Simulation of Network Attacks on SCADA Systems

Rohan Chabukswar*, Bruno Sinópoli*, Gabor Karsai[†], Annarita Giani[‡], Himanshu Neema[†] and Andrew Davis[†]
*Carnegie Mellon University
[†]Vanderbilt University
[‡]University of California Berkeley

Abstract—**Network security is a major issue affecting SCADA systems designed and deployed in the last decade. Simulation of network attacks on a SCADA system presents certain challenges, since even a simple SCADA system is composed of models in several domains and simulation environments. Here we demonstrate the use of C2WindTunnel to simulate a plant and its controller, and the Ethernet network that connects them, in different simulation environments. We also simulate DDOS-like attacks on a few of the routers to observe and analyze the effects of a network attack on such a system.**

I. Introduction

Supervisory Control And Data Acquisition (SCADA) systems are computer-based monitoring tools that are used to manage and control critical infrastructure functions in real time, like gas utilities, power plants, chemical plants, traffic control systems, etc. A typical SCADA system consists of a SCADA Master which provides overall monitoring and control for the system, local process controllers called Remote Terminal Units (RTUs), sensors and actuators and a network which provides the communication between the Master and the RTUs.

A. Security of SCADA Systems

SCADA systems are designed to have long life spans, usually in decades. The SCADA systems currently installed and used were designed at a time when security issues were not paramount, which is not the case today. Furthermore, SCADA systems are now connected to the Internet for remote monitoring and control making the systems susceptible to network security problems which arise through a connection to a public network.

Despite these evident security risks, SCADA systems are cumbersome to upgrade for several reasons. Firstly, adding security features often implies a large downtime, which is not desirable in systems like power plants and traffic control. Secondly, SCADA devices with embedded codes would need to be completely replaced to add new security protocols. Lastly, the networks used in a SCADA system are usually customized for that system and cannot be generalized.

Security of legacy SCADA systems and design of future systems both thus rely heavily on the assessment and rectification of security vulnerabilities of SCADA implementations in realistic settings.

B. Simulation of SCADA Systems

In a SCADA system it is essential to model and simulate communication networks in order to study mission critical situations such as network failures or attacks. Even a simple SCADA system is composed of several units in various domains like dynamic systems, networks and physical environments, and each of these units can be modeled using a variety of available simulators and/or emulators. An example system could include simulating controller and plant dynamics in Simulink or Matlab, network architecture and behavior in a network simulator like OMNeT++, etc. An adequate simulation of such a system necessitates the use of an underlying software infrastructure that connects and relates the heterogeneous simulators in a logically and temporally coherent framework.

II. C2WindTunnel

One infrastructure suitable for such an application is the Command and Control WindTunnel ([3]). The C2WindTunnel is an integrated, graphical, multi-model simulation environment for the experimental evaluation of congruence between organizational and technical architectures in large-scale C2 systems. It enables various simulation engines to interact and transmit data to and from one another and log and analyze the real time simulation results.

The C2WindTunnel framework uses the discrete event model of computation as the common semantic framework for the precise integration of an extensible range of simulation engines. Each simulation model, when incorporated into the overall simulation environment of C2WindTunnel, requires integration on two levels: the API level and the interaction level. API level integration provides basic services such as message passing, and shared object management, whereas interaction level integration addresses the issues of synchronization and coordination. C2WindTunnel offers a solution for multi-model simulation by decomposing the problem into model integration and experiment integration tasks. It facilitates the rapid development of integration models and use of these models throughout the lifecycle of the simulated environment. An integration model defines all interactions between federated models and captures other design intent, such as simulation engine-specific parameters and deployment information. This information is leveraged to streamline and automate significant portions of the simulation lifecycle. The integration modeling language combined with various sophisticated generation tools provides a robust environment for users to rapidly design and synthesize complex, heterogeneous command and control simulations.

A. High Level Architecture (HLA)

C2WindTunnel is based on the High-Level Architecture (HLA) IEEE standard 1.3 ([2], [4] and [5]) initially designed by Department of Defense (DoD) to ensure interoperability and reusability of models and simulation components. Reusability implies individual simulation models can be employed in different simulation scenarios, while interoperability implies an ability to incorporate simulations on different types of distributed computing platforms, with real-time operation.

