Integrated Simulation and Emulation Platform for Cyber-Physical System Security Experimentation

Wei Yan†, Yuan Xue†, Xiaowei Li†, Jiannian Weng†, Timothy Busch‡, and Janos Sztipanovits†
†Institute for Software Integrated Systems and EECS Department, Vanderbilt University, Nashville, TN, USA
‡The State University of New York Institute of Technology, New York, NY, USA
wei.yan@vanderbilt.edu, yuan.xue@vanderbilt.edu, xiaowei.li@vanderbilt.edu, jiannian.weng@vanderbilt.edu, timothy.busch@sunyit.edu, janos.sztipanovits@vanderbilt.edu

ABSTRACT
There is a pressing need to evaluate both cyber- and physical systems together and holistically for a rapidly growing number of applications using simulation and emulation in a realistic environment, which brings realistic attacks against the defensive capabilities of CPS (Cyber-Physical System). Without the support from appropriate tools and run-time environments, this assessment process can be extremely time-consuming and error-prone, if possible at all. In this paper, we present iSEE - integrated Simulation and Emulation platform for security Experimentation, as a "software supporting research infrastructure used for cyber security research and development". iSEE allows for the concurrent modeling, experimentation and evaluation of CPS that range from a fully simulated to a fully implemented system. iSEE has two major components: 1) modeling environment for system specification and experiment configuration and 2) run-time environment that supports experiment execution. iSEE employs the Model-Integrated-Computing (MIC) approach, which explicitly uses models throughout the experiment environments and integrates them at the domain-specific model level. The run-time environment of iSEE integrates Matlab and the DETERlab testbed to support realistic assessment of CPS on real distributed networking environments in its early design phase, before a fully implemented system is available. At run time, iSEE provides time synchronization and data communication and coordinates the execution of the security experiment across simulation and emulation platforms.

Categories and Subject Descriptors
I.6.7 [Simulation and Modeling]: Simulation Support Systems—environments

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HCoNS ’12, April 17–18, 2012, Beijing, China.
Copyright 2012 ACM 978-1-4503-1263-9/12/04 ...$10.00.

General Terms
Experimentation, Security

Keywords
Cyber-Physical Systems, Simulation, Emulation, Security Experiment Platform

1. INTRODUCTION
Cyber-Physical Systems (CPS) are characterized by the tight coupling and coordination among sensing, communications, computational and physical components. As CPS become increasingly complex in terms of distributed architectures and expanded capability, it becomes extremely challenging to formally analyze their security properties. Currently, many CPS are designed without considering the effects of the network operating environment (e.g., time-varying delays and packet losses). Such inadequacies during the system design can lead to catastrophic consequences after the actual systems are deployed, as these systems are interconnected to open networks and exposed to network dynamics and uncertainties. Thus, there is a pressing need to evaluate both cyber and physical systems together holistically for a rapidly growing number of applications using both simulation and emulation in a realistic environment. Without support from appropriate tools and experimental environments, this evaluation process can be extremely time-consuming and error-prone, even impossible at all.

Building such an experiment environment is a challenging task, especially if one wants to test the security properties of CPS under real security attack scenarios. Currently, the DETERlab testbed [2] is the only available emulation environment accessible by researchers to perform realistic security experiments. Yet using the DETERlab testbed alone for CPS evaluation has a number of limitations. First, experiments on the DETERlab testbed require a fully implemented software system. This prevents early evaluation of CPS when they are partially implemented. In a CPS design, a “virtual prototype” of a system can be a suite of simulation models and prototypes, each element representing a system (hardware and software) and its environment with a different fidelity. Many CPS design aspects (such as control algorithms) need to be evaluated on this “virtual prototype” in the early design phase, instead of on a fully implemented...
system. Second, experiment setup in the DETERlab testbed requires the user to perform significant amount of configuration work and have sufficient background knowledge in networking and security. Though SEER [1] provides a tool to facilitate this procedure, domain experts, who are not familiar with networking systems, may create inconsistent experiment setups (e.g., mismatching host IDs in the system configuration and in the attack generation) and find it very time-consuming for large-scale experiments. This creates a huge barrier for CPS security assessment. On the other hand, Matlab/Simulink [3] has been a very popular tool to model and evaluate the performance of CPS systems via simulation. Though network simulation is provided in Matlab/Simulink via Truetime toolbox [9], the accuracy of its simulation depends on the level of abstraction of the network protocol models. The operating system details, such as buffer operations, which are essential to evaluate security attack behaviors, cannot be simulated in these control simulators.

