Implementation Challenges in Real-Time Middleware for Distributed Autonomous Systems

Vincenzo Liberatore
Division of Computer Science
Case Western Reserve University
10900 Euclid Avenue
Cleveland, Ohio 44106-7071, USA
E-mail: vl@case.edu
URL: http://vincenzo.liberatore.org/NetBots/

1 Introduction

Exploration missions will achieve a sustainable human-robotic presence in space and on the surface of the Moon and of Mars. Sustained operations require higher performance, which in turn implies a higher degree of asset autonomy. For example, construction, maintenance, or in-situ resource utilization will be accomplished by efficiently executing complex tasks in the absence of long-haul tele-operation from Earth. Moreover, assets will communicate with each other and form a distributed system. For example, multiple units will collaborate on, say, a construction task. The resulting system is both autonomous (i.e., it can operate in the absence of direct Earth tele-operation) and distributed (i.e., multiple assets communicate and collaborate with each other). Robustness and economy further demand the re-usability of tested middleware whenever possible. Finally, the distributed autonomous systems will include assets that operate in real-time: examples are surface robots, rovers, CEV on-board actuation units, or space construction robots.

In this paper, we describe the transversal challenges in implementing real-time middleware components for distributed autonomous systems. The emphasis is on reusable components that address issues that are shared among distributed autonomous real-time systems. Specific applications are not the focus of this paper.

Real-time middleware has been extensively used in terrestrial systems. For example, Real-Time CORBA (implemented, say, in TAO-ACE [21]) aids in the development of distributed real-time applications. Other middleware paradigms are for example TMO (Time-triggered and Message-triggered Objects [11]) and Agents (as in Jack, used for example in UAV and surface autonomous vehicles, or as in Aglets [12], which we have employed for remote manipulation tasks [1]). Terrestrial real-time middleware has reached high technology maturity and solves for the most part three major issues: (1) a consistent and well-defined API for applications (e.g., a well-defined methodology for RPC or for resource discovery), (2) a way to specify real-time constraints, such as soft or hard real-time deadlines, and (3) an implementation that maps the real-time requirements into process or thread scheduling. However, the current state of the art fails to address network-related real-time issues.

In this paper, we discuss four critical network-centric real-time middleware services and the challenges in their implementation. First, applications specify network QoS in terms of, say, delay, jitter, losses, or bandwidth. The application requirements are then translated into network QoS. However, the translation is often difficult. For example, if the underlying network supports, say, assured forwarding [20], then it is not straightforward for the middleware to map application requirements into network Classes of Service. Furthermore, general network QoS may require substantial coordination for traffic shaping or for resource reservation, depending on the specific QoS model. An alternative is to adopt Local QoS [13], at the cost of reduced end-point control over the network. The second application-independent service is the so-called rate control module, which defines the rate at which communication end-points inject packets into a network. Although rate control is well-understood in the case of bulk data transfers [22] or of certain multimedia applications (audio, video) [7], it is an open problem for distributed autonomous systems consisting of sensing, actuation, and control units. Third, sensor samples and control signals are subject to losses and jitter in the network infrastructure even if QoS is supported. The end-point middleware should recover from losses and jitter via a play-back buffer. Although play-back has been extensively

studied in digital audio, play-backs are an open problem in sense-and-respond systems. Finally, middleware must be hosted on space assets that are constrained in terms of computation, memory, power, or communication. A major problem is to map high-level middleware services to such devices.

2 Background

2.1 Middleware

The terrestrial Internet has grown explosively in the last decade. The Internet success is due in large part to a simple network architecture combined with an open interface that lets developers invent and deploy novel applications and services. The Internet model is radically different from the traditional practice in space networks, where communication software has often been developed specifically for each application or mission. However, the TCP/IP interface addresses fairly low level communication issues. Sockets are similar to low-level disk I/O and force application developers to structure communication around the disk I/O paradigm. On the other hand, a distributed application may benefit from more complex abstractions, such as remote procedure calls, where locally running code will invoke a function to be executed transparently on a remote server. Fortunately, the open-interface Internet architecture makes it possible to implement more complex primitives on top of TCP/IP. The resulting system is *middleware*, a set of libraries that abstract the details of the underlying communication protocols. Middleware implements common functionality once, thereby significantly speeding up the time and cost of developing distributed applications. Furthermore, middleware implements state-of-the-art distributed algorithms while hiding their complexity from application developers. Commercial off-the-shelf middleware includes for example CORBA and Java RMI [23].

