Chapter 18
Transportation Networks

Gabor Karsai, Xenofon Koutsoukos, Himanshu Neema, Peter Volgyesi, and Janos Sztipanovits

1 Introduction

According to one of the widely accepted definitions, CPS are engineered systems where functionality emerges from the networked interaction of computational and physical processes. The tight integration of physical and computational components creates new generations of smart systems whose impacts are revolutionary; this is evident today in emerging autonomous vehicles, military platforms, intelligent buildings, smart energy systems, intelligent transportation systems, robots, and smart medical devices. A recent study by McKinsey (Manyika et al. 2015) estimates that the ongoing digitization of industry will potentially add an additional 1.5 trillion US $ to the GDP of the United States by 2025 and 1 trillion EUR to the GDP in Europe. Emerging industrial platforms such as the Internet of Things (IoT), Industrial Internet (II) in the United States (Evans and Annunziata 2012), and Industrie 4.0 in Europe (Kagermann et al. 2013) are triggering a "gold rush" toward new markets and are creating societal-scale systems, which importantly, in addition to the synergy of computational and physical components, involve humans (H-CPS). H-CPS is at the heart of today's sharing economy and the driver of new kinds of industry sectors that involve humans interacting with CPS. These sectors are now producing companies which are changing how we live. For example, the future of mobility is being determined by companies like Uber, Lyft, Olla, and Didi, which are transforming personal transportation into a service. In addition, shared use of the third aerial dimension is being used to determine the future of logistics and how we deliver goods through our urban and rural infrastructures.

G. Karsai · X. Koutsoukos · H. Neema · P. Volgyesi · J. Sztipanovits (✉)
Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA
e-mail: gabor.karsai@vanderbilt.edu; xenofon.koutsoukos@vanderbilt.edu; himanshu.neema@vanderbilt.edu; peter.volgyesi@vanderbilt.edu; janos.sztipanovits@vanderbilt.edu

A. Kott, I. Linkov (eds.), *Cyber Resilience of Systems and Networks*, Risk, Systems and Decisions, https://doi.org/10.1007/978-3-319-77492-3_18

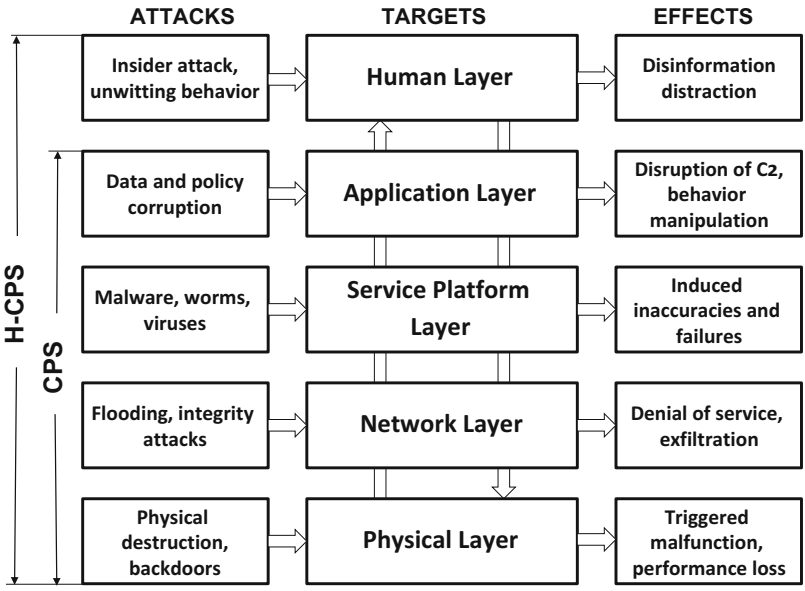

Fig. 18.1 H-CPS abstraction layers

It is not surprising that many of these systems are safety and mission critical that makes their resilience against faults and cyber-attacks an essential problem. Even under normal conditions, CPS and H-CPS face complex issues crosscutting many disciplines with significant implications on essential system functions. Designers must resolve complex constraints among physical configurations, disturbances, and software-based functions; they need to deal with significant differences in semantics of models and manage physical uncertainties and the always limited validity of physical models, as well as constraints caused by computational complexity. Adding faults and cyber-attacks in all their insidious variety creates a massive challenge that cannot be neglected due to their potential consequences. While conventional design of CPS and H-CPS implements functionalities that accounts for regular physical uncertainties in the system and tolerates environmental impact in some boundaries, resilient system design must handle larger variations in the external and internal physical and software environment and consider even maliciously injected physical faults and impacts of cyber-attacks.

H-CPS design processes use abstraction layers dictated by the heterogeneity of their component technology. Figure 18.1 shows a simplified view of abstraction layers that have distinct architectures, design patterns, composition principles, and vulnerabilities. The five fundamentally different abstraction layers are the *physical layer*, the three "cyber" layers (*network*, *service platform*, and *application layers*), and the *human layer*. The *physical layer* embodies physical components and their interactions which are modeled in continuous (physical) time. The mathematics of composition is based on linear algebra, topology, and differential equations. The

design requirements and key properties are described using well-understood abstractions of physical systems such as continuous (usually multi-physics), lumped, or distributed parameter dynamic models and geometry. The two cyber platform layers, *network* and *service platforms*, comprise the digital hardware side of CPS and include the networks and computation platforms that interact with the physical components through sensors and actuators. Execution of software-defined applications on processors and transfer of data through communication links "translate" their abstract behavior into physical, real-time behavior. The behavior of these platforms is usually modeled by discrete event systems. The mathematics of composition is timed automata, hybrid automata, algebra, and queuing theory. The *application layer* comprises the software components with behavior expressed in logical time. The mathematics of composition is discrete, based on logic, combinatorics, and universal algebra.

In H-CPS where humans are indirectly involved in system operation (by selecting architectures, weighting optimization criteria, and making investment decisions), incentives and regulations are used to guide the investment of resources in such a way as to improve the composite properties of the CPS system both at the strategic and tactical levels. For example, for DoD Information Systems (not CPS), the DoDI 8510.01 Risk Management Framework governs the certification and accreditation process to maintain the information assurance posture during the system's life cycle. The requirement for compliance forces organizations to follow design practices that provide improved assurance against cyber-attacks. In H-CPS where human behavior is directly involved in the properties of the overall system dynamics (as operators of vehicles or consumers of contested resources), incentive engineering is used to modulate the decision loop of individual players (drivers, consumers) such that the overall system behavior converges toward a societal optimum. The mathematical foundation for incentive engineering is mechanism design (Williams 2008), a field in game theory that uses an engineering approach to find the game (i.e., the game defined by the specific mechanism), in which Nash equilibria will be close to a desired optimum.