A complex simulation can be considered as a hierarchy of components with increasing levels of aggregation. At the lowest level is the model of a system component implemented in software to produce a simulation, referred to as a federate. Several such federates form a part of an HLA compliant simulation, called a federation. There are three components of an HLA:

1) HLA rules to ensure proper interaction among federates and to delineate the respective responsibilities.
2) Object Model Template (OMT) to prescribe format and syntax for recording and communicating information.
3) Interface specification to define Run Time Infrastructure services and interfaces and federate callback functions.

B. Run-Time Infrastructure (RTI)

The software implementation of HLA is called a Run-Time Infrastructure (RTI). There are several commercial and open-source RTIs available in the market, some of which have been verified by the US Defense Modeling and Simulation Office.

The RTI is basically a collection of software that provides a set of commonly required services, described by the HLA Interface Specification, to multiple simulation systems. Apart from federation and object management, the RTI handles time management and co-ordinates the exchange of interacting events and data among the federates in a system.

1) Time Management: The time management services provided by the RTI ensure advancement of the simulation time in an orderly fashion among all the federates. Initially, the federate manager uses HLA-specified synchronization points to guarantee that all federates are ready to proceed with the simulation. Only when each federate has reported readiness to proceed with the simulation, does the federate manager allow all federates to commence the simulation.

2) Event and Data Interaction: A publish and subscribe mechanism is used by the HLA to manage the distribution of messages between the federates in a federation. Each federate defines to the federation what data is to be published for each update or event. Each federate declares to the federation which updates and interactions it is interested in receiving by subscribing to those messages.

C. C2WindTunnel Modeling Environment

C2Wind Tunnel uses a custom developed domain-specific modeling language (DSML) for the definition of integration models and the design details for the simulation environment. A simulation environment is composed of multiple federates each of which includes a simulation model, the engine upon which it executes, and some amount of specialized glue code to integrate the engine with the simulation bus. Both the engine configuration and the integration code needed for each federate is highly dependent upon the role the federate plays in the environment as well as the type of simulation engine being utilized. While manually developing the glue code is possible, by leveraging the integration model, C2WindTunnel is able to synthesize all of the code, greatly reducing errors and effort. A suite of tools called model interpreters, integrated directly with the DSML automatically generates engine configurations, glue code, as well as scripts to automate simulation execution and data collection. The integration model DSML combined with the generation tools provides a robust environment for users to rapidly define complex, heterogeneous command and control simulations.

The Generic Modeling Environment is the foundation for the C2WindTunnel environment. GME is a meta-programmable model-integrated computing toolkit that supports the creation of rich domain-specific modeling and program synthesis environments. Configuration is accomplished through meta models, expressed as UML class diagrams, specifying the modeling paradigm of the application domain. Meta models characterize the abstract syntax of the domain-specific modeling language, defining which objects are permissible in the language and how they compose. The meta model is a schema or data model for all of the possible models that can be expressed by a language.

Figure 1 shows the structure of a simulation undertaken using C2WindTunnel.

III. Simulation

In this section, we demonstrate the simulation of network security attacks on a SCADA system simulated using C2WindTunnel.

The SCADA system chosen was a simplified version of the famous Tennessee Eastman Control Challenge Problem, proposed by N. Lawrence Ricker ([1]). The original challenge problem requires co-ordination of three unit operations, with 41 measured output variables (with added measurement noise) and 12 manipulated variables. The control challenge presented by this case study is quite complex, but a simplified version was proposed by Ricker in 1993.

The process schematic is shown in Figure 2. It consists of an isothermal fixed volume reactor with a combined separation system, in which a single irreversible reaction occurs:

$$\mathrm{A} + \mathrm{C} \rightarrow \mathrm{D}.$$

The reactants A and C are non-condensible, and the product D is a non-volatile liquid. The reaction rate depends only on the partial pressures of A and C. There are two controlled feeds to the reactor chamber. Feed 1 consists of the reactants A and C, and traces of an inert gas B. Feed 2 consists of pure A, which is used to compensate for disturbances in the partial pressures of A and C in feed 1. The solubilities of A, B and C in D are negligible, so the vapor phase can be assumed to consist only of A, B and C, and the liquid, pure D. Thus, the

