

Networked Control System Co-Simulation for Co-Design

Michael S. Branicky, Vincenzo Liberatore,¹ and Stephen M. Phillips
 Electrical Engineering and Computer Science Dept.
 Case Western Reserve Univ., Cleveland, OH 44106-7071 U.S.A.
 {msb11, vx111, smp2}@cwru.edu

Abstract

We provide a general framework for networked control systems (NCSs) and review previous theoretical results for NCS co-design. We present experimental studies of control and feedback scheduling of NCSs, consisting of dynamic system simulations for the control agents and environment and packet-level network simulations for the communications. To this end, we have extended the `ns-2` release in order to simulate the transmissions of plants and controllers that are modeled by ODEs (solved via a linked package). Our results show the overall control and network performances achieved while modeling the individual control and network components. Major co-design issues, such as scalability and network heterogeneity, are explored.

1 Introduction

Technological advances are delivering devices that have sensing and communication capabilities and that can be ubiquitously embedded in the physical world. A networked control system (NCS) consists of numerous physical and computational elements or agents, which have physical and informational interactions and dependencies, supported by overlapping network resources. Here, an “agent” could be a desktop computer; merely a sensor, an actuator, or a control module; or, hierarchically, an ensemble of such devices. Whenever there is distributed sensing, actuation, or computing of such kind, an NCS involves communication patterns in which both informational and physical control loops are closed through a real-time network.

Two examples are shown in Figs. 1 (left) and 2. The first consists of four aircraft in mountainous terrain. They are connected by line-of-sight communications; the available network connections may change in time as the vehicles’ communication is obscured by obstacles. In Fig. 2, a more complicated architecture is shown, where groups of agents are connected via multiple local area networks (LANs). There, Network A is wireless Ethernet, Network B is a real-time bus, and Network C represents satellite communications links.

In this paper, our objective is to develop a comprehensive mathematical and simulation framework for the modeling, analysis, design, and performance verifica-

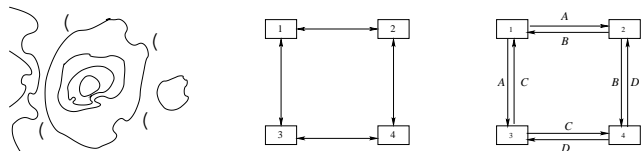


Fig. 1: Formation example: physical situation (l), information flow graph (m), network multigraph (r)

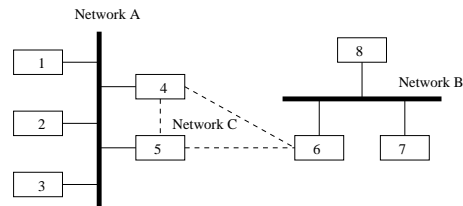


Fig. 2: Multiple LANs Example: network architecture

tion of NCSs. At the heart of our work is the point of view that the design of both the communication middleware and the interacting controlled systems should not be treated as separate. In the co-design approach we advocate, network issues such as bandwidth, quantization, survivability, reliability, scalability, and message delay will be considered simultaneously with controlled system issues such as stability, performance, fault tolerance and adaptability.

The paper is organized as follows. In the next section, we formulate a very general mathematical model of NCS architectures, and then discuss the system-level optimization problems that NCSs must solve to achieve performance within stability and network constraints. Section 3 reviews theory on the stability and performance analysis of NCSs. Section 4 overviews our co-simulation approach. Section 5 concentrates on NCSs connected via Ethernet.

2 General NCS Framework

2.1 Mathematical Model

An *NCS Architecture* consists of numerous physical and computational elements or *agents*, which have physical and informational interactions and dependencies, supported by multiple, overlapping network resources. Mathematically, we formulate an NCS Architecture is a three-tuple as follows.

¹Supported in part by NASA grants NAG3-2578, NAG3-2799.

