Integrating a Network Simulator with the High Level Architecture for the Co-Simulation of Cyber-Physical Systems

Thomas Roth, Cuong Nguyen, and Martin Burns
Smart Grid and Cyber-Physical Systems Program Office
National Institute of Standards and Technology
Gaithersburg, MD 20899
thomas.roth@nist.gov, cuong.nguyen@nist.gov, martin.burns@nist.gov

Himanshu Neema
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37212
himanshu.neema@vanderbilt.edu

Keywords:
cyber-physical systems, high level architecture, network simulation, smart grid

ABSTRACT: *Cyber-physical systems (CPS) use logical computation informed by measurements of the environment to actuate changes on the physical world. These systems have significant impact on people, and must be designed for resilience against fault and attack. However, due to their large scale, assurance of CPS trustworthiness is better suited to modeling and simulation than deployment of a real system. This paper describes an approach to integrate a network simulator with the High Level Architecture (HLA) to investigate the effects of different network conditions on CPS performance. Using this approach, an HLA interaction class can be configured to use network simulation rather than the default reliable HLA delivery mechanism. A technique similar to regions defined in HLA data distribution management is used to allow each federate to receive the same interactions at different logical time steps, as simulated by the network simulator. This is implemented in a piece of reusable code shared by federates that sits between the runtime infrastructure (RTI) and application code. The implementation can be used to create a test harness around the federates that represent the operation of a CPS to validate its behavior under unreliable network conditions.*

1. Introduction

Cyber-physical systems (CPS) are smart systems that include engineered interacting networks of both physical and computational components [1]. These systems have a high degree of complexity at numerous spatial and temporal scales and need highly networked communications to integrate the computational and physical components. The smart grid is an example of a CPS that is defined as the integration of digital computing and communication technologies and services with the power-delivery infrastructure. The smart grid is often referred to as a system of systems that enables the bi-directional flow of both communication and power. Because the smart grid integrates information communication technology (ICT) with the electrical grid, network communication is one very important system component.

Grid operations are becoming more complex with the widespread deployment of distributed energy resources (DER) and distributed sensors that provide intelligence at the grid edge [2]. DER are comprised of many different types of resources such as solar photovoltaic (PV), wind, battery, and electric vehicle (EV). Some of these resources such as PV and wind have volatility in energy production, and grid operators need robust communication capabilities to monitor and control them. Sensors distributed at the grid edge also have firm communication requirements to collect data, report, and fulfill their intended functions.

There are common standard-based communication protocols in use for grid operations. For substation automation, the two common protocols are IEC 61850 and Distributed Network Protocol 3 (DNP3). For DER communication, common protocols include IEEE 2030.5, DNP3, and SunSpec Modbus. Distributed sensors use protocols similar to both substation and DER. Beside these standardized protocols, there are manufacturer specific proprietary protocols. The choice of communication protocol for smart grid deployment depends on its performance capability, existing infrastructure, and the intended application. For example, if an operator plans to deploy an inverter for a PV installation, they need to consider what communication protocol their system can support to communicate with the inverter. If the operator plans to control DER, they will need a high-speed communication protocol for that application instead of a low bandwidth protocol that is only sufficient for monitoring alone. Other considerations may include whether the network connection between DER and the system is wired or wireless, and what cybersecurity mechanisms are required to protect the communication that still allow for the performance requirements.

Due to the complexity of the smart grid and its communication requirements, network simulation is essential and needs to represent the distributed nature of the evolving grid architecture. Sophisticated network simulation capabilities are needed to simulate the different communication protocols for the applications of interest to grid operators. Although such capabilities could be integrated into the implementation of grid simulators, it’s more intuitive to leverage the capabilities of existing network simulators. IEEE 1516-2010 High Level Architecture (HLA) is a standard for the co-simulation of distributed processes [3], such as the joint simulation of an electric grid and a network model. In HLA, the simulators that participate in a co-simulation are called federates, and the set of federates in the joint simulation are called a federation. The federates communicate and coordinate using software called the runtime infrastructure (RTI) that implements the common set of services described in the HLA federate interface specification [4]. An alternative to HLA is the Functional Mock-up Interface (FMI) standard often implemented by the developers of modeling tools [5]. Unlike HLA in which federates are independent processes in distributed system, FMI uses a master-slave architecture in which the master algorithm imports each simulator as a shared library and makes direct function calls into the slave code. The FMI standard for co-simulation prescribes the function definitions that each slave must implement to be interoperable with the master algorithm. This work uses HLA as its basis for co-simulation because it is more natural to consider a network simulator and a grid model as independent processes rather than sub-modules of one master program. In addition, networked co-simulation requires strict time management and distributed object management that are directly defined in HLA.

The remainder of this paper is organized as follows. Section 2 provides the motivation for the need of network simulation in the smart grid and lists the high level requirements that must be satisfied for network simulation to be meaningful in this context. Section 3 gives an overview of related work in the area, and Section 4 describes the specific approach to network simulation proposed by this work. The paper is then concluded in Section 5.

