Protocol For Code Exchange In Survivable Embedded Systems

Michael E. Locasto
Department of Computer Science
Fu Foundation of Engineering and Applied Science
Columbia University
locasto@cs.columbia.edu

Abstract

As the cost of both networking and producing powerful embedded devices drops, collections of these highly specialized and heterogeneous platforms will proliferate. These networks will face security threats and suffer traditional hardware failure. Failure of embedded devices is undesirable because these devices often perform critical functions and are difficult to take offline and upgrade. Networks of embedded devices require a method to accomplish functional survivability of essential computations in a hostile or volatile environment.

This paper presents a protocol and describes an implementation for migrating essential computations from failed devices. An essential computation is a device's primary algorithmic functionality. This migration is accomplished by specifying both an Area Controller (AC), which contains definitions for every essential computation, and a number of proxy agents, which periodically send the AC update statements. The AC and proxy agents negotiate the specifics of process migration when the AC has determined that a device has failed.

The implementation of this protocol in a network of embedded devices is a crucial step toward fault tolerance and survivability in networks of embedded devices. The protocol can also be applied to networks that do not involve embedded systems.

Keywords

Fault tolerance, embedded systems, process migration, network availability, survivability, security protocols

Introduction

Even though the dedicated efforts of hardware and software engineers have enabled computing devices to become more or less reliable appliances, there are many domains where any failure (either malicious or arbitrary) of a computing device is completely unacceptable. In the non-embedded domain, many organizations devote scads of money and countless human-hours to assuring that their web services systems remain in a High-Availability (HA) state. The amount of hardware, software, networking, and management in such systems is staggering, if not overwhelming.

We can observe how much effort is put into serving web pages, and then consider the extreme requirements for networks of embedded devices needed to fly airplanes, perform health monitoring in hospitals, and control nuclear power facilities. As embedded devices are networked together, not only do they face traditional failures, but also the growing legion of threats made possible by a networked environment.

Fault tolerance and survivability of computer systems and networks is often addressed by both replication of critical services (distributed databases, server clusters), and redundancy (UPS, dual network connections). Traditionally, work has been done to guarantee some level of service by the system or network in the presence of attack, failure, or high load [10].

The protocol presented in this paper continues this theme by precisely specifying the steps necessary for ensuring that an essential computation can be migrated to a target device based on some optional policy specification. The result of ensuring that an essential computation may be migrated is the continued execution of critical algorithms in the network.

1. Related Work

The impetus for this work is detailed in Keromytis, et al [2], which describes an ambitious and detailed plan to design programming languages, policy languages and compliance-checking mechanisms, and dynamic update mechanisms to meet the challenges presented by survivability in embedded network environments.

Faults in and failures of computing devices have long been an area of concern for computing professionals. The design of this protocol involves three areas of computer science: embedded systems, process migration and distributed computation, and network survivability.

1.1 Embedded Systems

Designing embedded systems to be robust is a difficult and time consuming process because of the extreme constraints involved in the available hardware environment. Embedded systems have more extreme power, heat, speed, and space requirements than ordinary computer hardware [2]. In addition, Edwards et al [9] note that most design of embedded systems was done on an ad hoc basis as recently as five years ago, with little or no formal specification or proof of correctness. Edwards et al [9] have written a detailed analysis and presentation of formal methods for specifying, validating, and synthesizing reactive real-time embedded systems. Their approach identifies "management of both design complexity and system heterogeneity as the key problem." It is clear that heterogeneity is a necessary challenge in embedded systems, and any work done for embedded systems networks would do well to adopt the approaches detailed in their paper.

1.2 Process Migration

Process migration is a technique that has been studied for some time with the intent of providing load balancing for clusters of servers or workstations. Some work [4] had suggested that process migration added too much overhead in general for the anticipated benefits, but Downey and Harchol-Balter [1] refuted this claim and presented several common environments in which process migration is largely beneficial. Process migration is well suited to clusters of computers and is most famously implemented as part of the MOSIX [3,14] distributed operating system software.

The Charlotte system is another environment for process migration research. Charlotte is built with the goal of ensuring that the migration completes successfully. Artsy and Finkel [12] provide both a succinct overview of process migration theory and some performance characteristics related to Charlotte.

The technique of process migration is often compared with remote execution, which is presented as a lower-cost alternative. However, process migration offers true continuity of service, whereas failure of a device providing remote execution services necessitates the availability of a complete replica - something that is not always practical in an embedded environment. Typical remote execution mechanisms are RPC, Java RMI, and CORBA. These mechanisms rely on a local proxy to call functions on a server over the network, rather than actually moving a process.