(1) *Agent Dynamics: a collection of stochastic hybrid systems.* The underlying dynamics of each agent is that of a *hybrid dynamical system*, which is impacted (a) directly by other agents' continuous and discrete states/outputs, (b) indirectly by the receipt of state or output information from other agents over one or more communication networks, and (c) by internal and external stochastic elements and disturbances. In equations, we may write

$$\begin{aligned} \dot{X}_i(t) &= f_i(Q_i(t), X_i(t), Q_{\bar{i}}[t], Y_{\bar{i}}[t], R(t)) \\ Y_i(t) &= g_i(Q_i(t), X_i(t), Q_{\bar{i}}[t], Y_{\bar{i}}[t], R(t)) \end{aligned} \quad (1)$$

Here, X_i , Y_i , Q_i are the continuous-state vector, continuous-output vector and discrete mode of the i th agent, respectively. $Y_{\bar{i}}$, $Q_{\bar{i}}$ are those corresponding to the other agents, and $R(t)$ is a vector of stochastic variables. The $[t]$ notation denotes a vector of event or update times, and reflects the fact that some of the continuous and discrete state/output information is asynchronously communicated and dated (say, with a time-stamp from the last communication time). For difference equations, we would similarly write

$$\begin{aligned} X_i[t+1] &= f_i(Q_i[t], X_i[t], Q_{\bar{i}}[t], Y_{\bar{i}}[t], R[t]) \\ Y_i[t] &= g_i(Q_i[t], X_i[t], Q_{\bar{i}}[t], Y_{\bar{i}}[t], R[t]) \end{aligned} \quad (2)$$

(2) *Network Information Flows: a directed graph.* This graph summarizes the information dependencies detailed in the dynamic equations above. For ease of notation, we will describe the case where each agent-to-agent information flow comes in a single "packet". Thus, we have a graph $G_I = (V, E_I)$, where each information node in $V = \{1, 2, \dots, N\}$ is an agent index and an edge $e = (i, j) \in E_I$ indicates that agent i sends information to agent j via a network connection. An example is shown in Fig. 1 (middle). The communication pattern shown is that required for formation-keeping in the given mission.

(3) *Network Topology: a colored, directed multigraph.* This multigraph summarizes the communication resources in terms of the available links/networks. It is a multigraph, because agent i and agent j may be connected in more than one way (say, by wired Ethernet and wireless line-of-sight, or by multiple buses). Each link is colored/labeled by its network membership. Thus, we have a multigraph $G_N = (V, \mathcal{C}, E_N)$. Here, each node in $V = \{1, 2, \dots, N\}$ is an agent, each $c \in \mathcal{C}$ is a network and an edge $e = (c, i, j) \in E_N$ indicates that agent i is connected to agent j across network c . An example is shown in Fig. 1 (right). The links colored by A in that diagram, e.g., represent the network communications that Agent 1 can form.

An NCS must perform its overall goals to the best of its ability based on the constraints arising from individual agents and the network resources by which they are connected. For the sake of argument, we assume that an NCS's performance can be captured by a function

$$J = \sum_{i=0}^N w_i J_i(\theta, K), \quad (3)$$

where J_0 represents overall NCS performance objectives, J_i that of the i th agent, and the w_i are weighting factors. The vectors θ and K are meant to generically capture all relevant parameters and design variables, respectively, impacting these performances. With this in hand, we define:

NCS Problem. Maximize J subject to stability and network communication constraints.

2.2 Technical Approach

The NCS Problem above may be attacked holistically using our *co-design* approach [5]. Designs may be verified using our newly-developed co-simulation framework (introduced below). As the design choices are many, and the interacting tradeoffs are complex, cycling this process is anticipated. See Fig. 3.

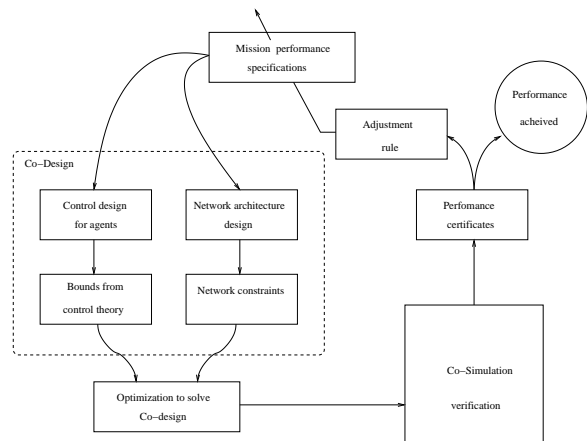


Fig. 3: Overall NCS Design Loop

The co-design approach begins by expressing both the control and network constraints. The stability constraints for each agent may be derived as (perhaps probabilistic) bounds on the latencies and/or transmission rates of all communicated variables, as determined by analyzing each agent's hybrid dynamical equations. The tools to accomplish this are described next.