2. Motivation and Approach

For holistic system of systems evaluations, CPS require complex co-simulations including an integrated simulation of the cyber communication network as well as hardware- and human-in-the-loop. Owing to their use in critical system operations, the performance and trustworthiness of CPS must be evaluated under a variety of communication network modes which include the extreme cases of failures and attacks. In the smart grid, there are a variety of distributed sensors that are deployed at the system edge to provide situational awareness for monitoring and control. These sensors provide the condition at various points in the grid to detect any potential issues that could lead to system failure. One type of widely deployed sensor in the smart grid is the phasor measurement unit (PMU) or synchrophasor. PMUs provide voltage and current phasor and frequency measurements that are synchronized against a common time reference typically provided by global positioning system (GPS) [6]. Since these sensors provide time sensitive measurements, they need reliable communication to send the data to the grid control center. An attack on the timing infrastructure used by these devices could cause them to provide erroneous data to the operator that could lead to incorrect operating decisions such as unnecessary tripping of a line or not acting on a potential failure. Similarly, with the emergence of a plethora of innovative Internet of Things (IoT) devices for industrial control systems, edge computing, remote system monitoring and control, and home automation, it is equally critical to analyze the operational impacts of communication network failures for systems that incorporate IoT devices.

In order to analyze how communication network failures impact the operation of CPS and IoT, a careful consideration of the networked communication is necessary. In particular, analysis of a simple cyber attack might consider the impact of delaying the network packets, data corruption, replaying or reordering network packets, and packet loss. The HLA standard does not provide any direct means to support the co-simulation of these attack effects and supports only two options for delivering messages: receive order and timestamp order. In receive order, the messages are sent over UDP transport protocol and are delivered to the receiving federate with the best effort, without any explicit guarantee that a given message will be eventually delivered. In timestamp order, the messages are marked with a timestamp for delivery and are scheduled for delivery at that time to the receiving federate. A naïve approach may involve simply adding a delay to the timestamp of the delivered messages via HLA, but that does not realistically represent the behavior of unreliable message delivery. Even in cases in which a piece of manual code could be added to associated HLA federates' source code, this approach is highly inflexible and not representative of the flow of network packets in a real network. This is in contrast to using a communication network simulator that is integrated into the federation as a separate federate, where the networked communication between federates flows through the network simulator. The use of an integrated network simulator achieves not only faithful, high-fidelity network simulation, but also enables the realistic network characteristics such as unreliable message delivery against which the CPS and IoT systems can be evaluated.

The remainder of this section summarizes several desirable features for approaches to network simulation of CPS that are shown in Figure 1. The figure contains three federates, which consist of a federate implementation and the local RTI component (LRC) at each federate that implements the HLA message bus. Each federate has a unique representation in the simulated network model, depicted on the right. The remaining features in this figure are described in the subsections that follow.

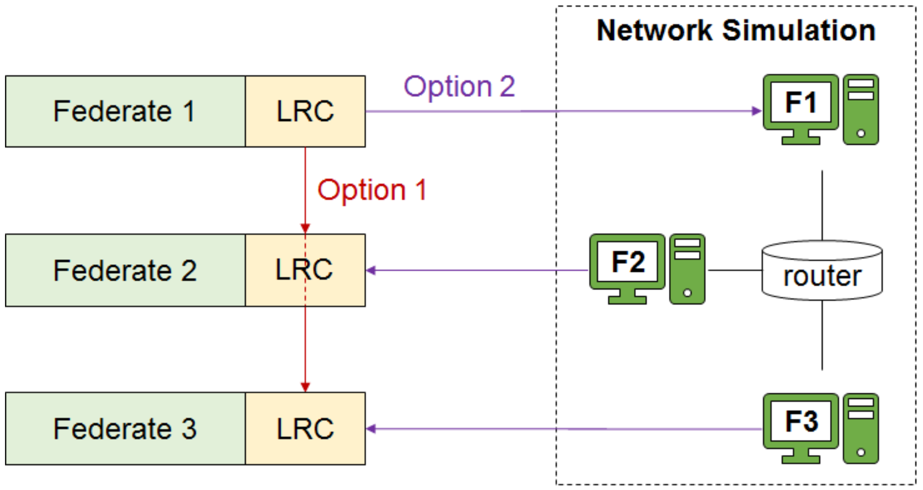


Figure 1: Approach to Network Simulation using the High Level Architecture (HLA)

2.1 The network model should contain nodes that represent a subset of the federates

At least two nodes from the network model should represent federates. The *Network Simulation* box from Figure 1 shows an example network model. A subset of the nodes in the network model indicated by labels F1 through F3 have a 1-to-1 correspondence to HLA federates. These federate nodes are connected through the simulated network topology. Although Figure 1 shows one network topology, the network model must be reconfigurable to allow the same federation to be executed with any number of different network configurations.

The network model does not need to define nodes to represent all federates in the federation. For instance, this federation could have a Federate 4 without the network model containing a corresponding node labeled F4. In this case, Federate 4 would not use the network simulation and all of its messages would use the default HLA provisions for object management. The network model also does not need to be fully connected. For instance, if node F1 cannot reach node F2 in the network topology, then none of the messages sent by Federate 1 using network simulation will be delivered to Federate 2.

