An Architectural Concept for Intrusion Tolerance in Air Traffic Networks

Jeffrey M. Maddalon
jeffrey.m.maddalon@nasa.gov

Paul S. Miner
paul.s.miner@nasa.gov

NASA/Langley Research Center
Hampton, VA

Abstract

The goal of an intrusion tolerant network is to continue to provide predictable and reliable communication in the presence of a limited number of compromised network components. The behavior of a compromised network component ranges from a node that no longer responds to a node that is under the control of a malicious entity that is actively trying to cause other nodes to fail. Most current data communication networks do not include support for tolerating unconstrained misbehavior of components in the network. However, the fault tolerance community has developed protocols that provide both predictable and reliable communication in the presence of the worst possible behavior of a limited number of nodes in the system. One may view a malicious entity in a communication network as a node that has failed and is behaving in an arbitrary manner. NASA/Langley Research Center has developed one such fault-tolerant computing platform called SPIDER (Scalable Processor-Independent Design for Electromagnetic Resilience). The protocols and interconnection mechanisms of SPIDER may be adapted to large-scale, distributed communication networks such as would be required for future Air Traffic Management systems. The predictability and reliability guarantees provided by the SPIDER protocols have been formally verified. This analysis can be readily adapted to similar network structures.

Introduction

The security of computer networks is often based on the idea of making a network impenetrable from outside attack. One example is the idea of a network “firewall.” A firewall sets a difficult barrier for attackers coming from the Internet to a local network. Encryption systems are also based on the making a system impenetrable. As long as the key remains secret, the information will be protected. If an attacker can circumvent the security measure—by any means—then these systems offer almost no resistance to further attacks on the system. For instance, a firewall cannot stop an attacker who is already behind the firewall (such as an employee of the company) and an encryption system does not protect any data once the key has been stolen.

Relatively few security systems have been built with the requirement that the system should continue to function even when some computers have been compromised. The few that have, usually assume that the nodes will be compromised in easily detectable ways and soon after the intrusion has been detected, some remaining “good” part of the system will be able to control the behavior of the compromised node. Frequently these systems do not have the ability to handle the situation when a compromised node actively tries to hide its behavior and/or attempt to corrupt good nodes in such a way that they begin to behave incorrectly. If one node in the system is compromised, then usually there is no expectation that the system will behave correctly.

An alternate design for a secure computer network is to have two goals: (1) make the system difficult to penetrate and (2) accept that the network may be breached. That is, the system must be built to continue operation even when some nodes are maliciously attempting to stop the system from functioning. Instead of making a system *intrusion resistant*, the network is *intrusion tolerant*.

One may question why it is necessary to build a system that is intrusion tolerant. Perhaps, one would argue, the resources spent on making a system tolerant of malicious nodes should instead be spent on making the system more difficult to enter in the first place. At this point one can examine the boundary conditions. Consider a system that is very difficult to enter, but cannot tolerate any malfunctioning of components. A determined attacker will examine the system and search for the weakest link in the system. Once this link is compromised, the system as a whole is worthless or even dangerous. The list below indicates some areas where an attacker could attempt to penetrate the system; an attacker may

- introduce malicious software, for instance viruses, Trojan horses, and Easter eggs[1];
- attempt to masquerade as a legitimate user (also known as “spoofing”) perhaps by intercepting communication or by stealing a password;
- directly infiltrate the system through malicious system administrators, controllers, pilots, or maintenance crew;
- coordinate an attack that uses several of these techniques.

Certainly measures can be taken to *reduce* the likelihood of any of these penetrations; however, these threats are very difficult to totally eliminate. The reason these types of attacks are so difficult to counteract is that they are not systemically solvable problems. They fundamentally require the people involved in the system’s operation to consistently take correct actions. An organization can put processes in place to lower the chance of these threats, but total elimination is difficult. For instance, an organization can install anti-virus software on all machines, but have *any* users turned it off[2]? Likewise, an organization can force users to follow a password policy, but can the organization ensure that users *never* reveal their passwords, such as by writing them down?

