a higher-level abstraction that allows the programmer to do tasks such as viewing a specific kernel symbol in memory, listing all processes, or viewing the memory space for a particular process. In addition, the XenAccess Library will be extended to support monitoring of the CPU context for each domain, all device I/O including network and disk I/O, and hardware management interfaces (e.g., performance counters). As discussed in Section 2.2.1, we have already shown the viability of the XenAccess Library and, in general, VMI using Xen. Furthermore, the Xen hypervisor requires no modifications to support this advanced functionality. Therefore, we are confident

that combining Xen with our XenAccess Library will create a powerful architecture to support host-based monitoring.

Leveraging this architecture, we will extend the behavioral distance work of Gao, Reiter, and Song [17]. The original approach to behavioral distance monitored system calls of multiple replicas of the same process. By comparing the system calls for each process, a distance can be calculated to represent the similarity between each process’ execution path. This approach makes mimicry attacks [47] more difficult because an attacker has further restrictions on allowable execution paths. Some attacks that would have bypassed other system call intrusion detection techniques would be caught by the behavioral distance approach. However, this approach is limited to the use of system calls. Clearly, system calls provide one important view into a program’s behavior since these calls represent all interaction with the kernel. But it is an incomplete view. For example, an attack can evade detection if it contains no system calls or only the ones that normally occur.

We will enrich the data used for behavioral distance measurements. Using VMI, we can collect data at different semantic levels. On one end of the spectrum, we can view each instruction executed by the process and on the other end we can monitor the application-level API calls. In between we will still use system calls and will also have access to information such as the process’ memory space, program text, hardware access, performance metrics, and more. In addition to using VMI, we can also incorporate information from the TrustBus (e.g., timing analysis results) for analysis. We will evaluate the usefulness of all of this data and create new behavioral distance models that can be used by the trust controllers to evaluate each host.

In addition to the behavior distance analysis, the trust controller also performs anomaly detection of local applications. Again, rather than limiting to system call data, we will study how to make effective use of the rich data sources provided by VMI and the TrustBus. We have extensive experience in feature construction and selection for anomaly detection [36, 37, 14, 13] that can help the proposed research.

### 2.2.1 Preliminary Study

In order to demonstrate the viability of this approach, we have begun implementation of the XenAccess Library. Currently, this library is capable of accessing most of the kernel memory region. Using this library, it is trivial to write monitors that perform integrity checks and other monitoring functions. For example, pseudo-code for a simple module that monitors the system call table is shown below:

```

xa_init();
memory = xa_access_kernel_symbol("sys_call_table");
while (1){
    sleep(SLEEP_INTERVAL);
    if (memcmp(memory, known_good_syscall_table) != 0){
        /* syscall table has changed! */
    }
}
munmap(memory);
xa_destroy();

```

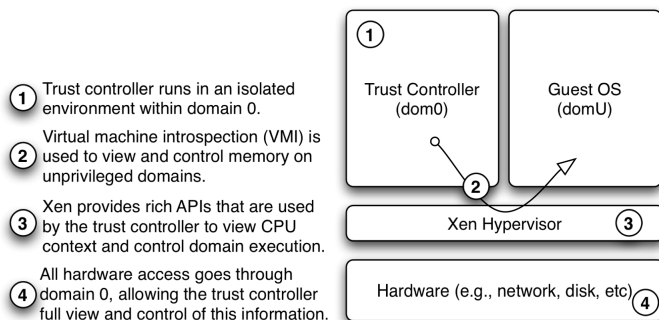


Figure 3: High-level view of the VMI architecture.

In this example, each function prefixed with `xa_` is part of the XenAccess Library. The following paragraphs provide a brief description of how the XenAccess Library operates.

Xen domains are not guaranteed to receive contiguous memory regions from the hypervisor. Therefore, each domain uses pseudo-physical addresses in order to make the memory appear contiguous. In order to distinguish the hardware's actual physical addresses from the pseudo-physical addresses, Xen refers to the actual physical addresses as machine addresses and the pseudo-physical addresses as physical addresses. Taking into account the additional abstractions used by Linux, the XenAccess Library must know how to translate memory addresses between machine addresses, physical address, and virtual addresses. The translation between machine and physical addresses is performed using lookup tables provided by the hypervisor. However, the translation between physical and virtual addresses requires knowledge of Linux.

