19th Mediterranean Conference on Control and Automation
Aquis Corfu Holiday Palace, Corfu, Greece
June 20-23, 2011

PaNeCS: A Modeling Language for Passivity-based Design of Networked Control Systems

Emeka Eyisi, Joseph Porter, Nicholas Kottenstette, Xenofon Koutsoukos, and Janos Sztipanovits
Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37235 USA
emeka.eyisi, joe.porter, nicholas.e.kottenstette, xenofon.koutsoukos, janos.sztipanovits@vanderbilt.edu

Abstract— **The rapidly increasing use of information technology in constructing real-world systems has led to the urgent need for a sound systematic approach in designing networked control systems. Communication delays and other uncertainties complicate the development and analysis of these systems. This paper describes a prototype modeling language for the design of networked control systems using passivity to decouple control design from network uncertainties. The modeling language includes an integrated analysis tool to check for passivity and code generators for simulation in MATLAB/Simulink using the TrueTime platform modeling toolbox and for running actual experiments. The resulting designs are by construction robust to platform effects and implementation uncertainties.**

I. Introduction

Complex engineered systems such as automotive vehicles, building automation systems, and groups of unmanned air vehicles are currently referred to as Cyber Physical Systems (CPS). CPS are characterized by intricate interactions between physical dynamics, computational dynamics and communication networks. CPS are often monitored and controlled by networked control systems (NCS). Although the pervasive use of NCS offers exceptional opportunities for the way we build CPS, it also adds heterogenity and complexity. This heterogeneity makes it difficult to use existing design approaches to design CPS.

Model-based design for embedded control systems involves creating models and checking correctness at different stages in the development process [1]. Model-based design flow progresses along precisely defined abstraction layers, typically starting with control design followed by system-level design for the specification of platform details, code organization, and deployment details, and the final stage of integration and testing on the deployed system. This design approach is ineffective for NCS because domain heterogeneity and tight coupling between design concerns create a number of challenges. Ensuring controller stability and performance for physical systems in the presence of network uncertainties (e.g. time delay, packet loss) couples the control and system-level design layers. In addition, downstream code modifications during testing and debugging invalidate results from earlier design-time analysis and any component change often results in "restarting" the design process.

A number of research projects seek to address the problems of model-based design for NCS. The ESMoL modeling language for designing and deploying time-triggered control systems explicitly captures in model structure many of the essential relationships in an embedded design [2]. The ESMoL tools include schedule determination for time-triggered communications, code generation, and a portable time-triggered virtual machine. AADL [3] is a similar textual language and standard for specifying deployments of control system designs in data networks [4]. The Metropolis modeling framework [5] aims to give designers tools to create verifiable system models. Metropolis integrates with scheduling, timing analysis, and verification tools.

We propose an automated model-based approach based on passive control theory. We used Model-Integrated Computing [1] to develop a domain specific modeling language (DSML) called the Passive Network Control Systems language (PaNeCS). Our approach is based on the passive control architecture presented in [6] which provides robustness to network delays and packet loss. We will briefly describe the DSML, compositional passivity analysis, code generation for simulations using Matlab/Simulink/Truetime, and code generation of executables from nonlinear models for running actual experiments.

Changes made during design, development, and testing cycles may cause extensive software revisions and force expensive re-verification. PaNeCS supports automated forward generation of both platform-specific simulation models as well as deployable executables. PaNeCS enables rapid configuration and code generation while at the same time reduces the chances of introducing errors as compared to manually designing NCS in Simulink directly. It also integrates passivity analysis of system components.

Control systems are often verified using complex optimization techniques such as linear matrix inequalities (LMIs), which can model many important controller properties (e.g. stability, response time, reachability). In a system built from the composition of multiple blocks, such analysis quickly becomes intractable. In contrast, a passive design ensures global stability compositionally by a combination of component analysis and specific rules for composition of passive components.