Once one accepts that some nodes in a network can be compromised, we ask can a system be built that is tolerant to these intrusions? One approach comes from the observation that we do not know how an attacker will modify the behavior of a compromised node. We say that the behavior of a compromised node is *arbitrary*. The behavior of such a node could range from obviously faulty behavior such as no longer producing any results, or it could be faulty in a way that is very difficult to detect such as the compromised node is actively trying to corrupt or confuse other nodes in the system. To have a system continue to function properly when some nodes do not contribute to the solution requires some redundancy of nodes. Over the last thirty years, the fault tolerance community has developed methods to allow a system to operate in the presence of a limited number of nodes that have failed in arbitrary ways. Depending on the architecture and the amount of redundancy in the system, these fault tolerant protocols can handle a coordinated attack from a limited number of compromised nodes. The idea of using fault tolerant concepts for solving security issues was described by Randell and Dobson [Ran86] and further explained by Rushby [Rus99].

NASA/Langley Research Center has developed a fault-tolerant computing platform called SPIDER (Scalable Processor-Independent Design for Electromagnetic Resilience). The protocols and interconnection mechanisms of SPIDER may be adapted to large-scale, distributed communication networks required for a future air traffic management system. Standard versions of fault tolerant protocols are viewed as too expensive in time and bandwidth to be of practical use in handling malicious attacks in a system as complex as an air traffic management

[1] An Easter egg is a software feature that was added without the knowledge of the corporate management. Most eggs that have been identified are artistic or humorous; however, a malicious corporate programmer could add dangerous features. An archive of Easter eggs is located at http://www.eeggs.com

[2] Anti-virus software can interfere with some software installations. For this reason, anti-virus software typically has a “disable” feature.

system [Pal00]. A variant of the protocols used in SPIDER may provide a solution to this criticism.

The proliferation of "buffer overflow" security problems has shown that even a correct concept with a poor implementation renders the system vulnerable. Similarly the concept that these protocols allow the system to continue functioning even with some compromised nodes requires that the protocols have a correct design and implementation. Error in the design or implementation is considered a *common-mode* failure. Such failures would affect all nodes in the system. Therefore if the attacker could exploit an implementation flaw in one node, all nodes would be vulnerable. Since we can expect that attackers will look for such flaws, it is critical that the system be designed and implemented correctly. The SPIDER protocols have been rigorously verified using formal techniques and it is expected that any modifications to SPIDER to support intrusion tolerance will also be formally verified.

System Requirements

The communication system we will describe provides *integrity* and *availability* in an environment with a limited number of malicious nodes. Integrity ensures that a message between two good nodes is not modified and availability ensures that when two good nodes want to communicate, they will be able to [Lap95]. Another common service in communication systems is *confidentiality*. This communication network does not explicitly provide this service; however, encrypting data would provide this capability.

By describing the behavior of compromised nodes as arbitrary, we make no assumption about the behavior of bad nodes; however, we do assume that the only way a bad node can communicate with a good node is through the defined communication path of the good node. This is a requirement on good nodes, not bad nodes. We accept that an agent (for instance, someone with physical access to the node) can corrupt a good node in any way.

To provide integrity and availability we must determine the required properties of the communication links in the system. If these properties are chosen carefully, then an intrusion tolerant system can be developed using these communication channels. The properties of a reliable communication channel are

validity – When a *good* node sends a message, the message received by all *good* nodes will equal the message sent. If the sender is *bad* then there is no guarantee about the validity of the communication. Equally, if a *bad* node receives the message, there is no guarantee about the message received.[3]

agreement – When a sender is *bad*, all *good* receivers see same message. In other words, all good nodes agree on the all messages sent. This property is not normally provided in computer networks.

A reliable communication channel will maintain these properties while malicious nodes are in the system.