The Linux kernel occupies the highest 1GB of virtual address space<sup>1</sup>. This memory region is broken into three distinct areas as described in [5] and [43]. The first area is `0xC0000000 - high_memory`, where `high_memory` is a variable set within the kernel. This region is linearly mapped into physical memory meaning that one can translate a virtual address into a physical address by subtracting `0xC0000000`. The second area is `high_memory - 0xFE000000`. This region is paged into physical memory meaning that one must follow a series of page tables in order to translate a virtual address into a physical address. The final area is `0xFE000000 - 0xFFFFFFFF`. This region uses a variety of different mappings for specialized functionality. Currently, XenAccess can access any memory that falls within the first two regions. This includes all of the kernel text, data structures, loadable modules, and the kernel's dynamically allocated memory.

Using the current capabilities of the XenAccess Library, we have implemented monitors for the system call table, the interrupt descriptor table, and the loadable kernel modules. We have also shown that kernel memory can be restored to correct values if a change is detected. Furthermore, micro-benchmarking has shown that VMI has a negligible impact on memory performance in the guest domain. These results indicate that VMI is a promising approach. Through additional work, it can be used to monitor a wide variety of information in the guest domain and provide this vital information to the trust controller.

### 2.2.2 Research Tasks

In summary, the primary research thrusts in support of the trust controller are outlined below:

- Extend the behavioral distance technique to detect differences in code execution on multiple platforms by incorporating richer data sources from VMI and the TrustBus. The technique must take into account expected variances on each platform while still identifying when two execution paths are different enough to suggest malicious intent or code tampering.
- Fully develop the XenAccess Library to support robust monitoring of memory, CPU context, disk access, network communication, hypervisor data, and performance metrics. For each of these techniques, ensure that the data can be captured efficiently while minimizing the ability for an attacker to corrupt the logging data.
- Leverage the additional data access techniques provided by the XenAccess Library and information provided by the TrustBus to provide new input data, and better accuracy, for host-based anomaly detection algorithms.

## 2.3 Distributed Dynamic Trust Management

The trust controllers observe the execution of local virtual machines and are able to monitor their actions. The processing done by a virtual machine and the data accessed by it at node *A* could belong to an application that is initiated at a remote node. The trust controller at such a remote node will be interested in the behavior of the virtual machines at node *A* that executes its application. However, the controller at the remote node may not be interested in the low level activity observed at the VMI and TrustBus levels. Ideally, a high level

---

<sup>1</sup>A recent Linux kernel patch allows for configuration of this number, but for simplicity of discussion we will refer to the traditional memory layout. Other layouts are easily addresses using similar techniques.

notion of trust that captures the ability of node  $A$  to meet the safety or performance needs of the application is what is desired. In this sense, our use of trust differs from trust management that has been explored in systems that are primarily concerned with authorization and access control (e.g., Keynote [39, 40] and more recent work [41]). Our notion of trust does share some characteristics with peer-to-peer trust or reputation that has been explored [1, 27]. However, as it will become clear, our focus is on supporting trusted passages and we explore a distributed trust model that is well suited for this abstraction.

### 2.3.1 Preliminaries

In an open environment, a distributed trusted passage relies on resources from nodes that are not completely under the control of the application that sets up the trusted passage. Furthermore, a binary characterization of a node as either one that is completely trusted or cannot be trusted at all is not very useful because any node can potentially be compromised or could fail to meet the needs of a trusted passage. An application would like to know how well a node is meeting its commitment to support its trusted passage. Trust is a quantitative measure of a node’s ability to meet a passage’s needs and it can be used in a number of ways. At the time a trusted passage is created, trust values can help determine the nodes that are best likely to meet the needs of the passage. If trust in some nodes degrades during the execution of an application, they can be replaced with other nodes that have higher trust values. For trust to be useful in such decisions, the model which defines trust values must have several properties. Trust must rapidly degrade when a trust controller determines that a node fails to meet safety or performance needs of a passage repeatedly. The trust of a node should be allowed to increase when it consistently meets a trusted passage’s requirements.

