# A Testbed to Simulate and Analyze Resilient Cyber-Physical Systems

Pranav Srinivas Kumar, William Emfinger and Gabor Karsai
Institute for Software Integrated Systems,
Dept. of EECS, Vanderbilt University,
Nashville, TN 37235, USA
Email:{pkumar, emfinger, gabor}@isis.vanderbilt.edu

***Abstract*—This paper describes a testbed for development, deployment, testing, and analysis of Cyber-Physical Systems (CPS) applications. The testbed incorporates smart network hardware, allowing high-fidelity emulation of CPS network characteristics, and CPS simulation environments to enable high-frequency sensor reading, actuator control and physical environmental changes. We discuss the architecture of this testbed and present the types of experiments and applications which can be run to study hardware and software fault tolerance, software reconfiguration, and system stability characteristics in distributed real-time embedded systems. We also describe the scalability, limitations, and potential extensions to this testbed.**

## I. Introduction

Distributed CPS are hard to develop hardware/software for; because the software is coupled with the hardware and the physical system, software testing and deployment may be difficult - a problem only exacerbated by distributing the system. Many systems require rigorous testing before final deployment, but may not be able to be tested easily in the lab or may not be testable in the real world without first providing the assurances that the tests produce. These types of systems must be tested for performance assurances, reliability, and fail-safety. Examples of these systems include UAV/UUV systems, fractionated satellite clusters, and networks of autonomous vehicles, all of which require strict guarantees about not only the performance of the system, but also the reliability of the system. Because of the need for such strict design-time guarantees, many traditional techniques for software testing cannot be used. Cloud-based software testing may not accurately reflect the performance of the software, since many of these systems use specialized embedded computers, and furthermore does not provide the capability to easily integrate a system simulation into the software testing loop. For such systems, a closed-loop simulation testbed is necessary which can fully emulate the deployed system, including the physical characteristics of the nodes, the network characteristics of the systems, and the sensors and actuators used by the systems.

Emerging industry standards and collaborations are progressing towards component-based system development and reuse, *e.g.* AUTOSAR[1] in the automotive industry. As these systems are becoming increasingly more reliant on collections of software components which must interact, they enable more advanced features, such as better safety or performance, but as a consequence require more thorough integration testing. Comprehensive full systems integration is required for system analysis and deployment, and the development of relevant testing system architectures enables expedited iterative integration. Developing these systems iteratively, by prototyping individual components and composing them can be expensive and time consuming, so tools and techniques are needed to expedite this process. Our testbed architecture was developed to help address these issues and decrease the turn-around time for integration testing of distributed, resilient CPS.

Examples of such systems which can be prototyped and tested using this architecture are (1) autonomous cars, (2) controllers for continuous and discrete manufacturing plants, and (3) UAV swarms. Each of these systems is characterized as a distributed CPS in which embedded controllers are networked to collectively control a system through cooperation. Each subsystem or embedded computer senses and controls a specific aspect of the overall system and cooperates to achieve the overall system goal. For instance, the autonomous car's subsystems of global navigation, pedestrian detection, lane detection, engine control, brake control, and steering control all must communicate and cooperate together to achieve full car autonomy. The control algorithms for each of these subsystems must be tested with their sensors and actuators but also must be tested together with the rest of the systems. It is these types of cooperating embedded controllers which are a distinguishing feature of distributed CPS. Integration testing for these distributed CPS can be quickly and easily accomplished using hardware-in-the-loop simulation, but must accurately represent the real physical system, hardware, software, and network.

In scenarios like the automotive networked CPS, one of the main challenges with system testing is the discord between the standardized networking protocols and communication methods e.g. CAN bus, and the manufacturer-specific implementations of these methods. It is difficult to obtain public access to the implementation details for such interaction patterns and therefore pure simulation of the communication protocols using event-simulation tools is not sufficient in validating resilient application performance. The comprehensive testing for such safety-critical systems require *replicating the CPS* by using a testing infrastructure that provides similar hardware and executes the exact embedded control code that would execute on the final system. Our proposed architecture aims at achieving this level of testing refinement.

The rest of this paper is organized as follows - Section II states the various requirements that we have identified for a resilient CPS testbed. Section III describes the overall architecture for the testbed and Section IV briefly presents our design and construction methods. Section V details our concrete experiments with the testbed and Section VI discusses the limitations we have identified. Lastly, Sections VII and IX present potential future extensions and concluding remarks.

