A **Cyber-Physical OS** for enabling Spatio-Temporal Coordination at Geo-distributed Scale

Sandeep D’souza and Raj Rajkumar
*Carnegie Mellon University*

1st Workshop on Next-Generation OS for Cyber-Physical Systems 2019
Cyber-Physical OS

- An OS which aids:
  - the development, deployment, scheduling and management
  - of cyber-physical applications over distributed infrastructure

CPS Applications

- System Services
- Hypervisor
- Kernel Modules
- Middleware
- Device Drivers

Distributed Infrastructure: Sensors, Actuators, Compute, Network, Storage

Layered OS with user-space to kernel-space components
Coordination in Space and Time*

Accurate Time and Location key for Spatio-Temporal Coordination

D’souza et al., HotCloud ‘17
Outline

- Motivation
- The Importance of Time & Localization
- Exposing Time & Location as First-Class Entities
- Realizing a Spatio-Temporal Cyber-Physical OS
Part I: The Importance of Time & Localization
A Shared Notion of Time

- *Ordering* of Events
- *Coordinated* Actions

A Shared Notion of Time is *useful*

→ Replace *Communication* with *Local Computation*

*Liskov, Distributed Computing ‘93*
Accurate Localization

- **Location-based** sensing and actuation
- **Low-latency** proximal computation
- **Safe Interaction** between physical endpoints

A Shared Localization Frame-of-Reference is *useful*
→ **Safe+Efficient Coordination** at the Correct Location
City-Scale Connected Vehicles

Required: A Cyber-Physical OS with *Time* & *Location* as First-Class Entities

"D'souza et al., HotCloud '17"
Coordinated Vehicles using *TimeNet*

- **TimeNet**: Cyber-Physical Internet
  - *perfect* time+geo stamping
- **Dynamic Traffic Management**
  - *city-scale* vehicular coordination
    - Timestamps+Location → *event ordering*
    - event ordering → *coordination policy*

*Uncertain* time and location estimates can *violate* safety constraints
Part II: Exposing Time & Location as First-Class Entities
Time & Location as *First-Class* Entities

- **Access to OS-supported Time+Location abstractions**
  - *read* the current time & location
  - *observe* and *schedule* events + computation

- **Specify & Observe Timing+Localization Uncertainty**
  - specifying safety-dictated uncertainty tolerances
    - allows the *system* to *autonomously meet* them
  - observing the delivered uncertainty
    - allows *applications* to *adapt* during failures

Exposing *uncertainty metrics* to applications
→ enables an *autonomous* system + *adaptive* applications
Quality of Time (QoT)*

- Quantified
  - using *clock parameters*: accuracy, precision, drift...
  - w.r.t a *reference clock* (time)
- Each timestamp has bounds
  - Timestamp $\in \{t-\varepsilon_1, t+\varepsilon_2\}$

The *end-to-end* uncertainty in the notion of time delivered to an application by the system

*Anwar et al., RTSS ‘16*
Quality of Location (QoL)

- Quantified
  - using *localization parameters*:
    - accuracy, precision, drift....
  - w.r.t a *reference frame*
- Each location estimate has bounds
  - Location $\epsilon \{L+/-E_h\}$
  - L is the location vector, and E the uncertainty vector

**The uncertainty radius** in a location estimate, with respect to a reference
QoT & QoL-based Connected Vehicles

- QoT & QoL Requirements based on
  - safety requirements
  - coordination policy
- If uncertainty *exceeds* tolerable limit
  - coordination policy can *adapt*
  - **Graceful Degradation:**
    - Increase vehicular spacing
  - **Safe Halt:**
    - Instruct vehicles to stop

Synchronized Clocks + Localization → *Scalable Spatio-Temporal Coordination*
Quality of Time + Quality of Location → *Fault Tolerance*
The Relation Between QoT & QoL

- QoT & QoL Requirements
  - influenced by *safety requirements*
  - can be *interdependent*

- Fleet of connected autonomous vehicles
  - *velocity & inter-vehicular spacing constraints*
    - decided by safety requirements
  - *decision-making system*
    - *higher localization accuracy* (lower uncertainty)
    - allows *less-accurate timestamps* (higher uncertainty)