In the past we have explored distributed trust models in two different domains where the trust value associated with a node ranges between 0 and 1. A trust value close to 1 indicates that the node consistently guarantees the availability of resources agreed by it and it returns correct results for processing done at it. In a system that efficiently disseminates information to a large number of nodes, a high trust value reflects the fact that the node forwards a message to its children in a timely fashion without modifying it [26]. In another work, we used trust values to choose nodes where processing of frames of a video stream could be done [46]. Other researchers have also explored trust aware scheduling as well as data replication [52, 11, 3]. In such past work, trust management is typically based on experiences that are observed by the application itself rather than trust controllers that run in an isolated manner at all nodes.

### 2.3.2 Research Tasks

Our research challenge is to develop and implement a distributed dynamic trust management framework that can take trust controller observations as input and produce meaningful trust values for supporting trusted passages. In particular, we need to explore what monitoring actions of trust controllers are relevant for trust values, and how to change trust values based on observations across nodes. We must also address how the resources allocated to a trusted passage must be changed as trust values vary.

1. The trust value of a node increases when there is no or little malicious or anomalous activity observed at it and it is able to correctly complete processing or communication tasks assigned to it by a trusted passage. However, the observation of events that affirm such behavior in a timely manner is challenging. Consider the scenario in which an application using a trusted passage is started at node  $A$  and the passage makes use of resources that are available at node  $B$ . The local trust controller at node  $B$  is in the best position to report anomalous activity at  $B$  and report it to the trust controller at node  $A$ . However, this trust controller may not be able to know if the virtual machine (VM) executing  $A$ ’s application has produced correct results. The requesting VM at  $A$  may not be able to establish correctness without executing the same computation at multiple nodes. If results are not returned in a timely manner, the application may report to its local trust controller the fact that node  $B$  is unresponsive. In case the application VM on  $A$  is compromised, it may falsely make such reports to degrade trust in node  $B$ . Clearly, we must establish events and communication paths between trust controllers to collect information that would be relevant to dynamic trust management. In our research, we will identify the variety of events and observations that are meaningful across nodes and their relative

importance for change in trust values. Thus, our goal is to move away from the simple model of positive and negative experiences that are typically used in dynamic trust models. Instead, we will focus on how to translate system level observations at the VMI and TrustBus levels into actions and events that permit trust controllers to dynamically change trust values they associate with each other and the VMs that are managed by them.

2. As an application makes use of a trusted passage, the question we like to ask is which nodes are meeting the requirements of the passage and which ones have failed to do so. Although detailed information can be useful, ideally one would like to separate highly trusted nodes from those that are less trusted based on a high level metric. In other words, how can we translate low level observations to a quantitative trust metric that is helpful in supporting trusted passages. We have explored models where trust values range between 0 and 1, where a value close to 1 indicates that the node can be highly trusted. Although reputation and other peer-to-peer trust systems have addressed the problem of trust evolution, we believe that generic models that have little domain related information cannot be highly effective. Thus, our challenge will be to determine the relative importance of various observations and the change in trust values as a result of such observations, and the entities that provided these observations. We will explore several models for upgrading and downgrading trust values dynamically based on the observations. Also, unlike other environments where there is little control over a node's behavior, the availability of trust controllers provides us a way to manage virtual machines when their trust values drop below a threshold. For example, such machines may be prevented from communicating or simply be terminated. Thus, active management of trust also becomes a challenge. We will explore the tradeoffs between maintaining certain levels of trust vs. the the benefits gained when resource allocation decisions for trusted passages are based on trust values. For example, the nature and frequency of monitoring actions and the overhead associated with them can be controlled based on the target virtual machine's trust level. Our expected key findings in this area include models for dynamically computing trust values and distributed algorithms that can be used by the monitoring system to manage virtual machines when their trust starts to degrade.
3. The idea of maintaining trust values in nodes is motivated by the fact that it will make it easier to find the set of nodes that could meet the needs of a trusted passage. The properties of trusted passages depend on the the application supported by them. For example, one application may be concerned with proper handling of data while another one may be most concerned about the correct execution of some code at nodes that have certain kind of data. In our research, we will develop techniques to bridge the gap between trusted passage properties such as confidentiality or availability and the choice of nodes which are chosen to support the trusted passage. Our initial work in trust sensitive scheduling has provided promising results for timely completion of tasks by choosing higher trust nodes. However, techniques for correlating trusted passage properties with trust values need to be developed and validated.

