Carnegie Mellon University
Research Showcase @ CMU

Institute for Software Research | School of Computer Science

2005

Challenges In Deeply Networked System Survivability

Philip Koopman
Carnegie Mellon University

Jennifer Black
Carnegie Mellon University

Priya Narasimhan
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/isr

This Working Paper is brought to you for free and open access by the School of Computer Science at Research Showcase @ CMU. It has been accepted for inclusion in Institute for Software Research by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Challenges In Deeply Networked System Survivability

Philip KOOPMAN, Jennifer MORRIS, Priya NARASIMHAN
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract.
Deeply networked systems are formed when embedded computing systems gain connectivity to each other and to larger enterprise systems. New functionality also brings new survivability challenges, including security across the embedded/enterprise interface. Addressing the needs of deeply networked system survivability is an open challenge that will require new approaches beyond those used for enterprise systems.

Introduction

A deeply networked system is formed when embedded computing subsystems are connected to each other and to enterprise systems, often via the Internet [20]. By increasing access to information and computing resources, these systems promise to provide new capabilities and opportunities. Unfortunately, deep networking also introduces survivability issues that have, thus far, received little attention.

Consider an automotive control application (such as a road-condition sensor or intake air quality sensor, Figure 1) which is connected via the vehicle's embedded networks to an automotive telematics infrastructure [2]. In this system, external servers could optimize performance for a given emissions requirement by reconfiguring the ratio of internal combustion to battery power in the car's engine, based on data from the internal vehicle sensors and other external sources (such as traffic conditions). However, what happens if someone penetrates the servers and commands all hybrid vehicles to perform 100% internal combustion on a smoggy day? What happens when a fault in a vehicle's telematics connection disrupts enterprise server operations? Worse, what happens when someone uses the enterprise-to-embedded communication channel to break into a vehicle and cause it to behave in an unsafe manner?

Given the proliferation of embedded applications that are increasingly connected to the Internet [19][16][15], including automobiles [9] and train control [6], it becomes imperative to find strategies that can safely and securely connect the two types of systems. In particular, deeply networked systems must ensure the survivability of embedded applications [12][10][18] with critical functionality.

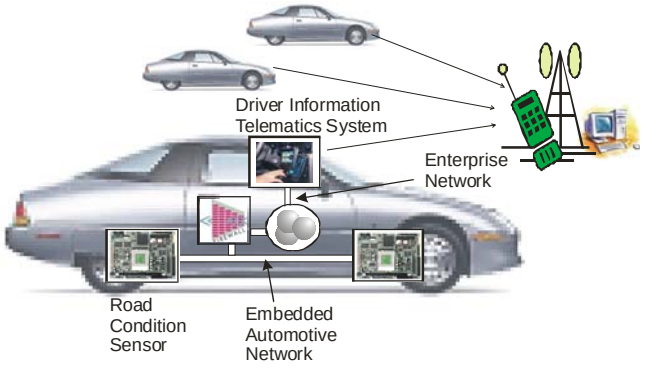

Figure 1: Example of a coupled embedded + enterprise system.

1. Embedded Survivability

Embedded systems are often used in critical applications, and there is much previous work in creating dependable systems for transportation applications, among others. In addition to newer security-based approaches, classical approaches have included hardware fault tolerance, software fault tolerance, and techniques to assure high software quality for critical systems ([4] are proceedings from a primary conference on these topics).

Classical work in this area assumes closed systems in which external attackers cannot gain access to the system. For this reason, most embedded systems have little or no native security capabilities. For example, the real-time embedded networks currently used in mainstream automobiles have no security mechanisms available for network messages.

1.1. Embedded system differences

It seems likely that traditional approaches won't solve many security problems in typical embedded systems, because the constraints and application domains differ tremendously from enterprise systems. A description of many of the differences can be found in [11]. The following points discuss the differences most relevant to survivability.