## II. Testbed Requirements

The purpose of this testbed is to provide researchers and developers a platform for development, testing, and analysis of distributed, resilient CPS applications. The testbed must provide (1) capable hardware on which applications can be deployed. (2) a networking infrastructure facilitating application interactions, (3) a set of simulation environments to integrate a physics model with sensing and actuation, and (4) an interface to couple application processes with the simulation. Critical to each of these components is the requirement that the sum of all characteristic behaviors must reasonably and reliably approximate the behavior of the real systems.

For the testbed hardware, this means that the processor functionality, speed, and memory should have similar characteristics to the systems' processor. This requirement is easily met since most CPS development or embedded system development starts with a prototyping or developer board from the processor manufacturer which allows for testing and development on the actual processor, before fabricating custom boards. These boards can be used as the hardware for the testbed.

For the application communications, it is imperative that the application network traffic receives reasonably similar network services on the testbed as it would in the real system. This means that the network traffic should see approximately the same latency, buffering, routing, etc.

The simulation is one of the most critical components of the CPS testbed, since it provides the feedback from the simulated physical world to the applications, which is a critical component of CPS software development. The environment and physical domain in which the system will be deployed affects the simulators which can be used, and the system itself affects the required timing properties of the simulator. Irrespective of those two concerns, the general requirement of the simulator is that it runs at least as fast as "real-time," where real-time really means faster than the highest sampling/actuation rate required by the system.

Coupled with the simulation requirements is the final requirement to close the loop between the physical simulation and the application hardware. Closing this loop allows for the applications on the testbed hardware to interact with and affect the simulated physical system. This communications layer must integrate with the simulator, a requirement generally met by a plug-in which exposes an API over sockets to communicate with the simulator. Furthermore, the communications layer must have reasonably accurate timing with respect to the sensing and control actions required by the applications. Because this communications layer simulates the direct physical connection a host has to a sensor or actuator, any extra overhead, jitter, or interference caused by the network must be taken into consideration when evaluating test results.

## III. RCPS Testbed Architecture

Cyber-Physical Systems require design-time testing and analysis before deployment. Several CPS scenarios require strict safety certification due to the mission-critical nature of the operation, e.g. flight control and automation. It is often times impossible to test control algorithms, fault tolerance procedures etc. on the real system due to both cost and hardware accessibility issues. To counter these issues, there are two principle methods in which a CPS can be tested and analyzed: (1) Construct a complete model of the CPS in a simulation environment e.g. Simulink [2] and simulate the system while accounting for run-time scenarios, (2) Establish a testing environment that can closely resemble the real CPS in both hardware and software. The problem with simulations is that it is hard to establish the network topology, emulate the application network and base processing power while running a physics simulation in the loop. Our RCPS architecture implements the latter alternative, as shown in Figure 1. We propose a generic testing environment that uses embedded boards, programmable network switches and physics simulation machines to emulate real CPS deployments.

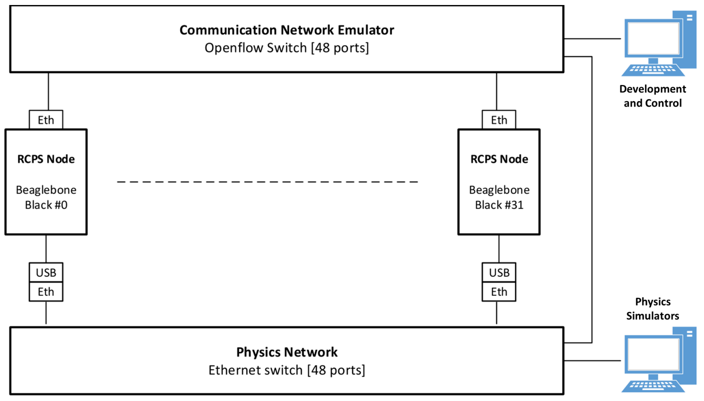


Fig. 1: Testbed Architecture

The testbed consists of 32 *RCPS nodes*, each of which is a Beaglebone Black (BBB) [3] development board running Linux. We execute a full software stack including a middleware and a component model, although much of this stack is not within the scope of this paper. For the subset of CPS we are interested in, the behavior of the CPS can be much more precisely emulated with these boards compared to running the applications inside of a standalone simulation. For example NASA's CubeSat Launch Initiative (CSLI) [4] provides opportunities for nanosatellites to be deployed into space for research. CubeSats are small (4-inch long) satellites running low-power embedded boards and being prepared for interplanetary missions [5] to Mars. A distributed set of CubeSats can be easily tested with this architecture if it can be integrated with a high-fidelity space flight simulator.