Before describing how a communication channel with these properties can be used to build an intrusion tolerant network, we will describe, in general, how to provide these properties. To build such a communication channel three capabilities must be provided: membership, synchronization and redundancy. Membership provides a capability for managing failures in the network. Synchronization provides a known time bound between when the receivers recognize a message. If this bound does not exist, then messages could be lost without the receivers being aware of it. This situation would clearly violate the agreement property. Finally, redundancy is also required. That is, if there is only one connection between nodes, then an attacker could sever that link and the validity property would be violated.

With these three capabilities, a communication channel can be built that provides the properties of agreement and validity. The description of how this is accomplished is beyond the scope of this paper. The interested reader should consult [MMTP02] for a treatment related

[3] "Spoofing" eliminates validity in most communication networks but not in our approach.

to the safety-critical version of the SPIDER protocols.

Some authors have argued for a relaxed synchronization requirement instead of the strong synchronization described above, examples include the Byzantine File System [Cas99] and Rampart [Rei96]. Instead of a fixed maximum delay between when messages are received, relaxed synchronization only requires that all messages will eventually be received. Under this weaker assumption, Castro and Liskov show that the Byzantine File System will always perform correct actions, but there is no guarantee about when these actions will be performed [Cas99]. Systems using the weaker synchronization requirement require less complexity in their protocols and provide economic benefits because fewer communication channels are required. However, for an air traffic network that provides safety-critical information, timely availability is vital to the functioning of the network. For this reason, availability is a requirement in our communication network. Kopetz observes that the time-triggered technique provides predictable availability for critical communication [Kop97]. To provide availability (along with other reasons), the SPIDER protocols use this time-triggered technique. Time-triggered distributed systems require strong synchronization guarantees.

Fault Tolerance and SPIDER

In a fault tolerant system, the goal is to ensure predictable system behavior in the presence of some subset of faulty nodes. One distinguishing characteristic of such architectures is the underlying fault model. The SPIDER architecture was designed using the hybrid fault model introduced by Thambidurai and Park [TP88]. Faults are classified according to the severity of their misbehavior. Nodes that fail in a manner that is locally detectable to all good observers are *benign* faulty. Nodes that are not benign faulty, but have the same error manifestation to all good observers are classified as *symmetric* faulty. All other faults are *asymmetric*. The class of asymmetric faults includes any arbitrary misbehavior, including the classic Byzantine fault model [LSP82]. Asymmetric faults are the most difficult to tolerate, but they are also the least frequent. Most failure manifestations are benign. Similarly, most non-benign faults are symmetric. The fault assumptions for the SPIDER architecture are defined with respect to the maximum number of simultaneous failures the system can tolerate while still providing guaranteed correct operation to the good nodes within the architecture.

The communication between nodes in the SPIDER architecture is provided by the Reliable Optical Bus (ROBUS). The ROBUS provides strong guarantees for communication between Processing Elements (PEs) in the presence of multiple internal ROBUS failures. Internally, the ROBUS consists of a collection of Bus Interface Units (BIUs) and Redundancy Management Units (RMUs) connected with a complete bipartite graph. This structure is illustrated in Figure 1. The ROBUS ensures interactive consistency, internal fault tolerant clock synchronization, and internal group membership if enough BIUs and RMUs are fault free. The general idea is that if there are good nodes present, the ROBUS can withstand an arbitrary number of benign failures and a limited number of other failures. The conceptual design of the ROBUS and an informal description of the fault tolerance protocols is presented by Miner et al. [MMTP02]. Geser and Miner have also formally verified an improved membership protocol for the ROBUS topology [GM03].