In this paper, we present iSEE - integrated Simulation and Emulation platform for cyber-physical system Security Experimentation 1. iSEE allows for the concurrent modeling, experimentation and evaluation of CPS that range from a fully simulated to a fully implemented system. iSEE has two major components: 1) a modeling environment for system specification and experiment configuration and 2) a run-time environment that supports experiment execution. iSEE employs the Model-Integrated-Computing (MIC) [16] approach, which explicitly uses models throughout the experiment environments and integrates them at the domain-specific model level. Using the model-integrated approach enables the experiment to be rapidly reconfigured and maintains the consistency among multiple simulation models and the real software components. The run-time environment integrates the Matlab/Simulink simulation tool with the DETERlab testbed. It provides time synchronization and data communication services and coordinates the execution of the security experiment across the simulation and emulation platforms.

The rest of the paper is organized as follows. Section II presents the related work. Section III provides an overview of the system architecture. Section IV presents the model-based design and integration of CPS using the MIC technique. Section V describes the Run-Time components. The evaluations of iSEE is presented in section VI. Finally, section VII concludes this work.

2. RELATED WORK

There have been some efforts that simulate and assess CPS using a variety of tools. These efforts can be categorized into several classes, as summarized in Table 2. Most of these tools have limited capability in simulating and evaluating CPS in different ways. Matlab/Simulink [3] has been widely used for modeling and simulating control systems and provides a Truetime toolbox [9] for approximating link layer network simulation. However, the network simulation accuracy, which depends on the abstraction level of the network protocol models, is greatly limited in the Truetime toolbox, due to its inability of supporting higher level protocols (e.g., TCP or UDP). In addition, Truetime toolbox cannot simulate operating system level issues, such as buffer operations, which are essential for detecting security attacks. Similar tools include Modelica [4] and Ptolemy [7], which allow for the simulation of CPS under various continuous and discrete dynamics. These tools still suffer from the similar challenges as Truetime in Matlab/Simulink in terms of network simulation.

Packet-level network simulators, such as ns-2 [5] and OMNet++ [6], provide a detailed implementation of network protocol stack. However, using ns-2 alone for NCS evaluation requires the control algorithm to be fully implemented in a high-level language such as C++. This becomes very difficult as the complexity of the control system increases.

Thus, several efforts have been made towards integrating multiple simulators in order to more effectively and accurately model and evaluate CPS. A tool chain PiccSIM [17] allows for the integration of Matlab/Simulink models with ns-2. It also provides a graphical user interface for the design of control systems and the automatic code generation of ns-2 and Matlab/Simulink models. Another work [13] couples several simulators, including ModelSim, Matlab/Simulink and ns-2, though establishing the communication between them. NCSWT [11] also integrates Matlab/Simulink and ns-2 for simulating networked control system and, in particular, uses MIC techniques to define behavior models for the control and networking systems respectively, which facilitate the code generation. Other similar works include [14, 8, 12].

Although the above integrated simulators can achieve better performances than individual simulators, the simulation accuracy is still bounded by the capability of network modeling provided by tools, such as ns-2 and OMNet++. Instead, DETERLab [2] is a realistic network emulation environment, where CPS security experiments can be carried out with better network accuracy and operating system level details. Therefore, our work iSEE, which integrates both control modeling environment (i.e., Matlab/Simulink) and network emulation environment (i.e., DETERLab), has evident benefits over the existing tools for simulating and evaluating CPS.

3. OVERVIEW ARCHITECTURE

Figure 1 presents the overall system architecture of iSEE. It has two major components: (1) the modeling environment for system specification and experiment configuration and (2) run-time environment that supports experiment execution. iSEE employs a model-integrated approach, which explicitly uses models throughout the experiment environments and integrates them at the domain-specific model level. The modeling environment of iSEE allows the user to specify (1) the system model of CPS and (2) security experiment scenario configuration. Currently, the modeling language of iSEE focuses on the modeling of networked control systems (NCS). iSEE provides an overarching modeling environment that integrates the operational semantics of Matlab simulation platform and the DETERlab emulation platform. The model interpreter generates the configuration files and necessary interface code to manage the interactions among the simulation and emulation platforms and configures the experiment scenario in the run-time environment. The model-integrated approach enables the experiment to be rapidly reconfigured and maintains the consistency among multiple simulation models and the real software components.

