Co-Emulation of Network Control Systems Using DETER

Darrel Brower, Suzanna Schmeelk, Saurabh Amin, Blaine Nelson, John Rivera, Joy Wang
University of California, Berkeley

Abstract – In order to experiment with attacking and defending network control systems, we need a topology and software to work with, where the software could be simulated or emulated. This article presents a comparison of a simulated network control system on a DETER topology to an emulated version of the same network control systems on DETER, and gives results from the use of both.

Introduction

In the past decade, energy companies have shifted from using proprietary network control systems towards using ubiquitous internet-based technologies to manage their electricity production and delivery. By doing so, these companies are reducing both time and money costs, since they do not have to pay the high costs of proprietary systems, and they can reduce the amount of training to give to manage their devices. Even more importantly, energy companies are able to leverage the internet to provide intelligent power delivery, which increases the efficiency of energy consumption, among other benefits.

With benefits, however, come risks. By relying on the internet, energy management systems are exposed to the risk of cyber attack because all an attacker has to do is probe for weaknesses and then manipulate the management system. By compromising the system, an attacker could, for example, direct electricity to a building without its billing being accounted for. Or the attacker may choose to disable the power to an individual customer. At worst, the attacker might decide to disable or destroy the power plant by sending malicious data.

Despite these risks, however, energy companies have still chosen to deploy internet-based technologies. This is due to several reasons. First, the world's energy consumption is expected to "increase worldwide by 44% … by 2030" (Lowe, 2010). To supply this need, energy companies are interested in intelligent power technology, such as smart grids, to reduce waste. Second, energy companies are also compelled to become more efficient due to governments government regulation such as the Kyoto Protocol (UNFCCC, 1998). Additionally, despite attempts by some power companies to isolate their network control systems, employees continue to connect the network control system to the internet, sometimes inadvertently. Realizing the reality that their systems will be connected to the internet in some way, most energy companies have chosen to embrace it.

Experiment Setup

Because of this unavoidable situation, we decided to research how to mitigate the risks. However, to research risk mitigation, we need a testbed, a network topology modeled after a major network, and a customization of the topology to make it a *control system*. Control systems "are computer-based systems that monitor and control physical processes" (Saurabh Amin, 2008). For the purposes of our experiments, this would be a power plant on one end of the network, and a controller on the other. Combined, these three components allow us to create varying network environments, control experiment factors such as time, and introduce events.

For the topology, we chose the Abilene network, which is a "high-performance backbone network the enables the development … and the deployment of leading-edge network services by .. universities and research labs across the country" (Internet 2, 2005). We chose this topology because it was designed for scientists.

With a topology decided, we determined that the DETER testbed would meet our experiment requirements. The DETER testbed is "a dedicated network testbed facility customized for cybersecurity research" (Terry Benzel, 2009). DETER uses a simulation platform called ns, which is a platform based on the programming languages TCL and C. Using simple TCL-like commands, we can create nodes, link them, and run software on top of them. These commands are mapped to real machines and virtual switches, which run the commands. The result is a contained network of nodes that behave exactly like a network connected to the internet.

For the control system, we found plant and controller simulation software called Network control system, or NCS. Written by Vincenzo Liberatore, this software allows for the creation of plant agents on the simulation platform ns-2, and has been proven to work correctly as a scalar network control system.

Definitions

To *simulate* a system means "to reproduce the conditions of (a situation, etc), as in carrying out an experiment" (Collins English Dictionary, 2009). For the purposes of our experiments, to simulate means to have one computer create virtual computers with virtual links to connect them. None of the virtual components are mapped to any whole physical entity, and all of the computation and communication is done within the computer.

To emulate a system means "to make one computer behave like (another different type of computer) so that the imitating system can operate on the same data and execute the same programs as the imitated system" (Collins English Dictionary, 2009). For our experiments, this means each node imitates a specific machine, and each link imitates whatever properties are desired, such as slower data rate or poor packet conveyance.

Network Control System Behavior

The central concept to the NCS software is the "plant". After deploying plants on nodes, we change the roles of each plant to be either a "controller interface, sensor, or actuator depending on its usage in the simulation script" (Liberatore, 2002). Each plant holds four values: state, plant input, time of last sample, and a stabilizing constant. These factors correspond to variables x_0, u, t_0, and a, respectively. The state represents the critical plant feature being output, such as an electrical load or temperature. The plant input represents a control on the state, much like a thermostat controls a heater's heat production. The time of last sample holds the exact time that the state was determined from sampling. The stabilizing constant is used to take into account delays in communication and calculation when the plants communicate.