As Fig. 18.1 illustrates, the abstraction layers are built on each other. They are associated with layer-specific architectures that utilize and provide services and interact via energy and information flows during operation. In CPS/H-CPS systems, architecture layering has the following implications on analyzing and providing resilience properties:

1. The layers are associated with technologies that have specific vulnerabilities and related attacks vectors. Figure 18.1 shows some examples for attacks and their implications. Understanding attacks and their potential implications requires the use of the appropriate abstractions and related theories.
2. Attacks and impacts are not isolated to a single layer. Cross-layer interactions can propagate impact: attacks on the physical layer can cause anomalies on cyber layers (e.g., blocking cooling mechanisms may lead to overheating of circuits that in turn may cause shutdown of services implemented by impacted processors). Similarly, cyber-attacks can have physical impact (e.g., integrity attacks on

sensor signals that may cause incorrect actuation on physical processes leading to potentially catastrophic system failures).

3. A more subtle but important implication of architecture layering is a new category of vulnerabilities caused by violations of assumption/guarantee relationships across layers. As an example, let's consider a safety critical physical layer property, stability. Control theory has developed methods (e.g., Lyapunov criteria (Bhatia and Szegő 2002)) and introduced architectural restriction (e.g., linear dynamics (Teschl 2012)) to make verification of stability solvable. However, multilayer implementation of controller dynamics that includes networking and computing introduces implementation side effects (such as time-varying delays in control loops, nonlinear effects of fixed-point arithmetic, coefficient quantization, and signal quantization) that violate key assumptions that were used in stability analysis on the physical layer. The result is that the integrated system may lose stability because these side effects can be easily manipulated by relatively simple timing attacks in the "cyber" layers. In conclusion, a layered system that operates correctly in each layer can be destroyed by attacking the frequently undocumented cross-layer assumptions that are vital for guaranteeing critical safety properties.

Designing large and heterogeneous CPS/H-CPS systems is a difficult problem, and one of the most difficult aspects is providing resilience. Beyond the problems mentioned above, what makes achieving resilience particularly hard is that cyber-attacks may be coordinated on different layers and combined with physical attacks to achieve maximum damage with the smallest investment. In this chapter we will examine two approaches to resilience: *passive* and *active resilience*.

Passive resilience refers to properties that are inherently robust against classes of uncertainties, "resilient-by-construction". Examples of passive resilience include decreasing safety and security risk by increasing safety margins, hardening access control policies, or using longer encryption keys.

Active resilience refers to the ability of a system to respond to attacks that implies some form of *reflexive* or *deliberative control*.

1. Reflexive methods employ a monitor-response scheme: monitors detect anomalies (or signatures) from a predefined library and rapidly associate those with predefined control actions that are expected to resolve or mitigate the problem. Reflexive methods are widely used in many safety critical systems, such as flight control. ClearView (Perkins et al.) uses this method to automatically patch errors in compiled binaries. It first observes normal, error-free executions to derive invariants that are true in all observed executions. The monitors respond to a program crash by finding violated invariants and trigger an action that reenforce the violated invariants by patches. Reflective methods are effective when the required response resolution is low, so the number of anomalies (invariant violations) and the associated actions are limited and can be well separated.
2. Deliberative methods expand anomaly detection with root cause analysis, isolation, recovery planning, and recovery actions using detailed information about the structure and expected behavior of the system. The approach involves

reasoning, and its implementation requires deeper integration with the overall system architecture (the deliberative controllers need to be made resilient as well).

After reviewing examples for passive and active resilience, we will discuss illustrative examples followed by simulation-based evaluation of resilience in CPS. We will conclude the chapter with presenting a simulation tool kit for resilience analysis.

2 Methods for Implementing Passive Resilience

Passive resilience of systems can be improved by increasing robustness against impacts of cyber-attacks. We demonstrate this concept using two very different examples: (a) designing attack-resilient sensor allocation and (b) improving resilience against cross-layer attacks by improving robustness via architectural constraints.

2.1 Attack-Resilient Sensor Allocation

The ability to control any system hinges on having accurate information about its evolving state, obtained through persistent system monitoring. In many applications, such as transportation networks, the system to be monitored can extend over a large area, with many possible points of observation. Although these areas can be very large, the number of sensors deployed is always limited by financial and/or technological constraints. Consequently, we faced the problem of finding locations for placing a limited number of sensors so as to minimize our posterior uncertainty about the quantities being monitored. This problem is further complicated by the presence of strategic adversaries, who may disable some of the deployed sensors in order to impair the operator's ability to make predictions.

Based on a Gaussian-process-based regression model, we formulated the problem of attack-resilient sensor placement as the problem of selecting a subset from a set of possible observations, with the goal of minimizing the posterior variance of predictions (Laszka et al. 2015). In order to illustrate the approach, we consider a transportation network. We assume that a set of possible traffic flow sensor locations is given, and a designer can place at most N sensors at some locations. The designer uses the observations of the deployed sensors to predict a value using Gaussian process-based regression. For example, traffic measurements obtained from induction-loop sensors can be used to predict traffic at unobserved locations or in a future time. Given the sensor measurements, the predicted value is a random variable which follows a Gaussian distribution. Next, we consider a model of denial-of-service attacks against sensors. We assume that the attacker is resource-bounded, so she can remove at most K of the sensors deployed by the designer. In

practice, removing a sensor can model all forms of denial-of-service type attacks, such as physical destruction, wireless jamming, or battery exhaustion. We also assume that the attacker is malicious in the sense that it will select a set of sensors to remove that will minimize the accuracy of prediction. We quantify the accuracy using the posterior variance; the lower the variance, the more accurate the prediction is. Then, the resilient sensor location selection problem can be formulated as an attacker-defender game where the designer first selects a set of sensor locations to minimize variance, and then the attacker removes a set of sensors to maximize the variance. We have shown that both finding an optimal resilient subset and finding an optimal attack against a given subset are NP-hard problems. Since both the design and the attack problems are computationally complex, we proposed efficient heuristic algorithms for solving them and presented theoretical approximability results. Finally, we have shown by using numerical results based on real-world datasets (Laszka et al. 2015) that the proposed algorithms perform well in practice.

2.2 Passivity-Based Architecture

As we mentioned earlier, an important cyber-attack category impacting networked control systems may exploit hidden and undocumented interdependences between the physical layer and cyber layers (such as networking and service platforms). For example, the physical stability of distributed CPS is usually verified using continuous time dynamic models of the plant and the ideal controllers. The networked, digital implementation of the controllers brings in implementation side effects in the timing properties; therefore stability needs to be reverified. However, sensitivity of the system-level stability to the timing properties of the controllers creates an exploitable vulnerability: by injecting time-varying delays, jitter, bandwidth changes, and packet drops into feedback loops, an attacker can destabilize the physical systems.