1Source code and demonstration video of iSEE can be found at http://vanets.vuse.vanderbilt.edu/dokuwiki/doku.php?id=research:isee.
Table 1: Summary of existing CPS experiment tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Type</th>
<th>Capacity</th>
<th>Limitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truetime [9] in Mat-</td>
<td>control system modeling and sim-</td>
<td>model complex control algo-</td>
<td>limited support for network sim-</td>
</tr>
<tr>
<td>lab/Simulink, Modelica [4],</td>
<td>ulation environment</td>
<td>rithms</td>
<td>ulation</td>
</tr>
<tr>
<td>Ptolemy [7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ns-2 [5], OM-Net++ [6]</td>
<td>network simulation environment</td>
<td>packet-level simulation of net-</td>
<td>limited in control system model-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>work protocol stack</td>
<td>ing and design</td>
</tr>
<tr>
<td>PiccSIM [17], ModelSim [13], NCSWT [11]</td>
<td>integrated modeling and simulation environment of both control and network systems</td>
<td>model control system dynamics and simulate network behavior simultaneously</td>
<td>lack of realistic network accuracy and operating system level details</td>
</tr>
</tbody>
</table>

Figure 1: Overall Framework

The run-time environment integrates the Matlab/SimuLink simulation tool with the DETERlab platform. It provides time synchronization and data communication service and coordinates the execution of the security experiment across these two platforms. In our current design, the integrated system runs at the same pace as the physical clock time. A novel virtual-time-based time skew adjustment mechanism is presented to compensate the time shift between the simulation and the emulation environment. A detailed description of the design-time modeling framework and the run-time execution framework are presented in Section IV and Section V respectively.

4. MODELING ENVIRONMENT

In order to facilitate the rapid security experiment of CPS with minimal effort, we employ the model integrated computing (MIC) techniques [16]. In iSEE, we focus on the support for NCS systems and define meta-models for three domain-specific modeling languages (DSMLs), for modeling and evaluating the NCS. The DSMLs are: (1) NCS HLA Simulation Language; (2) Control Subsystem Modeling Language; (3) Networking Emulation Modeling Language. The NCS HLA simulation language in iSEE is an extension of the DSML used in the Command and Control WindTunnel system [15]. The C2WT DSML provides all of the modeling primitives required to specify the integration, deployment, and execution of a federated simulation based on HLA. The Control subsystem modeling language in iSEE is based on the DSML used in the Networked Control System WindTunnel system [11], which defines the modeling elements for describing the dynamic behavior of the system components of the NCS.

We enhance the meta-models of these two systems with the network emulation meta-models to model (1) the networking emulation configuration, (2) the relationship between the control system and the networking system, and (3) the interaction between the simulation and the emulation platforms. In particular, our extension includes three aspects: (1) Network Topology Model, which specifies network topology configuration; (2) Application Process Deployment and Communication Model, which specifies the application deployment and communication configuration; (3) Network Emulation Interaction Model, which specifies the major attributes of the information exchanged between simulation and emulation environments. The CPS experiment configuration can specify the communication semantics through network modeling, which can in turn customize the communication protocol between the simulation and emulation environment to achieve the best tradeoff between experiment realism and performance. We will illustrate the above three aspects in the following sections.

4.1 Network Topology Model

The network topology model specifies the communication network topology. This model is particularly useful when DETERlab is used as the emulation platform. It will be used to generate the Tcl (Tool Command Language) script that configures the DETERlab network topology. If the experiment network is manually setup, this model will also
provide a blueprint for the setup. The meta-model of the network topology is shown in Figure 2, which includes four types of objects:

1. **NetworkElementBase**, which includes **Host** (end host), **Router**, and **Switch**.
2. **Link**, which connects the **NetworkElementBase**. A link has three attributes: **Bandwidth**, **Delay** and **Loss**.
3. **NetworkBase**, which models the network as a whole. For wired networks, the object **WiredNetwork** can be used to approximate a network whose topology is simplified.
4. **Connection**, which connects the **NetworkBase** and **NetworkElementBase**. When connecting to a **WiredNetwork**, the **Connection** approximates a network path, where its bandwidth is the bottleneck of the path.