Dynamic updates of software is a complex operation. Hicks [7] points to Smith [5] as an excellent review of process migration techniques. Hicks also describes the notion of state transfer as a process of automatically encoding current process information and details the difficulties involved with the checkpointing technique. The primary difficulty in state transfer is twofold: the target may not understand the state format, and some essential state may be hidden by the operating system of the original machine [7].

The initial version of the PCXSES protocol assumes that with a common virtual machine in the embedded network, state translation is not required. In addition, all interesting state information can be captured in the values of an object's instance data members, much like serialization of Java objects. However, the protocol does not limit what can be considered state; actually determining what state is important and translating it are left to the implementation.

On the other hand, heterogeneous process migration, as addressed in Smith and Hutchinson [8] does not make the assumption that all devices share a common runtime platform. Smith and Hutchinson [8] give a very precise analysis of the requirements for designing a process migration protocol. Future versions of PCXSES will address the issues raised by a heterogeneous network.

1.3 Network Survivability

The threat model for computer networks is significantly different from the threat model for a standalone system. Networks are vulnerable to a wide variety of attacks, and the technical report by Ellison et al [10] provides a very complete overview of modern network survivability. Fault tolerance and network survivability are two areas of network security that attempt to research and respectively address Byzantine [6] and malicious failures in networks.

Zhang et al [13] have commented that "the composition of most networks tends to converge on a single technology [often from the same vendor] at each layer of the network." This trend may boost interoperability; however, Zhang et al [13] have suggested that homogeneity in networks presents vulnerabilities and that survivability may be achieved through greater heterogeneity.

The weakness of this approach is the greater challenge in configuration and management when adding non-required complexity to the network. However, in the OASES [2] proposal, Keromytis et al argue that the natural heterogeneity of an embedded network provides a high degree of redundancy. This inherent redundancy may provide a platform for survivability. Furthermore, we should observe that embedded networks contain many highly specialized components that already require diverse configuration and management.

Employing process migration in a network of embedded devices is a highly desirable and cost effective method of assuring the survivability of this network. The PCXSES protocol performs this task.

2. The PCXSES Protocol

The PCXSES protocol is a straightforward series of steps that borrows from general principles of network routing protocols and the idea of timeouts as presented in Lamport et al [6]. The protocol has four phases:

1. GOOD MORNING phase (walk in the door)
2. PARTY phase (continuously greet host)
3. RECOVERY phase (host gives keys to sober friend)
4. GOODBYE phase (people leave party)

The underlying idea is that the AC can stores definitions (object code) for every process running on the devices, but needs to be alerted to changes in state. Each device joins the network with a GOOD MORNING message. The device then periodically updates the AC with altered state information via a HELLO or STATE message. Finally, if the AC has determined that a device has failed, it will enter the RECOVERY phase with that failed device and a target device. It will send a MORPH (Mobile Object RePlacement Header) message to the target device. The target device does not implicitly trust any MORPH message. Entering the RECOVERY phase with one device does not require the AC to abandon the PARTY phase or GOOD MORNING phase with other devices.

Figure 1: Protocol Sketch

```
[Client]                                                    [AC]
 GMP       -------- GOOD MORNING, mode -------->>
 GMP      <<----- GOOD MORNING, params --------
 PP     -------- STATE | HELLO ----------------->> (no ack)
 RP      <<---- MORPH (# obj, #bytes/obj, ctrl sigs -----
 RP             <-------- object def #1 --------------
 RP             <-------- object def #2 --------------
 RP             <-------- object def #n --------------
 RP         ---------- MORPH (ack) --------------->>
GBP        --------------- GOODBYE ------------>>
GBP         <<------------- GOODBYE -----------
```

2.1 Protocol Phases

2.1.1 Good Morning Phase

This phase allows a device to join the federation or network by contacting the AC. Thus, the AC does not have to poll the network and solicit new devices for their connectivity information.

2.1.2 Party Phase

In the party phase, each device sends notifications to the AC that the device is still alive. The device may also send updated state information as part of the notification or as part of a different message.

2.1.3 Recovery Phase

Naturally, the RECOVERY phase is the most interesting. [details of recovery phase...]