3 Previous Work

3.1 NCS Stability Analysis

The insertion of communication networks into feedback control loops makes the analysis and design of an NCS complex. Conventional control theories with many ideal assumptions (such as equal-distance sampling, synchronized control, and non-delayed sensing and actuation) must be reevaluated before they can be applied to NCSs.

Specifically, the following fundamental issues are of concern, as depicted in Fig. 4: (1) time-varying transmission period, (2) network schedulability, (3) network-induced delay (or latency), and (4) packet loss.

The transmissions of information on an NCS can be periodic or aperiodic, depending on the Medium Access

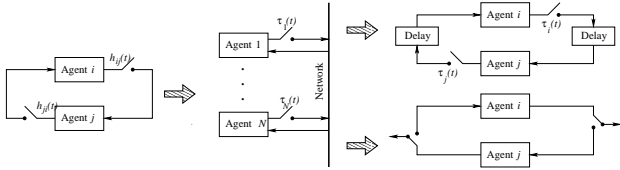


Fig. 4: Fundamental Issues in NCS

Control (MAC) protocol of the control network. So, the first issue is NCSs with time-varying transmission period. This results in a time-varying system model, making stability analyses non-routine. See Fig. 4 (left).

Network scheduling is the primary issue in NCS design when a set of agents are connected over the available network resources. With no coordination, concurrent transmissions will occur and someone has to back off to avoid collisions or bandwidth violations. This results in transmissions of some real-time data being delayed or even missing their deadlines. Good control scheduling algorithms will try to minimize such system performance loss. See Fig. 4 (middle).

Another issue is the network-induced delay that occurs while exchanging data among agents connected to the shared medium. This delay, either constant or time-varying, can degrade the performance of control systems designed without considering it and can even destabilize the system. See Fig. 4 (right, top).

One more issue is that networks can be viewed as a web of unreliable transmission paths. Some information packets not only suffer transmission delay, but even worse, can be lost during transmission. Thus, how such packet dropouts affect the performance of an NCS is an issue that must be considered. Finally, plant outputs may be transmitted using multiple network packets (so-called *multiple-packet transmission*), due to the bandwidth and packet-size constraints of the network. Because of the arbitration of the network medium with other nodes on the network, chances are, all/part/none of the packets could arrive at the time of control calculation. See Fig. 4 (right, bottom).

In our own previous work [2, 3, 4], we have addressed all these issues for the case of linear plants and controllers. Preliminary results were shared in [3]. In further studying the stability of NCSs with time-varying transmission period, sufficient conditions on the transmission period to guarantee the stability of linear-agent NCSs have been derived. Our main *NCS stability theorem* [4] is based on our Multiple Lyapunov Function approach [1], which aims to guarantee the decrease of a Lyapunov function at every transmission event. This technique is much less conservative than those previously used in the literature (including our own) [2, 3, 6] because those results required a Lyapunov function that is guaranteed to be decreasing at all times. Two versions of the bound were derived; their effectiveness was compared using specific examples. We also used an exhaustive search method to determine the bounds on the transmission period numerically. In many exam-

ples, these led to tight (necessary and sufficient) analytic bounds. Our NCS Stability Theorem is quite general. In particular, it is immediately applicable to the general situation described in Equations (1) and (2) in its sufficiency form.

3.2 Control and scheduling co-design

Now, consider the case of a set of agents that are connected to a network, as illustrated in Fig. 4 (middle). Each agent has a required transmission rate (a sufficient condition found using the theory above) of their data in order for the interactions among agents to be stable. However, when the transmission path is shared with other agents, transmission scheduling among the plants has to be considered.

Concepts of network scheduling in NCSs may be extended from those for CPU scheduling [5]. Both cases involve allocating a shared resource to a set of concurrent tasks; both involve frequent invocations of concurrent tasks, and both tasks have real-time constraints and have deadlines to be met. However, in the case of network scheduling in NCSs, the shared resource becomes the network instead of the CPU processor, and the execution of a real-time task has been replaced by the transmission of a data packet.

Already, we have analyzed the schedulability of agents using the RM scheduling algorithm [5]. We gave examples of how such optimization is carried out assuming both ideal transmission and with network-induced delay, packet dropouts, and multiple-packet transmissions taken into account.

4 Co-Simulation Methodology

Networked control systems pose novel challenges to mathematical analysis and design. A thrust of our work is to evaluate these issues using tools we have developed that combine dynamic system simulations for the control agents and environment (using an ODE solver) and packet-level network simulations for the communications (by extending `ns-2`). See Fig. 5.