2.2 The network simulator should be synchronized with the federation logical time

One responsibility of the network federate is to synchronize time progression of the simulated network with HLA federation logical time. The network federate is both time constrained and time regulating to operate in lock-step with HLA logical time. It also defines a function that maps a unit of HLA logical time to an exact number of seconds elapsed in the network simulation. This binding between the time representations of the federation and the network simulation ensures that a message is delivered to a federate only when the corresponding network packet is scheduled for delivery to that federate's node in the network simulation.

The optimal value for the logical step size of the network federate depends on the timing requirements of the federates using the simulated network. These timing requirements include concerns such as the smallest time interval between generation of network messages, and the shortest possible delivery time for messages sent from one federate to another. There is a trade-off between performance and simulation accuracy when choosing the logical step size. If the step size is too large, there will be delays in the delivery of messages to federates when a message arrives between time steps. If the step size is too small, the network simulator will synchronize more frequently with the federation which will lead to slower progression of logical time.

2.3 Network simulation should be configurable by both message type and sender

Even when a federate has a corresponding node in the network model, not all messages that originate from that federate are sent through the network simulation. A federate might want to coordinate with its peers or communicate with a federate that does not use the network simulation. For this reason, the use of network simulation is not configured per federate but rather per message that originates from a federate. This is shown in Figure 1 with two alternative paths for message flow listed as *Option 1* and *Option 2*. *Option 1* represents the normal HLA object management services where a federate can send and receive interactions and attribute updates using the RTI. *Option 2* is an alternative mode where specific messages are routed through a network simulator, rather than the usual set of HLA services.

In an ideal implementation, the LRC would perform this function of re-routing certain messages from the normal object management services into an alternative delivery mechanism based on the current network model. The RTI Initialization Data (RID) file could be modified to list the interactions and object classes that use network simulation. When the LRC received a message from the federate implementation, it would first check whether that specific message was configured for network simulation. If the message used simulation network, the LRC would send the message out-of-band to the network simulator. Otherwise, the LRC would continue to invoke the normal set of HLA object management services.

In this paper, the network simulator is a federate and *Option 2* is instead realized through re-encoding the message into a special interaction class that represents network packets. The network federate re-creates the original message once its corresponding network packet has propagated through the simulated network.

2.4 Federates should receive network simulated messages at different logical times

When the network federate receives a message from a LRC, it injects that message as one or more packets into the network simulation with the source of the packet set to the node representing the sending federate. If the network model is not based on multicast, then it is likely that one message will generate a unique packet for each federate node configured to receive that message that is reachable in the network model. All these packets will experience different delays, some may be dropped, and others might be modified through various forms of cyber-attacks. In the end, each federate node can receive a different packet, at different times, and perhaps with different content.

It is essential for cases such as network congestion and packet loss to break the reliable and uniform delivery of interactions and object classes guaranteed by HLA. Figure 1 shows an approach where this is implemented inside the RTI rather than the federates to reduce the amount of implementation required for each federate. However, the same effect could be achieved through implementation of a common library, shared by the federates, that sits between the LRC and the federate business logic.

2.5 Federate implementations should be agnostic to the presence of network simulation

Reusability is a desirable trait for federates developed for both CPS and IoT applications. Suppose a federate was developed to represent a PMU that reports time-synchronized voltage phasors measurements to some higher level application. This implementation could be useful for a number of different federation designs for different smart grid applications. Some of these applications might require realistic network delays to analyze the impacts of network congestion, some might require use of a specific communication protocol for hardware-in-the-loop testing, and some might just want to use a PMU with no network specific details. Despite differences in the interface on how the PMU is used, its basic implementation remains unchanged between these different applications.

While a typical HLA design flow might develop federations to achieve a specific purpose, in CPS and IoT applications, it is better to produce a federate like this PMU that can be composed into different scenarios. How this federate will be used is unknown at development time, and its implementation should support a broad range of scenarios without the need to develop additional code. For that reason, support for network simulation must be embedded into each federate as an option that can be enabled or disabled through configuration files. In addition, for different CPS and IoT federates to be interoperable, all the federates must implement their approach to optional network simulation using a consistent methodology.

3. Related Work

The integration of grid simulators and ICT into a co-simulation has over a decade of research. The first published approach in this area is the electric power and communication synchronizing simulator (EPOCHS) which uses HLA to integrate electromagnetic and electromechanical transient simulators with Network Simulator 2 (NS-2) [7]. Following EPOCHS, many co-simulation platforms were developed to integrate different grid simulators to different network simulators using different middleware [8]. The integrated co-simulation of power and ICT systems for real-time evaluation (INSPIRE) platform considers how to incorporate standard-based communication protocols into the co-simulation to support wide area monitoring, protection, and control (WAMPAC) applications [9]. The Hierarchical Engine for Large-scale Infrastructure Co-Simulation (HELICS) platform considers how to address scalability to handle grid scenarios that contain tens of thousands of independent agents [10].

The US National Institute of Standards and Technology (NIST) researched the effectiveness of different smart grid operating scenarios using the Framework for Network Co-Simulation (FNCS) developed by the Pacific Northwest National Laboratory that provides an integration of GridLAB-D, MATPOWER, and Network Simulator 3 (NS-3) [11]. The goal of this effort was to simulate a power grid segment that contained a substation and residential loads using different scenarios such as demand response and dynamic pricing. The work provided benchmarking for performance of the communication network under different operating conditions.