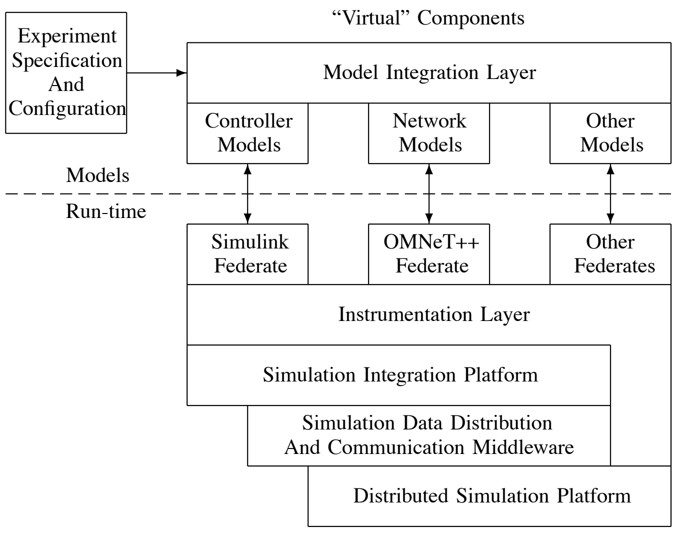


Fig. 1. C2WindTunnel Simulation Architecture

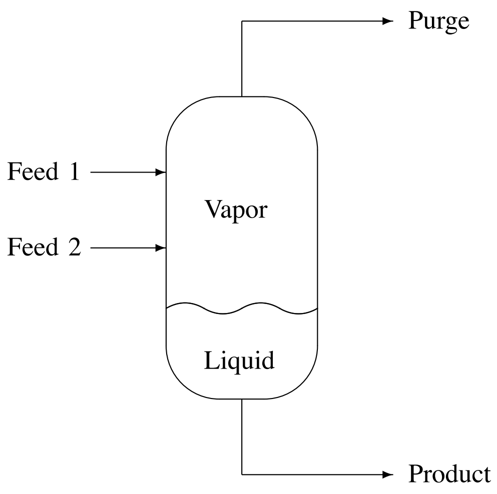


Fig. 2. Process Schematic, taken from [1]

only disturbance variables are the mole fractions of A, B and C in feed 1.

Isothermal conditions are maintained using independent controls. The product flow rate is adjusted using a proportional feedback controller which responds to variations in the liquid inventory. The purge rate depends on the pressure in the vessel and the position of the purge control valve.

Measured outputs include the four flow rates, pressure, liquid holdup volume, and the fractions of A, B and C in the purge flow. This composition measurement offers the pure time delay as in the original TE problem, as the simulated chromatograph operates on a 6 minute cycle.

The control problem is to maintain the product flow rate at a specified value by manipulating flows of feed and purge streams, and the liquid holdup volume. The restrictions come from the physical aspects of the plant: the operating pressure must be kept below the shutdown limit of 3 MPa, and the flow rates saturate at some point. A higher level control objective is to minimize operating costs, which are a function of the purge losses of A and C.

Tables I, II, III and IV list the different variables of the system.

In his paper, Ricker derives a linear time-invariant dynamic model of the plant at the base-case state. The LTI model matches the impulse response of the nonlinear plant well. The robust model-predictive controller is one of several proposed in [1]. It uses only four out of the ten available sensor outputs, and controls all four manipulated variables. The variables which necessarily must be monitored include the production rate (F_4), the pressure (P) and the liquid inventory (V_L). Failure to do so might upset other variables and profitability, or will allow violation of bounds on pressure and liquid holdup volume. The fourth variable chosen is the amount of reactant A in the purge flow (y_{A3}), though amounts of any of the other two components would perform just as well. The final structure of the simplified model as derived in [1] is:

$$\mathbf{y} = \begin{pmatrix} F_4 \\ P \\ y_{A3} \\ V_L \end{pmatrix} = \mathbf{Gu} = \begin{pmatrix} g_{11} & 0 & 0 & g_{14} \\ g_{21} & 0 & g_{23} & 0 \\ 0 & g_{32} & 0 & 0 \\ 0 & 0 & 0 & g_{44} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix}. \quad (1)$$

The individual transfer functions are given below (the unit

TABLE I
STATE VARIABLES (TAKEN FROM [1])