Similarly, networked multi-agent systems (such as swarms of interacting UAVs), like all large-scale distributed systems, have many entry points for malicious attacks or intrusions. If one or more of the agents are compromised in a security breach, it is crucial for the networked system to continue operating with minimal degradation in performance, and the success of the global objective should be assured. To achieve this, it is necessary for the cooperative control algorithms to be designed in such a way that they can withstand the compromise of a subset of the nodes and network links and still guarantee some notion of correct behavior at some decreased level of performance.

We have studied this problem in various application contexts and developed an approach that is based on improving decoupling of the stability property on the physical from implementation side effects of the cyber layers by an architectural constraint, called Passivity-Based Architecture (PBA) (Sztipanovits et al. 2012). Passivity is a fundamental concept in system science. It quantifies the property of a system to dissipate energy that is either being supplied to it through external inputs

or that is stored internally. In classical circuit theory, a dynamic system (e.g., a filter) implemented by using passive L, R, and C electronic components is passive; therefore it cannot ever become instable. While transitioning from analog filter technology to digital, a theory was developed for transforming passive continuous dynamics to digital implementation that preserves the property of passivity by maintaining passivity-related constraints among the digital representations of physical variables (Fettweis 1986). Passive systems (whether they are physical, digital, or their combination) have a unique property that when connected in either a parallel or negative feedback manner, the overall system remains also passive. Accordingly, large-scale and open systems organized as interconnections of passive structures (e.g., physical plants and digitally implemented networked controllers) will remain also passive; therefore they preserve stability independently from the timing and other uncertainties caused by the networks and computations in the digital controllers (Koutsoukos et al. 2012; Kottenstette et al. 2013). Further, the interconnection topology can be dynamically adapted or reconfigured in order to improve system availability. PBA, by decoupling physical stability property of CPS/H-CPS from timing uncertainties in the cyber layer, eliminates a significant category of exploitable vulnerabilities. While the progress in PBA design is significant, novel scalable analysis and design methods are necessary for understanding and eventually controlling the impact of dynamic topologies on stability and robustness (LeBlanc et al. 2013; LeBlanc and Koutsoukos 2013).

We briefly illustrate the approach with an example for safe semiautonomous driving (Dai and Koutsoukos 2016). An autonomous vehicle can be described as a CPS where the dynamics of the vehicle interact with the control software. Consider a scenario where a host vehicle is following autonomously a lead vehicle on a curved road. The control system must ensure that the vehicle behaves in a stable and safe manner avoiding collisions and skidding. The control system consists of an adaptive cruise control (ACC) system which controls the speed of the vehicle and a lane-keeping control (LKC) system which controls the angle of the steering wheel in order to maintain a desired position on the road. The overall system behavior emerges by the composition of the vehicle dynamics, the control software and network, and disturbances such as wind and the slope of the road. PBA is used for control design to ensure stability and safety in the presence of nonlinear dynamics, disturbances as well as uncertainties created by cyber-attacks, such as time-varying delay, jitter, and packet loss (Koutsoukos et al. 2012; Kottenstette et al. 2013).

Characterizing robust CPS behavior requires new notions of uncertainty that crosscut the physical, computation, and communication layers. These diverse notions of uncertainty require novel assessment metrics that capture different views of system stability and robustness. Extensions of passivity to discrete event and hybrid systems are required in order to characterize robust composition of software and physical components. Imposing such restrictions on the component dynamics will enable compositional modeling and reasoning for computing, sensing, and acting on the physical world.

Compositional properties such as passivity offer advantages for designing robust large-scale systems, but stability and robustness cannot be studied in isolation from

other design concerns. Understanding fundamental trade-offs between stability and robustness, system performance, safety and security, and properties of the physical platform is emerging research endeavors in CPS. They will help, for example, understanding how composition based on passivity affects performance, platform properties, and other design concerns.

Attack-resilient sensor allocation and PBA are examples for passive resilience against specific categories of cyber-attacks. Both methods improve robustness of selected properties of the CPS by introducing additional architectural constraints (optimality of sensor allocation and passivity) in the design process resulting decreased vulnerability. This is in sharp contrast with active resilience that requires the extension of the system architecture with new functional components dedicated to resilience.

3 Methods for Implementing Active Resilience

Design of systems with active resilience starts with deciding if reflexive or deliberative strategy is appropriate. In CPS/H-CPS a common challenge is that cross-layer propagation of anomalies (see Fig. 18.1) creates signatures that are nondeterministic and not specific to the underlying attack. This restricts the use of reflexive methods to trigger coarse reactions, for instance, switching from blacklisting to whitelisting under specific attack conditions. Deliberative methods require models, automated processes, and tools that provide a systematic, model-based reasoning method to intrusion detection, isolation, recovery planning, and plan execution in operation time. One possible approach to this has been developed and evaluated by our team (Pradhan et al. 2016), which is based on symbolic representation of the configuration space and automated, constraint-based reasoning.

The approach starts with the modeling and analysis of the system architecture; for example, applications that are built from components and deployed on a network of computing nodes providing platform services, interconnected through a network. Components are stateful objects that interact with each other through messages (via the network), and they implement a well-defined model of computation (e.g., single-threaded, event- or time-triggered actor model). The application(s) built from such distributed components provide some well-defined functionality that can be explicitly represented in an application model. The application model enumerates all the objectives ("functions") of the application and shows how the individual functions are linked to specific software components. This linkage may have cardinality, e.g., a function may require multiple copies of a component. A component may also require platform resources (e.g., hardware resources, like memory or file space; or specific hardware devices, like sensors or actuators). Thus the application model represents what functions the application provides, how these are mapped to (one or more) software components, and what platform resources those components require.

A second model, called the platform model, represents the hardware platform: its computing nodes and resources, the connectivity, and the capabilities and capacities

of those. Once these models are created, a calculation can be performed that represents a deployment (i.e., a mapping or assignment) of the software components to the platform. This calculation can be done by a heuristic search algorithm, but it is much simpler to implement using another approach based on constraint programming. Both the application model and the platform model can be expressed in the form of mixed-integer linear constraints, where decision variables indicate what component gets mapped to what resources. Solving this constraint programming problem is simple (using modern solver technology, like Z3 (de Moura and Bjørner 2008)), and the solution is the deployment of components on the platform such that the application functions are realized. Typically the solution is not unique, as multiple deployments are feasible – providing resilience for the implemented functions. If anomalies develop in a system (e.g., a component fails due to a local breach, or a network link gets jammed), this fact can be represented as an additional constraint that "negates" the failed resource. Now if the solver is run again, a new solution will be computed (if there is one) – which is a new deployment configuration of the system that (1) satisfies the application's objectives and (2) does not use compromised or failed node. Then the system should be reconfigured (the application must be redeployed) according to this new deployment plan. This process can be repeated upon new anomalies detected; while there is a viable configuration, the solver will find it.

