Position Paper: Deeply Embedded Survivability

Philip Koopman, Jennifer Black, Theresa Maxino
Carnegie Mellon University
{koopman, jenm, maxino}@cmu.edu

Abstract

This position paper identifies three significant research challenges in support of deeply embedded system survivability: achieving dependability at the enterprise/embedded interface gateway, finding a viable security patch approach for embedded systems, and surviving run-time software faults.

1. Introduction

Deeply embedded systems consist of one or more embedded systems connected to an enterprise system or to the Internet (e.g., [3]). To be survivable, such systems must continue to function in the face of faults, whether accidental or malicious, and whether the faults are caused by design errors or unexpected operating conditions. Embedded system survivability can be more challenging than enterprise survivability because embedded systems may not be able to perform frequent reboots, incorporate weekly patches, transfer large amounts of data, or be cared for by trained system administrators. Beyond this, the different natures of embedded control vs. enterprise systems present fundamental limitations to applying known techniques from either area to the other. [1]

2. Fundamental limitations

2.1 Time triggered to event triggered interfaces

A fundamental limitation to achieving deeply embedded system survivability is the inherent mismatch between time triggered and event triggered systems.

Embedded systems are often “time triggered,” meaning that they perform periodic computations and messaging in support of hard deadlines (e.g., [2]). Because of the dramatically different needs of real time control systems compared to desktop computing, they often use specialized network protocols such as CAN that provide low-cost, but low-bandwidth solutions optimized for very short messages (often 100 bits or fewer per message with network speeds on the order of 1 Mbit/sec).

Enterprise systems, in contrast, are usually characterized as “event triggered” systems with much larger, sporadic events, and typically have orders of magnitude more CPU power and network bandwidth.

The interface between the embedded and enterprise sides of a deeply embedded system is usually in the form of a “gateway” that provides a bidirectional transition between the time triggered and event triggered worlds. Given sufficient resources, each computing paradigm can be made to simulate the other. Event triggered systems can schedule events periodically to simulate time triggered operation. Time triggered systems can schedule periods so fast that they don’t miss events. But, those approaches only work in the fault-free case.

Deeply embedded system gateways will encounter fundamental limitations when attempting to map faults and responses in one computing paradigm into the other computing paradigm. For example, what happens when event triggered messages are clumped in transit, and arrive faster than the minimum inter-arrival rate assumed by the time triggered side of the gateway? Queues in the gateway provide only a partial solution, and can cause problems when the system encounters queue overflow or system instability as a result of queue lag time.

In the other direction, time triggered messages that contain too much value jitter can defeat whatever low pass filters are in place at the gateway and can potentially flood the enterprise system with messages. Leaky buckets and other throttling methods can provide some relief, but are not necessarily able to do the right thing in those cases where an event shower is representative of a true emergency situation rather than a fault or attack.

Despite a lack of understanding of these fundamental issues, deeply embedded system gateways are already being deployed, sometimes in critical systems.

2.2 Limits to the patch mentality

The approach of using security patches to address emergent attacks is pervasive in the desktop computing environment. Embedded systems have fundamentally different constraints that make patching difficult.

Safety critical systems must be recertified each time critical software is updated. Doing so is usually a costly and time-consuming process. Quick-turnaround security patches are currently impracticable if they affect critical code. Unfortunately, many embedded systems are designed in such a way that all their code is effectively critical (i.e., any change to the code might affect critical properties, so it must all be assumed to be critical).

Strategies to isolate critical from non-critical software on the same CPU are still a subject of research.

An additional issue with patching embedded systems is that many of them have a zero down-time requirement. Maintenance reboots and physical operator intervention are simply unacceptable in many unattended applications.

Finally, patching approaches typically assume that the owner of a system is trustworthy. This is often not the case in embedded systems. For example, it is relatively common for sports car owners to install engine controller software that circumvents pollution emission and fuel economy controls as a way to get more performance.

2.3 Limits to the perfect software mentality

Much research in computer science is based on the laudable goal of creating perfect software. Industry practices also employ the assumption that “perfection” (or a close approximation thereof) can be achieved by identifying all the “important” bugs and removing them.

In the real world, very few application domains have the time and resources to deploy low defect rate software. Getting the highest software quality possible within time and budget is certainly important. But, spending exponentially increasing resources to chase down the last few bugs is usually impractical. Instead, it might make more sense to spend a small fraction of available resources providing ways to survive bugs that will inevitably be encountered, rather than throwing all resources at an attempt to achieve absolute perfection.

3. Research challenges

There are several research challenges that stem from the limitations just discussed. They are:

Understand what goes into the embedded/ enterprise gateway. While some combination of queues and message filters can work in the fault-free case, mapping fault manifestations and survivability mechanisms across the time triggered to event triggered interface provides fundamental research challenges.

Make patching of critical embedded software viable. Patching of unattended, critical embedded systems provides fundamental challenges that aren’t encountered in most desktop systems. Creating patching approaches that maintain system integrity promises to be difficult.

Increase system survivability by tolerating inevitable software defects. Software defects are inevitable in most fielded systems. In some cases these defects will result in security vulnerabilities. In others they will result in failures to maintain critical system properties. Making software faults more survivable could offer improved cost effectiveness and reduced system fragility.

4. Promising innovations and abstractions

4.1 Safety invariants

Safety invariants, which are formal expressions of critical system properties that must hold true, offer new promise for increasing system survivability. Traditionally, analysis and testing are used to ensure the invariants are never violated. But, these techniques only work for the systems that are modeled (which are usually fault-free systems). One could also check safety invariants at run time to detect when a fault has occurred that is severe enough to compromise system safety. Safety invariant checks could act as failure detectors that activate recovery or safe shutdown mechanisms.

4.2 Graceful degradation

The term graceful degradation encompasses several meanings. The term was coined to describe modular redundancy in fault tolerant computing, and later evolved to encompass failover strategies and functional diversity. More recently, the term has been used to describe performability tradeoffs in Quality of Service research. The notion of providing systems that can partially work rather than only be fully working or fully failed is essential to achieving cost-effective survivability.

5. Possible Milestones

Survivability is an emerging research area, with the current emphasis more on understanding fundamental problems rather than on comprehensive solutions. Long-term milestones should include discovering fundamental tradeoffs, impossibility results, and workarounds applicable to realistic systems. Short term research milestones should emphasize characterizing practical limitations and exploring techniques to offer near-term improvement to system builders.

6. Acknowledgements

This work had been funded by the General Motors Collaborative Research Laboratory at Carnegie Mellon University and the Pennsylvania Infrastructure Technology Alliance.

7. References

[1] Koopman, P., Morris, J. & Narasimhan, P., "Challenges in Deeply Networked System Survivability," *NATO Advanced Research Workshop On Security and Embedded Systems*, August 2005

[2] Kopetz, H., *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer, 1997.

[3] Tennenhouse, D., “Proactive Computing,” *Comm. ACM*, 43(5): 43-50, May 2000.