Figure 3 shows an example network topology model. In this example, there are two unmanned aerial vehicles (UAVs) communicating with a **ControlStation** through wired links.

**Figure 3: An example network topology model**

### 4.2 Application Process Deployment and Communication Model

This model specifies the deployment of application processes on the emulation hosts and the transport-layer communication model that will be used. This model includes the following objects.

1. **NetworkApp**, which models the network application process that is running on the emulation host. It has at least one port. The **TCPBasicApp** and **UDPBasicApp** can be pre-built as two basic forms of network application processes to facilitate network experiments (e.g., for background/attack traffic generation).
2. **OnHost**, which models the deployment of a network application on an end host (i.e., **Host** in the topology model). Note the host defined in the network topology model is a virtual host. When DETERlab is used, the virtual host will be mapped to a real host where the IP address is assigned. In this case, the emulation gateway needs to keep track of the IP address and performs host mapping.
3. **NetworkAppInteraction**, which models the communication connection between network applications. There are two types of communication models between application processes: connection-oriented and connectionless. TCP protocol is connection-oriented, while UDP protocol can work in both connection-oriented and connectionless manners. For connection-oriented communication, two application processes need to establish a connection. The information of the destination application process is predefined (prewired) in the Application Deployment model, so that it does not need to be carried in the Network Emulation model (detailed in the next subsection). In the connectionless model, the destination process information is unknown during the deployment. Thus it is required to be carried in the Network Emulation model.

The application deployment model will be used to generate Tcl files that initialize the DETERlab deployment. At run time, this model can also help to validate the network communication pattern (a feature that can be explored for security purpose in the future.). Figure 5 shows an example of application deployment model over the network as shown in Figure 3. In this example, there are two network applications. The UAVs will transfer images to the ControlStation using UDP in the connection-oriented mode; the ControlStation will send commands to the UAVs via TCP. For each UAV, there are two application processes **SendImage** and **RecvCommand** deployed. The ControlStation also has two application processes: **RecvImage** and **SendCommand** deployed.

**Figure 5: Application Deployment Model Example**

### 4.3 Network Emulation Interaction Model

This model specifies how the network application processes in the CPS communicate through the network and, in particular, what information needs to be exchanged between the simulated elements and the real network applications through **EmuGateway** in the experiment. In our design, this function is enabled through **NetworkInteraction**, a special type of HLA Interaction.

As shown in Figure 6, each **NetworkInteraction** needs to have three attributes in order to be processed by the **EmuGateway**. These three parameters specify **NetworkInteraction** to be processed by which application (**ProcName**) lo-
Figure 6: Network Interaction Metamodel

1. **Timestamp**, the time when this interaction should be processed by the network application process. Note that this is not the timestamp of the `NetworkInteraction` itself. The `NetworkInteraction` needs to be sent at an earlier time before this Timestamp through RTI for the sake of time synchronization.

2. **ProcName**, the name of the network application process that will handle this `NetworkInteraction`. This name should be exactly the same as the process name used at the emulation host, as it will be passed directly by the `EmuGateway` to the `TapClient`.

3. **NodeName**, the name of the node that hosts the network application process which handles this `NetworkInteraction`. Note that the deployment of the application process onto emulation hosts may not be known at the modeling time and the `EmuGateway` will convert this name to the real hostname/IP address at runtime.