Each plant responds based on one of two functions. The first function, sysphy, is invoked when the plant receives a request to update its state from another plant, while the second function, smplschd, is invoked when a plant is told by another plant the time to sample itself again (if any). When sysphy is called, or the set time in smplschd has occurred, the plant will sample itself using the following equation:

$$x_1 = \left(x_0 + \frac{u}{a}\right) \times e^{a \times \Delta t} - \frac{u}{a}$$

Where x_1 is the new state, and Δt is the difference between the current time and t_0.

After computing this new state, the plant sends it to the controller so it can compute a new input. The controller uses the same equation as above, but additionally multiplies the result by a calibration number, such as -.8, to the result in order to stabilize the plant. The controller sends this new result to the plant, which updates its *u* value accordingly. Additionally, the controller sends a future sampling time value t_1, which the plant uses to sample itself in the future.

As the plant and controller communicate, the plant's state fluctuates while the controller keeps it stable. However, if communication is severed, the simulated plant's state will continue to increase by itself until a set point, when the plant program detects that the state is so high that it should both quit and notify the user that the plant exploded.

Implementing a Simulated NCS

Because the NCS software runs on ns-2, and the DETER testbed is built on the original version of ns, we needed to find a way to get the software to run. Eventually, we decided to try installing the ns-2 platform on each DETER node that would have an actuator, sensor, or controller, and then running the NCS software on top of that. If we could get a node to run the software, then we would try rerouting the communications of the simulated components to transmit over the DETER network, instead of within simulated links within the node.

However, due to differences between the version of ns-2 that the NCS software runs on, and the current version of ns-2, there are a few correctable discrepancies with the PlantAgent code. Despite these, we find that the NCS software runs correctly on DETER, and we are able to generate useful data within a node. However, because the components are running contained within the simulation program, we do not have a way for the simulated components to transmit between DETER nodes, and are unable to continue using the NCS software for security testing. To remedy this problem, we decided to write an emulated version of the NCS software.

Implementing an Emulated NCS

Averaging Program

We know that the programming language python is installed on each node of the DETER testbed. We also have code for a threaded echo server and echo client (Martelli, 2006). To make sure emulation works correctly on DETER, as well as to better understand the python code, we modify the client and server scripts to be a simple averaging program, where we call the client and server a plant and controller, respectively.

For the plant script, we initialize a state value to zero, and increment once per second. Immediately after incrementing, we send the state and the current time to the controller. During the entire run of the program, the plant is connected to the controller's TCP socket. Since this program only tests proper communication and some simple attack effects, the plant program does not expect feedback from the controller.

For the controller script, we initialize a variable *total* to zero. Upon receipt of a packet from a plant, the controller checks the time, enclosed in the payload. If the time is within one second of the controller's current time, the state value, in the packet payload, is added to the total. This behavior is applied to all plants connected to the controller. After one second, the total is divided by the number of plants, with total reset to zero. To get a proper average, the

controller is reliant on all plants starting at the same one second interval so that each plant transmits the same number during each interval.

Though simplistic, this program is useful in detecting when a disrupting event occurs. Under an ideal communication cycle, where all plants successfully transmit their state to the plant within the one-second interval, the controller's average will be the same as one of the states transmitted to it. If a plant's state should not make it to the controller in time, however, then the average will be reduced by the percentage of the number of successful plant transmits to the total number plants connected.

Upon running of the scripts, we determine that DETER is capable of reliably running an emulated version of the plant and controller. We also determine that the echo scripts are capable of being used as plants and controllers.

Emulated Plant and Controller Program

To emulate the NCS software, we translate the TCL-based NCS code into python code and integrate it into a copy of the original echo client and server scripts. Because the NCS software has some noticeable differences from the python scripts in execution, we provide the sequence of events during normal operation.

At plant start up, the plant handshakes with the controller by sending its *u* and *a* value to the controller. If the controller receives the packet, it creates a separate connection for the plant, stores the sent values, and sends the word "ACK" back to the plant. If the plant receives the ACK packet, it begins normal operation. However, if the plant doesn't receive the ACK within one second, or if it receives any differing packet, the plant outputs an error and terminates.