## 2.4 Evaluation

**Summary of Proposed Research** The system-level and distributed systems technologies proposed here will allow open distributed platforms to better service critical applications, despite external disruptions and attacks, even when applications or operating system components are compromised. In our *trusted passages* paradigm, trust controllers executing at each node continually assess the trust levels of their local machines and interact across multiple machines to maintain high level trust values for use by distributed applications. Such values help the applications choose a set of machines to deliver to them the services they require to operate. To demonstrate that this paradigm is effective in meeting the needs of critical applications, we will use well-defined service requirements of applications to evaluate this research.

The proposed research will not only produce new software technologies, but it will also produce concrete evaluation artifacts that will be of benefit to the broad community. We term these the Critical System Benchmarks. One benchmark will be derived from the operational information systems servicing the needs of companies like Delta Air Lines or Worldspan, with whom we are collaborating. The other will emulate the needs and behavior of remote instrument access and real-time collaboration, based on our ongoing interaction

with computational scientists at Georgia Tech and at Oak Ridge National Labs. These representative critical applications will run across an emulated Internet (using Georgia Tech’s NetLab facility, a GT offspring of Utah’s Emulab), and they will utilize known attack scenarios and be exposed to future attack methods. We actively collaborate with Internet Security Systems (ISS) and they will help us develop the attack models. We will also use results of other researchers, in particular, the DETER teams, in developing representative attack scenarios. Developing meaningful evaluation criteria for the security or trustworthiness of the trusted passages is all by itself a challenging task. Again, we will work with our industry partners and other researchers to develop application relevant measurable metrics as well as potential attack scenarios that seek to degrade them.

**Evaluation and Metrics.** Key issues concerning the viability of the proposed approach include (1) the effectiveness of trusted passages in meeting application requirements under various attacks, (2) the overheads implied by the presence of multiple virtual machines on a single platform, at minimum including the VM running the trusted controller and the actual VMs using the trusted passage, (3) the overheads of on-line monitoring, state collection, and analysis necessary for diagnosing compromised machines, (4) and the overheads implied by dynamic trust management across nodes along with the costs incurred from the use of replication or redundancy with machines and network links for meeting trusted passages needs in open distributed systems.

While industry is working hard to make it “cheap” to run many virtual machines on a single physical platform, detailed experiments and microbenchmarks must be run in which virtual machines with guest OSs and applications co-exist with different configurations of additional VMs running trust controllers. This may lead to alternative solutions for associating trust analysis and management actions with virtual machines, an example being to gain performance advantages or improved predictability via host offloading [19].

Multiple metrics will be used to evaluate our research results, ranging from the aforementioned overheads, to the ability to diagnose compromised single or set of VMs, to the speed with which repair actions can be carried out, to the delay with which trust can be re-acquired after compromises or when attacks are detected, to application-specific measures for critical applications like end-to-end delay and the predictability of delays experienced in trusted passages vs. in general open vs. closed system solutions. We are fortunate to be able to have the facilities and experiences with realistic applications that are required to carry out these complex tasks.

**Timeline.** With this background, the proposed research will be conducted so as to ensure both meaningful intermediate deliverables and integration of different research components.

*Year 1:* (1) Develop basic trusted passage technologies. This includes the software architecture comprised of trusted controllers, their on-machine and cross-machine interactions using some initial trust model and distributed overlay management techniques. Initial benchmark development by leveraging our multi-year investments in high performance middleware and in real-time remote instrument access, as well as our ongoing collaboration with Delta Air Lines in the Enterprise domain. Construction of suitable testbeds and attack scenarios using the facilities already present in our labs and interacting with our ISS industry partner. Initiative development of advanced diagnosis and introspection techniques. (2) Develop preliminary metrics of security or trustworthiness of the trusted passage.

