An End-to-End Quality of Time (QoT) Stack for Linux

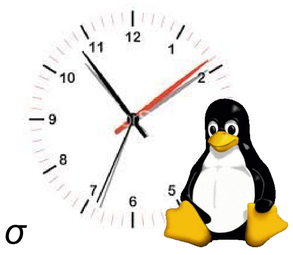
Award # CNS-1329755 (UCLA), CNS-1329644 (CMU), CNS-1329644 (UCSD), and CNS-1329650 (UCSB)
Type: Frontier; Start Date: June 2014

Andrew Symington (UCLA)

Co-Authors: Justin Pearson, Joao Hespanha and Masashi Wakaiki (UCSB), Fatima Muhammad, Paul Martin and Mani Srivastava (UCLA), Adwait Dongare and Anthony Rowe (CMU)

Motivation

- Linux exposes time to applications in a very limited way
  - No sense of uncertainty in current time estimate
  - Synchronization independent of application demand
  - Synchronization is not adpative or resource-aware
- What makes up Quality of Time (QoT)
  - **Knowing** time - *The current time is **x** with uncertainty σ*
  - **Keeping** time - *Wake me up after **x**, but no later than **x**±σ*
  - **Sharing** time - *Declare a group of networked applications to share time*
  - **Controlling** time - *Adapt a local sense of time to balance resources*
    - Switching between oscillators to drive local time representation
    - Election of master time source for synchronization
    - Rate at which synchronization is carried out with master
- **Goal**: An adaptive, end-to-end Linux stack for enabling QoT

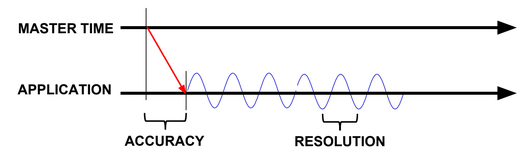


Key Idea

- Applications **bind** to **timelines** with a desired **accuracy** and **resolution**
- The system performs synchronization to achieve application demands
- Programmers use C++ interface to bind to timelines and query time

```
int64_t accuracy = 1e3;  // worst-case phase offset (ns) from timeline
int64_t resolution = 1e6; // minimum acceptable tick resolution (ns)
qot::Timeline timeline("my_test_timeline", accuracy, resolution);
timeline.SetName("my_application_name"); // name of this application
```

```
int64_t tval = timeline.GetTime(); // current time (ns)
int64_t terr = timeline.GetError(); // error in the time estimate
timeline.WaitUntil(tval + 1e9); // blocking wait until 1s from tval
```



- Also supports interaction with other cyber-physical systems

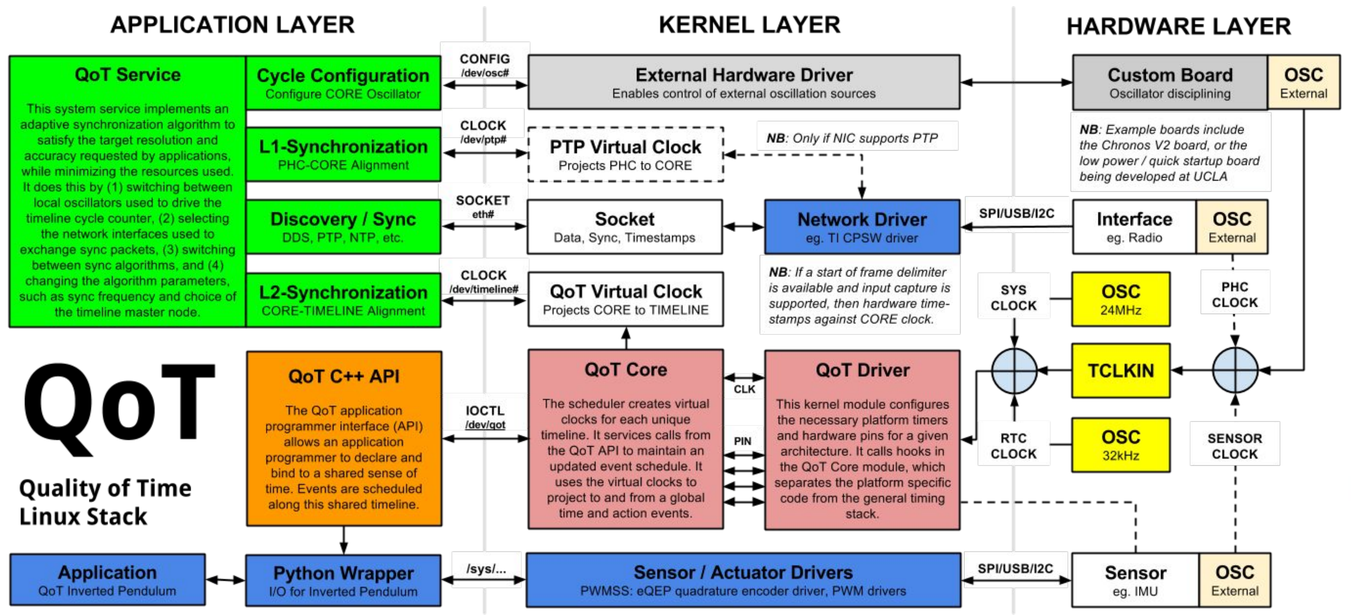
```
bool timeline.GenerateInterrupt(
    "timer_name", // name of the timer pin
    true, // true: enable, false: disable
    timeline.GetTime() + 1e9,    // time of first rising edge
    1e9, // PWM duty cycle high time (ns)
    2e9, // PWM duty cycle low time (ns)
    10   // number of PWM cycles - 0:infinite, 1+:finite
);
```

```
void callback(const std::string &tid, int64_t t) {
   std::cout << "Timer " << tid << " irq at " << t;
}
timeline.SetCaptureCallBack(callback);
```