Variable	Nominal Value	Description	Symbol	Units
x_1	44.49999958429348	Molar holdup of A	N_A	kmol
x_2	13.53296996509594	Molar holdup of B	N_B	kmol
x_3	36.64788062995841	Molar holdup of C	N_C	kmol
x_4	110.0	Molar holdup of D	N_D	kmol
x_5	60.95327313484253	Feed 1 valve position	χ_1	%
x_6	25.02232231706676	Feed 2 valve position	χ_2	%
x_7	39.25777017606444	Purge valve position	χ_3	%
x_8	47.03024823457651	Product valve position	χ_4	%

TABLE II
MANIPULATED VARIABLES (TAKEN FROM [1])

Variable	Nominal Value	Purpose	Range
u_1	60.95327313484253	Changes feed 1 valve position	0 – 100%
u_2	25.02232231706676	Changes feed 2 valve position	0 – 100%
u_3	39.25777017606444	Changes purge valve position	0 – 100%
u_4	44.17670682730923	Liquid inventory set point	0 – 100%

TABLE III
OUTPUT VARIABLES (TAKEN FROM [1])

Variable	Nominal Value	Description	Symbol	Units	Range
y_1	201.43	Feed 1 flow measurement	F_1	kmol/hr	0 – 330.46
y_2	5.62	Feed 2 flow measurement	F_2	kmol/hr	0 – 22.46
y_3	7.05	Purge flow measurement	F_3	kmol/hr	Complicated
y_4	100.00	Product flow measurement	F_4	kmol/hr	Complicated
y_5	2700.00	Pressure	P	kPa	$<$ 3000
y_6	44.18	Liquid inventory	V_L	% of maximum	0 – 100
y_7	47.00	Amount of A in purge	y_{A3}	mol %	0 – 100
y_8	14.29	Amount of B in purge	y_{B3}	mol %	0 – 100
y_9	38.71	Amount of C in purge	y_{C3}	mol %	0 – 100
y_{10}	0.2415	Instantaneous cost	C	\$/kmol	$>$ 0

TABLE IV
DISTURBANCE VARIABLES (TAKEN FROM [1])

Variable	Nominal Value	Description	Units
y_{A1}	0.485	Mole fraction of A in feed 1	—
y_{B1}	0.005	Mole fraction of B in feed 1	—

of s is assumed to be hr^{-1}):

$$g_{11} = \frac{1.7}{0.75s+1}, \tag{2}$$

$$g_{21} = \frac{45\,(5.667s+1)}{2.5s^2+10.25s+1}, \tag{3}$$

$$g_{23} = \frac{-15s-11.25}{2.5s^2+10.25s+1}, \tag{4}$$

$$g_{32} = \frac{1.5}{10s+1}e^{-0.1s}, \tag{5}$$

$$g_{14} = \frac{-3.4s}{0.1s^2+1.1s+1}, \tag{6}$$

$$g_{44} = \frac{1}{s+1}. \tag{7}$$

(The transfer function g_{23} is not given in [1], probably due to oversight. It was estimated using the method described in the paper.)

Ricker provides a FORTRAN simulator of the plant model. This was converted to C code and used as a S-Function block to create a Simulink model of the plant.

The plant has a very high time constant, as is characteristic of chemical plants. To properly study the effects of a network attack, we need a system which can respond to disturbances during network attacks of duration in minutes. To this end, the time constants for the plant were multiplied by 60, so that changes which took hours now take the same number of minutes to occur. The transfer functions were then suitably modified to convert the unit of time to seconds. The final equations used were:

$$g_{11} = \frac{0.02833}{45s+1}, \tag{8}$$

$$g_{21} = \frac{45\,(340s+1)}{9000s^2+615s+1}, \tag{9}$$

$$g_{23} = \frac{-900s-11.25}{9000s^2+615s+1}, \tag{10}$$

$$g_{32} = \frac{1.5}{600s+1}e^{-6s}, \tag{11}$$

$$g_{14} = \frac{-3.4s}{360s^2+66s+1}, \tag{12}$$

$$g_{44} = \frac{1}{60s+1}. \tag{13}$$

Using Matlab, a minimal state space model of the system was constructed, which was then discretized to run at one-hundredth of a second. This is the system assumed in order to implement the controller as a Discrete State Space System block in a separate Simulink model. The A, B, C and D matrices for the controller were then calculated by using a Kalman state estimator and linear quadratic state feedback regulator system.

