Modeling Plant Stability in a Networked Control System

Joy Wang

Graduate Mentors: **Saurabh Amin, Blaine Nelson, Dr. Suzanna Schmeelk**

Faculty Mentor: **Professor S. Shankar Sastry**

July 30th, 2010

Research Experience for Undergraduates (REU) Program 2010

Team for Research in Ubiquitous Secure Technology

Department of Electrical Engineering and Computer Sciences

University of California – Berkeley

Modeling Plant Stability in a Networked Control System

Jue Wang

Abstract — Assurance of system stability is of paramount importance in every control system. Without the maintenance of stability, plants could easily break down and explode, resulting not only in wasted time and capital, but the potential endangerment of human lives as well. This research project focuses on modeling plant stability through the emulation of a control system and the simulation of control behavior. Preparations for the investigation involve the creation of Python-scripted software that establishes the client-server relationships, automates communication between them, as well as synchronizes the timing of task delegation. The behavior of the control system is modeled by an ordinary differential equation, where a spring constant A directly affects plant stability. After the construction of the network is complete, the software is deployed on a given isolated network and tested repeatedly for stability. Results show that stability is guaranteed when the model contains a value of A between 0 and 1 for a simple two-node network. In addition to this finding, trial runs of the software were also conducted on different topologies to show an inverse relationship between information latency and stability. This paper investigates the limiting values of constant A for plant stability under different network conditions.

I. Introduction

Control systems are one of the most accurate creations of monitoring physical processes and serve as important computer-based structures in today's technological world [1]. Many control systems depend on the presence of sensors, such as heat sensors, speed sensors, and lighting sensors. In the absence of sensors, control system behavior may be simulated on a networked control system, which makes it possible to simulate physical systems attached to a network [2].

The goal of the overall project was to emulate a network, simulate network control behavior, deploy software, and attack the network. Simulation of the Abilene Network is performed to investigate and experiment with traffic behavior on an isolated network. The simulation is conducted on the DETER testbeds using the SEER application to monitor traffic and collect information [3]. Preparations for the

investigation involve python and TCL scripting to establish the client-server relationships, automate communication between them, as well as synchronize the timing of task delegation. After the construction of the network is complete, basic denial-of-service attacks are implemented to study and analyze the behavior of the network. The overall DETER summer project involves the procedures used in simulating the Abilene Network, examines the effects of DOS attack on the network, and discusses future developments and research in network security.

The goal of this specific paper is to discuss the emulation and simulation of a network and the stabilizing behavior of such a network, made possible by networked control systems [2]. The task will be to emulate networks by creating experiments of various network topologies, simulate network control behavior, and evaluate the range of the plant stabilizing constant A for those networks. The ordinary differential equation used to model the control system is given in the appendix.

II. Methodology

In order to simulate control system behavior, the first step is constructing the software needed to be installed onto the nodes. Python scripts were created for both plant and controller to manage the nature of their client-server communication [4]. The code for the plant starts off with gathering configuration information, which includes the plant's current state, input, identity, sample period, maximum value of state, and test duration. These properties define each testing session while running the software.

After the plant collects the configurations, it will initiate communication with the controller by updating the controller with its current state information. The configuration specifies the time period during which the plant should sample itself and report back to the controller From there on, regular updates will commence from plant to controller as the plant sends its current state and the controller sends back the state the plant should be at for the calibration of the plant. The script for the controller complements that of the plant and behaves like a threaded echo server that constantly listens for new plants. The controller recognizes a plant by the initial information exchange of the plant's input value and calibration constant A. If the exchange is successful, then the controller will wait for the plant to send its current state value and start the process of state control as described above.

Once the scripts are written and made compatible for the DETER testbed, the software will be ready for installation onto the computer nodes in a DETER network. The desired network must be created using NS script or a graphical user interface (GUI). Before running the software, one needs to edit the configuration files that determine the

settings and identity of a plant as well as the parameters for a new experiment with the software. The testing of plant stability necessitates the creation of a test value data file, which will log all the experimental A-values the user wishes to test. Deployment of software will then be done through logging into the node of an active experiment. Refer to the appendix for the necessary Python commands for running the plant and controller software on nodes.

