Decentralized and Dynamic Bandwidth Allocation in Networked Control Systems

Ahmad T. Al-Hammouri[1], Michael S. Branicky[1], Vincenzo Liberatore[1], and Stephen M. Phillips[2]

[1]Case Western Reserve University
Dept. of Electrical Engineering and Computer Science
Cleveland, Ohio 44106 USA
{ata5, mb, vl}@case.edu

[2]Arizona State University
Dept. of Electrical Engineering
Tempe, Arizona 85287 USA
stephen.phillips@asu.edu

Abstract

In this paper, we propose a bandwidth allocation scheme for networked control systems that have their control loops closed over a geographically distributed network. We first formulate the bandwidth allocation as a convex optimization problem. We then present an allocation scheme that solves this optimization problem in a fully distributed manner. In addition to being fully distributed, the proposed scheme is asynchronous, scalable, dynamic and flexible. We further discuss mechanisms to enhance the performance of the allocation scheme. We present analytical and simulation results.

1. Introduction

We are witnessing technological advances in VLSI, in MEMS, and in communication networks that have brought devices with sensing, processing, actuating, and communication capabilities. These devices have contributed to the formation of *distributed control systems* over communication networks, which can be used to monitor and to control the physical world around us, e.g., [2]. Sensors generate a stream of sensed data and communicate it over a network to controllers. Controllers process the samples of the sensed data and generate appropriate control signals to be delivered over the network to actuators. Actuators transform control signals into actions that affect the physical world.

Distributed control systems are often implemented over local and proprietary networks [6]. Recently, open-standard networks have emerged as a suitable and convenient communication media for distributed control due to lower costs, higher speeds, and easier installation and configuration [10]. However, distributed control implementations over the open-standard networks are still confined to local- and limited-area networks. In contrast, we address distributed control systems over wide-area and geographically distributed networks. More specifically, we lay foundations for closing the feedback control loops over IP networks. We envision that one day a term like *CoIP* (Control over IP) will become as common as VoIP (Voice over IP). In this regard, Liberatore has proposed in a recent paper an end-to-end algorithm, which integrates play-back buffers, adaptive sampling, and control strategies, to enable control systems to adapt to varying levels of network service [14].

In this paper, we propose a scheme that efficiently allocates the network bandwidth among several control systems. Network bandwidth arises as a crucial issue if distributed control is to be deployed ubiquitously over IP networks. This is true because when several processes compete for a finite resource, such as network bandwidth, and no proper coordination exists, *congestion* is a common consequence. In any bandwidth allocation scheme, a term that always accompanies efficiency is *fairness*. Our objective is thus to *fairly* allocate the bandwidth to avoid congestion and to meet each system's requirement as best as possible.

The idea presented in this paper is that control systems vary their *sampling periods* (thus their bandwidth consumption) based on the congestion level fed back from the network. Our proposed scheme has the following features:

- It allocates the bandwidth in a way to ensure stability of all control systems, if feasible.
- It allocates the bandwidth in a way to attain the maximum *aggregate* performance of all control systems.

- It makes use of network bandwidth efficiently; controls congestion, thus minimizes delays and losses; and achieves fairness by fulfilling performance objectives of different control loops.
- It provides a *fully distributed*, an *asynchronous*, and a *scalable* solution. Each node executes an independent algorithm using local information with no central managing entity. The approach scales up as the number of controlled systems and/or the size of the network increase.
- It is *dynamic* and *flexible*. It dynamically reallocates the bandwidth as different control systems acquire and release the network.

In the literature, it has become common to refer to distributed control systems implemented over communication networks as networked control systems (NCS) [15, 26], and so we use this term throughout the paper.

The roadmap of this paper is as follows. In Section 2, we contrast our work with related work. In Section 3, we formulate our problem mathematically as a convex optimization problem, and we explain its distributed implementation. In Section 4, we model the interaction between the network and control systems as a linear dynamical system, and we study its performance. In Section 5, we present our experimental methodology, and experiments to evaluate the proposed scheme. We conclude the paper in Section 6.