To complete the system, an Ethernet network was added for communication between the plant and its controller. The network was designed to be a realistic implementation of one in a chemical plant, where a single router would collect data from plant sensors which are physically close to it (and to each other). Similarly, it would send manipulation data to the valves which are physically close to it. There are four routers which route data from different sensors and to different control inputs. There is a three-level hierarchy in the network map, with a master controller distributing data to and collecting data from other routers at the plant site. Two relay routers are employed between the master router at the plant site and the router that communicates to the controller. To keep the network model simple, no redundancy was employed. The network model was simulated in OMNeT++, a generic discrete event simulation package using INET network protocols ([6]). The schematic of the network model is given in Figure 3.

A. Model Integration

The integration of models in different simulation environments has been described in more detail in [3].

1) OMNeT++: NetworkSim, a network simulator based on OMNeT++, provides a set of high-level communication protocols while maintaining full network stack simulation internally. NetworkSim utilizes network models built using OMNeT++. It translates messages from the RTI into appropriate network actions and vice versa, and injects these messages onto the correct simulated network node. This mechanism isolates the simulated network traffic from the general RTI traffic. Each OMNeT++ model deployed onto NetworkSim must have some code synthesized for integration with the RTI. When simulated via NetworkSim, some of the connected nodes in OMNeT++ become end-points, responsible for passing messages between the RTI and the OMNeT++ engine. The code that implements the communication between the RTI and these end nodes must be generated by the integration software.

A GME-based interpreter traverses the C2WT integration model and generates the C++ code needed for end-point nodes within an OMNeT++ model. For each federate, the integration model provides information about which interactions may be sent or received and which objects attributes may be published or updated. The interpreter understands these relationships and synthesizes code for each end-point in an OMNeT++ federate. The generated code builds upon the OMNeT++ API and is compiled directly into NetworkSim. Apart from the glue code, evolution of the OMNeT++ internal simulation clock must also be synchronized with the RTI. As a part of HLA compliance, NetworkSim includes a reusable class that extends the basic OMNeT++ scheduler. The function `getNextEvent()` is called by OMNeT++ to determine the next event, originating either internally or externally. If the timestamp on the next message places it outside of the window of time granted by the RTI, then a time advance is requested. An internal dispatch mechanism routes all RTI interactions to the appropriate OMNeT++ protocol module which interprets them and can schedule new internal OMNeT++ messages. A similar mechanism interprets and routes OMNeT++ messages bound for external dispatch into the RTI. Using these mechanisms both the evolution of time and message passing within an OMNeT++ federate is tightly coordinated via the RTI with the federation.

2) Simulink: Like in the integration of the OMNeT++ simulation engine, all of the engine-specific glue code is generated based on the overarching integration model. The GME-based model interpreter generates code that, in conjunction with several reusable Java generic classes, is used to directly integrate any Simulink model with a C2WT federation. The generic classes provide all of the fundamental RTI integration requirements: providing interfaces for converting between Simulink types and RTI types, encapsulating interfacing with the RTI for initializing the federate, synchronizing the Simulink engine's simulation clock, and managing any publish-and-subscribe relationships with other federates.

Within any given Simulink model the user must insert an S-function block for each interaction to which the model either publishes or subscribes via which blocks that the Simulink engine can interact with the rest of the federation. The modeler specifies whether the block either publishes or subscribes an interaction by instantiating the corresponding sender or receiver S-function from those that were generated from the integration model. The modeler must also specify which interaction the S-function block should call by passing the name of the interaction via a string parameter to the block. The naming convention of the `.m` files and of the parameters is standardized and easily derived from the primary C2WT model.