Control designers create models for both physical systems and controllers using tools like Simulink and Stateflow. Deployment of a control design such as a Simulink model

978-1-4577-0123-8/11/$26.00 ©2011 IEEE

to a networked architecture introduces uncertainties due to time-varying delay, data rate limitations, jitter, and packet loss. Current approaches extend Simulink models with platform behaviors, as in TrueTime [7], which adds networks, clocks, and schedulers. While platform-specific simulation is a major step in validating NCS designs, it does not decouple design layers as described above. A control designer can factor in implementation effects (e.g., network delays), but if the implementation changes the controller may need to be redesigned. Our approach imposes passivity constraints on the component dynamics, so that the design becomes robust to network effects, establishing orthogonality (with respect to controller stability) across the control and implementation design layers.

The paper is organized as follows: Section II presents a passive control architecture for NCS. Section III presents our prototype modeling language. Section IV discusses an integrated analysis tool for automatically checking passivity. Section V presents automated code generators. Section VI shows a case study. Section VII provides our conclusion.

II. Passivity-Based Control of Networked Control Systems

Our NCS design approach is based on passivity theory. Essentially all passivity definitions state that output energy must be bounded so that the system does not produce more energy than was initially stored [8]. Passive systems have a unique property that when connected in either a parallel or negative feedback manner the overall system remains passive. Passive systems are robust to certain implementation uncertainties [9], so passivity can be exploited in the design of NCS. The main idea is that by imposing passivity constraints on the component dynamics, the controller becomes robust to network effects. This allows the separation of concerns between control properties and platform behavior.

We briefly discuss the passivity based control architecture for multiple plants controlled by a single controller via a network [6]. Fig. 1 depicts a simple networked control system with only one plant shown. The Bilinear Transform block, denoted **b**, represents a transformation between signals and wave variables. Wave variables were introduced by Fettweis in order to circumvent the problem of delay-free loops and guarantee a realizable implementation for digital filters [10]. Wave variables also allow systems to remain passive while transmitting data over a network subject to arbitrary fixed time delays and data dropouts [9]. In Fig. 1, $u_{pk}(i)$ (where k=2,...,n), can be thought of as sensor output data in wave variable form from each plant, where $n-1$ is the total number of plants in the network. Likewise, $v_{cj}(i)$ (where j=1) can be thought of as a command output in wave variable form from the controller.

The power junction, denoted PJ in Fig. 1, is an abstraction used to interconnect wave variables from multiple controllers and multiple plants in parallel such that the total input power is always greater than or equal to the total output power. This provides a formal way to construct a NCS design. A power junction makes it possible for a single controller to control multiple plants over a network and guarantee that the overall system remains stable. In Fig. 1, the power junction has waves entering and leaving as indicated by the arrows. The blocks, z^{-ck} and z^{-pk} (k=1,2), represent network delays incurred by the wave variables. Waves entering the power junction from the controller are network-delayed versions of waves leaving the controller, as indicated by the time delay block. Waves entering the controller are delayed versions of waves leaving the power junction. The other waves in the diagram are similarly delayed.

Due to bandwidth constraints, the controller typically runs at a slower rate than the sensors and actuators of the plants. In order to preserve passivity in the multi-rate digital control network we use the passive upsampler PUS:M and passive downsampler PDS:M pair to handle the data rate transitions. The PUS:M and PDS:M as shown in Fig. 1 provide the upsampled and downsampled versions of their respective wave variable inputs while preserving passivity. The block parameter M is the sampling ratio – the data rate of the faster side of the connection divided by the data rate on the slow side. Based on the architecture described in Fig. 1, we can now describe our prototype modelling language, PaNeCS.

III. PaNeCs

The passivity-based modeling language (PaNeCS) is developed using the Generic Modeling Environment (GME), from the Model Integrated Computing (MIC) tool suite [11].