2. Related Work

The issue of bandwidth management in NCS has gained considerable research attention; see for example [3, 22] and the references contained therein. All of such research efforts have focused on bandwidth scheduling in limited (local) area networks, e.g., in a car, in an airplane, or in a factory. Several reasons hinder the extension of such bandwidth scheduling schemes to Wide Area Networks (WANs) domain. These schemes usually require time synchronization among the different devices in the network. The allocation schemes are either static or dynamic. Static schemes, where allocation is determined pre-run, lack flexibility and adaptability to dynamic changes. Dynamic schemes, on the other hand, require centralized implementations. In this paper, we propose a bandwidth allocation scheme for NCSs working over WANs that is asynchronous, dynamic and flexible, and fully distributed. To the best of our knowledge, there has been no prior research into bandwidth allocation for NCSs over WANs.

In our scheme, NCSs adapt their bandwidth usage by varying their sampling periods so as to avoid congestion in the network, and to preserve high performance level. It is worthwhile here to evaluate the ideas presented in [8] and in [22]. In [8], the authors proposed an algorithm to adapt the sampling period of controlled systems implemented over a CAN bus based on two factors, network load and stability threshold. The algorithm per se is special to CAN in the way it determines the network load. Moreover, the heuristic of increasing and decreasing the sampling period has no mathematical justification. The algorithm proposed in [22] uses the network's available bandwidth and the error in each system's state to adapt the sampling period. However, the paper fails to discuss an important implementation issue: measuring the occupied bandwidth (to be used along with the network's capacity to infer about the available bandwidth). In this paper, we introduce an approach that relies on solid mathematical foundations, and we discuss its implementation details over IP networks. We also present results from a network simulator that was extended to simulate control systems [4].

3. Problem Formulation

3.1. On the Wire

Figure 1 shows a configuration of a single NCS in which the feedback loop is closed over a network. In general, the *sensor* samples the values of physical quantities, writes them in a packet, and sends the packet to the controller. The *controller* examines the received sample to generate a control signal that is then sent to the actuators.

The time interval between two sample packets is called the *sampling period*, and it is denoted by h. In other words, the sensor sends one packet containing sample data every h seconds. The reciprocal of the sampling period,

$$r = \frac{1}{h}\,, \tag{1}$$

is the *rate* of transmission from the plant to the controller. The rate can be similarly defined in the reverse path from the controller to the plant. Although, in principle, the rates in the two paths could differ, in most applications, the two rates are identical. The transmission rate is the amount of bandwidth resources that a particular plant-controller pair consumes. If the rate exceeds the end-to-end available

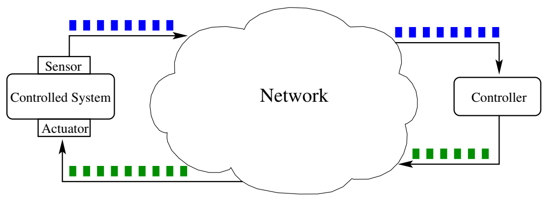

Figure 1. A networked control system with one controlled system (a.k.a. *plant*) and one controller.

bandwidth, the network is *congested*, and the communication is then characterized by packet losses, delays, and jitter. In principle, the rate should be small enough to avoid congestion. However, an NCS typically benefits from higher sampling rates. For example, the physical behavior tends to track more closely the intended reference behavior if the sampling rate is higher. In extreme circumstances, the sampling rate is so low that the physical system becomes *unstable*, in which case even small perturbations can cause massive breakdowns. Hence, the sampling rate r must strike a balance between network utilization and intended physical behaviors. The sampling rate is thus a critical tuning factor in NCSs.

The effect of the transmission rate r on the physical system dynamics is often captured by a *utility function*, $U(r)$. The utility value $U(r)$ expresses the degree to which a particular system can benefit from sampling rate r. In general, the utility function is a monotonically increasing function of the rate r, which reflects the fact that higher sampling rates lead to better control performance. In practice, the utility function is also often a strictly concave function of r, which reflects a law of diminishing returns as the rate increases. Finally, the utility function is defined only for $r \geq r_{\min}$, where $r_{\min}$ is the minimum rate below which the system becomes unstable. To carry out mathematical analysis easily, we pose an extra condition on $U(r)$ in which we require $U(r)$ to be doubly differentiable. Figure 2 shows two generic examples of utility functions associated with different applications. In the NCS literature, quadratic and exponential utility (performance) functions are commonly used [5].

3.2. Optimization Formulation

Defining a *fair* allocation to be the one that *maximizes* the sum of the utility functions of individual NCSs (the aggregate benefit of all NCSs), we formally state our objective as follows:

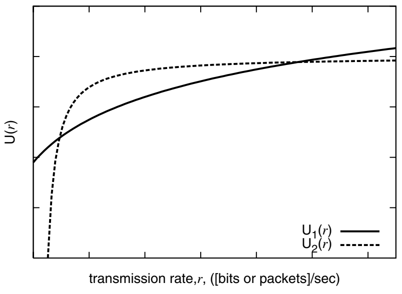

Figure 2. Examples of two generic utility functions.

