NMLab: A Co-Simulation Framework for Matlab and ns-2

Oliver Heimlich, Rudolf Sailer
Institut für Mathematik
Universität Würzburg
Würzburg, Germany
oliver.heimlich@stud-mail.uni-wuerzburg.de
sailer@mathematik.uni-wuerzburg.de

Łukasz Budzisz
Hamilton Institute
NUI Maynooth
Maynooth, Co. Kildare, Ireland
Lukasz.Budzisz@nuim.ie

***Abstract*—This paper presents NMLab, a proposal of new co-simulation framework for Matlab and ns-2 targeted for networked control systems. NMLab enables to combine flexible and powerful numerical operations together with a realistic support for a wide range of up-to-date communication protocols, and permits simulating networking scenarios of high grade of complexity. At the same time the proposed solution is kept simple. The capabilities of NMLab are illustrated through a series of experiments, using as an example the academic version of the pendulum on a cart.**

***Keywords*-Matlab; ns-2; co-simulation; networked control system**

I. Introduction

Recent years saw a growing role of networked control systems (NCS) within the control community. A NCS consists of several systems, controllers, actuators and sensors which share one bandwidth-limited digital communication channel (for a more detailed introduction to NCS, see e.g., [1] and the references therein). As NCS have become more and more complex, a pure analytical approach is often not possible (or too demanding), and consequently there is a strong need for development of appropriate simulating tools. Typically control system simulators, e.g., Matlab/Simulink [2], or Scilab/Scicos [3], focus on continuous dynamics having only some support for discrete events. Meanwhile, most common network simulators, such as Opnet or ns-2 [4], while providing realistic frameworks for simulations of packet based transmissions (discrete-event-driven) and supporting wide range of network protocols and scenarios, have limited support for continuous dynamics [5].

As hybrid dynamical systems, NCS require from their simulators to combine both continuous-time plant dynamics of control system simulators and discrete events present in network simulators. One of possible solutions is to rely on several existing hybrid-system simulators, such as Ptolemy II [6] or SimEvents [7] extension for Matlab/Simulink. However these solutions usually have high levels of abstraction which sees them either unsuitable, or requires large modelling effort in order to make a realistic NCS simulation feasible (e.g., SimEvents does not have wireless support, whereas Ptolemy II's wireless model can be seen as too simplistic). Therefore, the approach that federates existing control and network simulators, a so called *co-simulation* approach, seems much more appropriate in terms of simulating NCS. Bringing both types of simulators together could help to combine the advantages, i.e., preserve the computational flexibility of Matlab and provide the functionality of a tested and verified network simulator [5], however it also poses many challenges, e.g., versatility of the proposed solution.

One of the first examples of a co-simulation design is TrueTime extension for Matlab/Simulink [8]. TrueTime is a discrete-event simulator inside Simulink providing co-simulation of control task execution, network communication and plant dynamics. TrueTime, while taking the advantage of all flexibility and versatility of numerical computations, preserves also the main drawback of Matlab in context of NCS simulations, namely, limitation on complexity of the network and its dynamics [9]. In his thesis, Hartman points out this limitation of TrueTime while also giving a brief overview of a co-simulation framework between Matlab and ns-2 in context of NCS. Further overview of existing NCS co-simulation tools for ns-2 is given in [10]. Al-Hammouri et al. present in detail Agent/Plant extension for ns-2, as well as two other recent solutions, NSCSPlant and one that integrates ns-2 with Modelica.

The idea of a co-simulation between Matlab and ns-2 is not new, though existing co-simulation frameworks are lacking flexibility beyond the network simulation. Co-simulation in off-line mode, where the output data from one simulation package is stored in a file, permits no interaction between the network and the dynamical system [11]. Frameworks [10], [12, Section 3.3], [13], [14], which focus on the network simulator, generally benefit from the performance of compiled external simulators, but make changes to parameters of the dynamical system difficult. In contrast, while existing real-time co-simulation frameworks, e.g., PiccSim [15] for Simulink and ns-2, provide a valuable tool for simulation of real dynamical systems and real processes combined with simulated networks, they categorically cannot simulate asynchronous distributed systems with high grade of accuracy [16, Chapter 11].