A. Overview

PaNeCS encodes passive model constraints into the language structure to achieve compositional design and realization. The prototype language defines the allowable connections between components to ensure that any networked control system model built using passive components will be passive. PaNeCS provides the flexibility to easily model networked control systems and configure the system parameters. It also allows testing through simulations or by running actual experiments under various network conditions by simply adjusting parameters to generate appropriate software for each scenario. Fig. 2 shows the design flow in PaNeCS.

1) Components: In PaNeCS, the top level consists of four main components: the **PlantSystem**, the **ControllerSystem**, the **PowerJunction** and the **WirelessNetwork** modelling the components of an NCS. The modelling features of NCS components can be different in PaNeCS based on the goal of a NCS designer. For example, **PlantSystem** model for generating code to run simulations is different from a **PlantSystem** model for generating code for running real experiments.

The **PlantSystem** represents all the sub-components on the plant side of the network. Its components include *Plant*, *BilinearTransformP*, *PassiveUpSampler*, *PassiveDownSampler*, *Send* and *Receive*. *Plant* represents the system to be controlled. The *Plant* model can be any passive discrete linear time-invariant (LTI) system and can approximate a nonlinear system realized in the modeling and generation path. We discuss nonlinear NCS modelling for robotic arms

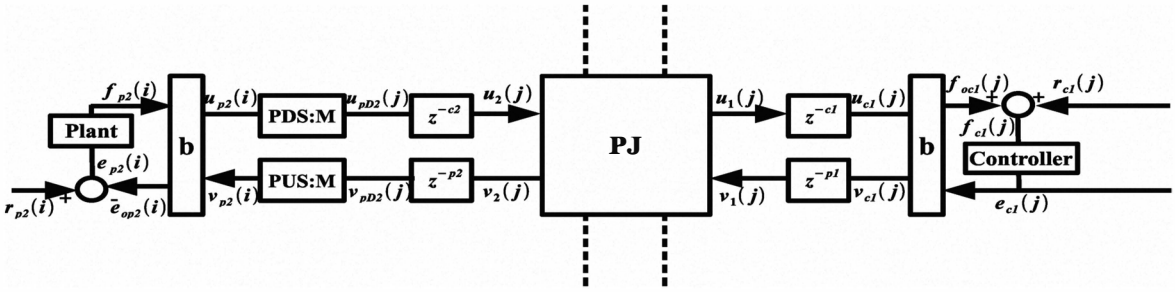


Fig. 1. Networked Control Architecture

using PaNeCS in [12] we omit the details for that case here for simplicity and brevity. The *Plant* dynamics are parameterized by matrix attributes A, B, C, D, and a scalar *SamplingTime*, and are specified using any valid Matlab expression evaluating to the proper dimensions. *BilinearTransformP* represents a model for wave scattering, which transforms the wave variables received from the power junction into control input to the plant and transforms the plant output signal into wave variables that are transmitted over the network. *PassiveUpSampler and PassiveDownSampler* pair represent the PUS:M and PDS:M pair discussed in Section II. *Send* and *Receive* pair are used to represent the transmission of data over the network. For creating models for running actual experiments additional attributes are used to specify the software components for sending and receiving data over the network such as port numbers, data size and dimension.

ControllerSystem represents all the sub-components on the controller side of the network. These include *DigitalController*, *BilinearTransformC*, *ZeroOrderHold*, *ReferenceInput*, *Send* and *Receive*. *DigitalController* is a model representing the algorithm for controlling the networked plants. Similar to the model of the *Plant* in the **PlantSystem**, the *DigitalController* is modeled as a passive LTI system. Therefore, the *DigitalController* parameters have similar attributes to the *Plant* for a LTI system case. *BilinearTransformC* is similar to the *BilinearTransformP* described in the **PlantSystem**. *ZeroOrderHold* represents a component that holds its input for the time period specified in the sampling time attribute. *ReferenceInput* represents the desired signal to be tracked by the plants. Similar to the **PlantSystem**, *Send* and *Receive* pair are used to represent the transmission of data over the network.

