Distributed Real-Time Systems: Challenges and Opportunities

Edward A. Lee
Robert S. Pepper Distinguished Professor and Chair of EECS, UC Berkeley

Invited Talk
NSF NeTS FIND Initiative PI Meeting (Future Internet Design)
Washington DC
November 27, 2007
**My Context: From Physical Layer Communications and Signal Processing through Embedded Systems to Cyber-Physical Systems**

CPS: Orchestrating networked computational resources with physical systems.
Today’s Application Drivers for Real-Time Networking

- Voip
- Video delivery
- Video & audio chat
- Social networks
- Internet gaming
- Industrial automation
- Sensor networks
- Large-scale instrumentation systems
- Transportation networks
- Automotive electronics
- Avionics
- Military systems
- Manufacturing
- Process control
Tomorrow’s Application Drivers: Cyber-Physical Systems (CPS)

- telepresence
- distributed physical games
- traffic control and safety
- financial networks
- medical devices and systems
- assisted living
- advanced automotive systems,
- energy conservation
- environmental control
- aviation systems
- critical infrastructure (power, water)
- distributed robotics
- military systems
- smart structures
- biosystems (morphogenesis,…)

Potential impact

- social networking and games
- safe/efficient transportation
- fair financial networks
- integrated medical systems
- distributed micro power generation
- military dominance
- economic dominance
- disaster recovery
- energy efficient buildings
- alternative energy
- pervasive adaptive communications
- distributed service delivery
- …
Real-time networking should not be about “quality of service” but rather about “correctness of service.”

Traditionally, “faster is better.”

This is like saying that for a roller coaster, “stronger is better.”

We have to change the mindset to “not fast enough is wrong!”
Abstraction Layers

The point of these abstraction layers is to isolate a system designer from the details of the implementation below, and to provide an abstraction for other system designers to build on.

In today’s networks, timing is a property that emerges from the details of the implementation, and is not included in the abstractions. For time-sensitive applications, the abstraction layers fail.
My Main Point

Timing needs to be a part of the network *semantics*, not a side effect of the implementation.

Technologies needed:
- Time synchronization
- Time-aware fault isolation and recovery
- Time-aware robustness

Note that the very premise of “net neutrality” makes these very difficult. Also needed:
- Fair, vendor-neutral heterogeneous service
Background - Domain-Specific Networks with Timed Semantics

- **WorldFIP** (Factory Instrumentation Protocol)
  - Created in France, 1980s, used in train systems
- **CAN**: Controller Area Network
  - Created by Bosch, 1980s/90s, ISO standard
- Various **ethernet** variants
  - PROFInet, EtherCAT, Powerlink, …
- **TTP/C**: Time-Triggered Protocol
  - Created around 1990, Univ. of Vienna, supported by TTTech
- **MOST**: Media Oriented Systems Transport
  - Created by a consortium of automotive & electronics companies
  - Under active development today
- **FlexRay**: Time triggered bus for automotive applications
  - Created by a consortium of automotive & electronics companies
  - Under active development today
Services Provided by Networks with Timed Semantics

- Frequency locking
- Time synchronization
- Bounded latency
- Fault isolation (sometimes)
- Priorities (sometimes)
- Admission control (sometimes)
Not so Domain-Specific Network Mechanisms

- Frequency locking
  - E.g., **synchronous ethernet**: ITU-T G.8261, May 2006
  - Enables integrating circuit-switched services on packet-switched networks
  - Can deliver performance independent of network loading.

- Time synchronization
  - E.g., **IEEE 1588** standard set in 2002.
  - Synchronized time-of-day across a network.
Time Synchronization on Ethernet with TCP/IP: IEEE 1588

Press Release October 1, 2007

Clocks on a LAN agree on the current time of day to within 8ns, far more precise than older techniques like NTP.

A question we are addressing at Berkeley: How does this change how we develop distributed real-time software?
The question we address:

Given a common notion of time shared to some known precision across a network, and given bounded network latencies, can we design better distributed embedded software?

Our answer (today):

Use discrete-event (DE) models for specification of systems, bind *model time* to *real time* only exactly where this is needed.
Using DE as a Programming Model for Distributed Real-Time Systems

Consider a scenario:
The Discrete Event (DE) Model of Computation

Software components (“actors”) send time-stamped events to other components, and components react in chronological order.

DE models are used in:
- Hardware description languages
- Network simulation languages (ns2, Opnet …)
- Financial trading systems and modeling languages,
- …
Using DE as a Programming Model for Distributed Real-Time Systems

Bind model time to real time at the sensors (physical inputs):

Output with time stamps $t$ is seen when real time $\geq t$
Using DE as a Programming Model for Distributed Real-Time Systems

Bind model time to real time at the *actuators* (physical outputs):

Input with time stamp $t$ must be delivered at real time $\leq t$.
Schedulability is not violating these timing inequalities.
Using DE as a Programming Model for Distributed Real-Time Systems

Assumption: Clocks on the distributed platforms are synchronized with bounded error $e$. 
Using DE as a Programming Model for Distributed Real-Time Systems

Assumption: network latencies are bounded by $d$. 

Diagram:

- Platform 1
  - Sensor 1
  - Computation 1

- Platform 2
  - Sensor 2
  - Computation 2
  - Trigger

- Platform 3
  - Computation 3

- Network communication

- Merge
  - Actuator 1

- Clock
  - Computation 4
Using DE as a Programming Model for Distributed Real-Time Systems

Assumption: computation times are bounded by \( c_1 \) and \( c_2 \).
Using DE as a Programming Model for Distributed Real-Time Systems

Static analysis reveals that this program is not schedulable!

The event here with time stamp t cannot be presented to the actuator until real time exceeds $t + d + e + \max(c_1, c_2)$.
Using DE as a Programming Model for Distributed Real-Time Systems

The program can be fixed with actors that increment the time stamps (model-time delays). This changes the semantics of the program, and makes its network behavior determinate.
Using DE as a Programming Model for Distributed Real-Time Systems

Opportunities and Challenges

- Time synchronization augmented with **location information** would greatly expand the possibilities. *Bring to networking what GPS brought to navigation.*

- Fault management cannot be based on *eventual* satisfaction of a request. See for example:
  - TTP Project (Kopetz et al., 1990s): Vienna
  - Tenet Project (Ferrari, Banerjea, Knightly, et al., 1990s): Berkeley
Conclusions

The next generation Internet should enable applications that the current generation cannot.

Thus, it cannot be just about performance improvements.

Putting temporal semantics in the network changes it qualitatively, not just quantitatively, and enables a whole new field of invention.