2.1.4 Goodbye Phase

This phase allows the protocol to terminate gracefully. If this phase were not included in the protocol, the AC would assume that a device had failed and being a needless Recovery Phase. [more]

2.2 Message Formats

All messages are encapsulated in a BOTTLE object with appropriate type identifiers. There are five types of messages:

1. GOOD_MORNING message
2. HELLO message
3. STATE message
4. MORPH message
5. GOODBYE message

The GOOD_MORNING message is used by the client to notify the AC of its existence and to negotiate parameters to be used in the rest of the protocol. The HELLO message is used to notify the AC that the device is still active, and may be used to pass state information. The STATE message is an optional message used to pass state information. Providing the option of separating state from the HELLO message is a performance enhancement; it also provides the device programmer more control over network traffic. The MORPH message is used to migrate a process to a target device and acknowledge that the migration has successfully completed. Finally, the GOODBYE message offers a graceful mechanism for a device to notify the AC that it has left the network and does not need to be "recovered."
[specific format of messages here]

3. Results

The protocol proof of concept and development environment was implemented using the Java 1.4.1 platform. The AreaController was hosted on a dual Xeon 2.0 GHz RedHat Linux platform with 1 gigabytes of RAM. The host machine had a number of other processes running, and it is anticipated that the protocol does not require such firepower to perform in a reasonable manner. Test devices were hosted on weaker workstations and some network cluster machines, running a variety of Microsoft Windows and Solaris operating systems. Of course, future testing will be performed on smaller devices and embedded platforms.

Three important results are the realization of additional requirements for survivable networks of embedded systems: the development of a clear, concise, and powerful policy language, the need to recognize and prevent "failure chaining", and the need to detect Byzantine failure, perhaps through some peer-based mechanism.

The protocol functions as anticipated and the table below details some test processes that were successfully migrated. In addition, the graph displays some measurements of the time it took to migrate the different processes. The protocol adds no significant amount of lag. In addition, the protocol adds only a medium amount of network traffic and should scale through X # of nodes.

4. Conclusions and Future Work

The PCXSES protocol is a first general solution to the problem of process migration and fault tolerance in embedded networks. Successive transformation of the protocol for performance and security is anticipated. Most notably, a challenge-response protocol will be integrated into the GOOD MORNING phase.

In addition, since the protocol currently assumes a common virtual machine layer on each device (to simplify the restart of processes), the protocol will be adjusted to account for different hardware targets. Perhaps standardization of lifecycle methods (in addition to saving state variables) can help achieve some level of granularity by presenting well-known hooks into object code, thus making transfer of a program counter unnecessary.

5. References

[1] Allen Downey and Mor Harchol-Balter. "A note on 'The Limited Performance Benefits of Migrating Active Processes for Load Sharing'," University of California at Berkeley Technical Report. UCB/CSD-95-888, November 1995

[2] A. Keromytis, S. Edwards, V. Prevelakis, and M. Hicks. TC: Open and Survivable Embedded Systems (OASES). NSF Grant Proposal and Project Summary. 2002.

[3] Barak A. and La'adan O., The MOSIX Multicomputer Operating System for High Performance Cluster Computing , *Journal of Future Generation Computer Systems*, Vol. 13, No. 4-5, pp. 361-372, March 1998.

[4] D. Eager and E. Lazowska and J. Zahorjan: The Limited Performance Benefits of Migrating Active Processes for Load Sharing. In *Conf. on Measurement & Modelling of Comp. Syst., (ACM SIGMETRICS)*, May 1988, pages 63--72.

[5] J. M. Smith. A Survey of Process Migration Mechanisms. *ACM Operating Systems Review*, *SIGOPS*, 22(3): 28-40, 1988.

[6] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, Vol. 4, No. 3, July 1982, Pages 382-401.

[7] M. Hicks. Dynamic Software Updating. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, August 2001.

[8] P. Smith and N. C. Hutchinson. Heterogeneous Process Migration: The Tui System. Software - Practice and Experience 28(6): 611-639, 1998.

[9] S. Edwards, L. Lavagno, E. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. *Proceedings of the IEEE* 85(3): 366-390, March 1997.

[10] R.J. Ellison, D. Fisher, R.C. Linger, H.F. Lipson, T. Longstaff, and N. R. Mead. Survivable Network Systems: An Emerging Discipline (CMU/SEI-97-TR-013) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1997.

[11] W. Du and M. J. Atallah. Secure Multi-Party Computation Problems and Their Applications: A Review

and Open Problems. In *Proceedings of the New Security Paradigms Workshop*, pages 13-22, Cloudcroft, New Mexico, 2001.

[12] Y. Artsy and R. Finkel. Designing a Process Migration Facility: The Charlotte Experience. *IEEE Computer* 22(9): 47-56, September 1989.

[13] Y. Zhang, H. Vin, L. Alvisi, W. Lee, and S. K. Dao. Heterogeneous Networking: A New Survivability Paradigm. In *Proceedings of the New Security Paradigms Workshop*, pages 33-39, Cloudcroft, New Mexico, 2001.

[14] http://www.mosix.org/ The official MOSIX website.