During normal operation, the plant waits to sample its state until it either receives a message from the controller, or the current time equals its future time variable, t_1. If either of these two conditions occur, the plant samples its state by using the ordinary differential equation, described in the simulation section, and updates its state to this new value. If the new state value is above a certain value, the plant sends a message to the screen that it has exploded, and terminates. Otherwise, the plant immediately sends its state, along with the current time, to the controller.

During the sampling, the plant determines the next time to sample based on how the sampling is triggered: if the plant's sampling is triggered by t_1, the plant sets t_1 to be the current time, plus an interval constant set at the beginning of program execution. However, if the sampling is triggered by the controller, the plant updates t_1 to be the sample time that is inside the controller packet. Finally, the plant updates t_0 to be the current time.

Whenever the controller receives a packet from a plant, it first checks the time located within the payload to make sure that the packet data are not old or invalid. If the data are timely, the controller find the plant's *u* value the same way that it did in the NCS program: it takes the packet's state value, and finds the plant's future state using the same ordinary differential equation. It then multiplies the future state by a predetermined constant, such as -.8, which results in the plant's updated *u* value. It sends this value back to the plant, along with the time that the plant should send its next value, which is the current time plus an interval constant. When the plant receives the controller's packet, it updates its *u* and t_1 value with the values in the packet payload.

We successfully implemented these behaviors into the python scripts, and have a working network control software to test on the DETER testbed.

Improved Plant and Controller

After creating the plant and controller, we find that we can customize the code, to allow for rapid experimentation, while still retaining the codes accuracy. For example, In the original code, the plant runs forever. Sometimes, it is useful to run a plant for a only limited time period, and deem that plant stable if it doesn't explode. We therefore add a method where the plant first grabs all of its initial configuration data, including maximum time period to run, from the file *plant.conf* before connecting to the controller. Furthermore, experiments showed that varying the values of the calibration variable *a* is be useful, as its value is a major factor in stability. We thus add a method for the plant to acquire a range of *a* values from the file *testvals.conf*. Using these two methods, a plant is able to run with customized settings for a specified time. At the end of the time, it terminates. If there is another *a* value, the plant restarts with the new *a* value, including reloading its configuration data from *plant.conf*. Finally, both the plant and controller have logging ability, whereby the plant saves any new data to a file called *plantstat.conf*, while the controller saves data it receives and sends to one or more conf files with the name of the sending or receiving plant as the filename.

Upon running these improved scripts, we find that the plant and controller function correctly. Moreover, we find that the customized values allow us to observe plant behavior, by analyzing the state trends for each custom plant.

Results

We find that both simulation and emulation of a network control system is possible on the DETER testbed. We also find that the emulated version is preferable to the simulated version, as the emulated version yields more realistic results, and is easily customizable. We are excited by the increased experiment possibilities given by the emulated version, and expect the emulated version to be relied on for future experiments.

Acknowledgments

Special thanks goes to John Rivera, Jue Wang, Vincenzo Liberatore, the DETER team, the Team for Research in Ubiquitous Secure Technology, and the University of California, Berkeley.

References

Amin, Saurabh. A. A. (2008). Research Challenges for the Security of Control Systems. *Hot Topics in Security* (pp. 2-6). San Jose: USENIX Association.

Benzel, Terry. B. B. (2009). Current Developments in DETER Cybersecurity Testbed Technology. *isi.edu* , 11.

Collins English Dictionary. (2009). *emulate.* Retrieved July 29, 2010, from dictionary.com: http://dictionary.reference.com/browse/emulate

Collins English Dictionary. (2009). *simulate.* Retrieved July 29, 2010, from Dictionary.com: http://dictionary.reference.com/browse/simulate

Internet2. (2005, February). *200502-IS-AN.* Retrieved July 29, 2010, from Internet2: http://www.internet2.edu/pubs/200502-IS-AN.pdf

Liberatore, V. (2002). Network Control Systems. 1-5.

Lowe, J. S. (2010). The Future of Oil and Gas Law. *Washburn Law Journal* , 235-45.

Martelli, A. (2006). *Python in A Nutshell, 2nd Edition.* New York City: O'Reilly.

UNFCCC. (1998). *Kyoto Protocol.* Retrieved July 29, 2010, from UNFCCC: http://unfccc.int/resource/docs/convkp/kpeng.pdf

World Bank. (2010, July 26). *Energy Consumption in the World*. Retrieved July 29, 2010, from Google: http://www.google.com/publicdata?ds=wb-wdi&met=eg_use_elec_kh_pc