4.1 Starting Point

Our most basic objective is to establish whether a certain network, plant, or controller configuration makes it possible to achieve a plant output that is close to the reference trajectory. One of our first findings is that in many instances the error is small when, say, there are few connected plants, but it becomes extremely large rapidly as soon as one or few more plants are connected. In other words, the observation was not the graceful degradation of the plant output, but a clear threshold behavior. Threshold behaviors lead to large errors after a small change of parameter values, and we observed that certain traditional metrics of network performance, such as drop rate, do not always follow a threshold behavior in correspondence to plant output behavior.

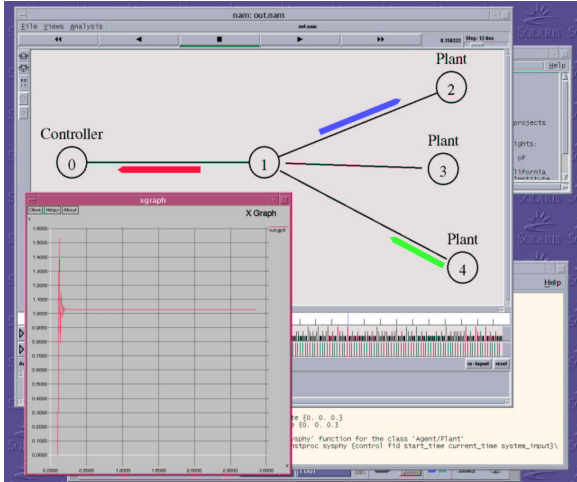


Fig. 5: A snapshot of the simulator’s graphical interface during a three-plant demonstration.

The identified threshold behavior has implications in two areas: feasibility and scalability. If the error is large even with only $n = 1$ plant connected, then the threshold occurs for $n \simeq 0$, and the network, plant, or controller configuration do not make remote control possible at all. An example of an infeasible network is shown in Section 5.5 where one concurrent FTP flow effectively prevents convergence of a single plant. Another type of effect is when the threshold is strictly positive. Ideally, we would like to connect as many plants as the network bandwidth can support, but this is not possible in many simulations, for example, due to contention for the use of a switch buffer. In these cases, certain buffer configurations sustained up to 38 plants, whereas other configurations only sustained 25. We will say that the former configurations are more *scalable* than the latter in that they scale up to a heavier workload. A main claim in this paper is that threshold behaviors of a distributed network control system can be as important a characteristic as individual plant output, and that, correspondingly, scalability as a performance metric is as important as output error. Our claim is particularly evident in the case of the feedback gain parameter K for the following reasons. When there are few plants connected the network, the error (defined in Section 4.3) was low for different values of K , and a larger value of K decreased an already small error. However, larger values of K cause practically unbounded output values once a threshold on the number n of plants is crossed, and thus a larger K is not as scalable as a smaller one. Thus, the most important effect of K was not to reduce error, but to hamper the system scalability. For these reasons, we believe that scalability (and the implied threshold behavior) are critical performance metrics in networked control systems.

4.2 Network Topology and Parameters

The network is simulated with the `ns-2` package, a widespread state-of-the-art open-source network simu-

lator package for studying packet dynamics in networks [7]. We have extended the `ns-2` release to simulate the transmissions of plants and controllers. Specifically, we can create a new “agent” to represent a plant. Such an agent samples its output at a specified interval, writes the sample output into a packet, and sends it to the controller. The `ns-2` simulator will then follow the packet through the network as it is transmitted over links, queued at router buffers, dequeued, and finally reaches the controller. The controller immediately generates a control u and writes u in a packet that is delivered through the network to the plant. Fig. 5 shows the graphical interface of the simulator as it is running an experiment with 3 plants linked to a controller through an intermediate router. The boxes over the links represent packets in transit over the link and move over time in an animated demonstration.