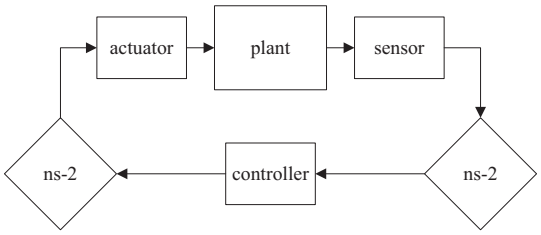

Fig. 1. NCS using ns-2. The control system is modelled in Matlab, whereas communication with the controller is subject to delays and message loss specified by the network simulator ns-2.

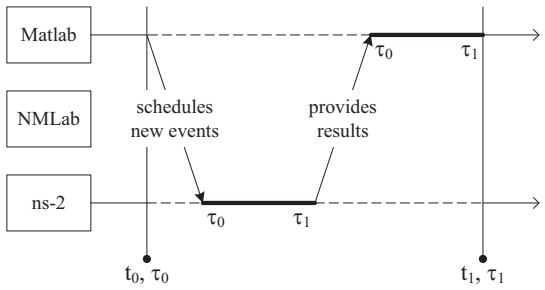

Fig. 2. Co-simulation coupling with NMLab. Both simulators have synchronised clocks at time t_0 (simulated time τ_0), Matlab commands ns-2 to start its simulation, ns-2 simulates until simulated time $\tau_1 \geq \tau_0$ and sends the updated time to Matlab, which in turn catches up with τ_1, so both simulators are synchronised at t_1 again.

This work further explores the idea of ns-2 and Matlab co-simulation, with a simple design called NMLab, which aims to improve the available NCS simulation frameworks while keeping the proposed solution simple.

The rest of the paper is organised as follows. The basic idea, as well as the capabilities of NMLab are presented in Section II. Section III provides simple, but illustrative example of NMLab application. The discussion of the obtained results is presented in Section IV, whereas final remarks are given in Section V.

II. NMLab Co-Simulation Framework

NMLab co-simulation framework extends Matlab with features for straight forward network modelling and automates simulated communication over the network below the application level of the so-called Open System Interconnection reference model (OSI model). More precisely, messages in terms of Matlab data objects are being automatically fragmented, transmitted, routed and defragmented by the network simulator. NMLab can be used to construct network topologies using selected network technologies supported by ns-2. Modelled plants can be situated in NCS contexts easily, as shown in Figure 1, so the network and its dynamics can be considered during controller design. The object-oriented approach makes it possible to virtually direct Matlab events via the simulated network and modify them (e.g., delay, or drop given event).

Communication between Matlab and ns-2 is established by NMLab using stream sockets to connect to ns-2's Tcl interface. This solution offers a good deal of flexibility, because it does not depend on a certain version or setup of ns-2. Furthermore, almost any functionality of NMLab can be provided by an expandable Matlab class library, which generates Tcl code for ns-2. The major drawback of this approach is that any required functionality of the network simulator has to be implemented via NMLab classes in advance to provide accessibility. However, NMLab can forward Tcl commands to ns-2 partially resolving that issue.

The major challenge for NMLab is to assure the correct impact of the (application level) data payload to the network simulation and vice versa. NMLab creates appropriate traffic on the packet layer in ns-2 for the messages to be send. Successfully transmitted packets on the other side are reassembled to their original message, commanding the network simulator to synchronise with Matlab which triggers a corresponding event in Matlab.

During co-simulation, although ns-2 is rendered passive by being utilised for simply generating message transmission times and providing network states, its scheduler is used to specify the common simulated time between both simulators, as explained in Figure 2. This design has been chosen due to the simple fact that the occurrence of message receiving events cannot be foreseen by Matlab. Such a design allows predictable state-based events of the dynamical system to trigger a network event, which would have been impossible otherwise (without modifications to ns-2's event scheduler).