PowerJunction model component describes the power junction. The **PowerJunction** can contain ports for the connection of the plants and controllers. They are briefly described as follows: *PowerInputPowerOutput* represents a port through which the **PlantSystem** connects to the **PowerJunction**. Through it, the **PowerJunction** sends calculated control signals to the **PlantSystem** and also receives sensor signals from the **PlantSystem**. *PowerOutputPowerInput* represents a port through which the **ControllerSystem** can connect to the **PowerJunction**. Through it, the **PowerJunction** sends the averaged sensor signal to the **ControllerSystem** and receives the calculated control signal from the **ControllerSystem**. Additionally, for creating models for running real experiments port number attributes are used to specify ports for sending and receiveing data over the network.

WirelessNetwork represents the network and its parameters for the NCS. The **WirelessNetwork** model provides modifiable parameters for simulation. *Data rate* sets the throughput for simulating network activity. *DisturbancePacketSize* configures the size of simulated disturbance attack packets on the network (introduces delays). This provides a way for simulating the NCS under non-optimal conditions. *DisturbancePeriod* configures the frequency of disturbance attacks on the network.

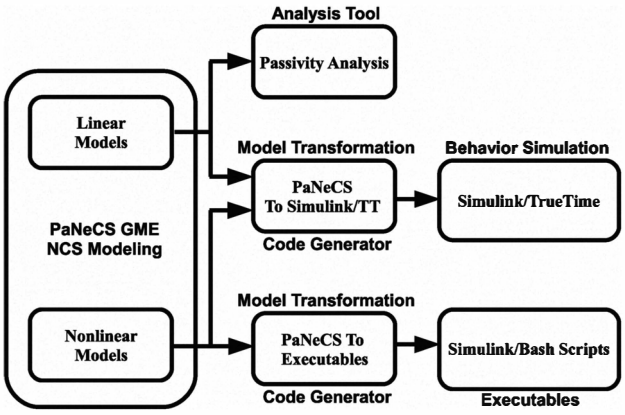


Fig. 2. PaNeCS Design Flow

2) Language Aspects: Our modeling language has two main aspects (GME aspects are similar to modeling views in other tools): **Control Design Aspect** and **Platform Aspect**. The **Control Design Aspect** visualizes the controller modeling layer. This includes the plants, controller, and power junction, as well as their interconnections – indicating the flow of control and sensor signals.

The **Platform Aspect** visualizes the physical platform components of the NCS. This view includes plants, controllers, and the wireless network as well as their interconnections – indicating the flow of data packets over the network. Though the plants and controller appear in both aspects, in the Platform aspect they represent physical devices rather than control design concepts.

In PaNeCS, when the design goal is for running actual experiments, an additional aspect, **Processor Assignment Aspect**, is used for depicting the mapping of software components to processors on which the computations and implementations are to be performed [12].

3) Structural Semantics: Our main language design goal is to ensure "correctness-by-construction" for passive NCS designed using PaNeCS. To achieve this objective we impose constraints on the NCS component properties as well as their interconnections. The metamodel notations described above do not capture all the required structural constraints. Using the Object Constraint Language (OCL), we can add well-formedness rules to precisely control the static semantics of the language. GME is embedded with an OCL engine – specified constraints are enforced at design time, giving direct feedback when the user attempts to create faulty connections in the model or violates any other constraints. In Section IV, we will describe an analysis tool for verifying that system components satisfy passivity constraints.

We implemented three classes of constraints: *Cardinality Constraints* ensure that the correct number of components are used in the NCS design. For example, for each **PlantSystem** model there must be one *Plant*. *Connection Constraints* restrict the number of allowable connections between components. For example, in the **PlantSystem** model there can only be one bidirectional connection between the *Plant* and *BilinearTransformP*. *Unique Name Constraints* ensure the uniqueness of the names of components in the Plant and Controller subsystems as well as in the top level model of the NCS.