We executed simulations with the network topology shown in the light gray window of Fig. 5. The plants (nodes 2 to 4) are connected to a Fast Ethernet switch (node 1). The plant-switch links have a $120\mu s$ latency, which includes propagation delay as well as per-frame processing at the end-points. The figure shows a network topology with 3 plants, but simulations were also executed with a different number n of plants attached to the switch. Plant 2 is a distinguished plant, whose output will be traced over time. The other plants are added to simulate a scenario where plant 2 is in contention with other units for network resources. Plant 2 samples its output every 10ms, and each plant $i > 2$ samples its output with constant period T_i , where T_i is a random value extracted uniformly at random between 5ms and 15ms. Output samples are represented by double-precision floating point numbers. Thus, a scalar output sample occupies a packet of length 48B: the double-precision payload takes 8B, the RTP header 12B, the UDP header 8B, and the IP header 20B. The switch is connected to the controller through a T1 line with a 1ms delay. Observe that the T1 line is the bottleneck and makes it possible to accommodate at most $n \leq 1.544 \cdot 10^6 / (48 \cdot 8 \cdot 100) < 41$ plants simultaneously.

Additional experiments were executed in the presence of FTP cross-traffic. In this case, additional FTP nodes are connected to the switch by Fast Ethernet links. These nodes are FTP sources that emulate the transmission of plant operation logs to the controller (e.g., [8]). The FTP session uses TCP SACK/DelAck with a segment size of 1460B, an advertised window of 16 packets, a slow-start threshold of 44 packets, a clock granularity of 10ms, and a DelAck time-out of 50ms (these parameters are typical for today’s Internet).

In terms of packet dynamics, the fundamental metrics are: (1) *delays* (one-way from plant to controller; the round-trip times—RTTs—can be directly calculated from the one-way delays in the network under consideration), (2) *drop rate* (the fraction of sample packets dropped), and (3) *inter-arrival gaps* (time between reception of successive control packets at the plants). In most cases, we have collected both average values and the distribution of these quantities.

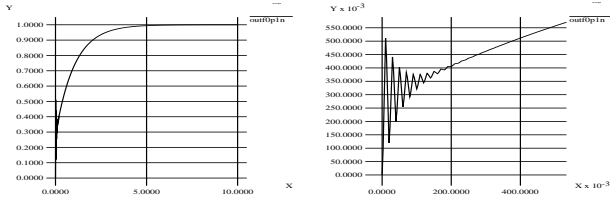


Fig. 6: Plant 2 output ($n = 1$, $K = 101$). 10s (l), 0.3s (r).

4.3 Plant and Controller Dynamics

We begin with a simple experimental setup for the network topology shown in Fig. 5. The distinguished plant (Plant 2) has a scalar output $y(t)$ defined by

$$\dot{x}(t) = 100x(t) + u(t), \quad y(t) = x(t),$$

where $u(t)$ is the plant input. The initial plant state is 0 and the target plant output reference trajectory is $r = 1$ (unit step response). If the output is sampled at time t , the (subsequently computed and sent) control signal comes back to the plant at time $t + h_{\text{rtt}}(t)$, where $h_{\text{rtt}}(t)$ is the *round-trip time* for feedback to arrive from the plant output measurement taken at time t . In general, this is a time varying, stochastic quantity depending on network loading and parameters. However, if no other plants are connected to the network, it is 2.745ms since the propagation time on the T1 line is 1ms, the propagation time on the Ethernet link is $120\mu\text{s}$, the transmission time on the T1 line is $48 \cdot 8/1.544 \cdot 10^6 \simeq 248.705\mu\text{s}$, and the transmission time on the Ethernet link is $48 \cdot 8/10^8 = 3.84\mu\text{s}$. Upon receipt of the output sample $y(t)$, the controller predicts the output $y(t)$ forward to $\hat{y}(t + h_{\text{rtt}}(t))$ and then applies the control $u = K(r - \hat{y}(t + h_{\text{rtt}}(t)))$ (the controller assumes a delay slightly smaller than the actual delay, and so u arrives slightly late at the plant). Differential equations for the plant and the controller are solved numerically with the `Ode` utility within a relative precision of 10^{-5} . In certain experiments, the value of the plant state exceeds the floating point representation for the `Ode` input (`%10f`), in which cases the simulation stops and reports that the plant has *crashed*. In these cases, the plant output is assumed to be growing without bound. The plant output $y(t)$ is shown in Fig. 6 for the case when $n = 1$ and $K = 101$. This aggressive control gain is chosen to more clearly show the effect of the network. The plant output converges to the target output. However, a damped oscillation is present in the output due to the interaction of our slight delay underestimate with the high control gain.