A follow-on work was on performance evaluation of DER and storage devices in terms of cost and impact on grid reliability [12]. This work was done by applying network traffic routing concepts to the routing of power in a grid segment with DER and storage devices. The premise for this work was that the resources are controllable, and the energy can be routed like network traffic management.

Additional research combined the smart grid operating scenarios (demand response and dynamic pricing) with the integration of DER in the grid [13]. This simulation work used a standard IEEE bus model with an integrated simulation platform that included GridMat, FNCS, GridLAB-D, and NS-3. The intent was to evaluate the performance of the grid with DER under different operating scenarios.

This paper attempts to address the feature from Section 2.4 on breaking the reliable delivery of HLA messages based on the results of network simulation. Other approaches largely limit their scope to adding message delays, and rarely consider the impact of packet loss or modification due to fault or cyber-attack.

4. Implementation Details

4.1 Universal CPS Environment for Federation (UCEF)

NIST has developed a software tool to expedite the development of federates and federations called the Universal CPS Environment for Federation (UCEF) [14]. UCEF is distributed as an Ubuntu virtual machine pre-configured with a suite of software useful in the development of different federate types. The latest 1.0.1 version of UCEF includes support for Java and C++ federates, and several grid simulators including GridLAB-D, TRNSYS, and EnergyPlus. The front end of UCEF is the Web-based Generic Modeling Environment (WebGME) developed at Vanderbilt University that provides a graphical web environment where users can model federations using simple building blocks. At the back end are JavaScript extensions to WebGME that perform code generation to transform the federate models into stub code for the different supported federate types. A core concept of UCEF is the separation of a federate implementation into two layers: a user layer that implements the intended function of the federate, and an infrastructure layer generated from WebGME that implements shared boiler plate code.

The UCEF infrastructure layer handles functions such as joining a federation, declaring publication and subscription interests, providing helper methods to send and receive interactions and object attributes, and other application independent utility functions. It also prescribes a basic federate lifecycle with hooks that an application developer can extend to customize the behavior of the federate at specific points in the HLA state machine, such as after the grant of an advance time request. The WebGME generated code closely resembles the structure of a Functional Mock-up Unit (FMU) as defined in the FMI standard for co-simulation.

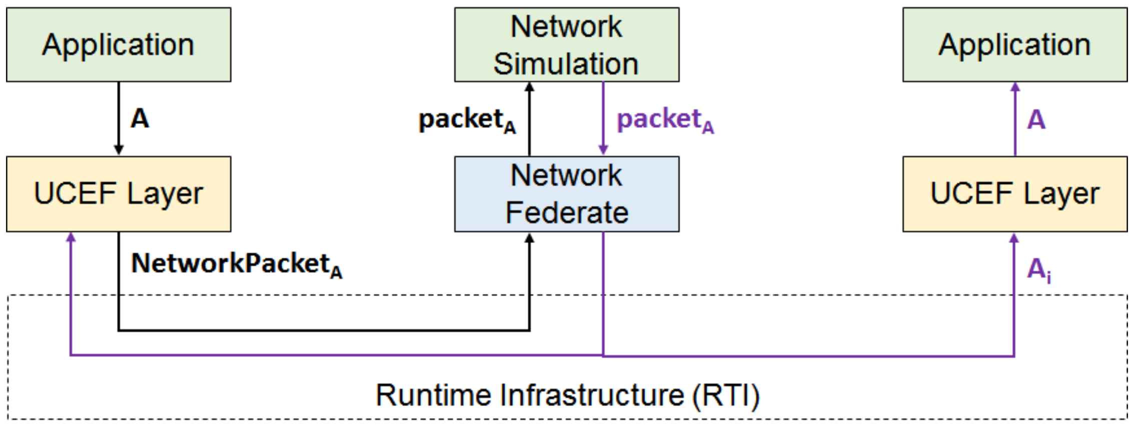


Figure 2: The UCEF Architecture for Network Simulation

Figure 2 shows how network simulation is implemented in UCEF to satisfy the requirements enumerated in Section 2. Rather than implement a new RTI, or modify an existing RTI, the logic related to network simulation was implemented in the UCEF infrastructure layer. In the figure, the user application on the left sends an interaction class A which has been flagged for network simulation. Before the UCEF layer passes this interaction to the RTI, it converts it into a special NetworkPacket interaction class to ensure it is routed to the network federate.

The network federate subscribes to the NetworkPacket interaction class, and encodes the received interaction instance as one or more packets in a format compatible with the network simulation. These packets are injected into the network simulation, where they propagate through the simulated network. When a packet arrives at its destination, it is delivered back to the network federate along with the name of the destination node from the network simulation. Then the network federate reconstructs the original interaction embedded inside the NetworkPacket and watermarks this new interaction with the destination node name. The network federate is implemented using a library called the UCEF Gateway [15], which enables it to create dynamic publications based on the current federation object model. As such, the network federate is not bound to a specific data model and can be used in any federation without code modification.