The sample OCL constraint below specifies that the number of connections from a BilinearTransformC model to a DigitalController must be at most one.

```
Description: There must be only one bidirectional
connection between BilinearTransformC to the
DigitalController

Equation: let dstCount = ...
self.attachingConnections("src",...
Controller_Bilinear)->size in
dstCount <> 0 implies dstCount = 1
```

IV. Passivity Analysis

In passive designs, we have to analyze the system components to make sure they satisfy passivity constraints. Using the analysis tool integrated in PaNeCS, we can currently analyze the passivity properties of any LTI system component modelling the plant or digital controller.

The analysis of the *Plant* and *DigitalController* components of the networked control system for passivity is done automatically by an integrated Matlab analysis function. Each component is assumed to have a linear time-invariant (LTI) discrete-time model, so we use LMIs together with the CVX semidefinite programming tools for Matlab [13] [14]. On invocation (i.e. the modeler presses a button), a C++ model interpreter within GME visits each component, and invokes the analysis function. Any components failing the passivity test are reported to the user.

The dynamics of the *Plant* and *DigitalController* models can each be defined by a state space representation and are characterized by the matrices A, B, C, D of size compatible with the number of inputs and outputs in the system and the number of states in the model. The passivity constraints for these models is defined by Linear Matrix Inequality (LMI) constraints [15]. For example, a LMI formula for strict output passivity for an LTI digital controller is given by

$$\begin{bmatrix} A^TPA - P - \hat{Q} & A^TPB - \hat{S} \\ (A^TPB - \hat{S})^T & -\hat{R} + B^TPB \end{bmatrix} \leqslant 0$$

$$\begin{aligned} \hat{Q} &= C^TQC, \quad \hat{S} = C^TS + C^TQD \\ \hat{R} &= D^TQD + (D^TS + S^TD) + R \end{aligned} \tag{1}$$

$$\exists \varepsilon > 0,\ Q = -\varepsilon I,\ R = 0,\ S = \frac{1}{2}I, \exists P = P^T > 0$$

The CVX semidefinite programming (SDP) tool is used in a Matlab script to solve the LMI for each component.

Due to the "correct-by-construction" approach we use in designing networked control, we only analyze the *Plant* and *DigitalController* elements for passivity. If those components satisfy the passivity constraints, the network control system as a whole also satisfies the passivity principles.

The component interconnections are restricted in such a way that they are "correct-by-construction". Only valid (parallel) connections are allowed to the power junction, so any interconnected system of passive components in the language will be globally passive. The modeling language and its constraints encode the passive composition semantics, greatly reducing the analysis burden for determining passivity (and hence stability [6]) of the composed system design.

V. Code Generation

The main objective of the code generators is to generate code that maps the models designed using the modeling language to Simulink models and network scripts that represent the networked control system. Fig. 2 shows the two code generators one for generating code for running simulation and for generating executables for running actual experiments. We will briefly discuss the two code generators used in PaNeCS.

We developed a model interpreter that synthesizes simulation code from PaNeCS models. The interpreter is developed in C++ using the Builder Object Network (BON2) API provided with GME [11]. The interpreter traverses all the entities of a particular networked control system instance model and extracts model parameters. These parameters and model structure are used to generate MATLAB files for configuring and building Simulink and TrueTime models to simulate the NCS. The **PlantSystem** and **ControllerSystem** are modelled as Simulink subsystems, which contain the respective *Plant* and *DigitalController* behavior blocks. Each generated system-level component is connected to a TrueTime Kernel block. The TrueTime Kernel models a processing node with a scheduler and I/O. Our models execute on periodic schedules within TrueTime. For this version of our language, the **PowerJunction** is implemented as a task in the TrueTime Kernel connected to the **ControllerSystem**. The task that implements the power junction is triggered by data

arrival events. Each TrueTime kernel has an initialization script and a function script specifying timing for I/O and task execution. The TrueTime Wireless network block simulates the transfer of data packets over a wireless network from one node to another.