To be more precise, the exchange of events is a general problem of asynchronous co-simulation. In Figure 2, if Matlab needs to schedule a new network event for ns-2 during its part of the simulation, this is not possible until the time t_1. However, at time t_0 Matlab can define an additional synchronisation event to occur between τ_0 and τ_1. So, if it was possible that a network event needed to be sent to ns-2 before next (original) synchronisation between both simulators, this has to be decided in advance, e.g., the dynamical system's state crosses a certain threshold which triggers a state-based sensor to send a message, or any part of the NCS has to send a "time-out" message as soon as it has not received a message for a certain period of time.

Both of the above given examples can be modelled using NMLab with some additional expense. *Time-discrete* simulations may need to conservatively schedule additional conditional synchronisation events, which possibly are going to be changed or deleted before occurrence, e.g., time-out events. On the other hand *time-continuous* simulations may need to be computed foresight in order to determine the exact time of a possibly needed synchronisation event. However, in the simulations presented in this paper such methods are not necessary.

One of the key features of NMLab is to keep the co-simulation together, namely modelling the control system in conjunction with the network in Matlab, while the latter is simulated by ns-2. Thus it becomes easy to change network

```
timer = ns2.NMTimer(network);
timer.setDelay(0.01);
timer.start();

addlistener(timer, 'Tick', @(s, evnt) ...
    sendMessage(plant, controller, ...
        readSensors(plant, netwock.clock)));
addlistener(controller, 'MessageReceived', @(s, evnt) ...
    sendMessage(controller, plant, ...
        createValue(controller, evnt.message)));
addlistener(plant, 'MessageReceived', @(s, evnt) ...
    applyValue(plant, evnt.message, network.clock));

network.run();
```

Listing 1. Example of Matlab code for setting up a networked control system after having created the network topology. Highlighted identifiers and events are provided by Matlab and NMLab.

	Variables	
φ	rad	pendulum's angle
r	m	cart's position on the rail
u	$\mathrm{kg\,m/s^2}$	external force
	Constants	
m_c	3.2kg	cart's mass
m_p	0.329kg	pendulum's mass
l	0.44m	distance between pendulum's axis and gravity centre
J	$0.0719944\mathrm{kg\,m^2}$	pendulum's moment of inertia
b_c	$25.28\mathrm{kg/s}$	cart friction coefficient
b_p	$0.009\mathrm{kg\,m^2/s}$	pendulum friction coefficient
g	$9.81\mathrm{m/s^2}$	gravity

TABLE I
VARIABLES AND CONSTANTS USED FOR THE PENDULUM. THE CONSTANTS HAVE BEEN CHOSEN TO MATCH AN EXISTING PENDULUM ENTITY.

parameters in conjunction with parameters of the control system. Additionally certain states of the simulated network, known exclusively to the network simulator, e.g., queue levels, can be passed to Matlab during co-simulation for live analysis or processing.

Listing 1 demonstrates NMLab's usage by giving an example of how the communication between parts of a NCS is set up. Every 10ms the plant reads its sensors and sends a message to the controller, which in return provides the appropriate control value. As soon as the value has arrived at the actuator, it is applied. Synchronisation between Matlab and ns-2 happens automatically each time an event is triggered.

III. The Pendulum as a Showcase

With the help of the well studied academic example of a pendulum on a cart we want to demonstrate the versatility and flexibility of NMLab and the need for simulating NCS in general, i.e., our aim is *not* to gain new experiences regarding the pendulum in the first place. We take a look at the four-dimensional pendulum-cart system. The following system's equations:

$$u + m_p l\dot{\varphi}^2 \sin\varphi = (m_c + m_p)\ddot{r} + b_c\dot{r} + m_p l\ddot{\varphi}\cos\varphi \quad (1)$$

$$g m_p l \sin\varphi = J\ddot{\varphi} + b_p\dot{\varphi} + m_p l\ddot{r}\cos\varphi \quad (2)$$

(see Table I for notation) are a model of the pendulum's dynamics and are integrated in Matlab using an ODE solver.