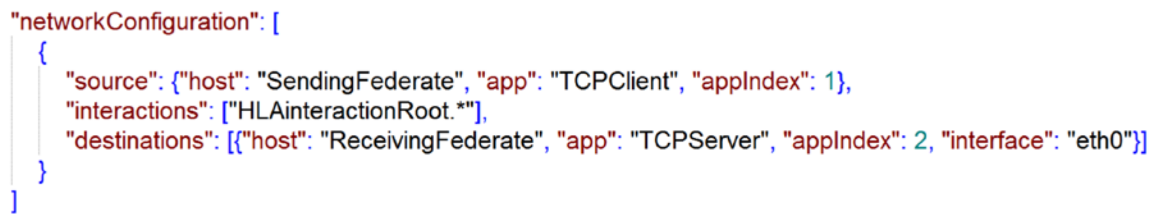
Because the network federate sends the watermarked interaction using the standard HLA services, all subscribed federates will receive it. However, each federate is configured to know the name of its representation in the network simulation, and the UCEF layer can use the watermark to check if an interaction was meant for its user application. In this manner, even though the left most federate receives a copy of its own interaction A_i, this interaction will be dropped at the UCEF layer before it reaches the user application.

One constraint of this approach is that all federates that use network simulation must have the same UCEF layer, which means they must be code generated from the UCEF virtual machine. The benefit of the approach is that the application never has to know about the network simulation and only must consider its native interaction class A. When network simulation is required, the infrastructure will take care of it. The remainder of this section describes how network simulation was embedded into the UCEF layer.

4.2 Network Configuration of UCEF Federates

There are three requirements to support network simulation in the UCEF layer: (1) the interaction classes that require network simulation must be specified in a configuration file, (2) the NetworkPacket interaction class must be defined, and (3) a filtering mechanism must be defined to ensure that interactions sent by the network federate are received only by the intended federates.

The same configuration file for network simulation is shared by all federates, including the network federate. This JavaScript Object Notation (JSON) file lists which interactions from which federates should use network simulation and how those interactions should be routed through the network simulation. Figure 3 shows an example instance of this configuration file. This JSON configuration was designed for use with the OMNeT++ network simulator using its INET Framework.



```
"networkConfiguration": [
    {
        "source": {"host": "SendingFederate", "app": "TCPClient", "appIndex": 1},
        "interactions": ["HLAinteractionRoot.*"],
        "destinations": [{"host": "ReceivingFederate", "app": "TCPServer", "appIndex": 2, "interface": "eth0"}]
    }
]
```

Figure 3: Example JSON for Network Configuration

The network configuration is a list of network rules. Each network rule defines the list of *interactions* for a given *source* federate that are configured to use network simulation. These interactions are injected into the network model at the *source* node and routed to each of the listed *destinations*. The same *source* federate can appear in multiple network rules for the case when different interactions from the same source have different destinations. This implementation assumes for simplicity that the federate name is identical to the host name of its equivalent network node. Under this assumption, the *host* fields are both the federate name and the network node name.

A network node in the OMNeT++ INET Framework contains submodules for different network applications. For example, a node could define a submodule for a Representational State Transfer (REST) server running on localhost:8080. These network applications are identified using an application name (*app*) and an application or submodule index (*appIndex*). The destination applications are bound to a specific network interface (*interface*) to handle cases where a node has more than one available network interface.

Figure 4 shows the WebGME representations of the two interaction classes *HLAinteractionRoot.InteractionBase* and *HLAinteractionRoot.InteractionBase.NetworkPacket*.

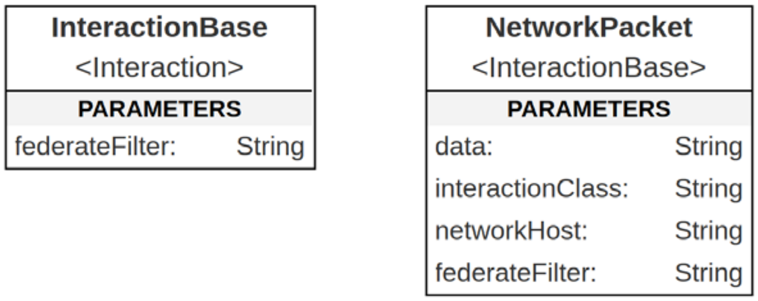


Figure 4: Example Object Model for Network Simulation

The UCEF layer encodes all interactions it sends to the network federate using the NetworkPacket interaction class. Besides the *federateFilter* parameter that will be discussed later, this interaction class has three parameters. The *interactionClass* and *data* parameters are used to embed the original interaction from the user application into the NetworkPacket. When an instance of interaction class A is converted into a NetworkPacket, the *interactionClass* parameter would be the fully qualified class path of A and the *data* parameter would be the serialized parameters of A. These fields are used by the network federate to reconstruct the original interaction after the packet propagates through the network simulation. The final parameter, *networkHost*, is set to the unique identifier of the sending federate. This allows the network federate to inject the packet into the network simulation at the correct node.

When the UCEF layer receives an interaction, it checks if that interaction was sent by the network federate. If the interaction was sent by the network federate, then it is possible that it was only intended for receipt by a single federate. The network federate uses the *federateFilter* parameter to specify this destination. If the *federateFilter* is empty, then the UCEF layer processes the interaction as normal. Otherwise, the UCEF layer discards the interaction unless the *federateFilter* is string equivalent to the federate's own unique identifier.