In summary, the above approach is based on (1) modeling the application (its functions and its architecture and the component's resource requirements), (2) modeling the platform (processing and communication capabilities of the resources), (3) translating the models into constraint form and representing the deployment configuration in the form of decision variables, and (4) generating a solution that represents a possible deployment of the application on the platform. In other words, we model the configuration space of the system, then use constraint-based pruning to find solutions that yield an acceptable deployment. This method can be obviously extended to various componentized and layered architectures, which may include security requirements in information flows, or extended with optimization of the deployments.

As this example illustrates, implementation of active resilience requires additional management functions that are outside of the system's functional architecture. It is important to recognize that these resilience management functions need to be resilient as well; therefore they need to be organically integrated into the overall system architecture as part of the control hierarchy.

4 Simulation-Based Evaluation of Resilience

Heterogeneity and the richness of interactions among system components are the key barriers for evaluating resilience in CPS/H-CPS. For some approaches, such as passivity-based design, formal analytical methods are available for proving passive resilience of *selected properties* (stability) for whole attack classes (Koutsoukos

et al. 2012), but this is rather an exception. The remaining option is simulation-based evaluation of resilience. In general, what makes evaluation of resilience challenging is its context dependence: resilience makes sense only for some *well-defined property* and against some *well-defined attack classes.* In addition, resilience is a *system-level property* which is usually non-compositional – except in rare cases such as passivity-based design, where compositionality for stability has been established (Kottenstette et al. 2013). In this section we establish requirements for simulation-based studies using a simple example.

Earlier we introduced an approach for selecting the locations of networked sensor devices so that their placement is resilient to denial-of- service type attacks that aim to degrade prediction accuracy for a networked physical process (e.g., traffic flow or flows in water distribution networks) (Laszka et al. 2015). The approach was built on a Gaussian process-based regression model (Rasmussen and Williams 2006), whose parameters are estimated using the simulation test bed prior to deployment. Since finding an optimal placement is an NP-hard problem, we developed a heuristic algorithm and evaluated it using simulations. The simulation-based analysis required the following problem setup:

1. *Sensor and Prediction Model*: We assume that a set V of possible sensor locations is given, and a designer can place at most N sensors at a set $S \subset V$ of locations. The designer uses the observations of the deployed sensors to predict a value using Gaussian process-based regression. For example, traffic measurements obtained from induction-loop sensors can be used to predict traffic situation at unobserved locations or in the future.
2. *Attacker Model*: Next, we introduce our model of denial-of-service attacks against the sensor network. We assume that the attacker is resource-bounded, so it can remove at most K of the sensors deployed by the designer. In practice, removing a sensor can model all forms of denial-of-service type attacks, such as physical destruction, wireless jamming, or battery exhaustion. We also assume that the attacker is malicious in the sense that it will select a set of sensors to remove that will minimize the accuracy of flow prediction.
3. *Problem Formulation*: We quantify the accuracy of predicting traffic flow using the posterior variance: the lower the variance, the more accurate the prediction is. Then, the resilient sensor location selection problem is defined as attacker-defender game.

While all mathematical details and the detailed results are omitted here (for details, see Laszka et al. (2015)), the example shows some interesting requirements for simulation-based studies:

1. *Modeling and simulation of CPS*. The simulation platform needs to support multi-modeling including realistic models of cyber and physical components and their interactions, as well as operational scenarios that can be used for evaluations of cybersecurity risks and mitigation measures.
2. *Attack modeling*. Attack modeling requires intricate connections and deeper insights in the cyber and physical infrastructure. As Fig. 18.1 shows, attacks

can be deployed in different system layers, and their effects can propagate up and down in the implementation hierarchy. The basic tenet of a simulation platform for resilience studies is to build and evaluate executable models. Thus we cannot sidestep important details about the execution of a cyber-attack. Such details in network attacks include the formation of network packets, routing information, or OS-level behavior of individual nodes. This requirement is in conflict with goals of providing a truly generic, technology neutral, and executable adversarial language that captures abstract high-level domain concepts only, since simulation of cyber-attacks relies on the details of the implementation infrastructure. The consequences of this requirement on the simulation infrastructure are significant. The declared goal of the resilience study (the selected system properties and attack types) strongly influences the level of abstractions to be used both in the attack models and system models. This makes flexibility and rapid reconfiguration of simulations a fundamental requirement. In some analysis, the required level of simulation fidelity may not be feasible; therefore applying Hardware-in-the-Loop (HIL) and System-in-the-Loop (SIL) solutions is the only possibility.

3. *Attacker-defender games*. Resilience studies are frequently scenario-based where real or virtual attackers and defenders play against each other. In these setups interactive access to simulated models are essential since attack scenarios unfold dynamically driven by interactions with players.
4. *Information management*. Simulation-based evaluations require information in the forms of models, software, and data that are interdependent, deeply versioned and linked to an extensive IT infrastructure. Services for managing this information are a frequently overlooked aspect of simulation-based evaluations.

In summary, simulation-based evaluation of CPS resilience against cyber-attacks involves multiple, heterogeneous, interacting modeling domains. While established modeling domains usually have model building and simulation tools, their integration into a multi-model simulation is time-consuming, labor-intensive, and error-prone task. This means that computational studies cannot be completed rapidly, and the process does not provide timely answers to the planners, operators, and policy makers. Furthermore, CPS/H-CPS behavior has to be tested in a number of scenarios and situations that usually involve large number of simulation runs so as to cover the space of possibilities. Designing and efficiently deploying such computational "experiments" by utilizing multi-domain tools for integrated CPS are a significant challenge.

5 Simulation Integration Architecture and Tool Kit

Multi-model simulation test beds require that domain-specific models and simulators are integrated into heterogeneous system models, and the simulations are executed on scalable computing platforms. This section will describe the Cyber-Physical

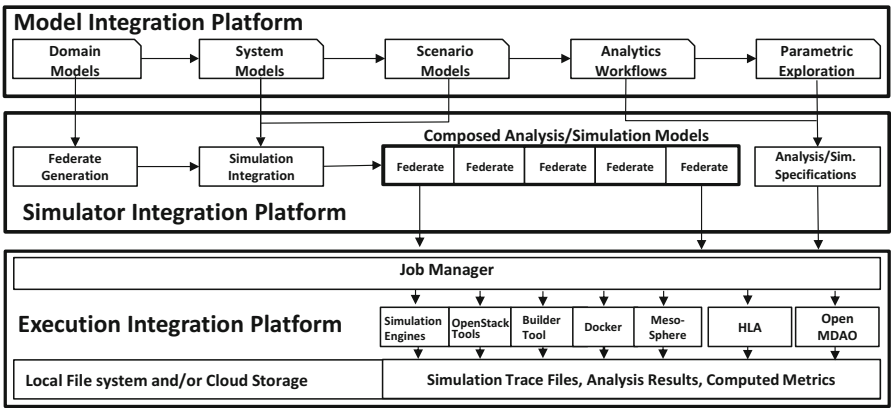

Fig. 18.2 Integration platforms for heterogeneous modeling and simulation

Systems Wind Tunnel (CPSWT), a simulation integration architecture and tool kit, which is the result of a decade-long research effort (Hemingway et al. 2012) and now used as foundation for several distributed simulators such as Vanderbilt's SURE Test Bed for resilience studies and the NIST CPS test bed called Universal CPS Environment for Federation (UCEF) (Roth et al. 2017).