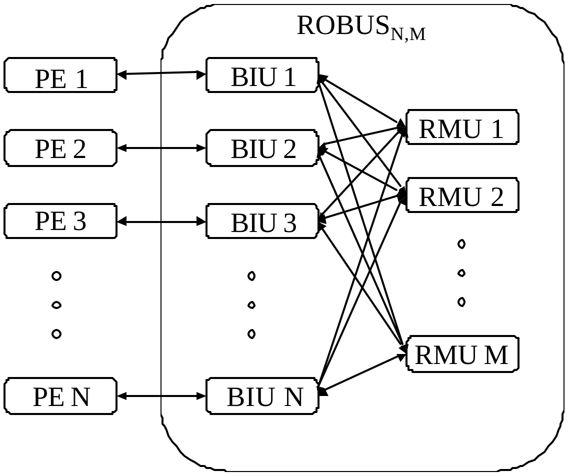

Figure 1: SPIDER Architecture

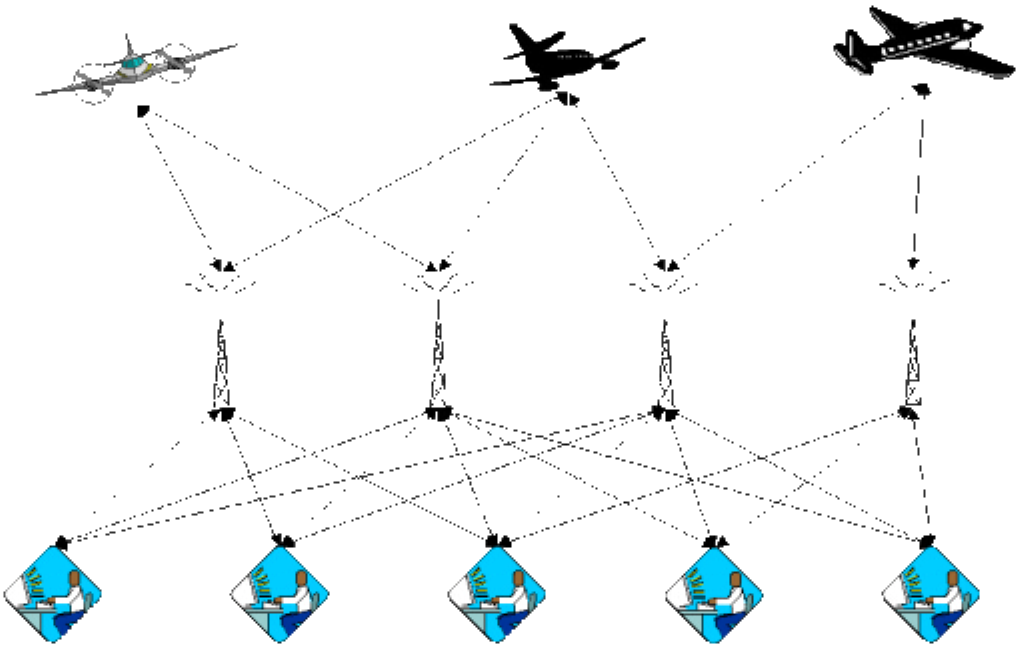

Figure 2: Network Concept

Adapting Fault Tolerance for Intrusion Tolerance

The failure modes that must be addressed for intrusion tolerance are similar to those for fault tolerance. Most networks are designed assuming that nodes fail silent. This corresponds to some benign faults in the hybrid model. Nodes behaving in an arbitrarily malicious manner correspond to the more severe fault modes. Since the class of asymmetric faults allows arbitrary behavior, systems that survive a bounded number of asymmetric faults can withstand a similar number of arbitrarily compromised nodes. The primary difference between fault tolerance and intrusion tolerance is that random hardware failures typically arrive slowly and independently. Multiple fault scenarios are rare. Stochastic models allow for reasonable predictions of system reliability. When the failure modes are human directed, multiple fault scenarios are anticipated. It is much more difficult to predict the survivability of the system. However, it is prudent to take measures to make a successful attack as difficult as possible. Furthermore, strong membership protocols that operate correctly even in the presence some compromised nodes will allow for targeted disabling of suspect nodes.

SPIDER, like many safety-critical architectures, is time-triggered. It uses strong synchronization guarantees to ensure correct operation of its protocols. This strong synchrony property also allows for predictable system behavior as new functions are added. This contrasts with the structure of most communication networks. These typically use event-based communication models. This gives good behavior if the typical failure mode is benign, but it renders the network susceptible to various denial-of-service strategies. A time-triggered approach, with known (or predictable) communication patterns provides guaranteed bandwidth for critical messages.