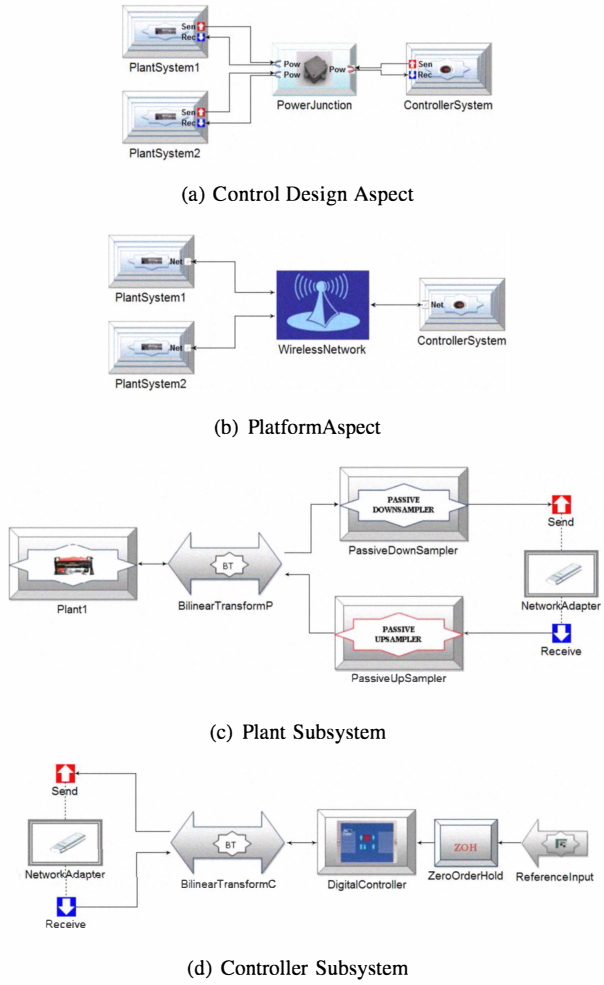
Similar to the case of synthesizing simulation code, we developed a code generator that can be used to synthesize software for integration, deployment, and testing of the networked system. The code generator implements translation rules from the model elements to Simulink blocks realizing the system behavior. For the experiments, the Simulink models alone are not sufficient to set up the network infrastructure for nonlinear experiments. A deployment model, which can be visualized through the **Processor Assignment Aspect**, describes assignments of models to processors. The integrated code generator also produces *bash* shell scripts in order to set up the networking infrastructure and run the experiments. The network infrastructure utilizes Netcat (a standard computer networking service tool) and SSH to build internet tunnels for system testing. The powerjunction is configured as a server, and plant and controller models use client sockets to attach to the power junction. The network system follows a globally asynchronous locally synchronous execution model. The controller and plant receivers execute periodically.

VI. Case Study

Fig. 3 shows a model for a simple passive linear controller regulating two passive linear plants to track a specified reference signal. Fig. 3a denotes the **Control Design Aspect** describing the control design concepts while Fig. 3b denotes the **Platform Aspect** describing the physical platform components. The **Powerjunction** does not appear in Fig. 3b because it is implemented on the the same physical component as the **ControllerSystem**, as described in Section III the **Platform Aspect** only shows the physical platform components. In this experiment, the goal is to model the NCS and generate a simulation of its behavior. Although this case study models two discrete-time plants, PaNeCS can model and simulate an arbitrary number of plants.

The two plants in the experiment (Fig. 3) are simple integrators (corresponding to models of inertial masses of 2kg and .25kg respectively) which are discretized in time. The plants' dynamics were modeled in state space form and the corresponding A, B, C, and D matrices as well as sampling time (T_s) were provided as parameters to the instance model.