*Year 2:* (1) Mid-year demonstration of sample, dynamically constructed and managed trusted passages for the two benchmarks developed in Year 1. Different management actions are demonstrated for each, focused on dynamic overlay management for the remote collaboration benchmark vs. for the enterprise benchmark, focused on online repair of compromised VMs coupled with network control to identify and isolate compromised VMs and networks. Trust models and passage management actions associated with them address relevant end-to-end characteristics, including end-to-end delay in information delivery and trust in the results produced by distributed components. (2) Microbenchmarks to measure monitoring, repair, and isolation overheads, the goal being to better understand how to better integrate trusted passage technologies into future systems. (3) Experimentation with new hardware platforms, including Intel’s VT-technology-enabled ia86 machines. The budget proposed for this effort does not contain hardware funds to acquire these

machines, as we expect to receive additional funding from companies like Intel for these purposes. (4) New methods for online diagnosis and possibly, repair, addressing both the efficacy of our initial approaches and their overheads. (5) A set of representative attack scenarios.

*Year 3:* (1) Experimentation with a rich set of benchmark behaviors and trusted passage solutions, focused on scaled hardware platforms using Netlab and PlanetLab (the latter principally to evaluate trust algorithms). Experimentation with variety of attack scenarios. (2) Documentation in order to deliver benchmarks to the broader community. (3) Creation of realistic trusted passages for VT-enabled hardware. (4) New approaches to trusted controller isolation in response to measured overheads.

### 3 Education and Outreach

The Georgia Tech Information Security Center (GTISC) has created a number of innovative educational programs that will benefit from the research described in this proposal. We have information security related certificates at the undergraduate and graduate levels and also have a professional Master's level degree program in information security. Students from all these programs will be engaged in the research. In particular, we will develop projects in courses taught by the PIs, including operating systems, secure computer systems, network security and information security laboratory, that incorporate aspects of research that is described here. We are actively working with our industry partner ISS to enhance the Georgia Tech Information Security Center HoneyNet [5]. We have and will use data collected in this HoneyNet and from our application work with Delta in both undergraduate and graduate courses.

GTISC also has a very strong outreach program that includes regular security summits, industry leaders lecture series and a visitor program. In January 2004, we hosted the Accelerating Trustworthy Internetworking Workshop 2004, and more recently we have organized summits on VoIP security and wireless security. We will continue incorporating our research results from our Cyber Trust research activities into education and outreach activities.

## 4 Results from Previous NSF Grants

### 4.1 System Area Networks and Cluster Computing

The ASAN (Active System Area Networks) project has been using the Intel IXP 1200s to explore the potential of smart communication co-processors in cluster machines. This project, led jointly by Prof. Schwan in the College of Computing and Prof. Yalamanchili in the School of Electrical and Computer Engineering, was funded by NSF and DOE for understanding the roles of the communication co-processors that will be part of future cluster machines. Tangible project results to date include: (1) the demonstration of configuration on network interfaces (NIs), including last generation NIs supporting 155MB ATM [44] and NIs that have direct access both to disks and to multiple network interfaces (using Intel's I2O boards) [31], (2) experimentation with field programmable gate arrays (FPGAs) to run certain data-intensive streaming computations at link speeds, with particular focus on packet scheduling [48] for dynamic quality management, (3) demonstrations of the advantages of using an NI's embedded processor to handle meta-information and perform computationally non-intensive operations best run 'close' to the network [44].