Due to the vagaries of the network level of service in networked control systems, performance metrics with inherent smoothing are more appropriate than classical bounds on step response transients (e.g., Fig. 8 and 10). Thus we consider integral squared error as a performance measure: $\frac{1}{T} \int_0^T (y(t) - r(t))^2 dt$.

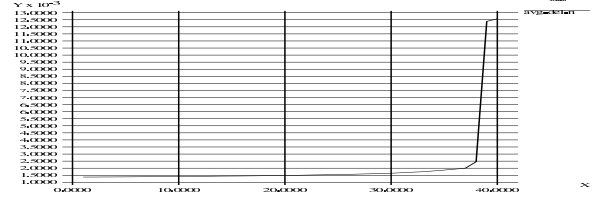


Fig. 7: One-way delays (plant to controller) as a function of n for the case of infinite buffers.

5 NCSs over Ethernet

Ethernet switches have output buffers on their interfaces. In the first part of this section, we discuss the implications of different choices for the output buffer sizes at switch 1. We then turn to the impact of different values of the control gain and of FTP cross-traffic.

5.1 Infinite Buffer

Our objective is to estimate the impact of buffer sizes and number n of plants on the plant output. The network topology and parameters make it possible to overrun only one such buffer, viz. the one on the T1 interface, so simulations were executed for different sizes of this one buffer. First, we assume that the buffer has infinite size. Since infinite memory cannot be installed in a switch, the results in this section serve mostly as a point of comparison for finite-buffer switches considered in the next sections. If the buffer is infinite, no packet is ever lost, but packets can be delayed by an arbitrarily amount of time in the buffer queue. Fig. 7 plots the average packet delays as a function of the number n of plants. The delay distribution has a knee for $n = 39$, at which point the delay skyrockets from 2.45ms to 12.63ms. At lower loads, the delays increase from 1.373ms ($n = 1$) to 2.45ms. We now focus on the plant output for $n = 38$ and $n = 39$. When $n = 38$, the system response is basically indistinguishable from that for $n = 1$ in spite of the longer delays (the integral squared error $e(S)$ remains basically unchanged). When $n = 39$, the plant crashes after around 2s. We conclude that even with switched networks and infinite buffers, the network should not be overloaded.

5.2 Finite Buffer

The obvious benefit of a finite buffer is that the delays are now bounded: a packet can be queued at most behind the maximum number of slots allocated in the buffer. The flip side, though, is that the switch can drop packets if there is no room left in the buffer. If a sample packet is dropped, the plant misses a control packet and the previous input is continuously applied until the next reception of a control packet. Fig. 8 gives the drop rate as a function of n for a buffer of size 4 (the average inter-arrival gap was basically constant across values of n). The error is typically small until the network is almost saturated, at which point it increases rapidly. The corresponding plant output can take unpredictable shapes at high loads (Fig. 8). We remark that the average inter-arrival gap only changed slightly,

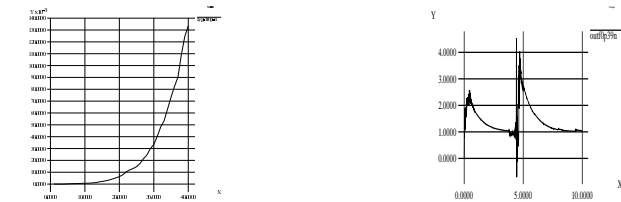


Fig. 8: Buffer of size 4. Drop rate up to $n = 39$ (l); plant output (r) for $n = 39$.

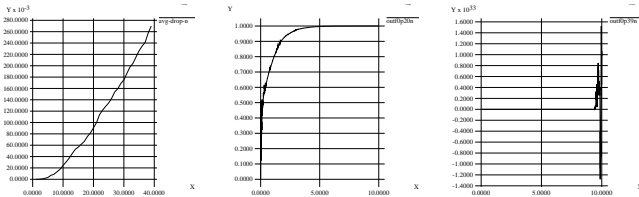


Fig. 9: Buffer of size 2. Drop rate up to $n = 39$ (l); plant output for $n = 20$ (m) and $n = 39$ (r).

and thus the plant output and performance appear to be tied more to the characteristics of the history of inter-arrival times than to the average gap values.