We used a digital proportional controller to command the plants to track the user-specified reference. We also used state-space form for the controller (A, B, C, D, and T_s). The parameters values for the plant and controller dynamics are given in Table I. The analysis tool checked and verified that the *Plant* and *DigitalController* models satisfied the passivity constraints. Then the code generator was used to create a platform-specific Simulink simulation model using the specified parameters and model structure.



(a) Control Design Aspect

(b) PlatformAspect

(c) Plant Subsystem

(d) Controller Subsystem

Fig. 3. Sample Model of a Networked Control System

Using PaNeCs, we tested the dynamics of an NCS by running experiments under different network conditions. Table II shows the parameters for the simulations.

A. Nominal Conditions

This experiment operates the sytem without disturbance attacks for three sample periods (0.1s, 0.5s and 1s). We only present plots for the results of the NCS having a sample period of 0.1s. Fig. 4a shows that the plants closely tracked the reference velocity. The round trip delay for each plant seemed to have very little effect on the stability of the plants' velocity response. The delay can be attributed to the internal processing of the plants and controllers rather than network delay itself.

TABLE I

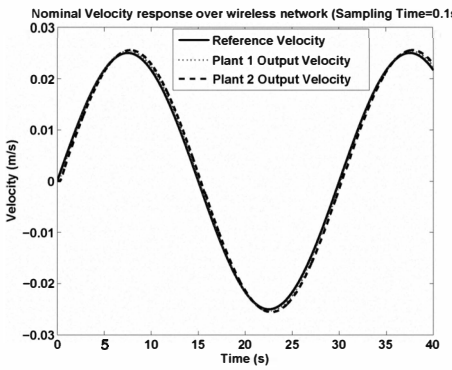
Plant and Controller Dynamics.

	A	B	C	D	T_s
Plant1	1	1	.005	.0025	$.01s$
Plant2	.996	1	.04	.02	$.01s$
Controller	0	0	0	10π	$.1s$

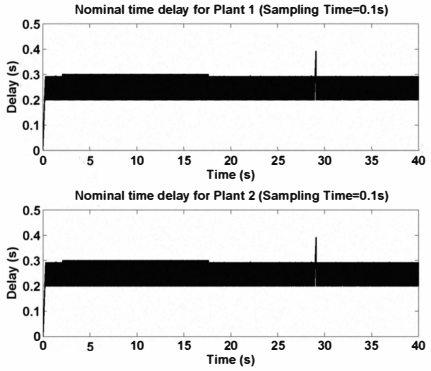
TABLE II

Simulation Parameters Summary.

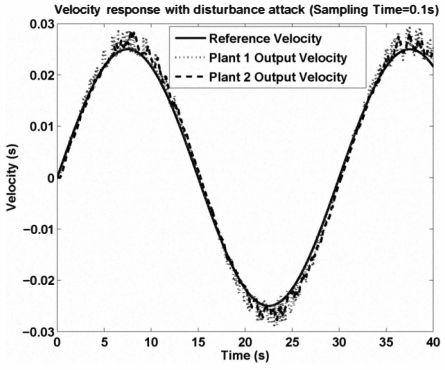
Sample Periods	$0.01s$	$0.05s$	$0.1s$
Plant1,M	10	50	100
Plant2,M	10	50	100
Disturbance	$T_s = 0.01$	$Packetsize = 110,000bits$	



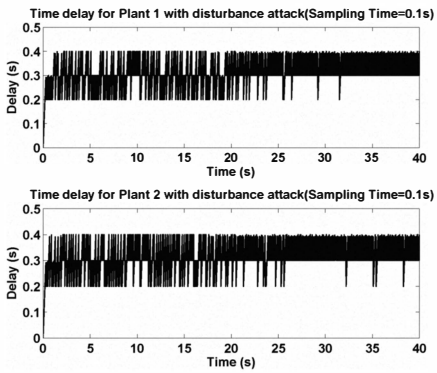
(a) Nominal velocity response.