Many embedded systems interact with the external world by reading sensor values and changing actuators. Several properties of this real-world interaction that increase the difficulty of maintaining essential functions in the face of a failure or attack are:

- Reactive and real time. Embedded systems often perform periodic computations to close control loops. Even small timing variations (less than one second in many cases) that destabilize a single control loop can cause complete system failure.
- Critical. Embedded systems are often used in life- or mission-critical applications. This means that even minor disruptions to service can have unacceptably high cost.

- Non-recoverable. Because embedded systems have actuators that change the physical world, it may be difficult or impossible to "roll back" a state change caused by a faulty system as can be done with errant financial transactions.
- Exceptional. Embedded systems often operate in harsh environments with analog inputs, potentially subjecting them to many hardware faults.

Embedded systems have to remain survivable even though they usually are far more constrained than enterprise systems. Common embedded system constraints include:

- Small size & weight; battery power. Severe constraints on size, weight, and power often limit the amount of memory and computational power available.
- Low cost. Cost pressure usually results in the least capable CPU possible being used. Indeed, 8-bit CPUs dominate the market by volume [21].
- 24x7 operation of single nodes. Continuous operation, often with a single CPU dedicated to a particular function, makes it impracticable to have periodic downtime for applying patches, updates, or other preventive maintenance functions.
- Use of embedded networks. Most embedded systems are too low-cost to permit the use of Ethernet, TCP/IP, or other enterprise communication techniques. Instead, they use specialized embedded real-time networks such as CAN and TTP/C that don't support TCP/IP efficiently.
- Lack of system administrator. While it might be realistic to have a system administrator for every personal computer, most embedded systems are not designed to be continually patched or require software management. (Who should be the sysadmin for an air conditioner?)

Because of these various issues, it is clear that techniques used in enterprise systems cannot be expected to work as-is in an embedded environment.

1.2. Issues at the Embedded/Enterprise Interface

Most embedded systems aren't designed to connect to the Internet. Rather, most designs assume that the manufacturer has complete control over the software and network interface to every node. Moreover, they are typically built under the assumption that the system designer has taken into account all likely failure modes, that there are no misbehaving nodes (with misbehavior due possibly to software defects, unforeseen hardware defects, or malicious attackers), and that all system inputs conform to system requirements. Once a system is connected to the Internet, even indirectly, these assumptions are no longer valid.

2. Embedded System Design Approach Differences

Because embedded systems have so many differences in constraints and domain characteristics from typical enterprise systems, it should come as no surprise that their design approaches are often fundamentally different. These differences affect which approaches are viable for creating survivable systems.

2.1. Event-based vs. time-based operation

Enterprise systems are typically transactional and event-triggered in nature, which means that they usually focus on preserving data and tend to center around end-to-end request-response semantics. Usually the emphasis is on statistically good performance under various loading conditions, and it is often acceptable to refuse admission to tasks during overloads. "Best effort" servicing of aperiodic tasks is often acceptable.

Embedded systems often focus on interacting with continuous, physical systems with hard deadlines. Even minor disruptions to service can have unacceptably high cost. Periodic operation of all aspects of the system to makes it easier to ensure that worst case timing properties are acceptable. Such operation is often called "time triggered" system design (e.g., as discussed in [13]).

Time triggered design makes possible optimizations such as leaving time stamps (or even message identifiers) off messages and instead relying on the fact that the system assures timely message arrival to identify messages. A focus on worst case performance leads, in many cases, to static periodic execution schedules to ensure that every task has the computational resources it needs to run at its worst-case highest frequency. (Dynamic scheduling techniques can also be used, but in the end resources must still be reserved for worst-case loading conditions.)

Embedded systems designed for worst-case situations are at first glance more survivable to overload situations than typical enterprise systems. This is because no matter how many events an attacker or fault from the enterprise system throws at it, excessive loads applied to one task will not compromise resources used for other tasks.