For each experiment, the plant configuration file, plant.conf, is set. A rough range of test values for A is also logged into the testvals.conf file to be collected by the plant software at the beginning of startup. Then the software may be deployed onto the nodes of the experiment. To log the information processed between the controller and its plants, a log file is created during each series of testing, which will note all initial configurations and each update communication between plant and controller until plant is determined to be stable or in a state of crash. The plan is to test three different types of network topologies to understand how each affects plant stability and how latency plays a role in stability. Figure 1 shows the three difference topologies below.

/

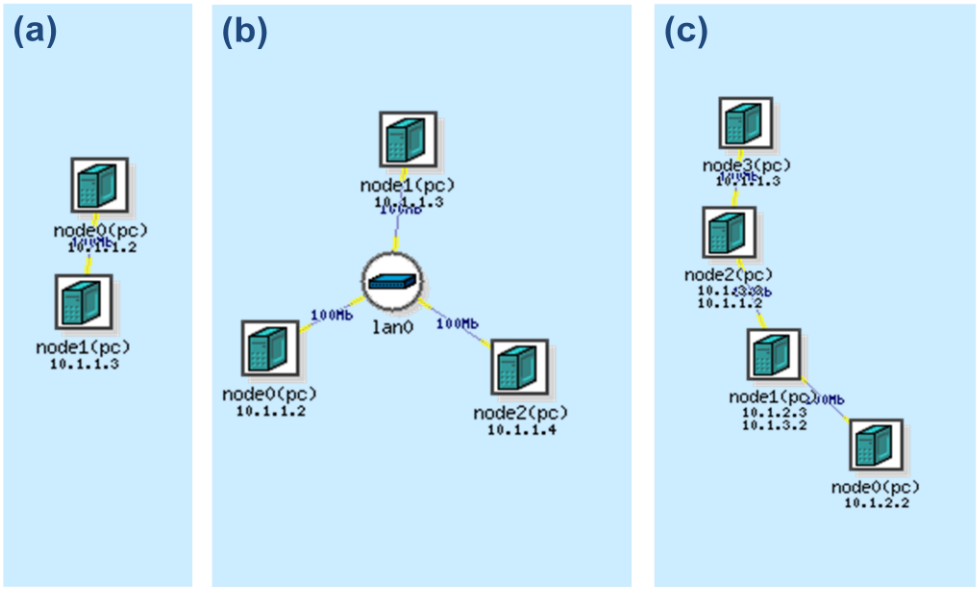

Figure 1. a) Simplest control network of a single controller and a single plant. b) 3 node network with a LAN connection. c) 4-node network

III. Results

For this project, experiments were conducted with three different types of topologies to investigate the ability of these three network structures to accommodate control system models. The first topology used was a simple two- node network. Multiple trials were processed with different values of A to figure out the limiting value of

state convergence and therefore, plant stability. Iterations of tests with the same A-values were also conducted to check for consistency of stability. After vigorous testing using the two-node network, the guaranteed stability range for constant A was found to be between 0 and 1. Stability is determined by how the state of a plant acts. If the state gradually converges to 0, then the plant is stable. Figure 2 below shows how a stable plant behaves as time passes.