Interface with devices, such as sensors and radios

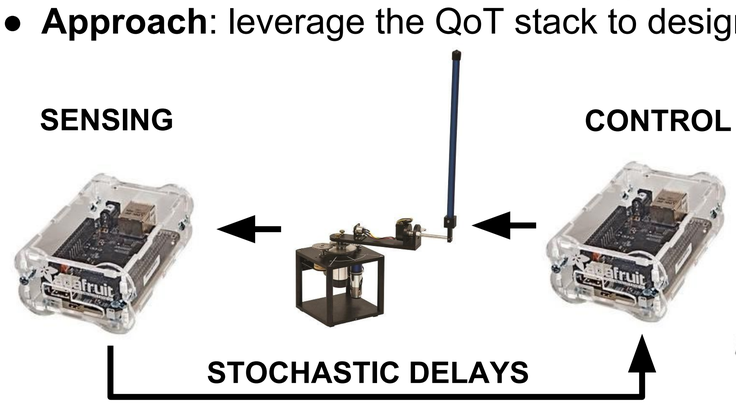
The Quality of Time (QoT) Stack for Linux

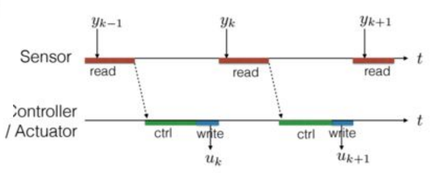
- **Interface** - the application programmer interface is a library that a third-party programmer links against in order to develop a time-aware application.
- **Module** - two kernel modules: one manages POSIX clocks that represent timelines, and the other provides hardware-dependent hooks to a timing architecture.
- **Service** - a data distribution service (DDS) driven daemon observes and shares timelines, and performs resource-aware synchronization to meet requirements.



Example: Distributed Inverted Pendulum

- Typical controllers assume sensing and actuation to occur simultaneously
- **Problem**: network-distributed sensing/control cannot stabilize pendulum
- **Approach**: leverage the QoT stack to design a delay-tolerant controller





- **Result**: we show that a QoT-aware controller stabilizes the pendulum

Current Research Directions

- **Improved synchronization algorithms** - negotiation of master and slaves, based on application demands, the network topology, and resources available.
- **Improved oscillation options** - development of boards to discipline a crystal oscillator directly (Utah) and provide a fast start-up, low-power clock source (UCLA).
- **Improved radio technologies** - developing PTP compliant Linux network drivers for an ultra-wideband radio, in order to enable wireless synchronization.
- **Quadrotor experimental platform** (pictured right) - a UWB system for indoor quadrotor stabilization is currently undergoing testing at UCLA.

Code is available online: https://bitbucket.org/rose-line