Determine a transmission rate r_i of each NCS i so as to maximize the sum of utilities $\sum_i U_i(r_i)$, subject to (i) each system i's stability constraint $r_i \geq r_{\min,i}$, and (ii) each link l's capacity constraint $\sum_{i\in S(l)} r_i \leq C_l$. Or, in compact form:

$$\begin{aligned} \max \quad & \sum_i U_i(r_i), \\ \text{s. t.} \quad & \sum_{i\in S(l)} r_i \leq C_l, l = 1,\ldots,L, \\ \text{and} \quad & r_i \geq r_{\min,i}, \end{aligned} \tag{2}$$

where $S(l)$ is the set of NCSs whose communication loops use link l, C_l is the capacity of link l, and L is the total number of links in the network. In this formulation, we assume that the communication loop for each NCS can use link l only once. This assumption is always valid if all links are full duplex (in which case, forward and backward traffic do not interfere).

Due to the concavity characteristic of $U(r)$, Equation (2) is a convex optimization problem, which means it can be solved quickly and efficiently to yield a global, optimal solution [21]. However, our objective is to solve this program with a distributed approach with no centralized coordination.

3.3. Distributed Implementation

Due to its convenient structure, Equation (2) can be decomposed into separable subproblems [17]. The solution can then be implemented in a distributed fashion, whereby individual controlled systems and links execute independent algorithms. This solution is achieved by considering a dual version of (2) that incorporates the Lagrange multipliers for link capacity constraints. We summarize next the distributed algorithm and the protocol.

To facilitate this, a special header field is introduced in the sensor and the controller packets. This field is meant to carry the congestion information from links back to plants. Each link l computes a congestion level, p_l, based on local information, such as the aggregate incoming traffic and/or the queue length. When the sensor generates a packet to carry the sampled data, the plant initializes the value of this field to zero. As the packet traverses network links in the directed path form the sensor to the controller and back to the actuator, each link adds it current value of p_l to whatever value has accumulated in the field. Thus, when the control packet arrives at the plant, this special field would contain the total sum of p_l values of all individual links along the directed path from the sensor to the controller and back to the actuator. Upon receiving the controller packet, the actuator applies the control signal and the sensor regulates its sampling (sending) rate r based on the fed-back congestion information as follows:

$$r(p_t) = \min\{\max\left\{U'^{-1}(p_t), r_{\min}\right\}, r_{\max}\}, \tag{3}$$

where

- p_t is the value of p in the received controller packet, which is the sum of p_l values of all the links along the path from the plant to the controller and back to the plant;
- U'^{-1} is the inverse of the derivative of the utility function;
- $r_{\min}$ is the minimum transmission rate that satisfies the stability condition of the plant; and
- $r_{\max}$ is the maximum sampling rate and/or the maximum transmission rate a plant can use, which may stem from inherent hardware limitations of the sensor.

Based on the newly computed $r(p_t)$, the value of h is then calculated according to (1), which defines the sleeping time before generating the next sample.

In [13,16], a similar approach was applied to flow control in ATM and in IP networks, respectively. However, neither did the authors analyze the effect of communication delays between sources and links on the convergence of the algorithm, nor did they address the issue of steady-state backlog in queue lengths. Actually, [16] only suggests (with no mathematical analysis) that the algorithm's step size must be chosen sufficiently small. The authors of [12, 19, 24] addressed the issue of delay only for special families of utility functions. In [18], the author analyzed the robustness of the solution in the presence of delays. Although the authors in [9] analyzed the steady-state backlog in queues, they did the analysis only for the case of the TCP congestion algorithm. None of these previous studies addressed the issue of steady-state backlog in queues in the context of optimization-based congestion control. Here, we use a similar approach as in [9] to design the link function for computing p_l, taking into account both delays and steady-state backlogs. Unlike [9], where the authors gave a single set of parameters, we obtain a complete *region* of appropriate parameters for the link function. This provides network designers flexibility in choosing more robust and resilient algorithm behaviors.