4.3 Implementation of the UCEF Layer

Figure 5 shows a flowchart for how interaction classes are handled from both the sending and the receiving federates. The first decision box at the top of the figure, whether to use network simulation, is determined by the content of the JSON file from Figure 3. The second decision box at the bottom of the figure, whether the received interaction should be delivered to this specific federate, performs the string comparison between the *federateFilter* parameter from Figure 4 and the federate name of the receiving federate. If the federate filter parameter is set but not equivalent to the receiving federate name, then the packet is dropped and not delivered to the user application.

This filtering mechanism allows different federates to receive the same interaction at different logical times, dependent on the results of network simulation. An equivalent implementation could have been achieved using regions from the HLA data distribution management (DMM) services instead of as a parameter of a base interaction type. However, as the configuration management of regions can be quite cumbersome, this simple filtering mechanism was implemented at the UCEF layer instead.

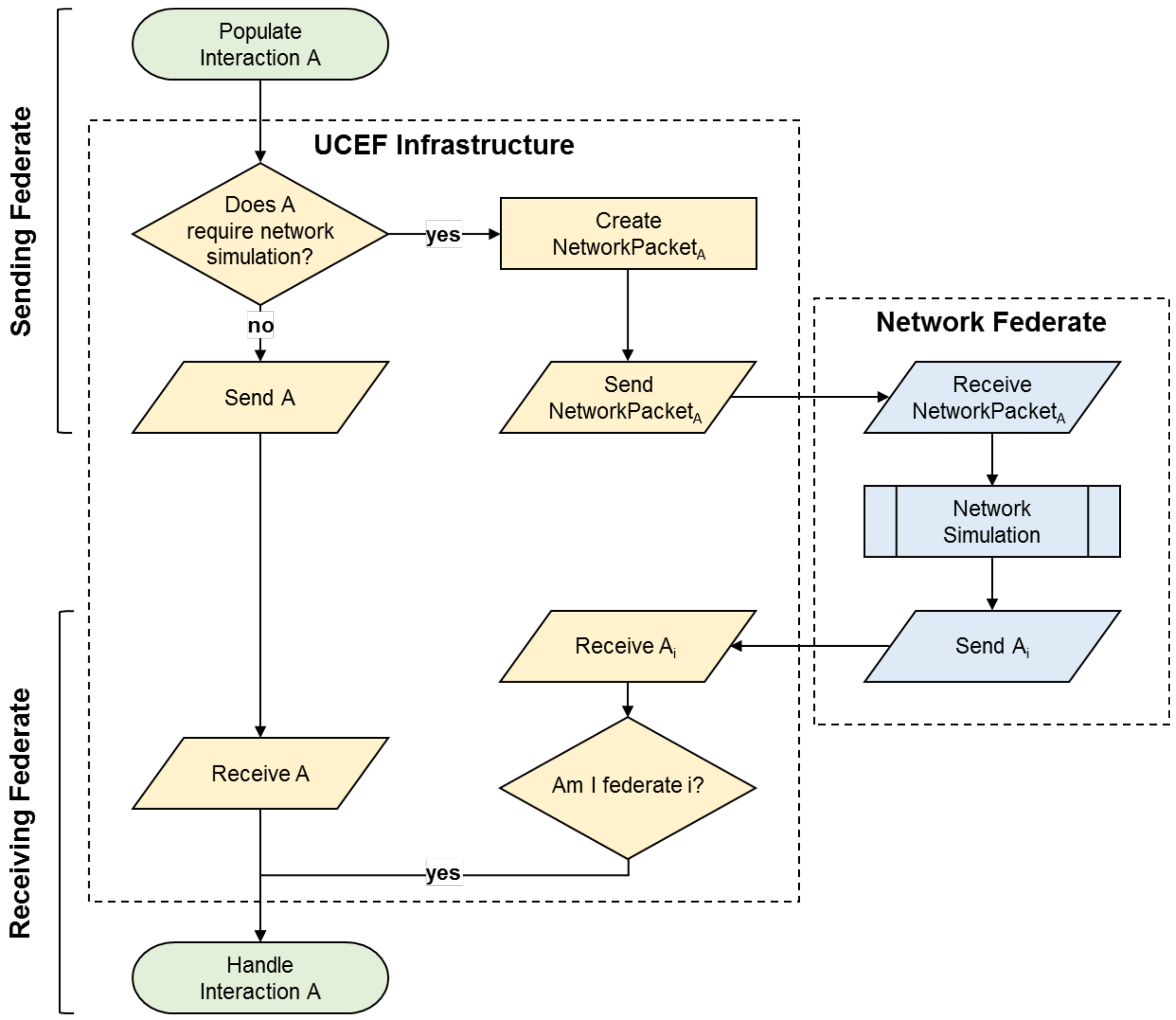


Figure 5: Flow of Interactions through the UCEF Layer

4.4 Implementation of the Network Federate

Algorithm 1 shows pseudocode for the network simulation from Figure 5. This algorithm uses the OMNeT++ network simulator, and extends the OMNeT++ *cSimpleModule* class which defines the *step* and *handleMessage* methods. The *step* method executes each HLA logical time step and checks for either packets from the network simulator or interactions from the federation. When an interaction is received, the *step* method parses the JSON network configuration to create packets in the network simulation for each configured destination. These packets propagate through the simulated network until they arrive at their destination, causing OMNeT++ to invoke the *handleMessage* method. In *handleMessage*, a customized interaction is created for the node that received the packet. The current implementation of time synchronization maps a unit of HLA logical time to a second of network simulation time. The *advanceTimeRequest* method could be replaced with an alternative implementation that uses a scaling function to make each logical time step some configurable multiple of seconds.