The control objective is to stabilise the pendulum in its top position ($\varphi = 0$) in the middle of the rail ($r = 0$). Different initial states with an arbitrary angle φ are chosen, and the remaining state parameters are set to $\dot{\varphi} = r = \dot{r} = 0$. In the following experiments it will be shown how long does the dynamical system need to reach a certain stability threshold for the first time, which will be denoted informally as "stabilisation time" in the figures below, with respect to different simulation parameters. Therefore the maximum norm of its state vector $x = \begin{pmatrix} r & \dot{r} & \varphi & \dot{\varphi} \end{pmatrix}^T$ is computed periodically until it falls below 0.01 (proposed threshold) for the first time.

$$\|x\| = \max\{|r|, |\dot{r}|, |\varphi|, |\dot{\varphi}|\} \quad (3)$$

Because of the angle's periodicity, φ is considered within the interval $[-\pi, \pi]$ during computation of the norm.

For the linearised model of the control system a linear controller is obtained via Matlab's built-in pole-placement algorithm, which proves useful for angles $|\varphi| < \frac{1}{4}\pi$ in the ideal scenario where there is no network control involved. Additionally, we have developed a simple non-linear controller with Lyapunov-based design, which is capable of uprising the pendulum with an initial angle $|\varphi| < \frac{5}{6}\pi$, again as long as there is no network involved. In the experiments presented below, a combined controller is used, utilising the linear controller for lower angles and the non-linear controller for greater angles. Switching between them takes place at $|\varphi| = \frac{1}{12}\pi$.

Using NMLab the system can be simulated as a networked control system with different network parameters. The pendulum's state is read by sensors at a default rate of 20Hz and sent over the network. The controller calculates the corresponding control signal and sends it back over the network, to be applied as soon as it arrives at the pendulum. The value is kept constant at the actuator until new information arrives. All data payloads have a fixed size of 100B. In order to keep the simulation simple, the sensors do not produce errors of measurement, the controller takes no time for calculation, and any further restrictions to the system are not considered, i.e., the actuator is able to apply the control value instantly. Though it could have been simulated with NMLab easily.

As a reference for classifying the following simulation results, the pendulum is simulated with zero message delay and zero message loss, see Figure 3.

A. No Background Traffic

At first, the network is run without the interfering background traffic, so the links produce nearly constant packet delay without packet loss, no matter which of the transport protocols (TCP or UDP) is used. Different round trip times (RTTs) between the pendulum and its controller are simulated. The initial angle of the pendulum φ is set to $\frac{1}{9}\pi$. Figure 4 shows that there is a local, but not global, maximum of the stabilisation time around the RTT of 40ms. Above 60ms the stabilisation time rapidly increases.

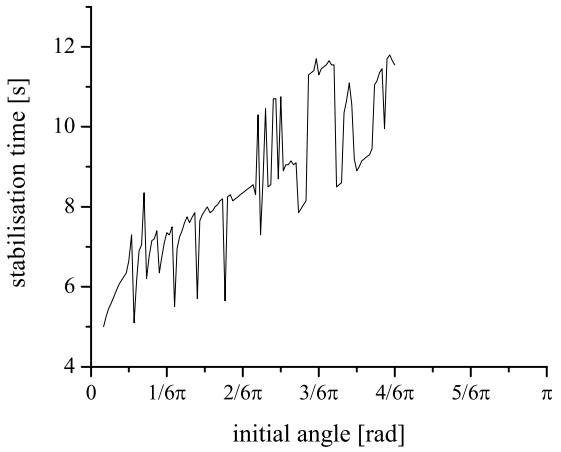

Fig. 3. Stabilising the pendulum under ideal conditions without packet delay or packet loss.