The Gigabit Ethernet port of each BBB is connected to a *Communication Network* switch. This is a programmable OpenFlow [6] switch, allowing users to program the flowtable of the switch to control the routes that packets follow and completely configure the full network and subnets required for their emulated deployment. Furthermore, the configurability of the communications network enables per-link or per-flow bandwidth throttling, enabling precise network emulation. The primary *Development and Control* machine, running our software development tools, communicates with the BBBs using this network. After software applications are deployed on this testbed, the characteristics of the real CPS network can be enforced on the application network traffic. Therefore, this network emulates the physical network which a distributed CPS would experience on deployment.

Each RCPS node is also connected to a *Physics Network* using a 10/100 USB-to-Ethernet adapter, since the BBBs only have one gigabit ethernet port. This network is connected to a *Physics Simulation Machine* running Cyber-Physical Systems simulations. Two such simulators are discussed in Section V. This network provides the infrastructure necessary to emulate CPS sensing and actuation in the loop, allowing application software to periodically receive sensor data and open interfaces to output actuator commands to the simulation.

The Physics Simulation Machine closes the interaction loop for the testbed nodes, allowing the physical dynamics of the RCPS nodes to be simulated in the environment in which it would be deployed, *e.g.* satellites' orbital mechanics and interactions can be simulated for a satellite cluster in low Earth orbit (LEO).

## IV. Design and Construction

The RCPS testbed was designed and constructed to be as self-contained as possible, while allowing for extensibility and modularity. The 32 BBB development boards were arranged on 4 laser-cut acrylic plates, with 8 boards on each plate, as shown in Figure 2. Holes on each plate route network and power supply cables from/to each board. With the communication network switch on top of these boards and the physics network switch on the bottom, the ensemble takes a total of 8U (rack units) of space. The primary power supply is an off-the-shelf 300 W power supply with custom cabling to fan-out power to all 32 boards in an efficient manner. By configuring the testbed in this self-contained way, we can easily maintain it, monitor it, and move it should the need arise.

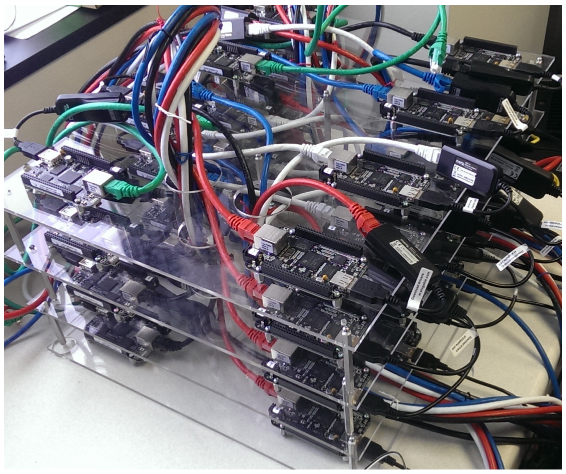


Fig. 2: Beaglebone Black Boards - Custom Mounting

Using a 13U server cabinet and standard mounting equipment, the network switches and simulation machines are mounted on either side of the development boards. Figure 3 shows the fully mounted testbed. The cabinet supports mountable base wheels which makes this setup easily portable.