Introducing the header field in the sensor and controller packets to carry the value of p_l, we assumed that routers are aware of and can manipulate this header. Also, we assumed that the overhead for this field is negligible (at most 64 bits for a double-precision floating-point number) compared to the size of each packet. Such assumptions are always implied for new congestion control protocols; see for example [11]. However, if practical implementations dictate otherwise, the p_l value in our protocol can be quantized and encoded by the two ECN bits that already exist in transport protocols, such as what has recently been proposed in [23].

4. Link Queue Controllers

In this section, we first model the interaction between controlled systems and links in the proposed allocation scheme as a time-delay dynamical system. Then, we design controllers for link queues to enhance the performance of the scheme. This gives rise to two types of feedback loops closed via the network. The first loop is for NCSs with distributed sensors, actuators, and controllers. The second is the loop of the developed model for the interaction between NCSs and links. Our focus in this paper is on the second loop, where we design controllers to enhance the performance of the bandwidth allocation algorithm among several NCS loops.

4.1. Modeling NCS-Queue Interaction

In practical implementations, routers connect two or more network links. So, the link algorithm is actually executed at the router that is deployed at the link's input. Routers use buffers to hold incoming packets while servicing others. Congestion at a link causes the buffer to build up and possibly to overflow. This results in long delays, jitter, and packet losses. The aim is to stabilize the buffer's queue around a controllable small size greater than zero. This has a two-fold advantage. First, a stable, small queue size eliminates excessive delays, jitter, and losses. Second, a queue size greater than zero avoids the scenario when network links are empty and underutilized because the queue will always have packets to transmit. Because of all of this, we study the performance of the allocation algorithm in terms of the queue length.

We model the interaction between N NCSs and a single bottleneck link. We assume that the incoming aggregate rate, $\sum_{i=1}^{N} r_i(t)$, at the bottleneck link is always larger than the capacity C. Based on this assumption, the rate of change in the queue length, $\dot{q}(t)$, at the link is related to the incoming aggregate traffic and the capacity of the link as follows:

$$\dot{q}(t) = \sum_{i=1}^{N} r_i(t) - C. \tag{4}$$

For simplicity, we assume that delays are constant and homogeneous among all NCSs. However, when we present our simulations, we allow for varying and different delays. We denote the delay from any plant to the queue by d_{pq}, and the delay from the queue to any plant by d_{qp}. At time instance t, plant j receives a new value of $p(t)$, and accordingly transmits packets at rate $r_j(p(t))$. This affects the queue length after a delay d_{pq}, whereby the queue computes a new value of $p(t)$. This new value of $p(t)$ reaches plant j after a delay d_{qp}. As a result, we obtain the following set of coupled equations (Equation (6) is same of (3) when neglecting the boundary conditions of $r_{\min}$ and $r_{\max}$):

$$\dot{q}(t) = \sum_{i=1}^{N} r_i(t-d_{pq}) - C, \quad (5)$$

$$r_j(t) = U_j'^{-1}\left[p(t-d_{qp})\right], j = 1,\ldots,N. \quad (6)$$

In general, $U(r)$, and thus $U'^{-1}(p_t)$ could be nonlinear functions. To study the performance of the system described by (5) and (6), we linearize both equations around the operating point (q_0, r_{j0}, p_0), where q_0 is the desired steady-state queue length, r_{j0} is the steady-state transmission rate of plant j, and p_0 is the value of p corresponding to the situation when every plant j transmits at rate r_{j0}. The corresponding linearized equations are as follows (details omitted):

$$\delta\dot{q}(t) = \sum_{i=1}^{N} \delta r_i(t-d_{pq}), \quad (7)$$

$$\delta r_j(t) = \frac{1}{U_j''(r_{j0})}\delta p(t-d_{qp}), j = 1,\ldots,N. \quad (8)$$

For simplicity we define $d = d_{pq} + d_{qp}$, $B_j = 1/U_j''(r_{j0})$, and $B = \sum_{i=1}^{N} B_i$. Combining (7) and (8) then yields

$$\delta\dot{q}(t) = \left[\sum_{i=1}^{N} B_i\right] \cdot \delta p(t-d) = B \cdot \delta p(t-d). \quad (9)$$

The only missing part here is the function that relates $p(t)$ and $q(t)$, which is the subject of the next subsection.