Although middleware has been primarily used in the terrestrial setting to support business applications, much research has also been devoted to middleware for real-time distributed embedded systems [21]. In this case, middleware must incorporate time as a first-class citizen because the stability, safety, and performance of applications depends on real-time middleware properties.

2.2 Networked Control

The literature commonly denotes the physical system to be controlled as *plant*. A controller acquires data from the *sensors* that monitor the plant. The controller uses the sensor readings to issues commands to *actuators* so as to change appropriately the plant state. The textbook example is the regulation of ambient temperature. A thermostat (the controller) reads the temperature in the ambient (the plant) from a thermometer (the sensor) and modulates the heating (the actuator) so as to bring the room to the desired temperature.

In general, the plant state depends on the actuator operations and also on exogenous disturbances. Furthermore, the sensors are subject to measurement errors. A *networked control system (NCS)* involves a plant that is controlled remotely, and networked control is complicated by delays, jitter, and packet losses. In the first place, network latency can delay the reception of sensor data at the controller site and the application of a control signal at the plant site. Furthermore, packets can be lost either because they are dropped out in the network infrastructure or because they are discarded by the communication end-points, for example, if they arrive late.

3 Network Quality-of-Service

Real-time middleware must satisfy Quality-of-Service (QoS) requirements. The QoS requirements can be classified depending on whether they focus on network parameters (latency, jitter, loss rates, end-to-end bandwidth), distributed systems concerns (predictability, scalability, dependability, and security) or programming languages and interfaces (programming abstractions for QoS). QoS requirements are translated into QoS provisioning by the middleware to meet the application requirements [8].

Unfortunately, QoS provisioning is made difficult by two fundamental architectural principles of the terrestrial Internet [5]. A first principle is autonomy and decentralized control. A decentralized architecture is responsible in part for preventing the adoption of QoS mechanisms that rely on the concurrent administration of multiple nodes or on the concerted efforts of multiple providers. Conversely, space exploration networks fall within the administrative control of a handful of government agencies, and make it relatively easier to coordinate administrative responsibilities. A second principle is best-effort service, which implies an unreliable service with no provisioning for QoS. A best-effort architecture can employ over-provisioning to guarantees a statistical form of QoS because an over-provisioned network is characterized by bandwidth larger than the arrival rates, so that queuing delays and loss rates are small. Moreover, over-provisioning also leads to redundancy and fault-tolerance. On the other hand, best-effort networks do not apply policing or isolation among flows and, as a result, QoS can suffer. Specifically, over-provisioned best-effort networks work well in the typical case, but do not provide guarantees and expose flows to the risk of poor QoS. Furthermore, over-provisioning is more difficult to use in space networks due to the exorbitant cost of deploying space assets.

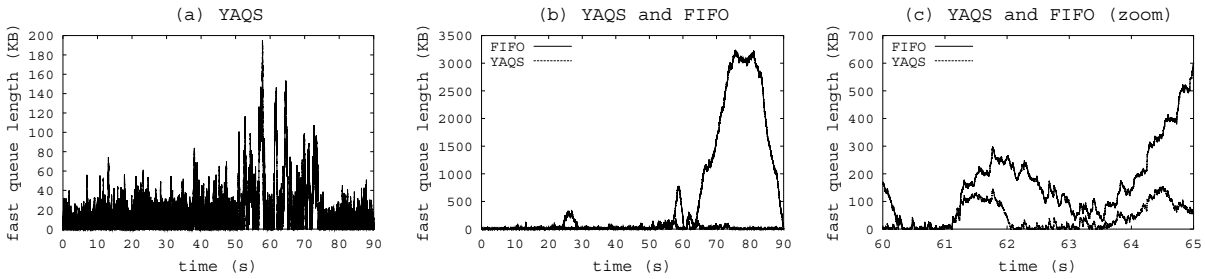

Figure 1. (Fast) queue length (trace 6). Chart (a) gives YAQS fast queue length, chart (b) compares YAQS and FIFO, and chart (c) zooms on the interval $[60, 70]$, where the FIFO queue starts its highest ascent. The vertical axis scale differs among charts [13].