(b) Time delays .



(c) Velocity response with disturbance attack.



(d) Time delays with disturbance attack.

Fig. 4. Velocity and Delay Plots.

B. Network disturbances

This experiment introduces a disturbance attack in the network using parameters on the wireless network block. Similar to nominal experiment, three different sample rates were tested, but we only present the results for the 0.1s sample period. Fig. 4c and 4d shows the velocity response and the time delay respectively for each plant. The results show that even with disturbance attacks, the plants remain stable in tracking the reference velocity although the performance is relatively affected as can be seen from the plots. This demonstrates the advantage of the passivity approach which guarantees the stability of the NCS in the presence of uncertainties due to network effects.

VII. Conclusion

Our model-based approach simplifies the process of designing passive networked control systems. We present PaNeCS, a prototype modeling language for designing, analyzing and generating code for simulations and executables. We use a case study to demonstrate our design approach.

References

[1] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Model-integrated development of embedded software," *Proc. of the IEEE*, vol. 91, no. 1, Jan. 2003.

[2] J. Porter, G. Karsai, P. Vlgyesi, H. Nine, P. Humke, G. Hemingway, R. Thibodeaux, and J. Sztipanovits, "Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation."

[3] AS-2 Embedded Computing Systems Committee, "Architecture analysis and design language (AADL)," Society of Automotive Engineers, Tech. Rep. AS5506, Nov. 2004.

[4] J. Hudak and P. Feiler, "Developing AADL models for control systems: A practitioner's guide," CMU SEI, Tech. Rep. CMU/SEI-2007-TR-014, 2007.

[5] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Paserone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," *IEEE Computer*, vol. 36, no. 4, pp. 45–52, Apr. 2003.

[6] N. Kottenstette, J. Hall, X. Koutsoukos, P. Antsaklis, and J. Sztipanovits, "Digital control of multiple discrete passive plants over networks," *Intl. Journal of Systems, Control and Communications, Special Issue on Progress in Networked Control Systems*, vol. 3, no. 2, pp. 194 – 228, 2011.

[7] M. Ohlin, D. Henriksson, and A. Cervin, *TrueTime 1.5 Reference Manual*, Dept. of Automatic Control, Lund University, Sweden, Jan. 2007, http://www.control.lth.se/truetime/.

[8] C. A. Desoer and M. Vidyasagar, *Feedback Systems: Input-Output Properties*. Orlando, FL, USA: Academic Press, Inc., 1975.

[9] P. Berestesky, N. Chopra, and M. W. Spong, "Discrete time passivity in bilateral teleoperation over the internet," in *IEEE International Conference on Robotics and Automation*, 2004, pp. 4557 – 4564.

[10] A. Fettweis, "Wave digital filters: theory and practice," *Proc. of the IEEE*, vol. 74, no. 2, pp. 270 – 327, 1986.

[11] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," *Workshop on Intelligent Signal Processing*, May 2001.

[12] X. Koutsoukos, N. Kottenstette, J. Hall, E. Eyisi, H. LeBlanc, J. Porter, and J. Sztipanovits, "A passivity approach for model-based compositional design of networked control systems," *ACM TECS special issue on the Synthesis of Cyber-Physical Systems (SCPS)*, To Appear.

[13] M. Grant and S. Boyd, "CVX: MATLAB software for disciplined convex programming," Feb 2009. [Online]. Available: http://stanford.edu/~boyd/cvx

[14] M. Grant and S. Boyd, "Graph implementations for nonsmooth convex programs," *Lecture Notes in Control and Information Sciences*, vol. 371, pp. 95–110, 2008.

[15] N. Kottenstette and P. Antsaklis, "Relationships between positive real, passive dissipative, & positive systems," in *American Control Conference*, 7 2010, pp. 409 –416.