4.2. P and PI Controllers

We analyze the linearized model (9) in the frequency domain [7]. Without loss of generality, we analyze the linearized model's response when starting from the origin (i.e., $q(0) = 0$, $p(0) = 0$, and $r_j(0) = 0$) and converging toward the steady-state point defined by (q_0, r_{j0}, p_0). Since $\delta q(0) = q(0) - q_0 = -q_0$, the Laplace transform of (9) is $s \cdot \delta q(s) + q_0 = B \cdot \delta p(s) \cdot e^{-sd}$, which after arrangement gives

$$\delta q(s) = \frac{-q_0}{s} + \frac{B}{s} \cdot \delta p(s) \cdot e^{-sd}. \quad (10)$$

Figure 3 shows the block diagram of (10), where $C(s)$ is the Laplace transform of the function that relates $p(t)$ and $q(t)$ and is called the *queue controller*.

The closed-loop transfer function is $1/(1 - BC(s)e^{-sd}/s)$. For the closed loop to be stable, the roots of the characteristic equation, i.e., $1 - BC(s)e^{-sd}/s = 0$, must lie in the left half of the complex plane.

We next design the controller $C(s)$ to stabilize and to improve the closed-loop response. Among different controllers, the simplest are the Proportional (P) and the

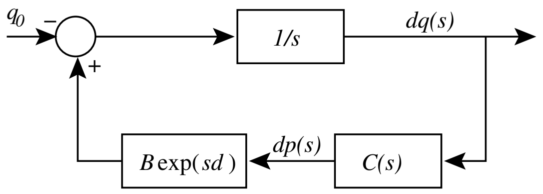

Figure 3. Linearized Model of NCS-Queue interaction.

Proportional-Integral (PI) controllers. Choosing such a simple controller algorithm enables the router to process huge amount of traffic efficiently and in real time.

The transfer function of a P controller is $C_P(s) = k_p$, and that of a PI is $C_I(s) = k_p + k_i/s$, where k_p and k_i are the proportional gain and the integral gain constants, respectively. Setting k_i to zero in the PI controller, $C_I(s)$, results in a pure P controller, $C_P(s)$. Therefore, we carry out the mathematical analysis for the general form of the controller, $C_I(s)$; when $C_P(s)$ is needed, we just let $k_i = 0$. The benefit of having a non-zero integral gain, $k_i \neq 0$, is to eliminate the stead-state error in the queue length (this will become clear when we present our results later on in the paper)

The characteristic equation with the PI controller becomes

$$s^2 - B(k_p s + k_i)e^{-sd} = 0. \quad (11)$$

The goal is then to determine the k_p and k_i values that result in a stable-closed loop response. Because of the exponential term, which originates from the delay in the feedback loop, Equation (11) has an infinite number of roots. Therefore, to ascertain the stabilizing values of k_p and k_i, Equation (11) needs special treatment beyond the classical techniques found in control theory textbooks [7]. In particular, we adapt the recent techniques found in [20] to obtain the complete set of k_p-k_i values that stabilize the closed loop. However, our system has the form of an IPD (integral plus time delay) not of a FOLPD (first-order lag plus time delay), which the authors analyzed in [20]. Following the material in [20, Ch. 7], we rework the elaborate analysis for the case of our system, and we arrive to the following conditions on k_p and k_i for a stable closed loop system (details omitted):

- The proportional gain, k_p, can take on any value in the interval $(0, \pi/(2Bd))$.
- For each given value of $k_p = \hat{k}_p$, the integral gain, k_i, can take on any value in the interval $[0, \lambda)$, where

$$\lambda = \frac{\alpha_0^2 \cos(\alpha_0)}{Bd^2},$$

and α_0 is the solution of

$$B\hat{k}_p - \frac{\alpha}{d}\sin(\alpha) = 0$$

in the interval $(0, \pi)$.

For given values of B and d, sweeping over all values of k_p values inside the range defined above and solving for k_i yields a region of stabilizing k_p and k_i gains, such as the one shown in Figure 4.

Obtaining a complete set of k_p and k_i gives the flexibility to choose values that would result in desired performance. Our model contains several sources of inaccuracies that might cause the performance of the real system to diverge form that of the linearized model. These inaccuracies are due to approximating the flow of discrete packets by a fluid-based model; linearization of the utility function; assuming delays are constant and homogeneous among different NCSs; and assuming the queue length is always > 0. Thus, it is advisable to choose k_p and k_i values inside the stabilizing region rather than at its boundaries because values inside the region result in a more resilient (in terms of model parameters) and a more robust (in terms of delay) controller.