The key idea in creating the CPSWT tool kit was to introduce three horizontal integration platforms for model integration, simulator integration, and execution integration (see Fig. 18.2).

5.1 Model Integration Platform

Model integration is required for expressing interactions across modeling domains – a basic need for multi-model simulations. The primary challenge to be addressed has been the semantic heterogeneity of domain models and the different model types required for the specification of a simulation experiment. The fact that domain models in CPS subdomains are provided by different simulation tools that evolve more or less independently further adds to the model integration challenge.

Resilience studies require the introduction of a number of model types:

1. *Domain models* (*DM*) that are specified in terms of the domain-specific modeling languages of various simulation tools integrated in the CPSWT tool kit. Examples are Simulink/Stateflow (dynamics), Simpower (Simscape Power Systems – Simulink), Colored Petri Net (CPN), ns-3 (network simulator), OMNeT++ (network simulator), DEVS (discrete event simulator), FMU-CS (Functional Mock-up Interface Co-Simulation Units), Modelica (multi-physical dynamics), SUMO (traffic flow simulator), and Delta3D (physics).

2. *Model Integration Language* (*MIL*) that captures the integration models for the overall integrated CPS/H- CPS to be simulated. Integration models include modeling constructs imported from individual domain models (semantic interfaces), as well as modeling constructs for representing interactions among the domain models.
3. *Scenario Models* (*SEM*) representing synchronization points, event conditions, and timed events driving a simulation run. Scenario modeling is part of the CPSWT language suite and essential for describing experiment scenarios.

Beyond these fundamental model types, there are several sub-languages for representing deployment and execution management information required for creating and running experiments.

To address heterogeneity, the CPSWT tool kit departed from the most frequently used approach to address heterogeneity: the development or adoption of a very broad and necessarily complex modeling language designed for covering all relevant views of multi-physics and cyber domains. Instead, emphasis was placed on the development of a model integration language –MIL – with constructs limited to modeling the interactions among different constituent models. The key to this approach is the concept of *semantic interface*. When integrating a new model type with its associated simulator, the semantic interface captures the sub-language exposed by the domain modeling language (e.g., Simulink) for integration. These exported sub-languages are complemented by integration constructs enabling the specification of integration models. Since CPSWT uses the High Level Architecture (HLA) standard for simulation engine integration, it is a natural choice that the MIL incorporates HLA interaction models and distributed object model (HLA Standard). It means that the interaction semantics among domain models is defined by HLA (and implemented by the HLA Run-Time Infrastructure – HLA-RTI).

5.2 Simulator Integration Platform

The role of the simulation integration platform is to establish interactions across the concurrently running simulators by coordinating time advancement and routing messages among the simulators. Since CPSWT uses HLA as the backbone for the simulation integration platform, a large range of services for configuring, running, and managing large-scale distributed simulations comes with the HLA-RTI that exists both open-source and COTS implementation. (Since CPSWT is an open-source project, it uses the Portico open-source RTI implementation (PORTICO).) Since the HLA standard documents the simulation services extensively (HLA Standard), it is not included in this discussion.

It is worth mentioning that there are still much discussion about the complexity and scalability of HLA implementations. In our experience, distributed time and object management are inherently complex problems, particularly when logical-time simulation needs to be synchronized with the real-time execution of HIL/SIL

components. Competing solutions such as Functional Mock-up Interface (FMI) (http://fmi-standard.org) or the Distributed Interactive Simulation (DIS) (Davis 1995) framework are simpler but provide limited services for integrating multi-model simulations on distributed computing platforms. Even using HLA, the integration of multi-model simulations by using directly the APIs of the HLA Run-Time Infrastructure (HLA-RTI) and writing integration code is an error-prone and time-consuming task. To alleviate this challenge, the CPSWT tool kit includes a suite of model-based generators that automatically integrate the executable federation from the integration models.

5.3 *Execution Integration Platform*

The CPSWT execution integration platform incorporates a range of tools for deploying and managing distributed simulations on desktops, on servers, and on cloud platforms. Availability of relatively low-cost computing resources offered a major simplification of running large-scale simulations. However it brought to the forefront the needs for structuring large simulations as federation of federations. Decomposing of large-scale models based on intensity of interactions and required time resolution have much need for automation.

5.4 *The Integrated CPSWT Tool Kit*

The horizontal integration platforms described above are supported by services and tools. The functional architecture of the CPSWT tool kit is shown in Fig. 18.3.

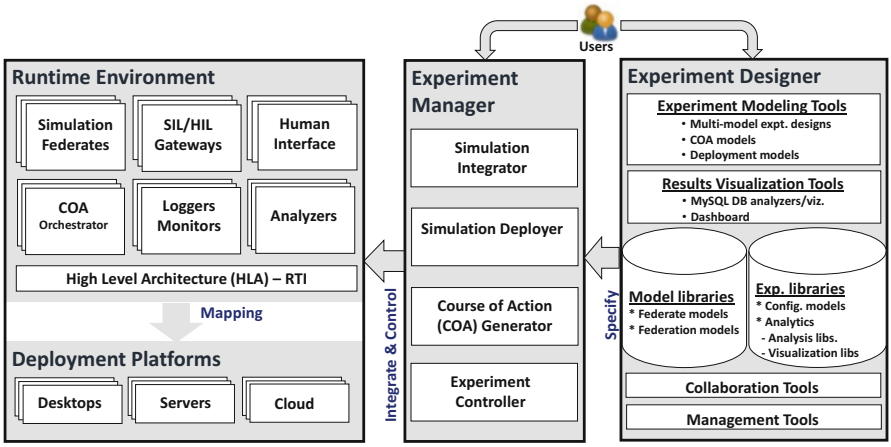

Fig. 18.3 CPSWT tool kit

Components of the runtime environment and the experiment manager belong primarily to the simulation integration layer and execution integration layer on Fig. 18.2, while the experiment designer incorporates tools supporting the model integration layer. The distributed runtime environment includes various domain-specific simulation engines wrapped as HLA federates, the SIL/HIL gateways, a range of HLA service functions. It also includes deployment platform alternatives as mentioned before. The overall tool kit incorporates two other functional components that were not mentioned before, the experiment manager and experiment designer.