Figure 7 shows an example of the Interaction Model for the network scenario outlined in Figure 3 and 5. In this example, the `UAVFederate` represents two simulated UAVs. The `UAVFederate` publishes `NetworkInteraction` `SendImageToNetwork`, which is subscribed by the `EmuGateway` federate. This interaction specifies UAV1 as the NodeName (the node which sends the image, could be other UAVs since UAV federate is responsible for multiple UAV simulations here) and `SendImage` as the ProcName (application process name). This `NetworkInteraction` has one parameter `ImageURL`, which is the information carried in this interaction. This implies that in this simulation experiment, only `ImageURL` is passed from the simulation environment to the emulated/real network. This experiment setup allows us to evaluate the impact of data corruption (e.g., by malicious attacks) on the CPS performance. If the experiment is only interested in the transmission delay of the data (the traffic load it incurs), instead of its actual content, we can set up another interaction model with lower communication overhead between the simulation and the emulation environment. In this case, no real data is exchanged for image transfer applications. Instead, dummy data is sent with the specified sending rate and packet size to generate the appropriate traffic load. Using the `SendImageToNetwork` `NetworkInteraction`, the simulated UAVs (`UAVFederate`) notify the network application process `SendImage` about two parameters: `FrameInterval` and `FrameSize`.

5. **RUN-TIME ENVIRONMENT**

Figure 8 shows the run-time environment of iSEE. The run-time environment is built on High-Level Architecture (HLA) [10], which is a standard framework for distributed computer simulation systems. Based on this standard, communications between different simulation tools are managed by Run-Time Infrastructure (RTI), which provides a set of services such as time management and data distribution. To integrate the simulation and the emulation platforms, HLA-compliant reusable components are developed as the interface between the simulation tool, the emulation platform and RTI, so that the simulation and emulation platforms become federates of RTI. The time synchronization and data communication between the simulation and the emulation environments is the key challenge in the development of iSEE. In our current design, the emulation gateway
federate will handle the data communication between the simulation environment and the DETERlab. RTI operates under the real-time mode to enforce the time advance of Matlab simulation to be at the same pace of the physical clock time.

5.2 Time Synchronization

CPS simulation is required to be run in the real-time mode, which means that the simulation will run at the same pace as the physical clock. Let \( t_s \) denote the simulation time, \( t_e \) the emulation time, and \( t_r \) the real (physical) time.

We propose a novel time synchronization mechanism, where the simulation runs under a virtual time that is separated from real time, but still keeps close pace with real time. In particular, the operating system time is synchronized with real time (\( t_e=t_r \)), while the simulation time is separated from real time. Figure 10 illustrates our proposed scheme. Assume that the communication delay from simulation environment to emulation environment is \( \delta_1 \) and \( \delta_2 \) is the delay from emulation to simulation. Also assume the simulation environment has a lag of \( L \) from real time (\( t_s=t_r-L \)). Simulation clock advances at the same pace as the real physical clock. All outgoing traffic events with time stamp \( t \) from the simulation environment will be actually scheduled/tunneled to the emulation environment at simulation time \( t-L \) to compensate the above lag so that it could arrive at the emulation host at real time \( t \). For incoming traffic with timestamp \( t+\tau \), it will arrive at the simulation environment at simulation time \( t_s = t - L + \tau + \delta_2 \). By making \( L \geq \delta_2 \), we can make sure that the event can be scheduled at time \( t_s = t + \tau \).

To implement this scheme, we have a time converter at each emulation host which converts the timestamp on each event. The time converter keeps the difference between the simulation time the emulation host OS time, which is initialized when the system starts. We use the following notations to illustrate the timestamp calculation:

- \( T_{os\_client\_start} \): the OS time (real time) when the TapClient registers itself to the TapServer.
- \( T_{os\_server\_start} \): the OS time at the TapServer when the client registration message is received.
- \( T_{s\_start} \): the value of the simulation time when the client registration message is received.

The difference between the simulation time and the OS time at the TapServer is \((T_{os\_server\_start} - T_{s\_start})\). We assume that the system clocks of the emulation hosts and the simulation host are synchronized via Network Time Protocol. Thus, this difference \((T_{os\_server\_start} - T_{s\_start})\) is also the difference between the simulation time and the real time at the TapClient. At the TapClient, each incoming event with timestamp \( t \) for scheduling will be converted from the simulation time to its emulation time with respect to the TapClient OS time according to the following relation.

\[
T_{timestamp\_emu} = T_{timestamp} + (T_{os\_server\_start} - T_{s\_start}) - L
\]

For outgoing event, the converter will convert its Timestamp with respect to the emulation time to the simulation time Timestamp as follows.
Timestamp = Timestampemu − (Tos_server_start − Tx_start) + L.