### 4.2 Network-Aware Middleware and Online Quality Management

The proposed work will leverage two ongoing NSF-funded projects, Netreact (award ANI-0230841) and Scientific-XML (ACI-0234323). Netreact is developing middleware that has the ability to be configured and/or interact with network-level functionality. One project outcome is a software architecture that enables network awareness for both publish/subscribe [6, 32] and grid [6] software. We intend to leverage this middleware architecture for this effort, thereby able to focus the proposed work on the trust and security implications of constructing, maintaining, and configuring overlays across multiple machines and their virtual machines. Another outcome of our middleware research relevant to the proposed effort are results that concern

platform-level support for high performance distributed applications and middleware. Both the enterprise and scientific applications to be used as examples in the proposed work have been developed in part through our ongoing middleware research [20, 49], thereby again permitting us to focus this research on trust and security issues instead of application development. Second, platform support for network-aware middleware and applications has led to our work on the DProc kernel-level monitoring infrastructure [2]. The system-level monitoring mechanisms developed in DProc constitute a concrete subset of the cross-machine, distributed trust and control algorithms this project will develop for Repair Partitions in conjunction with Trusted Controllers.

### 4.3 Morphable Software Services: Self-modifying Programs for Distributed Embedded Systems

This NSF ITR project (CCR-0326396, started in Sep. 2003) concerns the creation of new middleware and systems technologies for distributed embedded systems. To date, its main results have been contributions in which compiler techniques are shown useful to reduce the total amounts of state migrated across machines, system-level scheduling techniques can jointly improve power usage of CPU and associated devices (Schwan), and middleware-level solutions support network-aware operation in distributed embedded systems (Schwan, Eisenhauer). This ongoing ITR award has helped us with some initial work that is relevant here, including the ‘kernel plugin’ idea [16] and in new work on virtualizing heterogeneous multi-core architectures (with additional funding from Intel Corporation).

### 4.4 Agile Security for Storing Sensitive and Critical Information

CCR-0208655: Agile Security for Storing Sensitive and Critical Information (PI: Mustaque Ahamad, Co-PIs: Douglas Blough, Wenke Lee and H. Venkateswaran), September 2002 to August 2005. In this Cyber Trust program funded project, we are exploring a research approach in which the underlying systems deploys mechanisms to become aware of possible faulty behavior of services or malicious requests from clients. Such awareness is used to harden protocols to deal with the faulty behavior when necessary. In the absence of such behavior, the system can focus on delivering best performance or richer functionality. We have obtained results that explore these principles in the context of a storage service. We have developed methods to store secure data in a distributed system such that security requirements such as confidentiality, integrity and availability can be adapted based on application needs or system conditions. We have developed a variety of responsive protocols that adapt to observed malicious behavior of nodes or uses and the security overheads depend on the degree of malicious activity that is encountered. Several papers [34, 28, 33, 29] describe the results that have been obtained in the course of this research.

### 4.5 Intrusion Detection

The research accomplished under the support of **CCR-0311024** *Intrusion Detection Techniques for Mobile Ad-Hoc Networks* (Aug 2003 - Aug 2006) includes publication in the *ACM/Kluwer Wireless Networks Journal* [50], the 7th International Symposium on Recent Advances in Intrusion Detection (RAID) [22], and the 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005) [51]. These papers describe anomaly detection algorithms, and architectures and protocols for distributed intrusion detection, as well as an evaluation environment. Lee is working with two small companies to implement intrusion detection systems (IDS) on mobile devices for the U.S. military. The research accomplished under the support of **CCR-0133629** *CAREER: Adaptive Intrusion Detection Systems* (Aug 2002 - Aug 2007) includes publications in the 2001, 2003, 2004, and 2006 IEEE Symposium on Security and Privacy [37, 14, 13, 42]. These papers describe theoretical results as well as practical static and dynamic analysis techniques for program anomaly detection. In this research, we will build on these results when developing new techniques for building anomaly detection models for complex application software.