5. Simulations

In this section, we explain the experimental setup, and we present simulation results. We have conducted extensive experiments to evaluate the previous theoretical analysis. However, given the space limitations of this paper, we present only a single (yet a demonstrative) experiment.

Simulation Software. We have extended `ns-2` [1] by adding two new agents: *NSCSPlant* and *NSCSController*, which stand for networked-sensing-and-control-systems plants and controllers, respectively [4]. NSCSPlant is an abstract agent class, which can be used to instantiate several controlled systems, each of which simulates a physical system. NSCSController can be used to instantiate a controller to control a plant.

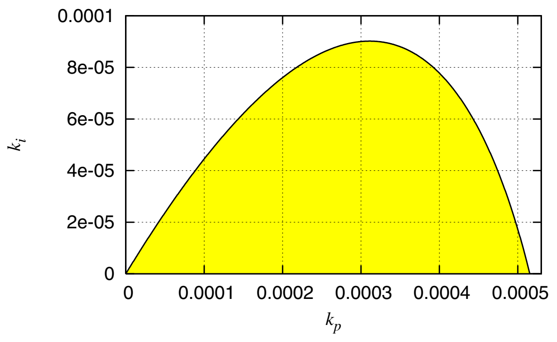

Figure 4. The stabilizing region of (k_p, k_i) for $B = 1523.8559$ and $d = 2.0$ sec.

With these two `ns-2` agents, we can combine the simulation of the dynamics of physical systems and of a communication network.

Network Topology. Our experiments are based on the dumbell topology shown in Figure 5. There, all NCSs share the single bottleneck link that connects the two routers, R1 and R2. Several plants are connected to R1; and their corresponding controllers to R2. The links' bandwidths and propagation delays are shown in the figure. In Figure 5, d is chosen to be a uniformly distributed random variable in the interval $[0, 50]$msec, except for the first plant where d is fixed to zero, and for the last plant where d is fixed to 50. Fixing the delays for the first and for the last plants makes it sufficient to show the results only for these two plants, and it ensures that the results for the remaining plants to fall somewhere in between.

Plants and Controllers. In this paper, we confine our focus to scalar LTI plants and proportional controllers. Each plant's state, $x(t)$, evolves according to the following differential equation:

$$\dot{x}(t) = ax(t) + bu(t),$$

where a and b are constants, and $u(t)$ is the input from the controller. The sensor samples $x(t)$ at discrete time instances and generates $x(t_0)$, $x(t_1)$, $\ldots$, $x(t_j)$. For each received plant sample $x(t_j)$, the controller calculates $u(t_j)$ as follows:

$$u(t_j) = -K(R - x(t_j)),$$

where K is the constant controller gain, and R is the reference point the plant is required to follow.

The authors in [5] proposed a performance measure for scalar LTI NCSs that is a function of the sampling period, h. Substituting $1/r$ in place of h, we obtain the following utility function for plant i:

$$U_i(r_i) = \frac{a_i - b_i K_i}{a_i} e^{a_i / r_i}.$$

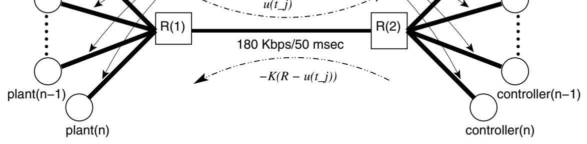

Figure 5. A single bottleneck topology for experimental simulation.

Such a utility function satisfies all required conditions mentioned at the end of Subsection 3.1. Moreover, $r_{\min}$ is derived in [25] for the same family of scalar LTI NCSs, and is given by

$$r_{\min,i} = \frac{a_i}{\ln\left(\frac{b_iK_i+a_i}{b_iK_i-a_i}\right)} .$$

Experimental Setup. We compare the performance of both a P and a PI controller. The experimental parameters are as follows. The number of plants is $N = 50$. All plants have same physical dynamics, $a_i = 0.01$ and $b_i = 1.0$, and all have same corresponding controllers, $K_i = 1.5$. All start and end at the same time, $t_{start} = 0.1$sec, and $t_{end} = 320.0$sec. All packets have a size of 100 bytes. Based on this, the capacity of the bottleneck link in Figure 5 is $180\text{Kbps}/(100 \cdot 8) = 225$ packets/sec. Each plant's transmission rate will converge to the steady-state value of $r_0 = 225/50 = 4.5$ packet/sec. We choose a conservative estimate of the round-trip delays, e.g., 2.0 sec. The k_p–k_i stabilizing region for $B = 1523.8559$ (obtained by calculating $\sum_{i=1}^{50} 1/U_i''(r_{i0})$ where $r_{i0} = 4.5$ for $i = 1,\ldots,50$) and for $d = 2.0$ is defined by Figure 4. For the PI controller, we choose $k_p = 3.0 \cdot 10^{-4}$ and $k_i = 4.0 \cdot 10^{-5}$; and for the P controller, we choose $k_p = 3.0 \cdot 10^{-4}$. Finally, we assume that we want to control the queue around 50 packets.