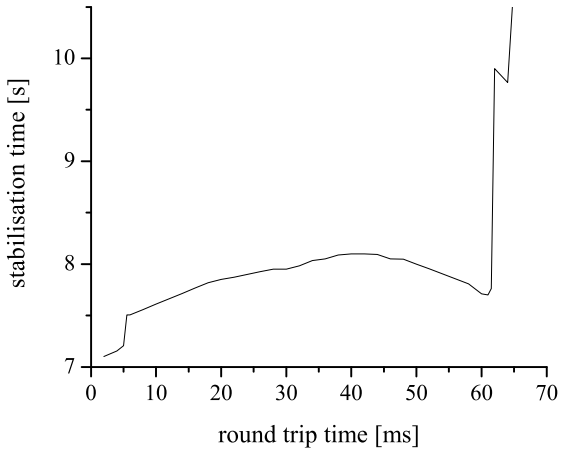

Fig. 4. Varying RTTs between the system and its controller. RTTs of 15ms and 60ms are both better for stability than 40ms.

B. Impact of Background Traffic

Next, a simple dumbbell topology (Figure 5) is simulated. This time, the RTT between the pendulum and the controller exceeds 40ms, subject to all connections' propagation delays settings. The bottleneck has a 10Mbps bandwidth and a queue size of 50packets in each direction. Apart from the communication between the pendulum and its controller which uses TCP protocol, there are three TCP traffic sources using the bottleneck at the maximum possible rate. During the simulation, when the pendulum is on the verge of reaching the stability threshold, two additional UDP traffic sources, each with a constant rate of 4Mbps, start sending packets at 2.5s and 3s respectively. Whereas already the first UDP source causes all TCP connections to reduce their bandwidth (the congestion window drops in Figure 6), the latter UDP source drastically influences the pendulum-network connection, as its

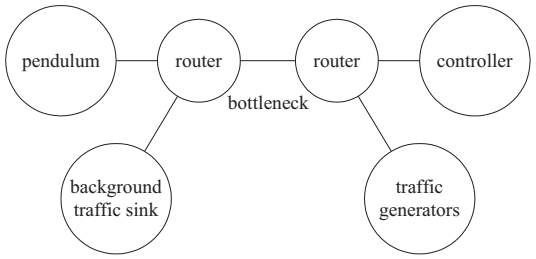

Fig. 5. Dumbbell Topology. The impact of background traffic can be studied by sending background traffic over the bottleneck link.

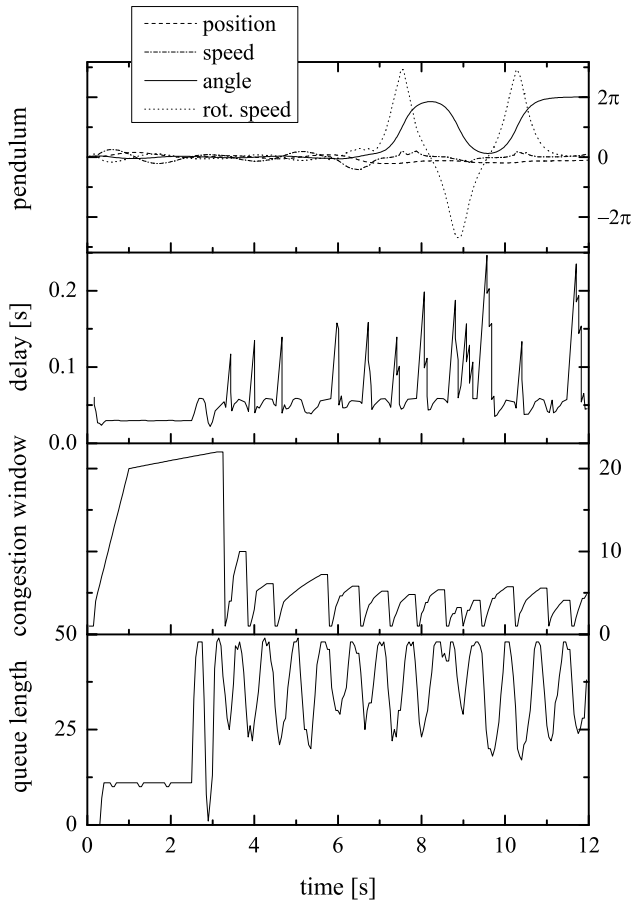

Fig. 6. At 2.5s and at 3s two UDP traffic generators start jamming 80 percent of the bottleneck's bandwidth. This results in raising message delays for the networked control system and the pendulum being unstable.