Here, L is the lag of the simulation time compared to the emulation time. As we have discussed earlier, L needs to be larger than δ2, the delay from the TapClient to the TapServer, which can be calculated as δ2 = Tos_server_start − Tos_client_start. In our experiments, the value of δ2 is around 30ms, and we use L = kδ2 (k = 2) considering the delay variation.

5.3 Data Communication

In the simulation environment, RTI handles the data communication, where the simulation federates and emulation gateways communicate with each other using publish/subscribe technology. The communication between the emulation gateways and the emulation environment is implemented via TCP connections to the control interfaces of the DETERlab nodes to avoid the interference to the data communication. In the DETERlab, the data communication are emulated between the data interfaces of the DETERlab nodes. In the example shown, for the traffic from Plant to Controller, the experiment data path is Plant + EmuGateway + Plant Emulation Host + Controller Emulation Host + EmuGateway + Controller. This routing path is reversed when the Controller sends traffic to the Plant.

6. EXPERIMENT

We design several experiments to evaluate iSEE system design and implementation. The CPS system used in the experiment is composed of an unmanned aerial vehicle (UAV), which is controlled over a network using a digital controller. The networked digital controller is designed to enable the UAV to track a desired reference trajectory. In these experiments, the simulation model Controller sends out reference signals, and the Plant will follow these signals. We record the signals received at the Plant side to evaluate the control performance.

6.1 Setup

Figure 11 shows the system topology and the related Tcl file. The PlantEmuHost represents the Plant emulation host, and ControllerEmuHost is Controller emulation host. All simulation environment, including RTI, Matlab federates, and EmuGateway federate are deployed in host named EmuGatewayHost.

6.2 Accuracy

In this experiment, we demonstrate the accuracy of iSEE experiments. In particular, we use the NCSWT [11] system as a benchmark. The NCSWT system is an integrated simulation platform for networked control system experiment, which uses ns-2 as the network simulator and Matlab as the control system simulator. The experiment is designed to compare the result produced by iSEE with the result produced by NCSWT.

In this experiment, the sampling period of the controller signal is 0.1 seconds. The network bandwidth is 100MB with no packet loss. Figure 12 plots the UAV positions as well as the reference trajectory. We can see from the figures that the experiment results produced from iSEE - the integrated Matlab and DETERlab environment are exactly the same as the results produced from NCSWT - the integrated Matlab and ns-2 environment.

6.3 Impact of Security Attacks

Now we demonstrate the usage of iSEE in evaluating the impact of security attacks. In this experiment, attack traffic is generated in the DETERlab to disrupt the communication between the UAV and the controller. The packet loss rate varies from 10% to 30%, and the results are shown in Figure 13. We can observe from the figure that as the loss rate increases, the UAV trajectory strays further from the reference trajectories.

7. CONCLUSION

The design and analysis of CPS is a critical task due to its distributed architectures and expanded mission capability. Network emulation environment can help realistically analyze the impact of network phenomena on the overall system performance. By integrating emulation environment with the simulation environment, we present iSEE, an integrated platform to evaluate CPS with greater realism in the network experiments. We present a case study using iSEE to evaluate our system performance. The results show that iSEE is capable of modeling and evaluating CPS accurately and performing security experiments. Currently, iSEE design only supports real-time experiment execution. In the future, we will enhance it with the support for fully virtualized time execution.

8. ACKNOWLEDGMENTS

We would like to thank Himanshu Neema, Gabor Karsai for helpful inputs and discussions on this work and anonymous reviewers for their feedback to improve the readability of the paper. This research is supported in part by the U.S. Army Research Office (ARO W911NF-10-1-0005), Lockheed Martin, NSF TRUST Science and Technology Center (CCF-0424422), NSF Grant OCI-1127396, the Air Force Research Lab Visiting Faculty Research Program and the Air Force Research lab Summer Faculty Extension Grant. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

9. REFERENCES

Figure 12: Output plots for reference, and plants in NCSWT and iSEE

![Output plots for reference, NCSWT, and iSEE](image)

(a) Reference Signals  
(b) Output plot for Plant in NCSWT  
(c) Output plot for Plant in iSEE

Figure 13: Plots of UAV trajectory for packet loss rates

![UAV trajectory plots for different loss rates](image)

(a) Output plot for 10% loss rate  
(b) Output plot for 20% loss rate  
(c) Output plot for 30% loss rate