## References

- [1] K. Aberer and Z. Despotovic. Managing trust in a p2p information system. In *CIKM*, 2002.
- [2] Sandip Agarwala, Christian Poellabauer, Jiantao Kong, Karsten Schwan, and Matthew Wolf. Resource-Aware Stream Management with the Customizable dproc Distributed Monitoring Mechanism. In *Proc. of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*, Seattle, WA, 2003.
- [3] F. Azzedin and M. Maheswaran. Towards trust-aware resource management. In *CCGRID*, 2002.
- [4] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proc. of 18th Symposium of Operating Systems Principles (SOSP-18)*, Bolton Landing, NY, 2003.
- [5] Daniel P. Bovet and Marco Cesati. *Understanding the Linux Kernel*. O'Reilly & Associates, Inc., 2nd edition, 2002.
- [6] Zhongtang Cai, Greg Eisenhauer, Qi He, Vibhore Kumar, Karsten Schwan, and Matthew Wolf. Iq-services: Network-aware middleware for interactive large-data applications. Technical Report 04-07, Georgia Tech Center for Experimental Research in Computer Systems (CERCS), 2004. <http://www.cercs.gatech.edu/tech-reports>.
- [7] Zongtang Cai, Vibhore Kumar, Brian F. Cooper, Greg Eisenhauer, Karsten Schwan, and Rob Strom. Utility-driven fault-tolerance in enterprise-scale information flows. Submitted to International Conference on Autonomic Computing 2006, 2006.
- [8] Emmanuel Cecchet and Julie Marguerite. Rice university bidding system (rubis). <http://rubis.objectweb.org/>, 2005.
- [9] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews. Xen and the art of repeated research. In *Proceedings of the USENIX 2004 Annual Technical Conference*, pages 135 – 144, June 2004.
- [10] Topspin Communications. Virtual i/o: Enabling the utility data center. [http://www.topspin.com/solutions/pdf/Virtual\\_IO.pdf](http://www.topspin.com/solutions/pdf/Virtual_IO.pdf).
- [11] E. Damiani, S. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *In 9th ACM Conf. on Computer and Communications Security*, 2002.
- [12] Greg Eisenhauer, Fabian Bustamante, and Karsten Schwan. Publish-subscribe for high-performance computing. *IEEE Internet Computing*, Jan/Feb 2006.
- [13] H. Feng, J. T. Giffin, Y. Huang, S. Jha, W. Lee, and B. P. Miller. Formalizing sensitivity in static analysis for intrusion detection. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, May 2004.
- [14] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly detection using call stack information. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, May 2003.
- [15] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, May 1996.
- [16] Ivan Ganey, Greg Eisenhauer, and Karsten Schwan. Kernel Plugins: When a VM is Too Much. In *Proceedings of the 3rd Virtual Machine Research and Technology Symposium (VM'04)*, San Jose, CA, May 2004.

- [17] Debin Gao, Michael K. Reiter, and Dawn Xiaodong Song. Behavioral distance for intrusion detection. In *RAID*, pages 63–81, 2005.
- [18] Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium*, February 2003.
- [19] Ada Gavrilovska, Kenneth Mackenzie, Karsten Schwan, and Austen McDonald. Stream Handlers: Application-specific Message Services on Attached Network Processors. In *Proc. of Hot Interconnects 10*, Stanford, CA, August 2002.
- [20] Ada Gavrilovska, Karsten Schwan, and Van Oleson. Practical Approach for Zero Downtime in an Operational Information System. In *Proc. of 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.
- [21] Jonathon T. Giffin, Somesh Jha, and Barton P. Miller. Detecting manipulated remote call streams. In *Proceedings of the 11th USENIX Security Symposium*, pages 61–79, Berkeley, CA, USA, 2002. USENIX Association.
- [22] Yian Huang and Wenke Lee. Attack analysis and detection for ad hoc routing protocols. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*, September 2004.
- [23] IBM. Personal communication. received by Dr. Karsten Schwan.
- [24] Information Sciences Institute. Future internet network design. <http://find.isi.edu>, 2005.
- [25] Ashlesha Joshi, Samuel T. King, George W. Dunlap, and Peter M. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–15, Oct 2005.
- [26] Seung Jun, Mustaque Ahamad, and Jun (Jim) Xu. Robust information dissemination in uncooperative environments. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 293–302, Washington, DC, USA, June 2005. IEEE Computer Society.
- [27] S. Kamvar, M. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the WWW Conference*, 2003.
- [28] L. Kong, D. Manohar, A. Subbiah, M. Sun, M. Ahamad, and D. Blough. Agile store: Experience with quorum-based data replication techniques for adaptive byzantine fault-tolerance. Submitted to *Dependable Systems and Networks (2005)*.
- [29] L. Kong, A. Subbiah, M. Ahamad, and D. Blough. A reconfigurable byzantine quorum protocol for the agile store. In *Proceedings of the Symposium on Reliable Distributed Systems*, 2003.
- [30] Lei Kong, Deepak J. Manohar, Arun Subbiah, Michael Sun, Mustaque Ahamad, and Douglas M. Blough. Agile store: Experience with quorum-based data replication techniques for adaptive byzantine fault tolerance. In *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS)*, 2005.
- [31] Rajamar Krishnamurthy, Karsten Schwan, and Marcel Rosu. A Network Co-Processor-Based Approach to Scalable Media Streaming in Servers. In *Proc. of International Conference on Parallel Processing (ICPP)*, August 2000.
- [32] Vibhore Kumar, Brian F Cooper, Zhongtang Cai, Greg Eisenhauer, and Karsten Schwan. Resource-aware distributed stream management using dynamic overlays. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS-2005)*, 2005.