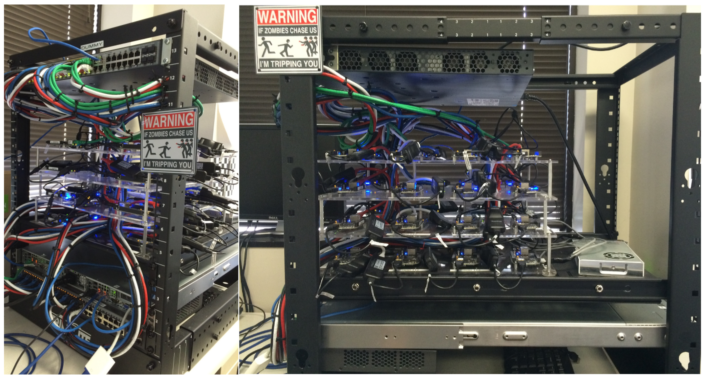


Fig. 3: Constructed Testbed

## V. Experiments

To validate our claims about the testbed, we have run several experiments that exercise its various aspects, primary of which is its utility in sensing, actuation and control of dynamic CPS scenarios. Our testing and experimentation parameters are as follows: (1) We use the Robot Operating System (ROS) [7] middleware and follow component-based software development principles, (2) the physics simulation machine is running an appropriate high-fidelity simulation with exposed interfaces for input and output, (3) the development machine has full network access to all RCPS nodes in order to successfully develop, build and deploy applications on this distributed setup, (4) the applications have direct access to the physics simulation sensors emulating the real-world case where the sensor is part of the very board running the application, and lastly (5) the processes are able to log real-time activity to aid the monitoring infrastructure. This monitoring not only validates the behavior but also provides important measures such as operational time stamps that enable post-deployment timing analysis to assess the overall system performance.

This section briefly describes two unique scenarios pertaining to flight control dynamics. These are mission-critical CPS environments using different kinds of sensors, actuators and physics simulations. The real-time requirements of both set of applications are different and the average processing power capabilities of both systems can be approximated to our testbed nodes. Below, we categorize these experiments based on the physics simulators used.

### A. *Orbiter Space Flight Simulator*

This is a scenario we have used in the past [8] to simulate a cluster of satellites. In our previous work, we used a minimal testbed to study mobile cloud infrastructure and integrated the Orbiter Space Flight Simulator [9] to analyze a 3-Sat scenario. Now, with our RCPS testbed, the same scenario can be extended up to 32 satellites. Given the flexibility of this testbed, we can easily change the scenario to a 16-Sat cluster with 2 embedded boards per satellite or any similar variations. The amount of process concurrency and actuation is dependent primarily on the deployment infrastructure used. If a new test simply requires a new deployment model and some artifact XML files, then the testing mechanisms can be easily automated.

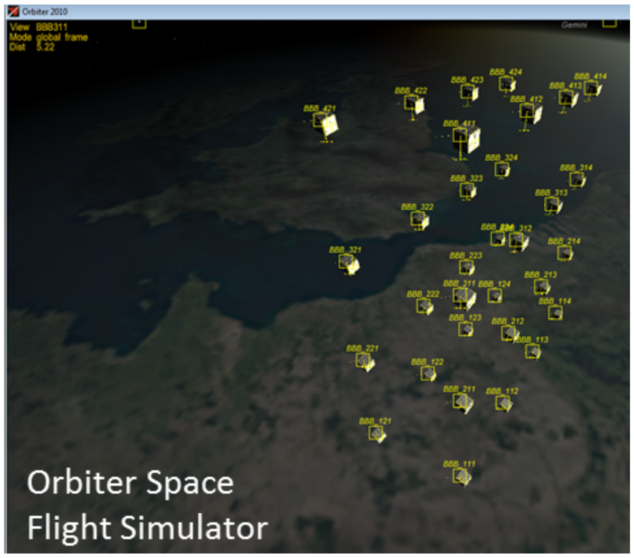


Fig. 4: 32-Sat Scenario

This scenario, shown in Figure 4, consists of 32 small satellites orbiting as a satellite cluster in LEO. The satellites each have different capabilities, with image sensors and imaging processors distributed throughout the cluster. Each satellite runs a satellite flight control application which determines the state of the satellite, distributes the state to the rest of the cluster, tracks the states of the other satellites, and controls the satellite's thrusters. This application communicates with a single orbit maintenance application in the cluster which maintains the orbital trajectories of all satellites in the cluster and instructs the cluster to perform maneuvers when needed. In addition to these to applications, the cluster's resources are provided to applications which perform image sensing, image processing, and communication tasks.