Results. Figures 6 and 7 show the results of this experiment. When using both the PI and the P controllers, the transmission rates of all plants converge nicely and in short time to the value where all plants share the bottleneck bandwidth equally (Figures 6-A and B, respectively). However, with the P controller, the queue exhibits a steady-state error, and thus the plants, in this case, suffer from long round-trip delays. On the other hand, with the PI controller, this problem is totally absent and the queue stabilizes around the desired set point of 50 packets; see Figures 6-C and D.

Figure 7 depicts the response of NCSs Nos. 1 and 50 when following a square wave input (Figure 7-A). With the PI controller, the NCSs stay stable and track the input signal accurately (Figure 7-B). On the other hand, with the P controller, long delays degrade the performance of the NCSs severely (Figure 7-C). From Figures 6 and 7, it becomes clear that it is not only important to allocate the bandwidth among NCSs, but it is of equal importance to also ensure that the queue length is controlled around a small value.

6. Conclusions

In this paper, we have presented a scheme for bandwidth allocation in networked control systems (NCSs). This scheme targets NCSs working over large distributed networks, i.e., WANs. While ensuring the stability of each NCS, the scheme allocates the bandwidth such that to maximize the aggregate performance of all NCSs. After presenting the optimization formulation, we discussed techniques to improve the performance of the allocation scheme.

7. Acknowledgment

This work was supported in part under NSF CCR-0329910, Department of Commerce TOP 39-60-04003, NASA NNC04AA12A, and an OhioICE Training grant.

References

[1] The network simulator - ns-2. [Online]. Available: http://www.isi.edu/nsnam/ns/.

[2] A. Al-Hammouri, A. Covitch, D. Rosas, M. Kose, W. Newman, and V. Liberatore. Compliant control and software agents for Internet robotics. *In Eighth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS)*, 2003.

[3] L. Almeida, P. Pedreiras, and J. Fonseca. The FTT-CAN protocol: Why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, 2002.

[4] M. Branicky, V. Liberatore, and S. Phillips. Networked control system co-simulation for co-design. In *Proc. American Control Conf.*, 2003.

[5] M. Branicky, S. Phillips, and W. Zhang. Scheduling and feedback co-design for networked control systems. In *Proc. IEEE Conf. on Decision and Control*, 2002.

[6] R. Carter, S. Parthasarathy, A. Pavan, and K. Nelson. Distributed real-time control over large scale networks. In *Large Scale Networking Workshop*, Vienna, VA, March 2001.

[7] G. Franklin, J. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, third edition, 1999.

[8] I. Gravagne, J. Davis, J. DaCunha, and R. Marks. Bandwidth reduction for controller area networks using adaptive sampling. In *International Conference on Robotics and Automation*, 2004.

[9] C. Hollot, V. Misra, D. Towsley, and W. Gong. On designing improved controllers for AQM routers supporting TCP flows. In *IEEE Infocom*, 2001.

[10] G. Kaplan. Ethernet's winning ways. *IEEE Spectrum*, 38:113–115, 2001.

[11] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings ACM SIGCOMM*, 2002.

[12] R. La, P. Ranjan, and E. Abed. Global stability conditions for rate control of discretized model with communication delays. In *Global Telecommunications Conference*, 2004.

[13] D. Lapsley and S. Low. An optimization approach to ABR control. In *In Proceedings of the ICC*, 1998.

[14] V. Liberatore. Integrated play-back, sensing, and networked control. In *IEEE Infocom*, 2006.

[15] V. Liberatore, M. Branicky, S. Phillips, and P. Arora. Networked control systems repository. [Online]. Available: http://home.cwru.edu/ncs/.

[16] S. Low and D. Lapsley. Optimization flow control—I: Basic algorithm and convergence. *IEEE/ACM Transactions on Networking*, 7(6):861–874, 1999.

[17] D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, second edition, 1989.