The *experiment manager* component incorporates facilities for building, configuring, deploying, and controlling experiments. These components are model-based; their actions are the result of interpreting experiment, scenario, and deployment models created by the experiment designer tool suite. The simulation integrator and simulation deployer components enable web-based modeling and configuration of the multi-model integrated simulations according to their deployment models. The *experiment controller* presents a front-end interface for experimentation where user can selectively deploy experiments, configure analysis, control running simulations, and analyze simulation results. The *experiment designer* tool suite incorporates a web-based modeling tool, WebGME (Maroti et al. 2014), for graphically building integration models, repositories for models and experiments, visualizer tools, and services that make the overall system accessible to users, manage user groups and communities, and enable collaborative modeling and experimentation.

A fundamental aspect of CPSWT is that it supports three distinct levels of users. At the top are the experiment designers and analysts who perform studies and evaluations by designing experimental scenarios on a configured system and running experiments using those scenarios. The second level of users include the system modelers and integrators who have the knowledge of the system-of-system architecture of the overall system that is being simulated in an integrated manner. These users can create models and artifacts needed for new studies. Lastly, the third level of users are the infrastructure developers and maintainers who have deep technical knowledge to incorporate new simulation tools in the infrastructure, to build and enable parameters for the individual models and experiments, and to develop and maintain the build systems and tooling infrastructure.

6 Example: Resilience Analysis for Transportation Networks

In this section we demonstrate simulation-based resilience analysis using a transportation network example.

Optimization of traffic lights in urban transportation networks is a known challenge in modern traffic control systems (Helbing and Siegmeier 2007). A common goal is to minimize congestion. Much prior work has now demonstrated that allowing for dynamic real-time control (as compared to fixed-time control) can

significantly improve performance of optimized traffic light controllers (Fouladvand et al. 2004; Gershenson and Rosenblueth 2012). A number of methods to perform optimization of closed-loop control systems have been proposed, where sensor measurements are used to dynamically adjust the timing of traffic light green-red cycles (Mikami and Kakazu 1994; Royani et al. 2010).

Although adaptive, state-aware strategies can offer gains in traffic control efficiency, they expose an attack surface that can be exploited to increase congestion, even to bring down a traffic network. For example, a common sort of adaptive control logic involves system state captured by vehicle queue lengths in each direction, with light switching between red and green as a function of relative queue lengths. While such state-aware switching can significantly increase efficiency, it also exposes a vulnerability of controllers to attacks on sensors from which queue length information is derived. An additional consideration which is crucial in modern complex traffic networks is that traffic lights on the network are often designed by multiple actors (e.g., municipalities).

We demonstrate how the SURE simulation-based traffic control test bed[1] (created using the CPSWT tool kit) can be successfully used to systematically and efficiently explore these challenges in multi-intersection closed-loop traffic light control, where (1) traffic light controllers take into account relative queue lengths to determine red-green state of the traffic lights at an intersection, (2) controllers for all lights must be designed to work jointly so as to optimize overall traffic network performance, (3) sensors feeding data into the controllers are vulnerable to denial-of-service attacks, and (4) intersections can be partitioned among a set of players, with own goals pertaining to congestion within their local municipal region, which are in general misaligned with global interests of the entire traffic network. The SURE environment integrates the SUMO traffic simulator,[2] OMNET++[3] network simulator, and Simulink for modeling controllers. Details of the theoretical approach can be found in (Lou and Vorobeychik 2016).

6.1 Traffic Model

The transportation network domain supports the use of real-world, detailed – street and lane-level – traffic maps. These maps can be imported from OpenStreetMap (OpenStreetMap Contributors 2017) with ready-to-use street network, intersection rules (i.e., permitted lane-to-lane graphs and priorities). The current test bed scenarios are developed on the Vanderbilt University campus network (Fig. 18.4a). Once the road map is imported, the traffic demand patterns are to be modeled. We have built a graphical demand modeling tool that can be used to easily to create different

[1] http://cps-vo.org/group/sos/sure

[2] sumo.dlr.de/index.html

[3] https://omnetpp.org/

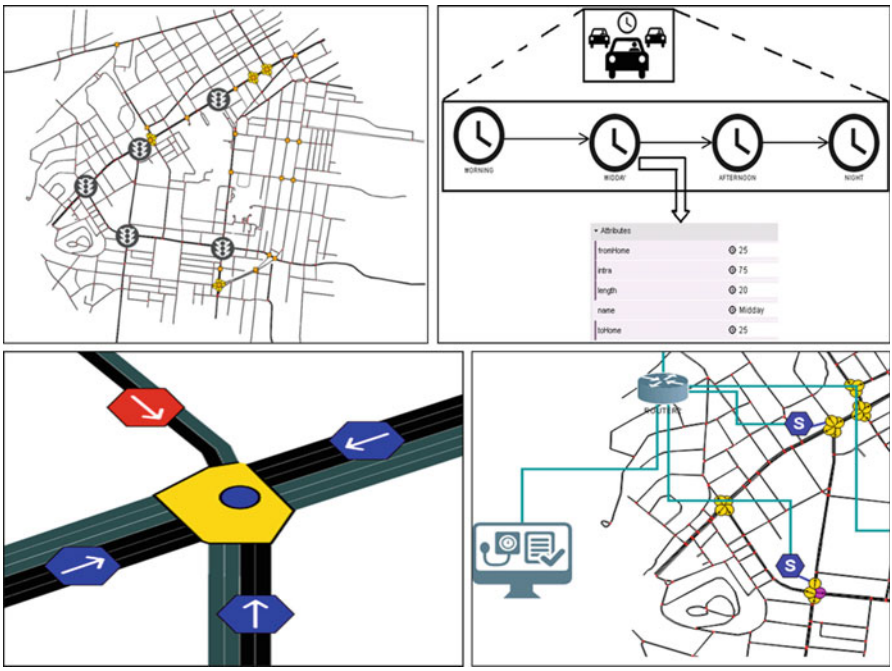

Fig. 18.4 Traffic control models. (**a**) Vanderbilt campus street network. (**b**) Stochastic traffic demand model. (**c**) Intersection model. (**d**) Communication network model

stochastic traffic demand patterns. Figure 18.4b shows an example of demand modeling in the WebGME modeling environment. The model allows for arbitrary number sequential demand phases, each with a time interval and weights for traffic heading into (*from Home*), from (*to Home*), and within (*intra*) the network.