On the whole, terrestrial networks have followed the two fundamental Internet design principles of decentralized control and best-effort, and mostly guarantee QoS through over-provisioning. However, space networks are not subject to the same types of constraints. First, space exploration networks have a more centralized administrative structure. Second, over-provisioning is not a sustainable solution. As a result, space networks are a better fit for QoS provisioning, including selected mechanisms to achieve scheduling, isolation, and policing. For example, a surface network could establish a priority scheme to differentiate among data, video, audio, and control flows.

Although planetary networks can support QoS mechanisms, a planetary network is also an autonomous system that must be ready to operate without direct Earth supervision. In particular, a distributed autonomous system should be able to arbitrate among competing QoS demands even without human supervision. For example, suppose that a certain flow has low priority until an unexpected event renders it critical to the mission. The distributed autonomous system should adapt to the change and adapt QoS levels accordingly. By contrast, the flow cannot suffer poor QoS until a remote Earth administrator realizes the danger and rearranges priorities. In other words, QoS administration must also be autonomous and distributed.

An approach to autonomous and distributed QoS is *Local QoS*, a set of fully distributed mechanisms to protect flows and ensure QoS. Local QoS is typically an addition (or even a replacement) to explicit and coordinated QoS provisioning.

In previous work, we have addressed local QoS with a scheduling algorithm that maintains short queues in best-effort networks that are generally over-provisioned but occasionally congested [13]. The algorithm requires no support for distributed coordination. The YAQS (Yet Another Queuing Strategy) mechanism attempts to obtain short queuing delays in a best-effort router that is crossed by aggressive flows. The algorithm involves the following two elements. First, per-flow queue occupancy is maintained with high accuracy and limited state. Second, when the queue occupancy of a certain flow exceeds a threshold value, subsequent packets belonging to that flow are segregated in a different queue. The forwarding decision is then made by an approximately fair schedule between the regular and the segregated queue. The intuition is that flows with small buffer occupancy should benefit from a shorter queue once the more aggressive flows have been separated.

Figure 1 gives the (fast) queue length of YAQS and FIFO on a Internet trace. YAQS is a clear improvement over FIFO in terms of fast queue length. In fact, the YAQS line is but noise close to the horizontal axis when compared to FIFO. The comparison is better seen by zooming in, as in Figure 1(c).

Simulations afford us the opportunity to compare YAQS with an Active Queue Management mechanism such as RED [6]. Figure 2 gives the (fast) queue length of YAQS, FIFO, and RED during the activity of an aggressive flow. YAQS is a clear improvement over FIFO and RED in terms of fast queue length. In fact, the YAQS line looks like noise when compared to FIFO or RED. RED improves over Drop-Tail, and it prevents long queues after an initial transient where it behaves like FIFO (i.e., until time 1.9s). However, RED did lead to long queues initially due to its slower dynamics. The aggressive flow goodput is 24Mbps under YAQS, 29Mbps under FIFO, and 17Mbps under RED. In summary, YAQS was the most effective method to maintain short queues, and simultaneously the aggressive flow achieved higher goodput than under RED.

Future work should develop the ideas of local QoS. For example, YAQS should be compared with Stochastic Fairness Queuing (SFQ) [17].

