Power Comparison of Time and Event-Triggered Paradigms: A Case Study

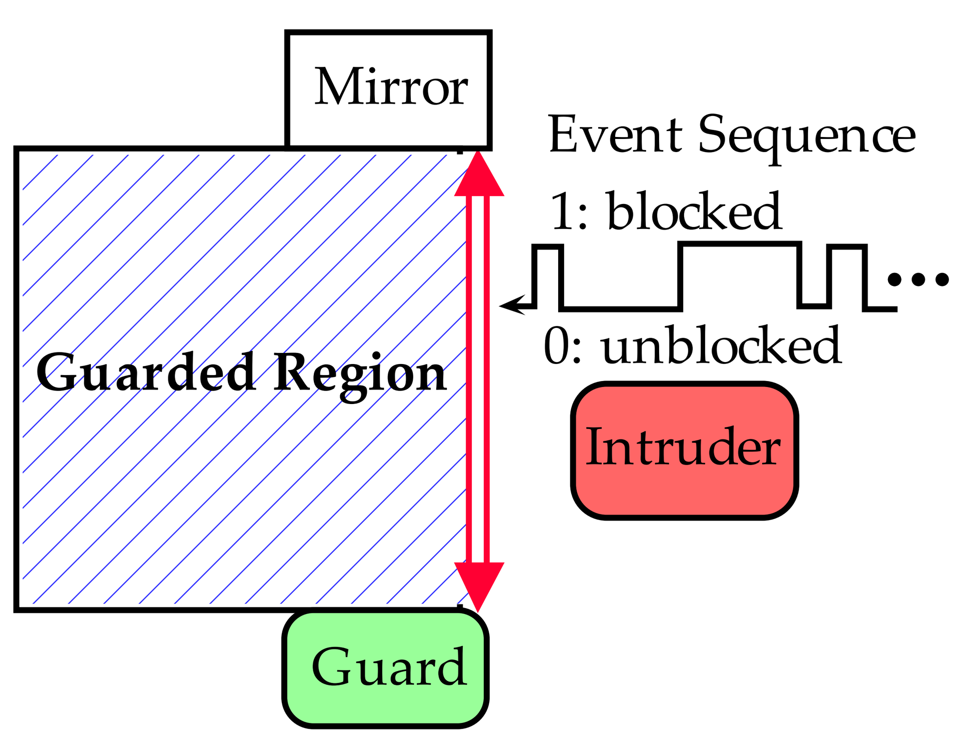
Taylor Johnson and Sayan Mitra

Software Solutions for Real-Time Systems

- Control software written using periodic tasks
- Simple model, fairly straightforward to reason about
- Guarantees from **programming models** and **runtime environments**
- e.g.: Giotto [1] and Hierarchical Timing Language (HTL) [2] use logical execution time (LET)
- Assuming tasks can be scheduled on a given hardware platform and RTOS, implementation conforms to LET model
- Otherwise, problem detected at runtime

Intrusion Detection Problem

- Guard detects if an intruder has entered a certain area by measuring light from laser LED detected via light sensor
- Randomly generated event sequence of intrusions
- 100 events of two types: duration 300ms and 2.5s
- Events are whether light is striking sensor
- Events generated by another Lego unit moving an actuator to block and unblock light source



Experimental Setup

- Hardware: Lego Mindstorms NXT
  - — 32-bit ARM7 processor and 8-bit Atmel microcontroller
  - — Powered by DC bench supply with shunt resistor inserted on low-side output for current measurement
- Software
  - — OS: nxtOSEK, an OSEK-compliant RTOS for Lego
  - — Time-triggered: Giotto [1] for OSEK ported to nxtOSEK
  - — Event-triggered: FIFO queue of events
  - — Equivalent programs trigger an alarm if intrusion occurs

Typical Experimental Results

- Event Features and Issues
  - — Potentially **rare** and **narrow**, e.g. intrusions in this case
  - — Event detection capability is a function of power usage
  - — TT **needs high sampling rate** for detection and **must move data every cycle** to maintain LET assumptions
  - — ET **can sleep** most of the time while waiting and detect
  - — Displays clear power savings (20%) for ET versus TT
  - — **Flexibility in event handling** if using ET and TT

Paradigm	ET	TT 150ms	TT 300ms	TT 450ms	TT 1000ms
Average Power (W)	0.87	1.10	1.10	1.11	1.08
Missed Events (#)	0	0	0	5	44
Error (%)	0.08	0.08	0.08	0.20	1.10

- Interrupts vs. Polling
  - — Power savings obvious at this low level
  - — Not previously established at level of abstraction software solutions mandate

Future Work

- Working towards bridging the **formal** gap between controls and embedded software
- New programming language and execution environment for distributed embedded systems
- Language will support **both ET & TT** computation
- Semantics defined in terms of Hybrid Input/Output Automata (HIOA) [3]
- New classes of faults, e.g., OS-level, actuator, sensor, etc. in addition to crash and Byzantine
  - — Create new types of behavior
  - — Detected more easily than crash and Byzantine
  - — e.g.: node with an actuator failure could just announce it
- Middleware and runtime environment will guarantee degraded safety and liveness in the face of faults
  - — Will deploy on 25-node wireless Linux cluster at Illinois [4]
  - — **Guarantees** at the software level of abstraction

References

[1] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in Proceedings of the IEEE. Springer-Verlag, 2001, pp. 166–184.

[2] A. Ghosal, T. A. Henzinger, D. Iercan, C. Kirsch, and A. L. Sangiovanni-Vincentelli, "Hierarchical timing language," EECS Dept., University of California, Berkeley, Tech. Rep. UCB/EECS-2006-79, May 2006.

[3] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O Automata," Inf. Comput., vol. 185, no. 1, pp. 105–157, 2003.

[4] P. Kyasanur, C. Chereddi, and N. Vaidya, "Net-X: System eXtensions for Supporting Multiple Channels, Multiple Interfaces, and Other Interface Capabilities," ECE Dept., UIUC, Tech. Rep., August 2006.