Algorithm 1: Network Simulation Pseudocode

```
variable: toHlaQueue
variable: toNetQueue
variable: networkConfiguration

method step() // inherited from cSimpleModule
    foreach $interaction \in toHlaQueue$ do
        sendToHla($interaction$)
    foreach $interaction \in toNetQueue$ do
        rule ← getNetworkRule($interaction.networkHost$, $interaction.interactionClass$)
        // create unique packet for each destination
        foreach $destination \in rule.destinations$ do
            packet ← convertToPacket($interaction.data$)
            packet.destination ← rule.destination
            sendDirect($packet$, $rule.source$) // inject packet into OMNeT++
    advanceTimeRequest() // synchronize with HLA logical time

method handleMessage($packet$) // inherited from cSimpleModule
    if $interaction$ ← convertToInteraction($packet$) then
        toHlaQueue.push(interaction) // ignore packets that do not contain embedded interactions

method receiveInteraction($networkPacket$)
    toNetQueue.push(networkPacket)

method getNetworkRule($host$, $interaction$)
    foreach $rule \in networkConfiguration$ do
        // the $\in$ operation must handle the wildcard character
        if $host = rule.source.host \wedge interaction \in rule.interactions$ then
            return rule
    return $\emptyset$
```

5. Conclusion

This paper proposed an approach to incorporate network simulation into the High Level Architecture (HLA). Whether the proposed approach is HLA compliant depends on the interpretation of the rule that "*During a federation execution, all exchange of FOM data among federates shall occur via the RTI*" [4]. From the federate perspective, this rule is upheld because the NetworkPacket interaction is transmitted to the federation via the RTI. However, from the user application perspective, the interaction that the user wants to send is automatically converted into a different interaction class and the RTI is never used to transmit the data in its intended format. This approach was chosen because co-simulation of CPS requires network simulations that support the concepts of message delay and packet drop, and integrating the semantics of network simulation into each individual federate - while feasible - is far too burdensome.

The next step is to complete the implementation of this approach in UCEF. An implementation that addresses the need for packet loss and modification was developed by Vanderbilt University for their Command and Control Wind Tunnel (C2WT) platform [16][17]. However, it requires source code modifications when changes are made to the network configuration. Another implementation was released by Calytrix for their open-source Portico RTI [18]. However, it requires the use of multiple network federates which leads to poor scalability for large federation sizes, and the approach may be incompatible with the popular OMNeT++ INET Framework. Future work will merge these implementations to produce one network federate configurable using JSON that is compatible with the OMNeT++ INET Framework. This future work will aim to improve the time synchronization strategy to be more flexible than a 1-to-1 equivalence between HLA logical time and the network simulator time.

6. Acknowledgement

Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreement 70NANB19H100. Official contribution of the National Institute of Standards and Technology; not subject to copyright in the United States. Certain commercial products are identified in order to adequately specify the procedure; this does not imply endorsement or recommendation by NIST, nor does it imply that such products are necessarily the best available for the purpose.

7. References

[1] Griffor, E. R., Greer, C., Wollman, D. A., & Burns, M. J. (2017). *Framework for cyber-physical systems: Volume 1, overview* (NIST-SP-1500-201). doi: 10.6028/NIST.SP.1500-201

[2] Greer, C., Wollman, D., Prochaska, D., Boynton, P., Mazer, J., Nguyen, C., FitzPatrick, G., Nelson, T., Koepke, G., Hefner Jr., A., Pillitteri, V., Brewer, T., Golmie, N., Su, D., Eustis, A., Holmberg, D., & Bushby, S. (2014). *NIST Framework and Roadmap for Smart Grid Interoperability Standards, Release 3.0*. (NIST SP-1108r3) doi: 10.6028/NIST.SP.1108r3

[3] Institute of Electrical and Electronics Engineers. (2010). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Framework and Rules* (IEEE Std 1516-2010) doi: 10.1109/IEEESTD.2010.5553440

[4] Institute of Electrical and Electronics Engineers. (2010). *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-- Federate Interface Specification* (IEEE Std. 1516.1-2010) doi: 10.1109/IEEESTD.2010.5557728

[5] *Functional Mock-up Interface for Model Exchange and Co-Simulation 2.0* (2014, July). Retrieved November 27, 2019 from http://fmi-standard.org

[6] Terzija, V. (2011). Wide-Area Monitoring, Protection, and Control of Future Electric Power Networks. *Proceedings of the IEEE*, 99(1), 80-93. doi: 10.1109/JPROC.2010.2060450