A complementary approach is to hide latency through overlay QoS. In an overlay network, end-points are organized at the application layer into a virtual connectivity net-

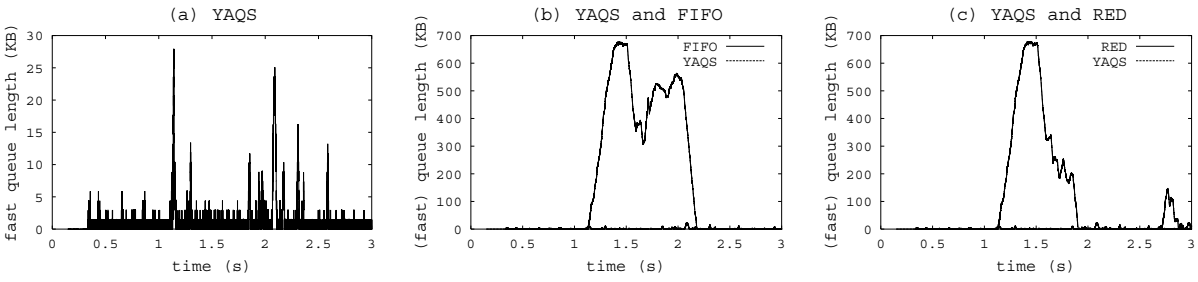

Figure 2. (Fast) queue length in simulations. Chart (a) gives YAQS fast queue length, chart (b) compares YAQS and FIFO, and chart (c) compares YAQS with RED. The vertical axis scale differs between charts [13].

work on top of the underlying IP connectivity. Overlay QoS dynamically manages an overlay network to guarantee higher levels of service to applications. For example, a reliable overlay can reduce the periods of network partition [4] that we found to be a significant problem for networked control performance. We expect that overlays and buffers will act on different time scales, and thus both should be investigated and used.

A related issue is the the use of multihoming, as in the real-time version of SCTP. Multihoming allows hosts to be reached from different end-to-end paths, thus improving the reliability of the distributed applications. In a real-time distributed application, signals would be replicated via forward error correction over multiple paths, and would increase the application's performance and continued operations [18]. Future research should also involve a comparison of SCTP multihoming with resilient overlays.

4 Rate Control

A second issue is the rate at which embedded devices exchange data. The issue of bandwidth management has gained considerable research attention in NC, see for example [25, 3] and the references contained therein. Most research has 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), such as a full planetary network. Previous work usually requires time synchronization among the different devices in the network. The allocation schemes are either static or dynamic. Static schemes determine allocation before running and consequently lack flexibility and adaptability to dynamic changes. On the other hand, dynamic schemes often required centralized implementations.

In our earlier work, we have developed a dynamic, decentralized bandwidth allocation scheme for networked control systems that have their control loops closed over WANs [2]. The algorithm is depends on a formulation of the bandwidth allocation problem as a convex optimization program that can be solved in a fully distributed manner.

The idea behind our approach is to have the control systems vary their sampling periods (thus their bandwidth consumption) based on the congestion level fed back from the network, and to preserve high performance level. Our scheme has the following features:

- It allocates the bandwidth 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, asynchronous, and scalable solution. Each node executes an independent algorithm using local information with no central managing entity. The approach scales up with the number of controlled systems and the size of the network.
- It is dynamic and flexible. It dynamically reallocates bandwidth as different control systems acquire and release the network.

A 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 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, $U_i(r)$. The utility value $U_i(r)$ expresses the degree to which a particular system i 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.

The rate control objective can be formulated as follows: determine a transmission rate of each NCS so as to maximize the sum of utilities subject to (i) each system stability constraint, and (ii) each link's capacity constraint:

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

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). The optimization problem (1) can be decomposed into separable sub-problems [2]. The solution was 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 the basic optimization problem that incorporates the Lagrange multipliers for link capacity constraints. The innovation involves the combination of an optimization congestion control approach with a PI AQM method in the case of networked control, and the exact analysis of the PI's stability region.

A basic issue is the derivation of utility functions that are relevant to networked control. A possible approach is to derive a per-plant utility function depending on how close the plant is approaching the instability region at the chosen rate, and using these utility functions in our framework. Future work should explore an innovative way to measure plant instability that is based on the time-scales approach for solving differential equations [9]. The advantage of time-scales is that the utility functions can change over time depending on the "degree of instability" currently exhibited by a plant.

In the development of congestion control algorithms, it is important to consider the effects of slow convergence of a congestion control algorithm, which can have substantial impact on the performance of connected plants. In countering the slow convergence rates, it is also important to not to vary the sampling rates of the NCSs as this may have detrimental effects on system performance. A promising approach would be based on equation-based congestion control [7] or on TCP-friendly SIMD congestion control [10].