congestion window drops further and delay for messages from the controller to the pendulum increases, and in consequence the pendulum fails to reach the stability threshold.

C. Multiple Dynamical Systems

Communication between the dynamical system and its controller can be disturbed in many different ways, not necessarily by supplementary background traffic. A good example is a corporate network with many pendulum instances. Thanks to NMLab's flexibility and scalability we can provide simulation

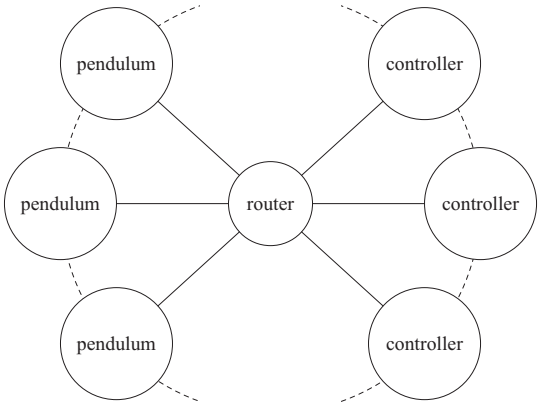

Fig. 7. Multiple Dynamical Systems. Many networked control systems share a corporate network topology interfering each other.

of such a system. In order to amplify variation in the network's load, the pendulum instances are capable of varying their sensor rates according to their distance to the equilibrium point. The sensor rate is limited to the interval $[10, 30]\mathrm{Hz}$ and can be calculated as follows:

$$30\mathrm{Hz} - 20\mathrm{Hz} \cdot \exp(-\|x\|), \tag{4}$$

where $\|x\|$ is the individual system's norm defined in Equation (3). In short: systems which have nearly reached their equilibrium reduce their sensor rate, and thereby increase the global systems performance. Pendulum and controller instances are arranged in a star topology, see Figure 7. The central routing node has an internal bottleneck with a bandwidth of $10\mathrm{Mbps}$ and a queue size of $50\mathrm{packets}$ which affects each routed packet. Data payload is being sent using TCP protocol again.

One hundred and thirty pendulum instances with different initial angles φ are simulated until all of them have reached the stability threshold. Within Matlab the queue length at the routing node can be monitored during the simulation. Here, it is shown, together with the "mean norm" of all pendulum systems, which is computed as the arithmetic mean of all pendulum state's maximum norm values, and the number of pendulums which meet the stability threshold in Figure 8. All states have been measured every millisecond, but a Fast Fourier Transform (FFT) filter with a window size of $50\mathrm{ms}$ has been applied to the queue length to mask high frequencies and smooth the graph. The initial peak in the queues (see queue length between 0 and $2\mathrm{s}$) is caused by many TCP connections starting simultaneously. Some of the initial handshakes time out that leads to another smaller peak at $3\mathrm{s}$ (TCP's default initial retransmission timeout value). Figure 8 shows the congestion window value for both of such flows. As the mean norm of the pendulum diminishes, the network load decreases slowly.