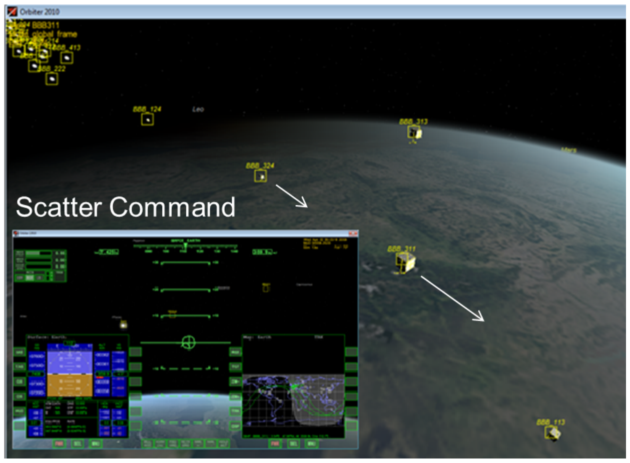


Fig. 5: Scatter Maneuver

It must be noted in this scenario that Orbiter executes only in Windows. The Physics simulation machine is a dual-boot setup capable of supporting a wide range of simulation due to both its CPU power (16 cores with 16 GB of RAM) and its NVIDIA Quadro K1200 graphics card. Another limitation to Orbiter is that the interface used to communicate with Orbiter, OrbConnect, does not provide the full range of attitude control for the satellites, although Orbiter supports this level of actuation. Therefore, even though the scenario exercises a scatter maneuver on receiving a critical command from the ground station, as shown in Figure 5, the behavior of this CPS is slightly inconsistent with the real-world scenario. Due to this, we have moved to a different flight simulator, as described below.

One of the relevant and interesting aspects of this satellite example is the networking required between the satellites. Because of the nature of the orbital mechanics governing the motion of the satellites, the distances between each of the satellites in the cluster varies periodically over time as a sinusoidal function. This deterministic variation in distance directly correlates to a deterministic network capacity variation which also is periodic according to the orbital period of the cluster. Because this information can be easily retrieved from the simulation, it can be used to alter the emulated network characteristics. Since the variation is deterministic and calculable, the emulated network characteristics can be measured and compared against the calculated, predicted network characteristics.

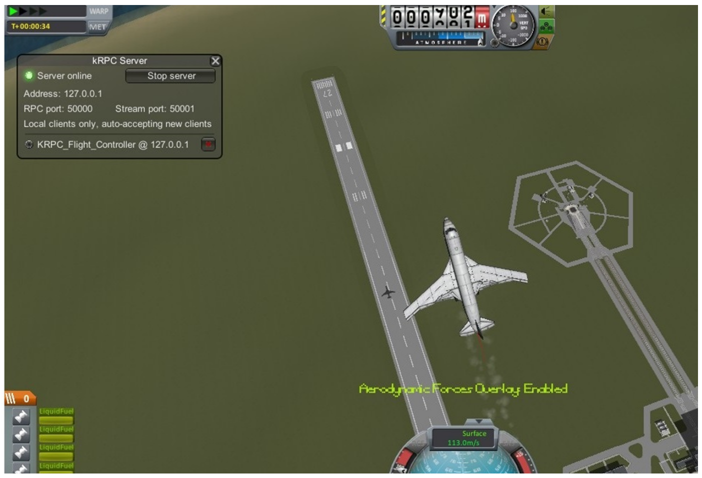
### B. *Kerbal Space Program*

Kerbal Space Program [10] (KSP) is a widely popular space flight simulator for a variety of platforms including Linux, OS X and Windows. In this game, players get to manage a space program, designing and building spacecrafts and exploring celestial bodies.

While KSP does not provide a perfect simulation of reality, it has been widely praised for its component-based design and development process coupled with aerodynamic, gravitational, and rigid-body interaction and simulation. In this simulation, every man-made object follows Newtonian dynamics. Rocket thrust and aerodynamic forces are accurately applied to the vehicles based on the directions and precise positions in which the force-affected elements are mounted on the vessel. Using KSP, we have modeled scenarios for a variety of flight missions including interplanetary travel. In this section, we briefly describe an aircraft flight controller that was designed and tested using the RCPS testbed and KSP.