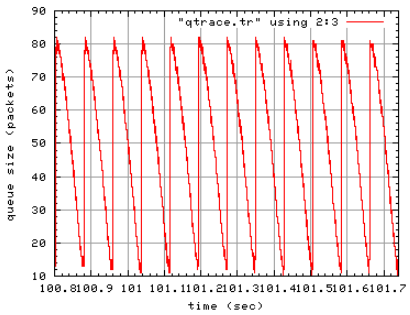

Figure 4. Queue oscillations.

Sense-and-respond congestion control differs from TCP congestion control in that rate regulation must be based on relatively fewer packets. On the other hand, the number of packets corresponds to the speed at which an individual plant receives information about the queue. As a result, the queue cannot be controlled at frequencies that are comparable or higher than the packet frequency. In turn, the paucity of packets causes visible queue oscillation and flow synchronization, as in the simulations in Figure 4. The oscillations are a problem in congestion control but are unremarkable in TCP congestion control even though the two methods are based on the same original techniques because TCP's packet rate is much higher and tends to mask the oscillations with noise. Future work should investigate methods to suppress this oscillations, and our on-going work suggests that a particularly effective technique would be to introduce a non-linear element prior to the PI AQM.

5 Play-Back

A play-back method smooths out delays, jitter, and packet losses by applying control signals to the environment during predictable time intervals [20]. Figure 3 exemplifies a possible signal exchange. In this example, the plant sends samples to the controller at regular intervals of length T. A sample was dropped out after t_3 but the plant kept sampling every T seconds and, in this example, the next signal was successfully delivered. The unbuffered control is commonly adopted in related work [15, 19] and applies received control signals from the time when they are received until the reception of a new signal. The unbuffered mechanism suffers the problem that no signal is applied during a predictable time interval. A traditional play-back buffer holds the signal u_1, u_2, and u_4 that are received before their play-back time and applies the corresponding control signal only

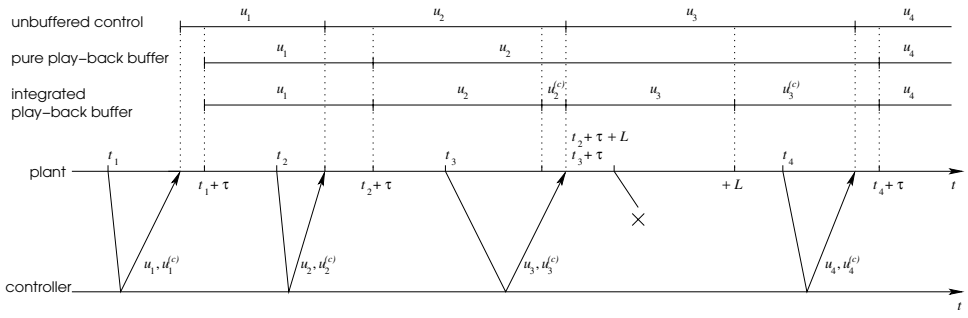

Figure 3. The timeline of network events in the remote control of a plant. In this example, τ_i and $L_i = T$ are constants [14].

at the planned instant. The buffer discards u_3 because it is late and applies u_2 until otherwise instructed by a new control signal. Traditional buffers suffer from the long application interval of signals such as u_2.

In our earlier work, we have developed a play-back algorithm for tolerant and adaptive networked control [14]. Figure 3 shows one of the main improvements over traditional buffers. The improved algorithm times out and switches to $u_2^{(c)}$ when u_3 is late and to $u_3^{(c)}$ when a sample is lost and no control signal is forthcoming.

Our playback algorithm employs a reverse play-back buffer and the estimation of play-back times, an expiration time to curb an aggressive controller, a contingency control to deal with loss episodes, and an observer that explicitly accounts for network vagaries. The intuition is that the control signals should be applied during a predictable interval. If the control is applied too early, the plant will not have reached yet the state for which the control was meant. Symmetrically, a control signal u_i can negatively affect a plant if it is applied continuously well after the sampling time t_i because the plant state would in general have changed significantly in the meanwhile.