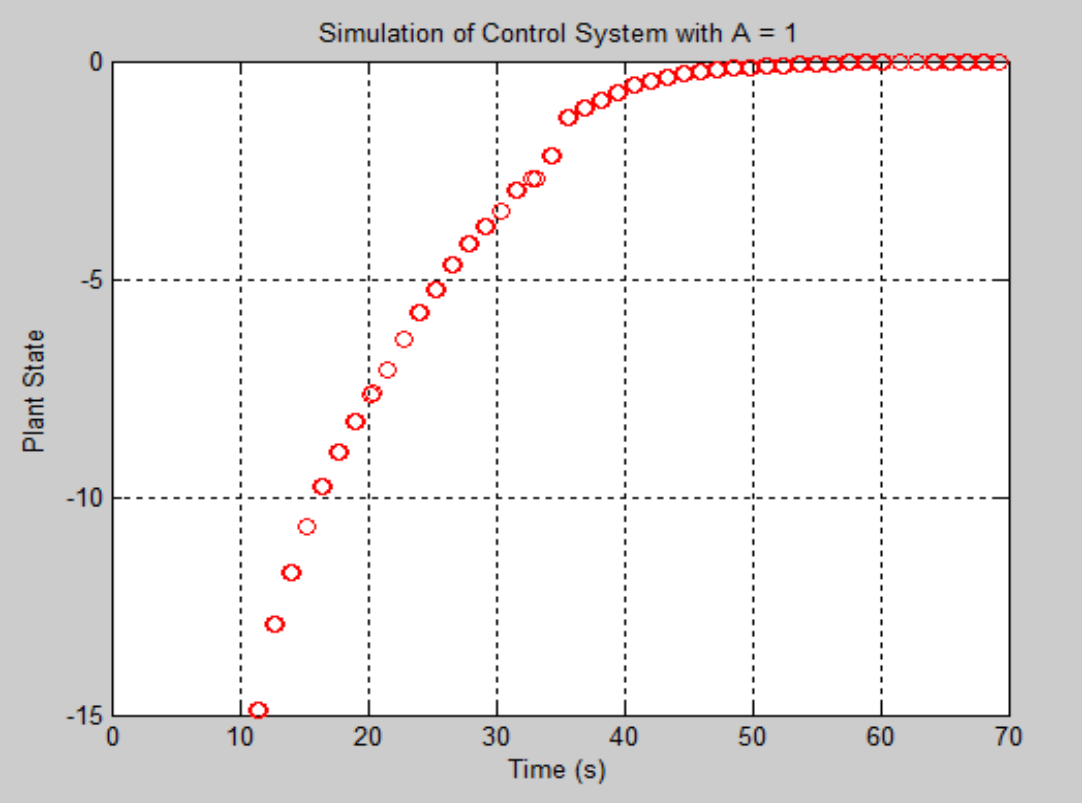

Figure 2. Example of a stable plant occurs when A = 1. Plant state gradually converges to 0 as time passes.

However, when A becomes greater than 1, stability becomes unpredictable and occurrence of stability rapidly decreases as the value of A increases. After approximately 1.25, plants are almost guaranteed to crash, as shown in Figure 3.