Once the S-function blocks have been incorporated and their values set, no further manual steps are typically necessary to prepare the model to be integrated. Some effort has to be spent to properly order the signals entering and exiting the S-

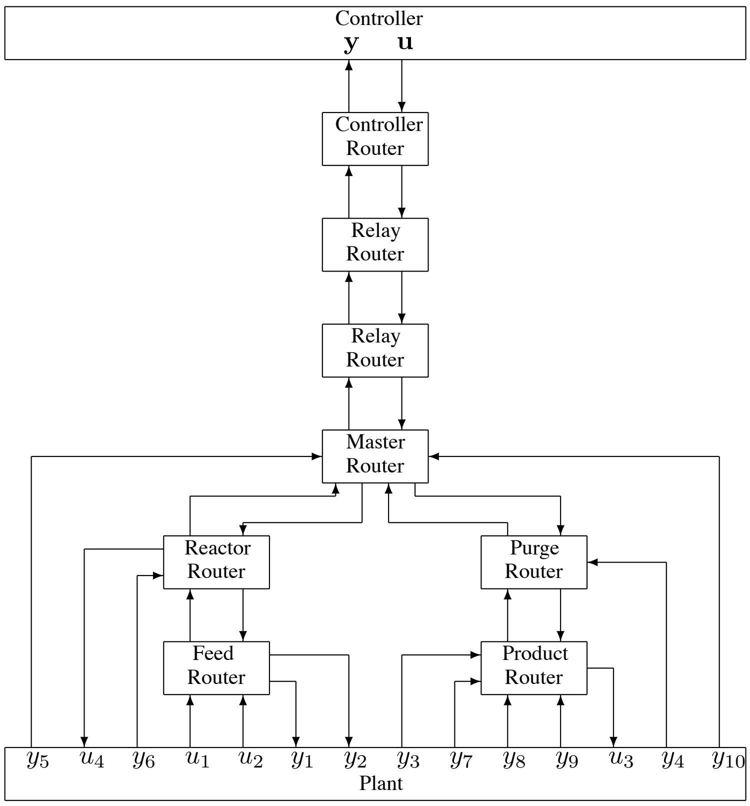


Fig. 3. Network Map

function blocks so that they correspond to the attribute ordering of the corresponding RTI interaction. The key mechanism for synchronizing the clock progression of the Simulink model with that of the RTI is the basic time-progression model for S-function blocks. During its execution, the Simulink engine consults each block in a model about when it can generate an output. With all S-function blocks, code must be supplied to respond to this request from the engine. The synthesized integration code in an S-function block uses this method to synchronize the model with the RTI and allow simulation time within Simulink to progress only when the RTI allows it to proceed. Until the RTI allows federation time to progress, we do not return from the method call within the S-function block, thus not allowing the Simulink engine to progress. We keep the Simulink engine step-size low (typically 0.1 seconds) to minimize any event timing errors due to the passing of input and output events between the Simulink model and the HLA. For incoming events, the glue code uses a polling scheme at every time step to check if the federate has received an input from the remainder of the federation. Very small step-sizes in any Simulink model can lead to a significant slowdown in simulation speed. In the context of the C2WT, possible performance penalties due to having small step-sizes must be weighed against minimizing timing errors due to overly large time-steps.

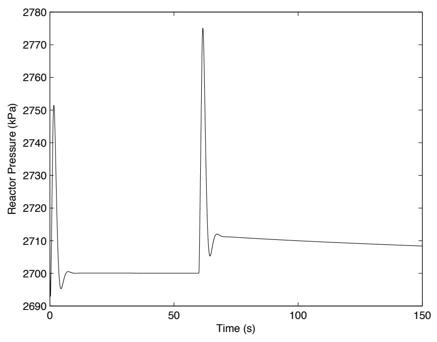
B. Attacks

Several DDOS-like attacks are simulated on the SCADA system, targeting various routers of the network. In each such attack, the target is saturated with external communication requests from a large number of zombie nodes so that it cannot handle the legitimate traffic of the system, or at least, is rendered so slow in handling the traffic, that it is effectively unavailable for transfer of legitimate data. The targets, durations and the number of attacks in the simulation are specified beforehand. In these simulations, the controller, feed and product routers are attacked. In each case, the simulation was run for 150 seconds, and attack started at the 30-second mark and continued till the 60-second mark.

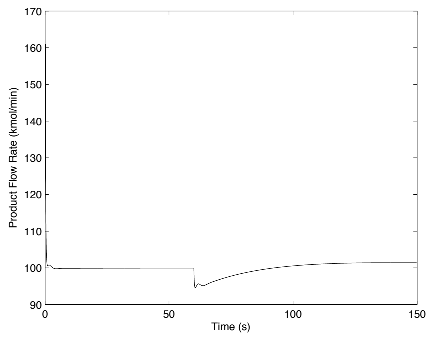
IV. Observations

When one of the routers is under a full-fledged DDOS attack, the network is essentially broken at that point. The controller will be rendered blind to sensors from which the router collects data. The plant will also be rendered unresponsive to such controller commands as are handled by that router. This will result in a loss of the regulatory function of the controller, which can potentially cause a variety of damage to the plant, from an unwanted change in the operating cost and production rate, to physical damage of plant equipment.