In our algorithm, the controller sends to the plant *two* control signals, which will be called the *regular control* (denoted by u_i) and the *contingency control* (denoted by $u_i^{(c)}$). Additionally, the controller sends also a *play-back delay* τ_i and the *duration* L_i of the regular control. The play-back time τ_i is used to implement a relaxed play-back buffer: the regular control is applied at time $t_i + \tau_i$, or immediately if the play-back time has already passed. Thus, the play-back delay only fixes the earliest time at which a control can be applied. After a duration L_i since the time when the control was applied, if a new control has not been received, the plant switches to the contingency action $u_i^{(c)}$. The duration L_i and the contingency control $u_i^{(c)}$ prevent the plant from applying the regular control for an unpredictably long period of time, thereby damaging the system. Therefore, the contingency control should be a low performance but presumably safe action. Additionally, the controller sends the plant a *sampling period* T_i, which denotes the time interval until the plant collects the next sample from the sensor and sends it to the controller. A predictable sampling schedule T_i effectively establishes time-out protection against losses. The play-back algorithm is integrated with sampling (through the dynamic setting of sampling time) and control (through expiration times and performance metrics).

The sampling rate $1/T$ normalized by the packet size is the instantaneous bandwidth used by networked control. Since T is related to bandwidth and τ to communication delays, T and τ are largely independent of each other. In particular, nothing prevents $T \ll \tau$, in which case a sequence of samples and control signals are typically in flight in the network. Such a scenario is called a *control pipe*. Notions similar to pipes were also discussed in [16, 24]. Although the pipe is, in some sense, stored in the network, the plant does not keep explicit state for the signals in the pipe. The main advantage of using control pipes comes from the fact that shorter sampling periods tend to improve control performance. A second advantage of pipes is that it enables a controller to quickly countermand a previous control signal, in case of transient delay spikes.

The algorithm was extensively simulated and emulated. The evaluations demonstrated that the algorithm produced low variability around the set point and it was robust to varying levels of connectivity. Extensive simulations showed that the combined approach was able to remove 75% of the additional (non-inherent) plant variability. Furthermore, the play-back control performed roughly as well as any proportional controller on an ideal network with pure delays. The algorithm was robust to parameter choice and its performance gracefully degraded when network connectivity worsened (Figure 5). The algorithm addressed several problematic issues in networked control systems. A control signal u_i is applied during a time interval whose bounds are defined by the play-back delay and by the expiration time. The

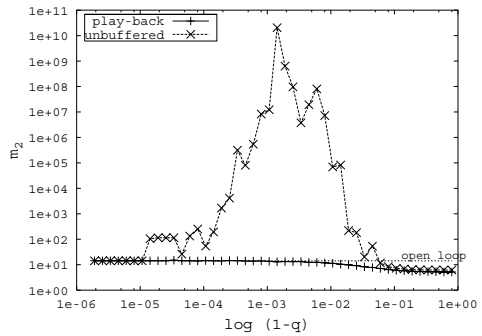

Figure 5. Root-mean-square m_2 as a function of the probability q of remaining in the lossy state [14].

expiration time was especially effective in the case of losses in that it prevented the continued application of an old control that was appropriate at the time it was originally applied but that later on would take the plant output away from its reference value. Symmetrically, the play-back buffer prevented the application of a signal when it was not appropriate yet for the current plant state. The introduction of a playback delay was very effective in mitigating the effects of jitter, at the price of a minimal additional average delay.

A major open question is the integration of play-back buffers with congestion control. In the first place, the playback algorithm also fixes the sampling period of T as a balance of two competing objective: a smaller T leads to better plant performance, but a smaller T also increases the number of control pipe imperfections that are caused by jitter. The algorithm dynamically attempts to decrease T until T reaches a value so small that it would start causing control pipe imperfections. An integrated algorithm can take one of following two approaches. First, T could be generated by the rate control algorithm, but a final check is made to ensure that T is higher than a jitter-aware lower bound. The second approach is to remove the lower bound altogether by using more aggressive buffers at the plant site. In principle, if the plant had an infinite buffer, the effect of jitter would vanish. However, larger buffers also pose significant questions: its memory consumption is heavier, and more significantly, the prediction of the future plant state becomes extremely complicated. As a consequence, the observer's prediction could be far less accurate. In summary, it is not clear whether the better solution is to keep the pipe simple while restricting the smallest value of the sampling period T.