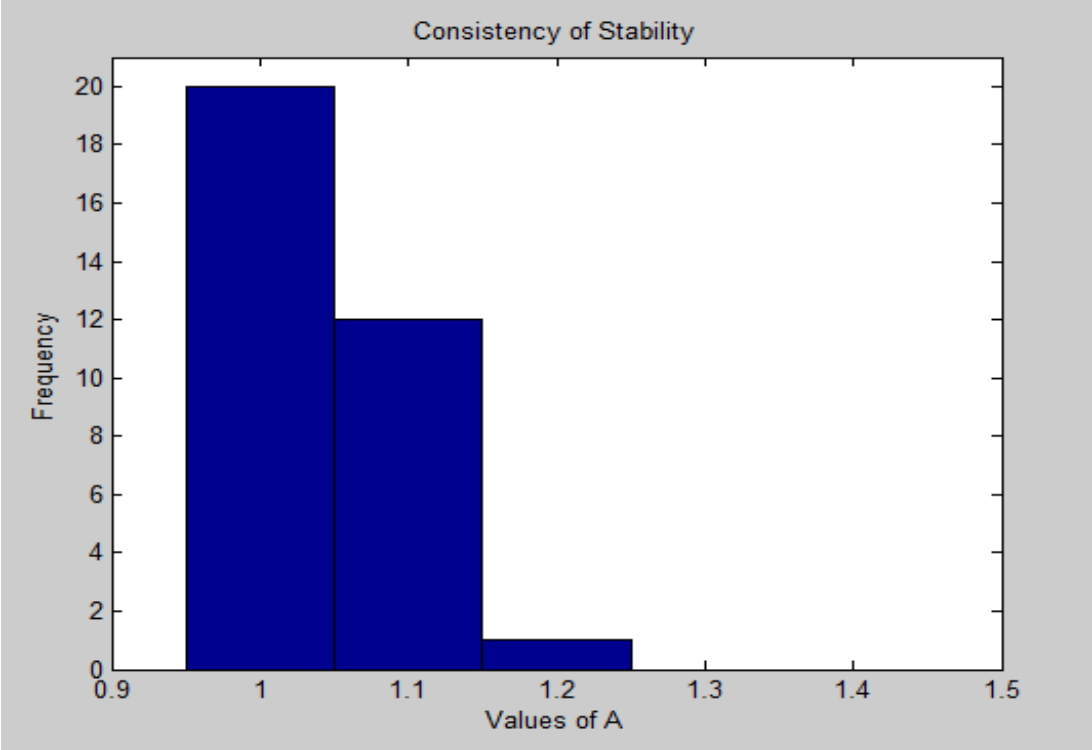

Figure 3. A histogram of repeated trials of various A values in a two-node network topology. Each A value was tested 20 times.

After experimenting with the simple two-node network, the software was deployed onto the other topologies mentioned. Analysis shows that the latency definitely affects plant stability. As the information path from point A to point B becomes more complicated and involves more nodes, the probability of plant stability decreases. Table 1 demonstrates the significant differences between different topologies.

	Two-Node	Four-Node	Three-Node LAN
Highest Stable A-Value	1	0.85	0.45

Table 1. Comparison of the stability between the three different network topologies. There is a clear effect between latency and stability.

The explanation for this result is probably that packet drop rates are higher through a LAN connection than through a second node. Future work may be done as to investigating the reason behind this significant drop in stability.

IV. Conclusions

The modeling of plant stability is essential in today's world, especially in light of the fact that simulation saves many resources and helps to analyze the reliability of a control system. Many real world applications could be applied to this simulation, such as the regulation of temperature in a factory, and other feedback-based control infrastructure. While this project tackles the problem of stability limits, many related topics still remain as opportunity for future research. For example, the packet drop rate is also an influential variable and it would be interesting to examine how it affects stability. In the next phase, one can also analyze the measures of dispersion in the final plant states of repeated tests. This will help determine the likelihood of instability by calculating the probability of a plant explosion based on its A-value. In conclusion, this project opens up paths to many new questions and considerations in the area of modeling plant stability.

V. References

[1] Alvaro Cardenas, Saurabh Amin, S. Shankar Sastry. "Research Challenges for the Security of Control Systems." Proceedings of the 3rd USENIX Workshop on Hot topics in security, USENIX, Article 6, 25, July, 2008.

[2] Liberatore, Vincenzo. "Networked Control Systems". Cleveland: 2002.

[3] Terry Benzel, Bob Braden, Ted Faber, Jelena Mirkovic, Steve Schwab, Karen Sollins, John Wroclawski, "Current Developments in DETER Cybersecurity Testbed Technology," catch, pp.57-70, Cybersecurity Applications & Technology Conference for Homeland Security, 2009

[4] Martelli, Alex. Python in a Nutshell. O' Reilly Media, 2006.

VI. Acknowledgments

I would like to thank the Team for Research in Ubiquitous Secure Technology, Dr. Kristen Gates, Larry Rohrbough, Ted Faber, Jelena Mirkovic, as well as the National Science Foundation for funding the program. Additionally, I would like to thank my mentors: Dr. Suzanna Schmeelk, Blaine Nelson, and Saurabh Amin; and my teammates in the REU program for all their help and support.

VII. Appendix

A. Equation used to model control system behavior:

$$\frac{dx}{dt} = Ax + u$$

$$\int \frac{1}{Ax+u}\,dx = \int dt$$

$$x + \frac{u}{A} = ce^{At}$$

$$x(u,t) = \left(x_0 + \frac{u}{A}\right)e^{At} - \frac{u}{A}$$

X = final state of plant

X_0 = initial state of plant

U = plant input

A = stability constant

T = time

B. Python commands:
    a. Controller: sudo python ./controller.py
    b. Plant: sudo python ./plant.py –s 10.1.2.2 > test.txt & (given a connection address 10.1.2.2)