This CPS scenario is a flight controller application used to completely control a KSP aircraft from the primary space-plane hanger to a destination airport. The application processes require inputs from KSP e.g. sensor data about pitch, roll, yaw, mean altitude etc. and interfaces to control the flight dynamics e.g. thrust, pitch and heading. If these interfaces are setup, then the processes can periodically retrieve flight telemetry and provide commands for course correction and feedback control.

Using an open source project called kRPC [11] (Kerbal Remote Procedure Call Server), the BBB nodes running CPS processes are provided with an interface to the simulation. Figure 6 shows the Stearwing A300 aircraft taking off from the space-plane hanger and stabilizing at a cruising altitude



Server online

Address: 127.0.0.1

RPC port: 50000 Stream port: 50001

Local clients only, auto-accepting new clients

KRPC_Flight_Controller @ 127.0.0.1

Fig. 6: Stearwing A300 PID Control

of 2000 meters, as shown in Figure 7. Each control unit in the aircraft is simulated by a BBB. It is critical to incorporate redundancy in all components and connections to assure that the system can survive runtime failures, especially in airborne software [12] [13]. So, many of the RCPS nodes in our testbed act as redundant sensors, all connected to KSP, receiving and periodically publishing messages. If one of the nodes in the testbed fails, either due to a hardware anomaly or a software fault, then the rest of the nodes are used to arrive at a general agreement regarding the sensor value to be considered for control.

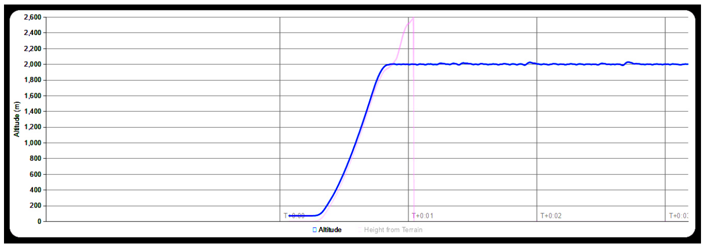


Fig. 7: Stearwing A300 Altitude Profile

Like Orbiter, using KSP also has its limitations. However, many such limitations are characteristic of the simulation interface and not the simulation itself. The C++ interface library we have used does not provide for the means to control multiple *vessels* in the simulation. Instead, developers are provided the complete control API for the *active vessel*, which is the vessel currently in focus in the game. Although such an interface is currently in the works, as it stands, KSP is a good solution for testing CPS use cases where a set of embedded boards are coordinating and controlling a single vessel or medium. Also, the KSP community provides several crowd-made modifications to the base game that supports various vehicle parts, aerodynamic changes, and drag models that enable researchers to build and test a variety of systems including cars, rovers, UAV swarms, robots, and even laser-guided systems. The open nature of this simulator makes this a realistic testing environment for dynamic CPS specifications.

## VI. Discussion

Despite the capabilities of this RCPS testbed architecture in analyzing distributed resilient CPS, its limitations must be evaluated with respect to system applicability and experimental validity. Such limitations govern the types of CPS which can be developed and tested with this architecture, as well as the scale of the systems to which this architecture can be used.

Because much of the testbed infrastructure is networking, and because much of the distributed application development and testing comes from the network infrastructure provided by the CPS (and therefore emulated by the testbed), one of the most critical aspects of the CPS testbed to analyze is the networking infrastructure. In this testbed's architecture, there are distinct networks reserved for application level network traffic, i.e. data which would flow on a real network in the deployed system, and for physics simulation network traffic, i.e. data which would be relayed by hardware directly connected to the embedded boards. The simulation network traffic allows the embedded hosts in the network to retrieve sensor data and send actuator commands to simulated hardware on the physics simulation node. Because this network is one of the major differences between the testbed and the real system, it must be carefully analyzed to determine how accurately it behaves with respect to the real system.

The actual network in the testbed is comprised of two parts: (1) a Gigabit network switch which is dedicated to the physics simulation traffic, and (2) the 10/100 USB-Ethernet adapters on each board which allow a direct connection to the physics network. The USB-Ethernet adapters on the boards do not allow the boards to utilize the full capabilities of their Ethernet connections to the physics network, but provide sufficient bandwidth for many sensor applications. For a single host, many sensor streams or a few high-data-rate sensors can adversely affect the host's sensor timing and services, thus affecting the system overall. High-resolution or high-rate image-based sensing on a certain host may degrade the performance of the host's physics network link, but should not affect the rest of the physics network. However, many such high-bandwidth sensing applications distributed across the cluster may degrade the overall performance of the physics network. Because in the real system, these high bandwidth data streams would be completely isolated and independent, this type of network performance degradation can cause the testbed applications' behavior to deviate from the real system applications' behavior.

