PRINCETON UNIVERSITY
Williams

CPS: Towards Secure, Privacy-Preserving, Verifiable Cyberphysical Systems

PIs: Esin Tureci and Kelly Shaw

CNS 1739701 (UR) and CNS 1739674 (Princeton), Oct. 1, 2017 - Sept. 30, 2020

Introduction

CPS/IoT systems are distributed over numerous sensing, actuation and computing devices, with a diverse set of specifications includiing

- physical and often safety-critical aspects to their operation and
- time sensitive requirements and energy constraints.

Challenges for achieving correctness and security include:

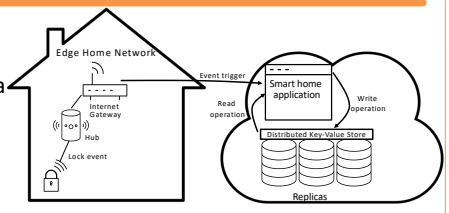
- increasing frequency of dynamic code compilation and deployment and
- heterogeneity across devices, systems, and within individual devices.

Proposed work:

- Develop formal, static verification techniques for correctness
- Develop hardware support for formal, dynamic system verification
- Open-source release of techniques and tools

Challenges and Opportunities in CPS/IoT

CPS/IoT devices communicate, update, and store state in distributed cloud storage. Data may be updated inconsistently in a variety of ways, creating potential correctness, security, and reliability flaws.



- Message delays or reorderings can leave device state inconsistent with data state stored in cloud.
- Interaction of different processing elements and applications in systems may introduce data inconsistencies and incorrect functionality.
- Lack of atomic read-modify-write operations in device programming interface can result in lost data updates or non-intuitive results.
- Weak data consistency models used by cloud infrastructure make it challenging for application developers to reason about correctly updating device and cloud state as order of updates is not guaranteed.
- Complexity in testing and deploying CPS/IoT systems and software is massive, due to the needs to ensure security and reliability.

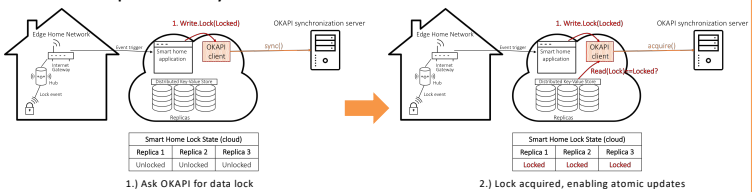
Case study: Smart Home Environments

Smart Home devices use cloud-based platforms to store application state, compute in response to sensor events, and trigger device actuation.

Problem 1: System support for cloud-based platforms is insufficient:

- Asynchronous events and distributed devices lead to event reordering.
- Concurrent event processing leads to race conditions due to non-existent read-modify-write atomic primitives.

Solution: OKAPI provides a platform synchronization service that orders events and provides synchronization mechanisms.

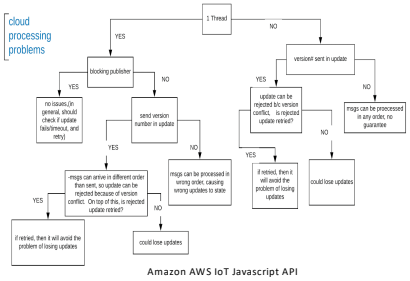


1.) Ask OKAPI for data lock

2.) Lock acquired, enabling atomic updates

Problem 2: Application developers do not factor eventual consistency and concurrent event processing into program development, resulting in incorrect applications.

Approach: Create static analysis tools to identify application code that may result in lost updates or reordered updates to cloud and device state.



Amazon AWS IoT Javascript API

Dynamic CPS/IoT Verification

Goal is to create a full suite of verification tools for design-time, compile-time, and run-time checking of state update orderings and atomicity.

General approach: Take computer architecture concept of "litmus tests" for verifying correctness of memory consistency implementations and apply to CPS/IoT systems.

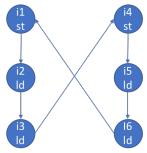
Cycle = Forbidden outcome, No Cycle = Allowed outcome

	Observable	Unobservable
Permitted	OK	OK (stricter than necessary)
Forbidden	BUG	OK

Core 1	Core 2
(i1) st[x] <- 1	(i4) st[y] <- 1
(i2) ld[x] -> r1	(i5) ld[y] -> r3
(i3) ld[y] -> r2	(i6) ld[x] -> r4
Forbidden Outcome: r1=1,r2=0,r3=1,r4=0	

Initially: [x]=[y]=0, We assume sequential consistency

ISA-level happens-before graphs

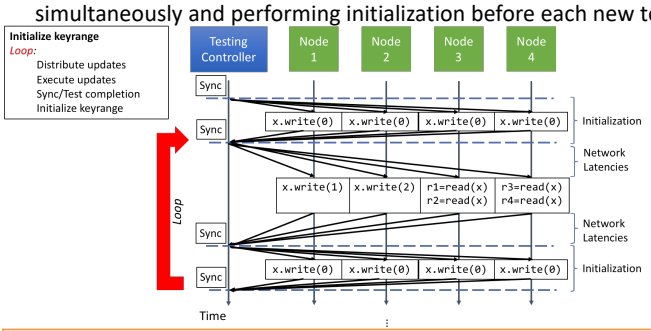


Litmus tests are targeted tests that access data, looking for outcomes that are not allowed under specific guarantees. Repeated runs of the tests ensure an implementation is tested with many different timings.

Challenge 1: Create and use litmus tests on distributed systems

Node 1	Node 2	Node 3	Node 4
Client 1	Client 2	Client 3	Client 4
(i1) x.write(1)	(i2) x.write(2)	(i3) r1=read(x) (i4) r2=read(x)	(i5) r3=read(x) (i6) r4=read(x)
x=0 initially			
Outcome tested: r1=1, r2=2, r3=2, r4=1			
For a system implementing strong consistency (e.g. linearizability): r1=1, r2=2, r3=2, r4=1 is a FORBIDDEN OUTCOME			

- Adapt litmus tests to distributed key-value stores and distributed consistency models. Execution and storage of a client are **decoupled**. Updates are not local operations.
- Develop **Musli Tool** to repeatedly execute test, executing updates simultaneously and performing initialization before each new test.



Analysis of 1000 iterations of Concurrent Writes test. Selected outcomes correspond to Permitted/Forbidden outcomes under different consistency guarantees.

Outcome	Cassandra (Eventual)	Explanation
1,2,1,2	15	Permitted
1,2,2,1	2	Forbidden under: Linearizability
1,2,2,2	12	Permitted
2,0,1,2	2	Forbidden under: Monotonic Reads
2,0,2,1	1	Forbidden under: Monotonic Reads
2,1,1,1,	47	Permitted

Challenge 2: Create synchronization free litmus tests

Perpetual litmus tests remove synchronization after every test iteration. Instead each test's immediate values replaced with monotonically increasing sequences of values, so iteration number can be recovered. Improves performance and stress testing, but correctness detection harder.

Conclusions

- Correctness and security are difficult to achieve in CPS/IoT.
- Correctness and security rely on appropriate static and dynamic verification techniques, as security attacks often exploit design mistakes and their implications.
- Our work is creating static and dynamic verification techniques in support of these correctness and security goals.