[7] Hopkinson, K., Wang, X., Giovanini, R., Thorp, J., Birman, K., & Coury, D. (2006). EPOCHS: a platform for agent-based electric power and communication simulation built from commercial off-the-shelf components. *IEEE Transactions on Power Systems*, *21*(2), 548-558. doi: 10.1109/TPWRS.2006.873129

[8] IEEE Task Force on Interfacing Techniques for Simulation Tools (2016). Interfacing Power System and ICT Simulators: Challenges, State-of-the-Art, and Case Studies. *IEEE Transactions on Smart Grid*, *9*(1), 14-24. doi: 10.1109/TSG.2016.2542824

[9] Georg, H., Müller, S. C., Rehtanz, C., & Wietfeld, C. (2014). Analyzing cyber-physical energy systems: The INSPIRE cosimulation of power and ICT systems using HLA. *IEEE Transactions on Industrial Informatics*, *10*(4), 2364-2373. doi: 10.1109/TII.2014.2332097

[10] Palmintier, B., Krishnamurthy, D., Top, P., Smith, S., Daily, J., & Fuller, J. (2017, April). Design of the HELICS high-performance transmission-distribution-communication-market co-simulation framework. In *2017 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)* (pp. 1-6). IEEE. doi: 10.1109/MSCPES.2017.8064542

[11] Moulema, P., Yu, W., Griffith, D., & Golmie, N. (2015). On Effectiveness of Smart Grid Applications Using Co-Simulation. *24th International Conference on Computer Communication and Networks (ICCCN)*. doi: 10.1109/ICCCN.2015.7288438

[12] Xu, G., Yu, W., Griffith, D., Golmie, N., & Moulema, P. (2016). Towards Integrating Distributed Energy Resources and Storage Devices in Smart Grid. *IEEE Internet of Things Journal*. 4(1): pp 192-204. doi: 10.1109/JIOT.2016.2640563

[13] Mallapuram, S., Yu, W., Moulema, P., Griffith, D., Golmie, N., & Liang, F. (2017). An Integrated Simulation Study on Reliable and Effective Distributed Energy Resources in Smart Grid. *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*. pp 140-145. doi: 10.1145/3129676.3129684

[14] Burns, M., Roth, T., Griffor, E., Boynton, P., Sztipanovits, J., & Neema, H. (2018). Universal CPS Environment for Federation (UCEF). In *2018 Winter Simulation Innovation Workshop*.

[15] Roth, T., & Burns, M. (2018). A gateway to easily integrate simulation platforms for co-simulation of cyber-physical systems. In *2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)* (pp. 1-6). IEEE. doi: 10.1109/MSCPES.2018.8405394

[16] Hemingway, G., Neema, H., Nine, H., Sztipanovits, J., & Karsai, G. (2012). Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach. *Simulation*, *88*(2), 217-232. doi: 10.1177/0037549711401950

[17] Neema, H. (2018). *Large-Scale Integration of Heterogeneous Simulations* (Doctoral dissertation, Vanderbilt University). Retrieved from https://www.isis.vanderbilt.edu/node/4925

[18] *Federate Base* (2019, July). Retrieved November 27, 2019 from https://github.com/openlvc/federate-base

Author Biographies

THOMAS ROTH leads development of the technology behind the cyber-physical systems testbed at the National Institute of Standards and Technology as a member of its Smart Grid and Cyber-Physical Systems program office. His research interests are in formal methods for the composition of cyber-physical systems, and the detection of compromised cyber-physical devices through comparison of their reported behavior against the constraints of the physical system.

CUONG NGUYEN leads the Smart Grid Testing and Certification Project in the Smart Grid and Cyber-Physical Systems Program Office of the Engineering Laboratory at the National Institute of Standards and Technology. He works with industry to support standards-based interoperability test programs to help accelerate smart grid deployments. Cuong is the chair of the Smart Electric Power Alliance (SEPA) Testing and Certification Working Group (TCWG). Cuong coordinates international outreach efforts through bilateral and multilateral engagements.

MARTIN BURNS is the Associate Director for the CPS/IoT Testbed in the Smart Grid and Cyber-Physical Systems Program Office at NIST. With his background in IEC and ANSI standards development for semantic models and data exchange, he has facilitated the development of the underlying Green Button technologies which define energy usage information and APIs in the Smart Grid in the US and internationally. He co-chairs the data interoperability working group for the NIST led Framework for Cyber-Physical Systems (CPS) and is a key contributor to NISTs architecture for UCEF-federated testbeds for investigating the behaviors of CPS/IoT.

HIMANSHU NEEMA is a Research Assistant Professor of Computer Science at Vanderbilt University. He holds a M.S. and Ph.D. in Computer Science from Vanderbilt University. Dr. Neema researches in the general area of model-based design and modeling and simulation of Cyber-Physical Systems and their integrated simulation with hardware- and humans- in the loop. His research interests include: Modeling & Simulation, Model-Integrated Computing, Distributed Simulations, Artificial Intelligence, Constraint Programming, Planning & Scheduling, Smart-Grids, Transactive Energy, Service-Oriented Architectures (SOAs), Semantic Web, and Automated Document Analysis & Classification. Dr. Neema has 20 years of experience in research and development of software applications covering the above areas and has co-authored ~50 publications.