The other network which plays an critical role in the testbed is the application network, which handles all network traffic for the system which would actually be transmitted on a network in the deployed system. The purpose of this network is to enforce on the applications' network traffic the same network characteristics as would be seen in the real system. Because the wired testbed application network is managed by a smart network switch using OpenFlow, many different and concurrent topologies can be enforced, and every link in the networks can be configured with different, varying bandwidth. For many types of networks, this type of network emulation is sufficiently accurate and will provide meaningful, useful test results. However, certain types of networks in deployed systems cannot be easily emulated with this setup, for instance, wireless networks with high error rates and high packet-loss or collision rates are difficult to emulate using this network infrastructure. Moreover, the environmental effects such as multi-path or obstruction may be difficult or impossible to emulate.

## VII. Future Work/Extensions

### A. Experiments

With this testbed, we are able to analyze different systems and multiple different aspects of those systems. Our primary goal with this testbed is the general analysis of distributed CPS through certain aspects such as resiliency and security. Based on this goal, we will analyze how certain systems behave with respect to fault tolerance, fault propagation, failure mitigation, and reconfiguration. These experiments govern some aspects of the resilience of the systems, and we will specifically focus on systems such as the aircraft software example mentioned above, as well as UAV swarms, autonomous cars, and traffic grid control systems.

Another critical aspect of these distributed CPS which we wish to analyze is their security. Because our testbed encompasses the physics, the hardware, the software, and the network of the CPS, we can analyze multiple attack vectors and their impact on the applications and the software infrastructure. These experiments can be coupled with the resilience experiments mentioned above to analyze the capability of attackers to achieve specific goals of bringing down certain subsystems or the overall system.

### B. Testbed Analysis

To better understand the capabilities and limitations of the testbed, we will measure such testbed characteristics as: (1) the effect the USB/Ethernet adapter has on communications delay, timing, and jitter, (2) the effect OpenFlow link management has on applications' usable bandwidth, and (3) the effect OpenFlow dynamically altering flows and bandwidth re-allocation has on active traffic.

### C. System Analysis

Using our deployment and management tool suite, we can analyze CPS deployments for system-level properties such as (1) Buffer Space Requirements satisfaction and per-link network traffic delay, (2) timing and schedulability satisfaction without deadline violations or system-wide deadlocks.

*1) Network Analysis:* Because we can accurately and precisely emulate the CPS network characteristics on the testbed, we can analyze the performance of the networked applications by measuring their utilized buffer space and their communication delay. These measurements, performed automatically throughout the system, provide a clearer insight into the network services provided by the system and the resource utilization required by the applications on the system. Comparing these requirements and services we can analyze the performance of the applications to determine the affect the network and its service capacity has on the applications' behaviors.

*2) Timing Analysis:* Timing analysis of models of CPS are useful only when the modeling abstractions for the CPS domain are uncompromising and the analysis techniques can be validated by testing in an approximated CPS testbed. This is one of our primary goals with the RCPS testbed. In previous work, we have shown our timing analysis methods [14] [15] for component-based distributed CPS. Using a generic *I/O component* for integration with physics simulation, we are working on modeling and analyzing the temporal behavior of software components while interacting with simulations. Our monitoring framework used with this testbed provides a strong infrastructure and the necessary inputs for timing verification.

## VIII. Related Research

Developing resilient software for CPS presents a unique challenge. These systems execute software pieces that are often times distributed across a collection of machines, presenting risks and challenges to both human safety and developmental costs. Research in such areas require tools and testbeds that enable an approximated evaluation of the real system. As a widely acknowledged challenge, several testbed architectures have been proposed in the past tackling heterogeneous concerns in CPS design such as security, fault tolerance and determinism.

Among the various application design challenges, many testbeds have focused on security research for CPS. Often, the interactions between hardware components and software control code in the real system is hard to replicate for testing, mainly in dynamic real-world-like environments. For example, in automotive networks, although the networking methods are standards like CAN, the implementation details are left to the hands of the board manufacturers and not easily accessible.