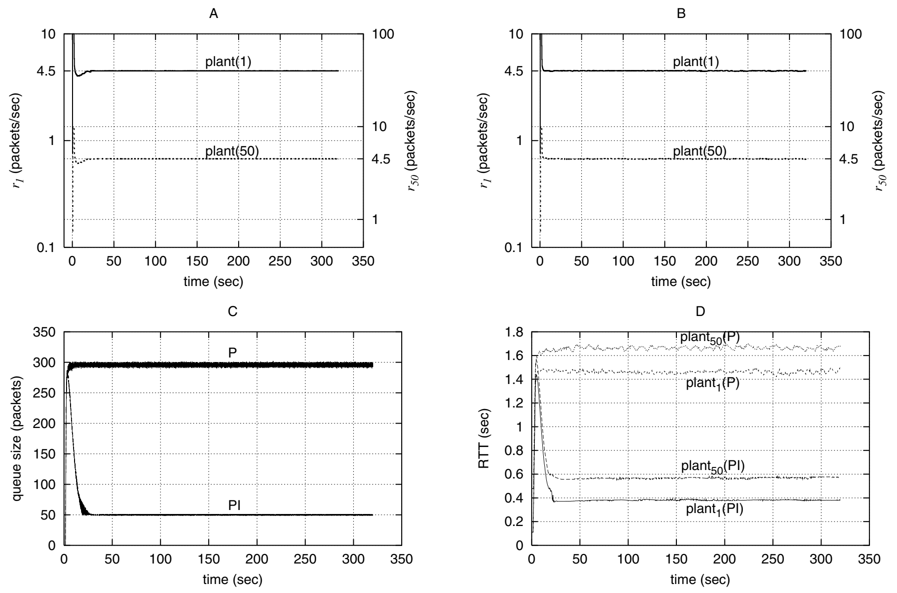

Figure 6. Transmission rates, queue lengths, and round-trip delays when using PI and P controllers.

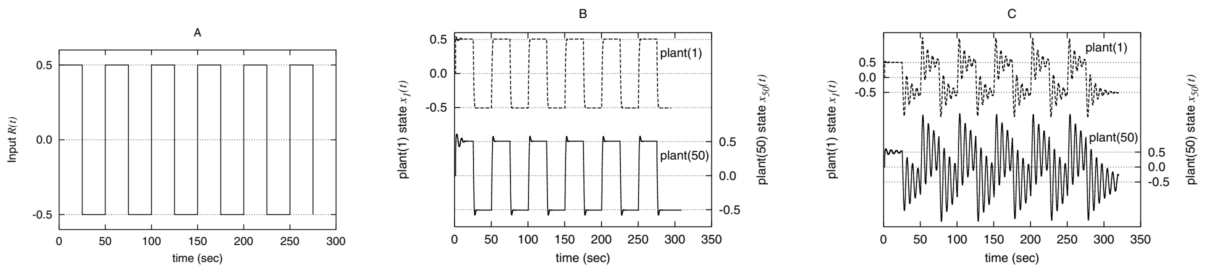

Figure 7. Input signal and output responses when using PI and P controllers.

[18] F. Paganini. Flow control via pricing: a feedback perspective. In *Proceedings of the 2000 Allerton Conference*, 2000.

[19] F. Paganini, J. Doyle, and S. Low. Scalable laws for stable network congestion control. In *IEEE CDC*, 2001.

[20] G. Silva, A. Datta, and S. Bhattacharyya. *PID Controllers for Time-Delay Systems*. Birkhäuser, 2005.

[21] F. Systems. Convex optimization. [Online]. Available: http://www.solver.com/probconvex.htm.

[22] M. Velasco, J. Fuertes, C. Lin, P. Marti, and S. Brandt. A control approach to bandwidth management in networked control systems. In *30th Annual Conference of the IEEE Industrial Electronics Society (IECON04)*, 2004.

[23] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanarama. One more bit is enough. In *Proceedings ACM SIGCOMM*, 2005.

[24] L. Ying, G. Dullerud, and R. Srikant. Global stability of Internet congestion controllers with heterogeneous delays. In *Proceedings of the 2004 American Control Conference*, 2004.

[25] W. Zhang and M. Branicky. Stability of networked control systems with time-varying transmission period. In *Allerton Conf. Communication, Control, and Computing*, 2001.

[26] W. Zhang, M. Branicky, and S. Phillips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21(1):84–99, February 2001.