2.2. Discrete vs. continuous applications

The interface between event- and time-based portions of deeply embedded systems creates additional types of vulnerabilities to faults and attacks. Beyond the usual issues of authentication and integrity, there are also timing vulnerabilities in continuous-time applications. Assuming that the embedded system has a typical time-triggered design approach, only one incoming message of a particular type can be processed per processing cycle (for example, one message of a particular type every 250 msec). If the data is being used for a control application, the system is likely designed to expect a fresh value for each and every control cycle. Even small disruptions in timing on the enterprise side, whether from congestion, faults, or malicious attacks, force exception handling mechanisms to be developed for the interface to the enterprise side. Exceptions that almost certainly must be handled include: missed messages (there may not be time for a successful retry on the enterprise side), erratically spaced messages (if two messages arrive during a single cycle, does the system queue one to let it get stale, combine the messages, or just throw one away?), severely clumped messages (if ten messages arrive all at once after a long delay, how does the system catch up given only enough capacity to process one incoming message per cycle?), duplicated messages, and messages that arrive too often over an extended period of time. Dealing with many of these scenarios will force tradeoffs between spending money on extra resources to deal with some fraction of overloads vs. discarding data.

Message transfer from embedded to enterprise systems requires a low pass filter between the periodically generated time-triggered messages and the event-triggered processing paradigm of the enterprise system. For example, it may be important to transmit the status of an embedded airbag sensor to an enterprise system. In a time-triggered embedded system, the state of the airbag (deployed or not) might be reported ten times per second via a network message to achieve 100 msec latency. But reporting ten times per second from millions of vehicles is an unacceptable enterprise server load. One job of the embedded/enterprise gateway on each vehicle must be to apply a low-pass filter to values, and only generate an event-driven enterprise message when an airbag is deployed. The gateway must also deal with spurious messages due to sensor failures or coordinated attacks.

2.3. Fault handling approaches

Enterprise systems typically use a checkpoint-rollback recovery strategy to make their significant amounts of state more survivable. Rollback reverts to a consistent, previously saved state snapshot to facilitate recovery or restart in the event that a failure occurs.

Embedded systems often use roll-forward recovery, because they cannot roll back in dealing with the irreversible physical world. Typically, embedded systems contain far less state than their enterprise counterparts. Thus, while enterprise systems focus primarily on data-integrity, ordering and state-consistency protocols, embedded systems tend to focus more on time-sensitive, scheduling protocols where data is extracted and processed from the system in real time, often grows stale quickly, and can be discarded. This makes fault recovery for embedded systems very different from enterprise recovery approaches.

2.4. Physical security & repair incentive

In general, enterprise survivability relies on the assumption that equipment owners have a vested interest in keeping their entire system secure and fault-free, so as to obtain full value from their capital equipment investment. Another underlying assumption is that it is possible to limit access or turn off machines in a crisis. For example, centralized service providers often deny individual users access to their equipment (for example, cut off network access or shut down the machine) if that equipment has been compromised, in order to avoid disruption to other users.

Embedded system owners may not have incentive to perform repairs and maintain physical security. Indeed, there is financial incentive to break into some smart cards used to store cash value or keep satellite TV access logs. In other instances, physical tampering can remove externally imposed constraints such as increasing vehicle performance at the expense of flouting anti-pollution laws or risking unsafe operating conditions.

Even if such faults or tampering could be detected, simply shutting down an embedded application and/or blocking its communication are likely to be unacceptable. A shut-down function would be complicated by the fact that it would have to be owned by someone other than the owner of the physical equipment. (Would you want the manufacturer of your vehicle or your local police department, for example, to have a “kill switch” for your car?) If the embedded system send safety critical information, (fire alarms from a dwelling;

airbag deployment alarms from a vehicle; medical alert alarms from a home security system), termination of communication might be prohibited without a lengthy process of warnings and opportunities for repair. And of course an external kill function would likely prove a tempting target for attackers to trip maliciously.