The maintainers of OCTANE [16] tackle these issues by providing a software package and a hardware infrastructure to reverse engineer and test automotive networks. By replicating the interactions between the system hardware and the control software, users can focus on security aspects in such networks instead of configuring and setting up the testing tools.

Similarly, the UPBOT [17] testbed provides a testbed for cyber-physical systems used primarily to test security threats and preventive measures. It presents a testing infrastructure to study several points of attack on programmable component-based systems where the on-board intelligence may be exhibiting safety-critical properties. The low cost and ease of use makes this an appreciable learning tool for students and researchers, especially ones lacking access to such testing environments.

In an alternate study, the Pharos [18] testbed provides a testing framework for mobile CPS. Using a networked system of autonomously mobile communicating controllers, the testbed demonstrates its utility in live testing of mobile CPS deployments, with comparisons against system simulation schemes. The study shows the absolute importance of validating simulation results with real-world experimentation and how Pharos enables such testing.

## IX. Conclusions

In this paper we have described a testbed architecture and implementation which enables researchers and developers to quickly prototype and develop distributed CPS applications and analyze their performance with hardware-in-the-loop simulation on networks which emulate the real system's networks. We have described with a few examples the types of experiments which this testbed can support and shown integration with multiple different simulators. Further, we have analyzed the limitations of this testbed for application to certain systems and applications. Finally, we have covered some proposed work for improving the testbed and testing framework in the future through further analysis and extensions to the testbed.

## ACKNOWLEDGMENT

This work was supported by DARPA under contract NNA11AB14C and USAF/AFRL under Cooperative Agreement FA8750-13-2-0050, and by the National Science Foundation (CNS-1035655). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA, USAF/AFRL, or NSF.

## REFERENCES

[1] Autosar GbR, "AUTomotive Open System ARchitecture," http://www.autosar.org/. [Online]. Available: http://www.autosar.org/

[2] "Simulink," http://www.mathworks.com/products/simulink/.

[3] "Beaglebone Black," http://beagleboard.org/BLACK/.

[4] "NASA CubeSat Launch initiative," https://www.nasa.gov/directorates/heo/home/CubeSats_initiative.html.

[5] "NASA CubeSats Mission to Mars," http://www.nasa.gov/press-release/nasa-prepares-for-first-interplanetary-cubesats-on-agency-s-next-mission-to-mars.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[8] D. Balasubramanian, A. Dubey, W. R. Otte, W. Emfinger, P. S. Kumar, and G. Karsai, "A rapid testing framework for a mobile cloud infrastructure."

[9] "Orbiter Space Flight Simulator," http://orbit.medphys.ucl.ac.uk/.

[10] "Kerbal Space Program," https://kerbalspaceprogram.com/en/.

[11] "Kerbal Remote Procedure Call Server," https://github.com/djungelorm/krpc/.

[12] A. J. Kornecki, "Airborne software: communication and certification," *Scalable Computing: Practice and Experience*, vol. 9, no. 1, 2001.

[13] A. J. Kornecki and K. Hall, "Approaches to assure safety in fly-by-wire systems: Airbus vs. boeing."

[14] P. S. Kumar, A. Dubey, and G. Karsai, "Colored petri net-based modeling and formal analysis of component-based applications," 2014, p. 79–88. [Online]. Available: http://ceur-ws.org/Vol-1235/paper-10.pdf

[15] ——, "Integrated analysis of temporal behavior of component-based distributed real-time embedded systems," 2015. [Online]. Available: http://conferences.computer.org/isorc/2015/papers/8781b019.pdf

[16] C. E. Everett and D. McCoy, "Octane (open car testbed and network experiments): Bringing cyber-physical security research to researchers and students." in *CSET*, 2013.

[17] T. L. Crenshaw and S. Beyer, "Upbot: a testbed for cyber-physical systems," in *Proceedings of the 3rd international conference on Cyber security experimentation and test*. USENIX Association, 2010, pp. 1–8.

[18] C. Fok, A. Petz, D. Stovall, N. Paine, C. Julien, and S. Vishwanath, "Pharos: A testbed for mobile cyber-physical systems," *Univ. of Texas at Austin, Tech. Rep. TR-ARiSE-2011-001*, 2011.