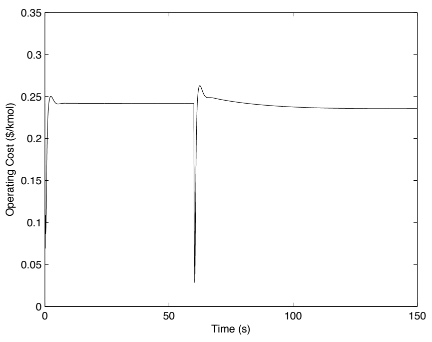
In case the target is one of the routers which handle all of the data (controller, master or relay routers), such an attack causes a complete loss of communication between the plant and



(a) Pressure

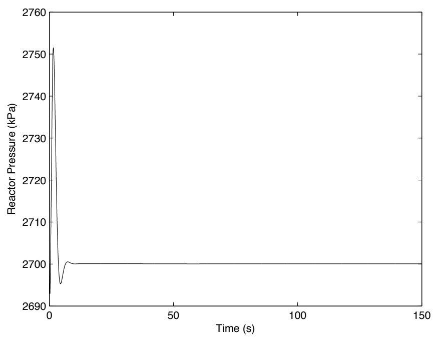


(b) Product Flow Rate

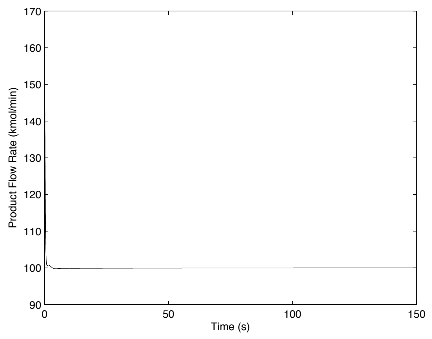


(c) Operating Cost

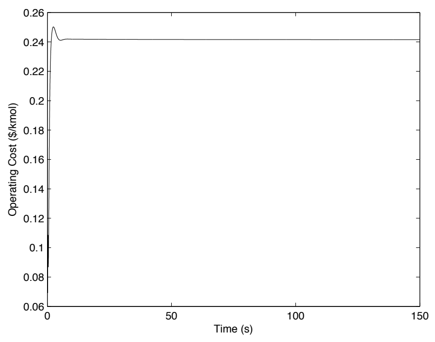
Fig. 4. Attack on Controller Router, from 30s through 60s



(a) Pressure



(b) Product Flow Rate



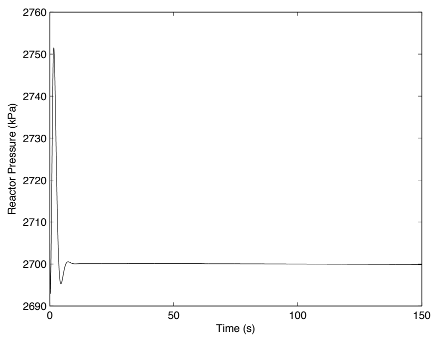
(c) Operating Cost

Fig. 5. Attack on Feed Router, from 30s through 60s

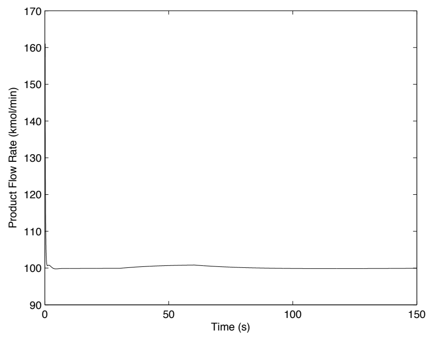
controller. The plant undergoes a severe change of state when the attack begins, from which it recovers and resumes normal operation. Such an attack is the first to be simulated. The effects can be observed in the change in pressure, production rate and operating cost over time, especially during the attack (Figures 4a, 4b and 4c). In case other routers are targeted, the controller will generate outputs based on the sensors outputs it has, and will try to control the inputs which are not unresponsive. These effect can be observed by monitoring the operating cost during the simulation.