3. Embedded Enterprise Gateway Requirements

The usual approach for attempting to resolve problems at the embedded/enterprise interface is to use a gateway or "firewall" node to isolate the embedded system from faults and attackers originating on the enterprise side (e.g., [1] [17] [22]). However, there is little or no guidance available on the types of services that have to be in such a gateway node to ensure the resulting system is survivable. Typical proposed approaches for this interface currently focus on the use of encryption (e.g., [7]) and in industrial applications often use VPN. But, based on our observations, the following types of additional services are likely to be needed in at least some systems to ensure survivability for a wide variety of deeply networked system applications.

3.1. Trusted Time Base

A trusted time base that is shared among all embedded gateways and enterprise servers within a deeply networked system could provide a foundation for resolving timing disruptions and ambiguities. This could improve survivability by:

- Distinguishing whether a tightly spaced group of messages arriving at a gateway were generated at almost the same time, were bunched up due to congestion, or were subject to a man-in-the-middle timing attack.
- Detecting timing jitter in messages sent between embedded subsystems via an enterprise network due to load variation or a control-loop destabilization attack.
- Enabling compensation for message aging in closing inter-subsystem control loops.

In some applications the embedded network will have to make available "freshness" data for various values transmitted periodically, because the assumption of end-to-end periodic operation doesn't hold for data that has been exposed to the enterprise side of the system.

3.2. "Firewall" protection in both directions

It is just as important to protect the enterprise system against an embedded subsystem as the other way around. Thus, we expect enterprise/embedded gateway nodes to be composed of a matched pair of gateway functions in opposing directions. Each side of the gateway will have opposite notions of whom to trust, complicating gateway management.

As the embedded-to-enterprise side of the gateway converts periodic time-triggered data to event-triggered messages, it will have to manage issues such as ensuring delivery via acknowledgements, self-throttling of message loads, filtering of inappropriate messages, time stamping messages, and in general ensuring that faults or attacks on the embedded side of the interface don't propagate to the enterprise side.

In addition to traditional firewall functions, the enterprise-to-embedded side of the gateway will have to manage the conversion of incoming event-based messages to time-triggered messages. This will include deciding what values to provide to periodic tasks when event-based messages are missed, delayed, clumped, repeated, or sent too fast.

3.3. Limiting damage from compromised servers

A significant potential vulnerability in deeply networked systems comes from enterprise servers that are given direct or indirect control authority over embedded system actuators. We believe that such control authority will inevitably creep in to most deeply networked systems. As an example, Koopman [12] describes a real-time energy pricing scenario in which a malicious failure of an enterprise server can cause an arbitrary number of houses to change their power usage, resulting in a potential physical attack on the electric power grid.

Avoiding vulnerabilities due to compromises of enterprise servers might be difficult. A starting point might be to limit, by design, the number of embedded systems that are permitted to take information from any particular enterprise server (even via indirect paths), thus limiting the consequences of a fault or compromise of that server.

4. Related work

Firewall designs for enterprise systems are well known, and secure the connection between internal and external systems by blocking unauthorized traffic [3]. This might be achieved by applying filters to the packet level, the application level, or the physical port level. Although these enterprise-centric security designs can be effective at blocking unauthorized communication, they are inadequate for the attack scenarios that we have identified for deeply networked systems. The SCADA community has been active in embedded security (e.g., [7][8]). To this point published results have focused on patch management and encryption of data sent over physically insecure links.

The TTP safety-critical embedded network protocol incorporates the concept of a "temporal firewall" [14] to isolate time-critical activities, but does not deal with embedded systems connected to the Internet. That refers more to the isolation of time-sensitive and non-time-sensitive tasks from each other within an embedded system.

Duri et al., have proposed a framework for automotive telematics applications to ensure the privacy and integrity of user-supplied data in the enterprise system [5]. That technique uses trusted processors to collect and aggregate user data to be sent to authenticated application. However, it focuses on authorization rather than message timing attacks.

5. Conclusions

Deeply embedded systems combine embedded and enterprise computing, offering tremendous potential but also new survivability challenges. Embedded systems have significantly different assumptions and approaches to computing, necessitating different

approaches to survivability than those used in enterprise systems. Moreover, the interface between time-triggered, real-time embedded computing and event-triggered, transaction-oriented enterprise computing presents unique survivability challenges.