Mapping the SPIDER protocols onto an air traffic management network requires several decisions. The mapping presented below is but one of many possibilities. Air traffic control computers and aircraft can be mapped into PE/BIU pairs. The communication towers can be mapped into RMUs.   This structure is illustrated in Figure 2.   The resulting network is much larger than existing SPIDER prototypes, and the graph structure is not as regular. However, we believe that we can adapt the SPIDER protocols to run on several overlapping bipartite graph structures. In this manner, the protocols can scale to a large air traffic management network.

Conclusions

Important properties of safety-critical air traffic networks include integrity and availability. Integrity ensures that messages are either correctly delivered or are detected as bad. Availability ensures that the system is able to provide critical communication on demand. An intrusion tolerant system is able to provide assured integrity and availability, even when a bounded subset of nodes is controlled by malicious agents trying to corrupt the network. Intrusion tolerance is intended to complement other approaches to network security. Other approaches seek either to prevent intrusions or to quickly detect and isolate them when they occur. Intrusion tolerance provides an additional layer of protection and can protect against malicious nodes subverting attempts to detect and isolate corrupted nodes.

In this paper, we presented an architectural concept that adapts techniques from fault tolerance to provide intrusion tolerance. Fault tolerance uses protocols that ensure correct behavior of a system with a limited number of nodes behaving in a completely arbitrary manner. In an intrusion tolerant system, a node that is misbehaving arbitrarily can be viewed as a node that has been taken over by a malicious entity. By adapting concepts from fault tolerance, an intrusion tolerant network can be built to withstand a limited number of corrupted nodes.

Modifications to the SPIDER fault tolerant architectures may be used to provide intrusion tolerance over a wide-area distributed network. Since any flaws in the protocols for a intrusion tolerant system could create an opening for an attacker to compromise the entire system, it is critical that users of the these protocols have an assurance that they are correct. We accomplish this in SPIDER by formally verifying the protocols and we expect that modifications to support intrusion tolerance would also be formally verified.

References

[Cas99] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *ACM Proceedings: Operating Systems Design and Implementation (OSDI)*, pages 173–186, February 1999.

[GM03] Alfons Geser and Paul Miner. A New On-line Diagnosis Protocol for the SPIDER Family of Byzantine Fault Tolerant Architectures. NASA Technical Memorandum. 2003. to appear.

[Kop97] Hermann Kopetz. *Real-time Systems Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[Lap95] Jean-Claude Laprie. Dependability—its attributes, impairments and means. In *Predictably Dependable Computing Systems*. pages 3–24, Springer, 1995.

[LSP82] Leslie Lamport, Robert Shoshtak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems.* 4(3):382–401, July 1982.

[MMTP02] Paul Miner, Mahyar Malekpour, and Wilfredo Torres. A conceptual design for a reliable optical bus (ROBUS). In *Proceedings of the 21st Digital Avionics Systems Conference (DASC)*, October 2002.

[Pal00] P. Pal P, F. Webber, R.E. Schantz, and J.P. Loyall JP. Intrusion tolerant systems. In *Proceedings of the IEEE Information Survivability Workshop (ISW-2000)*, pages 24–26, October 2000. Boston, MA.

[Ran86] B. Randell and E. Dobson. Building reliable secure systems out of unreliable insecure components. In *Proceedings of the IEEE Conference on Security and Privacy*, pages 187–193, April 1986. Oakland, CA.

[Rei96] Michael K. Reiter. Distributing trust with the Rampart toolkit. *Communications of the ACM*, 39(4): 71–74, 1996.

[Rus99] John Rushby. Security and fault tolerance perspectives. DARPA Information Technology Security Workshop, October 1999. Williamsburg, VA. http://www.csl.sri.com/~rushby/slides/darpa-its99.ps

[TP88] Philip Thambidurai and You-Keun Park. Interactive Consistency with multiple failure modes. In *7th Symposium on Reliable Distributed Systems*. Pages 93–100. October 1988. IEEE Computer Society.