The next simulation involves attacking the feed router, which blocks the feed 1 and feed 2 flow measurement sensors (which are not used by the controller), and the valve 1 and valve 2 controllers (u_1 and u_2). The controller is thus not blind to any of the required sensors, but its regulation function could be hampered by it not being able to control the two valves. The effects can be observed by monitoring the same sensor outputs over time (Figures 5a, 5b and 5c). We see that due to the robust nature of the controller, an attack on the feed router has no effect on the state of the plant, which continues to operate normally despite the attack.

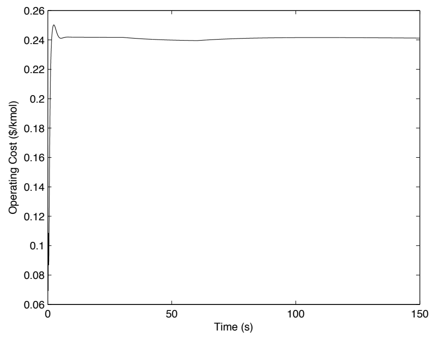
The last simulation involves attacking the product router, which blocks several sensors, the only required one of which being the amount of A in purge (y_7), and the purge valve controller (u_3). The controller is thus blind to one of the required sensors, and it is not able to control the purge valve. The effects of this attack are different than the previous two simulations. The plant goes into an uncontrolled state for



(a) Pressure



(b) Product Flow Rate



(c) Operating Cost

Fig. 6. Attack on Purge Router, from 30s through 60s

the duration of the attack, from which it can recover and resume normal operation only after the attack has ceased. The effects can be observed by observing the usual output variables (Figures 6a, 6b and 6c).

V. Conclusion

A DDOS-like attack was simulated on a plant and controller system and the effects of attacks on different routers were observed. While the effects of the total communication disruption might have been estimated, the effects of the other two attacks are harder to predict. The same attack on different routers causes no change in one case and severe problems in another. If the system were more complicated, then obtaining the effects would require intensive analytical computations, or indeed, could very well be intractable. In such a case, a simulation is the best way to estimate the effects, and to implement and compare different network configurations and redundancies.

The chemical plant was thus a proof-of-concept implementation of a simulation system composed of models in different domains and environments. The use of C2WindTunnel facilitated the interaction and data transfer between the environments, and in setting up the attacks and monitoring the response.

VI. Future Work

The results of the simulation can be used to analyze the current network and controller, and develop more robust control algorithms and improve the network, for example by using redundancies. The SCADA system itself might be expanded to employ a Fault Detection and Isolation and/or an Intrusion Detection System.

The attack that was simulated is one attack on the availability of a system. Future work involves observing the effect of other common network security attacks on integrity and confidentiality of the data as well, like eavesdropping, misdirection and spoofing.

Another direction for future work involves simulation of systems including hardware-in-the-loop.

Acknowledgment

This work was supported in part by TRUST (Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422) and the following organizations: AFOSR (#FA9550-06-1-0244), BT, Cisco, DoCoMo USA Labs, EADS, ESCHER, HP, IBM, iCAST, Intel, Microsoft, ORNL, Pirelli, Qualcomm, Sun, Symantec, TCS, Telecom Italia and United Technologies.

References

[1] N. Lawrence Ricker, *Model predictive control of a continuous, nonlinear, two-phase reactor*. Journal of Process Control, Volume 3, Issue 2, May 1993, Pages 109-123.

[2] J. O. Calvin, R. Weatherly, *An introduction to the high level architecture (HLA) runtime infrastructure (RTI)*. Proceedings of the 14th Workshop on Standards for the Interoperability of Defence Simulations, Orlando, FL, March 1996, pp. 705-715.

[3] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, G. Karsai, *Rapid Synthesis of HLA-Based Heterogeneous Simulation: A Model-Based Integration Approach*. in review for Simulation.

[4] R. Crosbie, J. Zenor, *High Level Architecture*. http://www.ecst.csuchico.edu/~hla/.

[5] *HLA standard - IEEE standard for modeling and simulation (M&S) high-level architecture (HLA) — framework and rules*. IEEE Std. 1516-2000, pp.i-22, 2000

[6] *OMNeT++ Simulation Package*. http://www.omnetpp.org/