6.2 Signalized Intersection Models

The design interface of SURE uses hierarchical decomposition and multiple abstraction levels to work at the map level, for capturing intersection-level parameters or fine-tuning attack strategies. Figure 18.4c shows the intersection-level model. Each signalized intersection is equipped with a full set of sensors on all incoming lanes capable for measuring the current queue length (number of cars) waiting on the corresponding lanes to cross the intersection. Also, each intersection is equipped with a traffic light running a predefined program. In each phase of the traffic light program, one or more incoming lanes are being emptied – this information is available for the controller algorithm. The controller logic – using the queue length measurement and driven by a few design parameters – can advance the phase of the traffic light, thus modulating the periodicity and the timing of the program.

6.3 Resilient Monitoring and Control Model

Formally, a feedback traffic light controller has a predefined phase sequence (p_0,..., p_n). For each phase p_i, m_i is the minimum interval, M_i is the maximal interval, q_i is the average queue length of the lanes related to the ith phase, and θ_i is the threshold on the queue length of lanes blocked in the ith phase. The controller parameters we need to tune are the thresholds $\Theta = (\Theta_0,...,\Theta_m)$, where $\Theta_i = (\theta_0,...\theta_{ni})$ are the thresholds of the ith intersection. The global objective is to maximize average speed, $s(\Theta)$, over the entire traffic network.

6.4 Communication Network Model

Apart from the road network and traffic demands, one also needs to model the associated cyber communication network for the traffic sensors, controllers, and communication network elements such as routers. In SURE, this can be done on the same web-based interface, by instantiating the appropriate elements from the part browser and creating the necessary communication links between them. Figure 18.4d shows an example of a cyber communication network model. Here, the large computerlike icon is for a centralized control center, connected to sensors – depicted as hexagons with the letter "S" – using intermediate routers and communication links.

6.5 Attacker-Defender Games

The current attack model allows for arbitrary denial-of-service (DoS) attacks to be deployed on the map – targeting the sensors or other elements of the cyber infrastructure. The task of defender (*blue*) team is to tune the controller parameters (interval and queue thresholds), while the attacker (red) team selects one or more – but limited number – of attack targets. Each round is evaluated by executing the C2WT-based simulation from the web interface. The results – i.e., average travel time – are shown once the simulation finished. A typical experiment workflow is shown in Fig. 18.5.

The high-level results (average speed) of various configurations on the Vanderbilt University campus network with five selected intersections are shown in Fig. 18.4.

We can make two important observations: (1) controller parameters which are jointly optimized can result in a significant increase in average speed and (2) explicitly building resilience into a controller can improve its resilience against attacks, while maintaining high performance when no attacks are present.

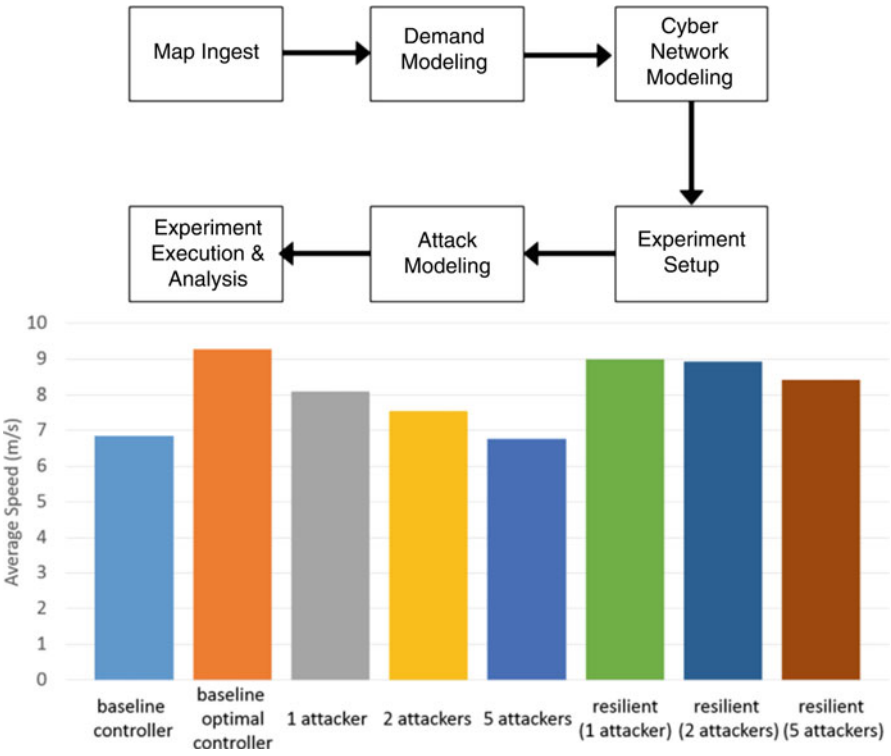

Fig. 18.5 SURE experiments and results. (**a**) SURE experiment workflow. (**b**) Performance of optimized and baseline controllers with and without attacks

7 Future Developments and Promising Research Directions

Engineering resilience into cyber-physical systems has made significant advances in recent years, but it is clearly far from a well-defined engineering discipline. Systems are still often unprepared for unforeseen situations, and they are relatively weak compared to, for instance, biological systems. However, there are a few interesting research directions that can and should be pursued. Resilience is a system-level property, and it is emerging from (the design of) a system, not from an individual component of a system. Given this, resilient system design that focuses on architecting and engineering issues related to resilience is always going to be in focus. New engineering principles and architectures are sought that could be formally analyzed to prove that the system functions can be recovered, even under adverse conditions.

Formal or simulation-based analysis of resilience is desired, because exhaustive testing on real-life systems is frequently not feasible, especially in cyber-physical systems. Classic techniques based on simple redundancy need to be replaced by intelligent management of resources that achieves the same level of resilience at a

lower cost. Classical redundancy is feasible but typically very expensive; hence more cost-effective solutions are needed. Today, these solutions are often centralized, meaning that they could have a single point of failure. In the future, such mechanisms need to become more decentralized and adaptive. Along these lines, collaborative, consensus-based approaches to resilience are of major interest. Much broader categories of anomalies need to be defined that a system should be resilient to. For example, temporary performance degradations or unintended side channels need to be detected and mitigated, but the list is probably much longer. The physical aspects of CPS is another example that could potentially provide novel uses cases for simulation-based study of resilience – e.g., can a system recover from the impact of physical attacks through software actions directed to reallocating functionalities among physical components?

There is a well-recognized gap between theoretical results for improving resilience and their experimentally validated applicability. The primary problem is that opportunities for doing experimental work on resilience in real systems are highly limited, and simulation-based experiments require tools that are expensive to use and models that are expensive to develop. To narrow this gap to enable experimental research on resilient system design using integrated high-fidelity simulations is an important goal of our work. The urgency for progress is further increased by the rapid expansion of data-driven methods that use machine learning for developing both reflexive and deliberative control methods for active resilience.