- [33] L. Lakshmanan, M. Ahamad, and H. Venkateswaran. Responsive security for stored data. *IEEE Transactions on Parallel and Distributed Systems*, 2003.
- [34] S. Lakshmanan, D. Manohar, M. Ahamad, and H. Venkateswaran. Collective endorsements and the dissemination problem in malicious environments. In *Proceedings of the Conference on Dependable Systems and Networks (DSN)*, 2004.
- [35] Marcos Laureano, Carlos Maziero, and Edgard Jamnhour. Intrusion detection in virtual machine environments, 8 2004.
- [36] W. Lee and S. J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3(4), November 2000.
- [37] W. Lee and D. Xiang. Information-theoretic measures for anomaly detection. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, May 2001.
- [38] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy*, pages 120–132, 1999.
- [39] Joan Feigenbaum Matt Blaze and Jack Lacy. Decentralized trust management. In *IEEE Conference on Privacy and Security, Oakland*, 1996.
- [40] John Ioannidis Matt Blaze, Joan Feigenbaum and Angelos D. Keromytis. Request for comments (rfc) 2704, 1999.
- [41] John C. Mitchell Ninghui Li and William H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *Journal of ACM*, 2005.
- [42] Roberto Perdisci, David Dagon, Wenke Lee, Prahlaad Fogla, and Monirul Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, May 2006.
- [43] Nick L. Petroni, Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot – a coprocessor-based kernel runtime integrity monitor. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.
- [44] Marcel-Catalin Rosu, Karsten Schwan, and Richard Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The Virtual Communication Machine-based Architecture. *Cluster Computing, Special Issue on High Performance Distributed Computing*, May 1998.
- [45] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 144, Washington, DC, USA, 2001. IEEE Computer Society.
- [46] Ramesh Viswanath, Mustaque Ahamad, and Karsten Schwan. Harnessing non-dedicated wide-area clusters for on-demand computing. In *Proceedings of IEEE International Conference on Cluster Computing (Cluster 2005)*, 2005.
- [47] D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems, 2002.
- [48] Richard West and Christian Poellabauer. Analysis of a Window-Constrained Scheduler for Real-Time and Best-Effort Packet Streams. In *Proc. of the Real-Time Systems Symposium*, 2000.
- [49] Matt Wolf, Zhongtang Cai, Weiyun Huang, and Karsten Schwan. Smart Pointers: Personalized Scientific Data Portals in Your Hand. In *Proc. of Supercomputing 2002*, November 2002.
- [50] Y. Zhang, W. Lee, and Y. Huang. Intrusion detection techniques for mobile wireless networks. *ACM/Kluwer Wireless Networks Journal*, 9(5), September 2003.

- [51] Yongguang Zhang, Yian Huang, and Wenke Lee. An extensible environment for evaluating secure manet. In *Proceedings of the 1st International Conference Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*, September 2005.
- [52] S. Zhao and V. Lo. Result verification and trust-based scheduling in open peer-to-peer cycle sharing systems. In *Proceedings of Fifth International Conference on Peer-to-Peer Systems*, 2005.