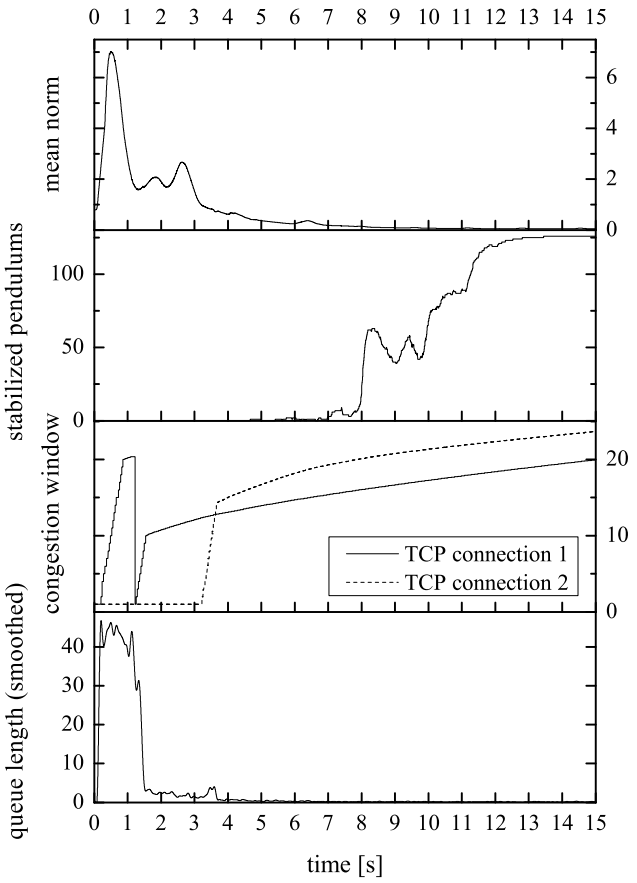

Fig. 8. Simulating 130 pendulum instances with a corporate bottleneck for communication with their controllers. TCP congestion window sizes have been monitored at two selected flows. Pendulums reduce their sensor rate along with their distance to the upward equilibrium point, so the network load decreases, as the pendulums come up to their control target.

IV. Discussion

Considering Figure 6, it can be seen, that higher message delays cause the pendulum to be unstable. Although this is quite obvious qualitatively, usually it is not perspicuous, to what extent the impact of non-trivial delay and bandwidth dynamics will affect the plant quantitatively. Currently, it is not possible to give tight bounds on the maximum delay that an arbitrary control systems can stand before losing stability. Even in the case of no interfering traffic, see Figure 4, which leads to much simpler dynamics of the TCP flows, it is not known, how to predict the resulting performance of the overall system precisely.

Figure 4 also shows that the heuristic statement, greater delays result in worse performance, is not true. This can be seen by the decreasing amount of time between 40 and $60\mathrm{ms}$ until the pendulum reaches the stability threshold. These observations explain the need for a simulation framework, which is capable of simulating the network in a realistic manner, evaluating the influence of various network parameters (e.g., TCP's parameter settings, buffer sizing, packet loss, etc.).

Many systems sharing the same network bottleneck can lead to interesting TCP dynamics, even without any

background traffic. Real existing NCS, e.g., modern cars, are much more complex than the system simulated in Section III-A. They often consist of hundreds of sensors, actuators and electrical as well as mechanical systems. In Section III-C we demonstrated the capability of NMLab to deal with such situations by simulating 130 instances of the pendulum. In this particular simulation, the individual subsystems adopt their sampling rates dynamically, i.e., they adjust their sensor rates according to their distance to the stability threshold. The ability of NMLab to have access to various parameters, more precisely describing the network (thanks to coupling with ns-2), makes possible to design and simulate controllers, which also take into account the specific states of the network, e.g., a controller that depends on the last measured delay.

V. Conclusion and Further Work

The non-trivial dynamics of an underlying communication network makes the design of stabilising controllers a demanding task. It is often not clear, how the delays and losses introduced by the network will impact the performance or the stability of a NCS.

Testing the performance of the controllers in many different networking scenarios (e.g., with different network delays, different network topologies, or different protocols) requires usually many simulation runs, and may turn to be very complex. Presented co-simulation framework NMLab, provides through Matlab, the capability to change easily both, the network parameters, and the parameters of the control system. We demonstrated flexibility and versatility of NMLab through simulations of the well known pendulum on a cart, but we purport it may be also applied to more complex NCS.