Acknowledgment This work was supported in part by FORCES (Foundations Of Resilient CybEr-physical Systems), which receives support from the National Science Foundation (NSF award numbers CNS-1238959, CNS-1238962, CNS1239054, CNS-1239166), by the Air Force Research Laboratory under award FA8750-14-2-0180, and by National Institute of Standards and Technology.

References

Bhatia, N. P., & Szegő, G. P. (2002). *Stability theory of dynamical systems*. Springer-Verlag Berlin Heidelberg. ISBN 978-3-540-42748-3.

Dai, S., & Koutsoukos, X. (2016). Safety analysis of automotive control systems using multi-modal port-Hamiltonian systems. In *19th international conference on hybrid systems: Computation and control* (pp. 105–114) LNCS Vol 10012, Springer.

Davis, P. K. (1995). Distributed interactive simulation in the evolution of DoD warfare modeling and simulation. *Proceedings of the IEEE, 83*(8).

de Moura, L., & Bjørner, N. (2008). *Z3: An efficient SMT solver*. In Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29–April 6, 2008. Proceedings, volume 4963 of Lecture Notes in Computer Science (pp. 337–340). Springer.

Evans, P., & Annunziata, M. (2012, November 26). *Industrial internet: Pushing the boundaries of minds and machines*. GE.

Fettweis, A. (1986). Wave digital filters: Theory and practice. *Proceeding of the IEEE, 74*(2), 270–327.

Fouladvand, M. E., Shaebani, M. R., & Sadjadi, Z. (2004). Intelligent controlling simulation of traffic flow in a small city network. *Journal of the Physical Society of Japan, 73*(11), 3209–3214.

Gershenson, C., & Rosenblueth, D. A. (2012). Self-organizing traffic lights at multiple-street intersections. *Complexity, 17*(4), 23–39.

Helbing, D., Siegmeier, J., & ?ammer, S. L. (2007). Self-organized network flows. *NHM, 2*(2), 193–210.

Hemingway, G., Neema, H., Nine, H., & Sztipanovits, J. (2012, February). Gabor Karsai: Rapid synthesis of high-level architecture-based heterogeneous simulation: A model-based integration approach, simulation. *Transactions of the Society for Modeling and Simulation International, 88*(2), 217–232.

HLA standard. IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) – framework and rules ieeexplore.ieee.org/servlet/opac?punumber=7179

http://fmi-standard.org

Kagermann, H., Wahlster, W., & Helbig, J. (2013, April). Recommendations for implementing the strategic initiative INDUSTRIE 4.0, ACATECH report. *National Academy of Science and Engineering*.

Kottenstette, N., Hall, J., Koutsoukos, X., Sztipanovits, J., & Antsaklis, P. (2013, May). Design of networked control systems using passivity. *IEEE Transactions on Control Systems Technology, 21*(3), 649–665.

Koutsoukos, X., Kottenstette, N., Hall, J., Eyisi, E., LeBlanc, H., Porter, J., & Sztipanovits, J. (2012, December). A passivity approach for model-based compositional design of networked control systems. *ACM Transactions on Embedded Computing Systems, Special Issue on the Synthesis of Cyber-Physical Systems, 11*(4), 31.

Laszka, A., Vorobeychik, Y., & Koutsoukos, X. (2015). "Resilient observation selection in adversarial settings", *54th IEEE conference on decision and control (CDC'15)*, Osaka, Dec 15–18.

LeBlanc, H., & Koutsoukos, X. (2013, April). Algorithms for determining network robustness, *ACM International Conference on High Confidence Networked Systems (HiCoNS 2013)*. Philadelphia, PA, April 8–11, 2013. Heath LeBlanc and Xenofon Koutsoukos. "Resilient synchronization in robust networked multi-agent systems", Hybrid Systems: Computation and Control 2013 (HSCC 2013). Philadelphia, PA, 8–11.

LeBlanc, H., Zhang, H., Koutsoukos, X., & Sundaram, S. (2013, April). Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas on Communication, Special Issue on In-Network Computation: Exploring the Fundamental Limits, 31*(4), 766–781.

Lou, J., & Vorobeychik, Y. (2016). Decentralization and security in dynamic traffic light control. In *Proceedings of the ACM symposium and Bootcamp on the science of security* (pp. 90–92). IGI Global.

Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., & Aharon, D. (2015). *The Internet of things: Mapping the value beyond the hype*. McKinsey Global Institute.

Maroti, M., Kereskenyi, R., Kecskes, T., Volgyesi, P., & Ledeczi, A. (2014, June). Online collaborative environment for designing complex computational systems. In *The international conference on computational science*. Cairns: ICCS.

Mikami, S., & Kakazu, Y. (1994). Genetic reinforcement learning for co-operative traffic signal control. In *Proceedings of the first IEEE conference on evolutionary computation, 1994* (Vol. 1, pp. 223–228). IEEE World Congress on Computational Intelligence.

OpenStreetMap Contributors. (2017). Planet dump retrieved from https://planet.osm.org, https://www.openstreetmap.org

Perkins, J., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., Wong, W., Zibin, Y., Ernst, M., & Rinard, M. Automatically Patching Errors in Deployed Software. SOSP'09, October 11–14, Big Sky, Montana.

PORTICO. An open-source Run-Time Infrastructure for HLA-based distributed simulations – www.porticoproject.org

Pradhan, S., Dubey, A., Levendovszky, T., Kumar, P. S., Emfinger, W. A., Balasubramanian, D., Otte, W., & Karsai, G. (2016). Achieving resilience in distributed software systems via self-reconfiguration. *Journal of Systems and Software, 122*, 344–363.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. University Press Group Limited.

Roth, T., Song, E., Burns, M., Neema, H., Emfinger, W., & Szipanovits, J. (2017). "Cyber-physical system development environment for energy applications". In *ASME power and energy conference 2017*, Charlotte, USA, 06/2017.

Royani, T., Haddadnia, J., & Alipoor, M. (2010). Traffic signal control for isolated intersections based on fuzzy neural network and genetic algorithm. In *Proceedings of the 10th WSEAS international conference on signal processing, computational geometry and artificial vision, ser.ISCGAV'10* (pp. 87–91).

Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., Goodwine, B., Baras, J., & Wang, S. (2012). Toward a science of cyber-physical system integration. *Proceedings of the IEEE, Special Issue on Cyber-Physical Systems, 100*(1), 29–44.

Teschl, G. (2012). *Ordinary differential equations and dynamical systems*. American Mathematical Society. Providence, Rhode Island. ISBN 978-0-8218-8328-0.

Williams, S. (2008). *Communication in mechanism design*. Cambridge, UK, Cambridge University Press.