5.3 Minimal Buffer

The minimum buffer size in a store-and-forward switch is 2 packets (one to forward a frame, and the other to store an upcoming frame). Plant 2 crashed at $n = 40$, but the simulation completed successfully for $n \leq 39$. Fig. 9 gives the drop rate as a function of n , and the plant output for $n = 20$ and $n = 39$. At $n = 20$, the output y is basically the same as in the unloaded scenario $n = 1$ (Fig. 6), although some small jitter can be noticed in Fig. 9. At $n = 39$, the plant output exhibits wide deviations, especially towards the end of the simulations. The integral squared error as a function of n was qualitatively similar to that for a size 4 buffer (Fig. 8), with negligible errors for small values of n and a sudden increase for larger value of n . However, the smaller buffer caused the error to begin diverging for a smaller number of plants ($n = 25$).

In summary, both infinite and size-4 buffers effectively supported 38 plants with minimal error, whereas a size-2 buffer supported only 24 plants. We conclude that switch buffer size is a critical factor in scalability.

5.4 Control Gain

A larger value of K should result in better performance, but could also lead to large errors in a networked control system. We have explored $K = 110, 120$, but at $K = 120$ even a single plant crashed. At $K = 110$, the integral squared error $e(S)$ was better than that for $K = 101$ (roughly 0.03 vs. 0.27) for small values of n but it then diverges at $n = 29$ (buffer of size 4). Thus, higher gain led to slightly better performance at low utilization, but can negatively impact the system scalability.

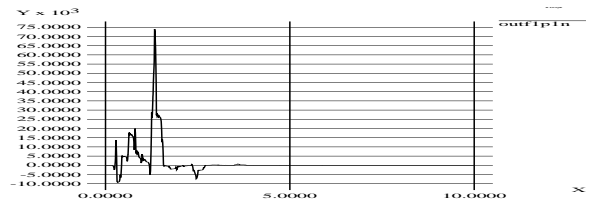


Fig. 10: Plant output in the presence of one FTP flow.

5.5 TCP Cross-Traffic

The obvious problem with infinite buffers is that infinite memory cannot be installed in a physical switch. The not-so-obvious problem is that infinite buffers are an open invitation for a concurrent TCP flow to dump an arbitrarily large file in the switch buffer, and thus force plant packets to long waiting times. Thus, the survival of the plant flows demands a buffer of finite size in the presence of concurrent TCP flows. Furthermore, TCP's performance is usually optimized when the buffer size is between 1.3 and 2 times the bandwidth-delay product (BDP) of the link [9]. In this case, the BDP is extremely small, and so we henceforth take arbitrarily a buffer of size 4. Fig. 10 gives the plant output with one FTP flow for the cases when the buffer has size 4. In the case of infinite buffers, the plant crashes after about 1s. The behavior with a finite buffer is erratic, but the plant eventually converges to the target output (the integral squared error was about $523 \cdot 10^6$). Moreover, we observe that TCP's throughput was substantial: about 128kB per second, which corresponds to 68% of the link bandwidth.

References

- [1] M.S. Branicky, Multiple Lyapunov functions and other analysis tools for switched and hybrid systems, *IEEE Trans. Automatic Control*, **43(4)**:475–482, 1998.
- [2] M.S. Branicky, S.M. Phillips, and W. Zhang, Stability of networked control systems: Explicit analysis of delay, in *Proc. ACC.*, pp. 2352–2357, Chicago, 2000.
- [3] W. Zhang, M.S. Branicky, and S.M. Phillips, Stability of networked control systems, *IEEE Control Systems Magazine*, **21(1)**:84–99, 2001.
- [4] W. Zhang and M.S. Branicky. Stability of networked control systems with time-varying transmission period. *Proc. Allerton Conf.*, Urbana, IL, Oct. 2001.
- [5] M.S. Branicky, S.M. Phillips, and W. Zhang. Scheduling and feedback co-design for networked control systems. *Proc. IEEE CDC*, Las Vegas, 2002.
- [6] G.C. Walsh, H. Ye, and L. Bushnell, Stability analysis of networked control systems, in *Proc. American Control Conf.*, pp. 2876–2880, San Diego, 1999.
- [7] L. Breslau *et al.* Advances in network simulation. *Computer*, **33(5)**:59–68, 2000.
- [8] S.-K. Kwon *et al.* Achieving real-time communication over ethernet with adaptive traffic smoothing. In *Proc. IEEE Real Time Tech. Appl. Symp.*, 2000.
- [9] M. Christiansen *et al.* Tuning RED for Web traffic. In *Proc. Sigcomm*, pp. 139–150, 2000.