Future work may contain integration of existing topology generators into NMLab, which will result in simpler creation process for different large-scale networks with realistic traffic behaviour. Modelling movement of wireless nodes with NMLab in Matlab would permit co-simulation of NCS with radio connections.

Acknowledgment

This work has been partially supported by the German Research Foundation (DFG) as part of the priority programme 1305: Control Theory of Digitally Networked Dynamical Systems.

The work on this article at Hamilton Institute was supported by SFI grant 07/IN.1/I901.

References

[1] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proc. of IEEE* Special Issue on Technology of Networked Control Systems, vol. 95, no. 1, pp. 138–162, Jan. 2007.

[2] The MathWorks, Inc. (2010) Matlab (rel. 2008b – 2010a). [Online]. Available: http://www.mathworks.com/

[3] Digiteo. (18.02.2010) Scilab (ver. 5.2.1). [Online]. Available: http://www.scilab.org/

[4] S. McCanne and S. Floyd. (17.06.2009) The network simulator ns-2 (ver. 2.34). [Online]. Available: http://www.isi.edu/nsnam/ns/

[5] G. F. Lucio, M. Paredes-Farrera, E. Jammeh, M. Fleury, and M. J. Reed, "OPNET Modeler and ns-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed," in *3rd WSEAS Transactions on Computers*, N. Mastorakis, Ed., vol. 2, 2003, pp. 700–707. [Online]. Available: http://nsl.csie.nctu.edu.tw/NCTUnsReferences/weas.pdf

[6] UC Berkeley. (26.02.2010) Ptolemy II (8.0.beta). [Online]. Available: http://ptolemy.berkeley.edu/ptolemyII/

[7] The MathWorks, Inc. (2009) SimEvents (ver. 3.1). [Online]. Available: http://www.mathworks.com/products/simevents/

[8] D. Henriksson, A. Cervin, M. Andersson, and K.-E. Årzén, "Truetime: Simulation of networked computer control systems," in *Proceedings of the 15th World Congress*, The International Federation of Automatic Control, Ed., 2002.

[9] J. R. Hartman, "Networked Control System Co-Simulation for Co-Design: Theory and Experiments," Master's thesis, Case Western Reserve University, Cleveland, 03.06.2004. [Online]. Available: http://dora.cwru.edu/msb/pubs/jrhMS.pdf

[10] A. T. Al-Hammouri, M. S. Branicky, and V. Liberatore, "Co-simulation Tools for Networked Control Systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, M. Egerstedt and B. Mishra, Eds., vol. 4981. Springer, 2008, pp. 16–29. [Online]. Available: http://dora.cwru.edu/msb/pubs/hscc08.pdf

[11] M. S. Hasan, H. Yu, A. Griffiths, and T. C. Yang, "Co-simulation framework for Networked Control Systems over multi-hop mobile ad-hoc networks," in *Proceedings of the 17th World Congress*, The International Federation of Automatic Control, Ed., 2008, pp. 12 552–12 557.

[12] V. Dham, "Link establishment in ad hoc networks using smart antennas," Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, 2003. [Online]. Available: http://scholar.lib.vt.edu/theses/available/etd-05072003-180228/unrestri%cted/etd.pdf

[13] G. Perbellini. (2005) SystemC - ns-2 Co-simulation using HSN. Verona. [Online]. Available: http://esd.sci.univr.it/pages/docs/SC_NS2_cosim.pdf

[14] U. Hatnik and S. Altmann, "Using ModelSim, Matlab/Simulink and NS for Simulation of Distributed Systems," in *International Conference on Parallel Computing in Electrical Engineering, 2004*. IEEE Computer Society, 2004, pp. 114–119.

[15] S. Nethi, M. Pohjola, L. Eriksson, and R. Jäntti, "Platform for Emulating Networked Control Systems in Laboratory Environments," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007*. IEEE, 2007.

[16] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: Concepts and design*, 4th ed., ser. International Computer Science Series. Harlow: Addison-Wesley, 2009.