This work is supported in part by the General Motors Collaborative Research Laboratory at Carnegie Mellon University, NSF CAREER Award CCR-0238381, an NSF Graduate Research Fellowship, and Army Research Office grant number DAAD19-02-1-0389. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation or other sponsoring organizations.

6. References

[1] AEEC Letter 01-112/SAI/742. *Draft 1 of Project Paper 664,'Aircraft Data networks' Part 5 'Network Interconnection Devices'*, May 2001.
[2] Robert Bosch GmbH. Audio, *Navigation & Telematics in the Vehicle: Bosch Technical Instruction*. Bentley Publishers, July 2002.
[3] Cheswick & Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison Wesley, June 1994.
[4] *Dependable Systems and Networks Conference Proceedings*, IEEE Computer Society Press, June 2005.
[5] S. Duri, J. Elliott, M. Gruteser, X. Liu, P. Moskowitz, and R. Perez. Data protection and data sharing in telematics. *Mobile Networks and Applications*, 9(6):693–701, December 2004.
[6] A. Fabri, T. Nieva, and P. UMILIACCHI. Use of the internet for remote train monitoring and control: the rosin project. *In Proceedings of Rail Technology '99*, 1999 September. 16
[7] AGA/GTI SCADA Encryption Web Site, *http://www.gtiservices.org/security/index.shtml* , August 1, 2005.
[8] W. Iversen. Hackers step up SCADA attacks. *Automation World*, October 2004.
[9] L. Kahney. Your car: The next net appliance. *Wired News*, 2001 March.
[10] P. Kocher, R. Lee, G.McGraw, A. Raghunathan, and S. Ravi. Security as a new dimension in embedded system design. In *Proceedings 41st Design Automation Conference*, pages 753–760. IEEE Press, June 2004.
[11] P. Koopman, "Embedded System Design Issues – the Rest of the Story", In *Proceedings of the 1996 International Conference on Computer Design*, Austin, October 7-9 1996.
[12] P. Koopman. Embedded system security. *IEEE Computer*, 37(7):95–97, July 2004.
[13] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer, 1997.
[14] H. Koptez and R. Nossal. Temporal firewalls in large distributed real-time systems. *In Proc. 6th IEEE Comp. Soc. Wkshp. on Future Trends of Distributed Comp. Sys.*, pp 310–315. IEEE Computer Soc., October 1997.
[15] P. Krishnamurthy, J. Kabara, and T. Anusas-Amornkul. Security in wireless residential networks. In *IEEE Transactions on Consumer Electronics*, volume 48, pages 157–166. IEEE Press, February 2002.
[16] S. Kuo, Z. Salcic, and U. Madawala. A real-time hybrid web-client access architecture for home automation. In *Proceedings of the 2003 Joint Conference on Information, Communications and Signal Processing and the Pacific Rim Conference on Multimedia*, volume 3, pages 1752–1756. IEEE Press, December 2003.
[17] P. Polishuk. Automotive industry in europe takes the lead in the introduction of optical data buses. *Wiring Harness News*, November 2001.
[18] S. Ravi, A. Raghunathan, P. Kocher, and S. Hattangady. Security in embedded systems: Design challenges. *ADM Transactions on Embedded Computing Systems (TECS)*, 2(3):461–491, August 2004.
[19] J. W. Szymanski. Embedded internet technology in process control devices. In *Proceedings 2000 IEEE International Workshop on Factory Communication Systems*, pages 301–308. IEEE Press, September 2000.
[20] D. Tennenhouse. Proactive computing. *Communications of the ACM*, 43(5): 43–50, May 2000.
[21] J. Turley, "The Two Percent Solution," *Embedded Systems Programming*, January 2003.
[22] C. A. Wargo and C. Chas. Security considerations for the e-enabled aircraft. In *Proceedings of the IEEE Aerospace Conference*, volume 4, Big Sky, MT, March 2003.