Relation between clock synchronization and localization
→ Quality of Time & Quality of Location *requirements are interdependent*
Part III: Realizing a Spatio-Temporal Cyber-Physical OS
Coordination in CPS - Challenges

- **Scalability**
  - Both *numerical* and *geographical*

- **Autonomy & Fault Tolerance**
  - *adapt* to application requirements
  - *graceful degradation* during adversity

- **Ease of Programmability**
  - *coordination framework* with APIs

- **Ease of Deployment & Management**
  - *complex applications* on *heterogeneous infrastructure*

- **Security & Privacy**
  - *protect* users and infrastructure

Need for a *Cyber-Physical OS* which meets all these requirements
Enabling Spatio-Temporal Coordination

- Autonomous Time and Location Services
  - scale across *numerous distributed* nodes
  - adapt to *application demands* and *faults*

*Expose* time and localization *as adaptive & autonomous services* to cyber-physical applications
Case Study: Time-as-a-Service (TaaS)

- Leverage mature open-source
  - protocols: NTP, PTP ...
  - technologies: GPS, hardware timestamping
- Adapts to Application QoT Demands
  - tunable clock synchronization
  - probabilistic QoT-estimation mechanisms
- Autonomous & Fault-Tolerant
  - adapts to clock-sync failures
  - notifies apps if QoT degrades beyond spec

The same ideas can be extended to Localization-as-a-Service (LaaS)
  → adaptive layer leveraging mature localization technologies
Design 1: LAN-scale QoT Stack for Linux*

Support for ARM and x86 platforms + QEMU-KVM Virtual Machines^
open source, modular implementation, no change to the Linux kernel

^Dsouza et al., RTAS ‘18  *Anwar et al., RTSS ‘16
Design 2: Geo-Scale Quartz TaaS*

Micro-service architecture for providing *Time-as-a-Service* over *wide-area* scales across embedded endpoints, to the edge and the cloud

*Developed in collaboration with Nutanix Inc.*
User-space vs Kernel-Space

● Implement general functionality as **user-space** micro-services
  ○ good for *scalability, portability* and *security*
  ○ easy to *develop, deploy* and *manage*
  ○ *performance* and *accuracy* can suffer

● Rely on **kernel/driver support** for high accuracy
  ○ great for utilizing *specialized* hardware
  ○ *loss of portability*, kernel-version/OS dependencies
  ○ *security risks*

Need a **balance** between placing functionality in user-space and kernel space
  → **balance performance with scalability and ease-of-use**
Deploying Applications @ Scale

- CPS: Physical Access is Challenging
  - nodes in the *real-world*
- Heterogeneous Infrastructure
  - binary compatibility and dependencies
- Software Management Layer
  - Application+Services *Lifecycle Management*
  - *Multi-tenant* resource allocation
  - Infrastructure Management

*Virtualization* technologies like *containerization* in conjunction with *orchestration* technologies like *Kubernetes* can help in app deployment
Ease of Development: API

- Timeline*: Virtual reference time base
- Coordinated actions on distributed components
  - all components bind to a common timeline
  - each specifying its required QoT
- Timeline-based API
  - specify uncertainty tolerances
  - read the current time with associated uncertainty
  - schedule events in space and time

Timelines can be extended to provide a common location reference to all the bound components, each specifying their desired QoL

*Anwar et al., RTSS ’16
Summary

● Geo-Distributed Coordination in CPS
  ○ *Time* and *Location* are key

● A Cyber-Physical OS for Spatio-Temporal Coordination should
  ○ expose *time and localization as a service* to applications
  ○ expose *uncertainty metrics* to applications

● Key Objectives:
  ○ Scalability, Autonomy, Ease of Use, Fault-Tolerance, Security & Privacy

● Implementation Challenges
  ○ use of *virtualization* and *orchestration* technologies
    ■ *overheads*, low-level *device access* (sensors/actuators)
  ○ balance between *kernel-level* and *user-space* functionality
    ■ *performance* vs *portability & scalability* trade-offs
    ■ *security* risks due to kernel-level access