The play-back algorithm often generates control pipes in which multiple samples and control signals are outstanding. Meanwhile, the plant uses timers to clock the sample generation process. Control pipes present an opportunity for congestion control to be based on a congestion window, thus becoming self-clocked and dispensing of timers. Future work should include an investigation of congestion control based on a congestion window. Finally, the communication flow is driven by the plant timers, and alternative sampling schemes should be investigated.

6 Middleware Implementation

Terrestrial middleware is modular and flexible, but often demanding in terms of computational resources or bandwidth. A major open question is to map the clean designs of distributed real-time embedded systems on constrained resources, such as those that are going to be deployed on space assets.

A number of pragmatic issues must also be addressed before the algorithms can be integrated into a working system. In the first place, samples and controls should be sent over RTP. Although we are confident that the RTP protocol is appropriate for real-time flows, we had problems with several of its implementations. Future work should undertake an investigation of existing RTP libraries and implement modifications as needed. We expect that the modifications will be confined to the implementations and that no changes would be needed in the RTP/RTCP protocols. Similarly, our preliminary investigation suggests that SCTP implementations may suffer from problems similar to RTP implementations.

Acknowledgments

Work supported in part under NSF CCR-0329910, Department of Commerce TOP 39-60-04003, and NASA NNC04AA12A. We would like to thank Ahmad Al-Hammouri for Figure 4.

References

[1] A. Al-Hammouri, A. Covitch, D. Rosas, M. Kose, W. S. Newman, and V. Liberatore. Compliant control and software agents for Internet robotics. In *WORDS*, 2003.

[2] Ahmad Al-Hammouri and Vincenzo Liberatore. Decentralized and dynamic bandwidth allocation in networked control systems. In *14th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, 2006.

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

[4] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. *Operating Systems Review*, 35(5):131–145, 2001.

[5] V. Cerf and R. Kahn. A protocol for packet network interconnection. *IEEE Transactions on Communications Technology*, COM-22(5):627–641.

[6] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. Tuning RED for Web traffic. In *Proc. Sigcomm*, pages 139–150, 2000.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. *Computer Communication Review*, 30(4):43–56, 2000.

[8] Christopher D. Gill, Jeanna M. Gossett, Joseph P. Loyall, Douglas C. Schmidt, David Corman, Richard E. Schantz, and Michael Atighetchi. Integrated adaptive QoS management in middleware: An empirical case study. *Real-Time Systems*, 29(2–3):101–130, March 2005.

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

[10] Shudong Jin, Liang Guo, Ibrahim Matta, and Azer Bestavros. A spectrum of TCP-friendly window-based congestion control algorithms. *IEEE/ACM Transactions on Networking*, 11(3), June 2003.

[11] K.H. (Kane) Kim. APIs for real-time distributed object programming. *Computer*, 33(6):72–80, June 2000.

[12] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[13] Vincenzo Liberatore. Local flow separation. In *The Twelfth IEEE International Workshop on Quality of Service (IWQoS)*, pages 87–95, 2004.

[14] Vincenzo Liberatore. A play-back algorithm for networked control. In *Proceedings of the Twentififth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2006)*, 2006.

[15] Vincenzo Liberatore et al. Networked Control Systems Repository. `http://home.cwru.edu/ncs/`.

[16] B. Lincoln and B. Bernhardsson. Optimal control over networks with long random delays. In *Proc. International Symposium on Mathematical Theory of Networks and Systems*, 2000.

[17] P. E. McKenney. Stochastic fairness queueing. In *INFOCOM 1990*, 1990.

[18] T. Nguyen and A. Zakhor. Path diversity with forward error correction (PDF) system for packet switched networks. In *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM)*, pages 663–672, 2003.

[19] J. Nilsson. *Real-Time Control Systems with Delays*. PhD thesis, Lund Institute of Technology, 1998.

[20] Larry L. Peterson and Bruce S. Davie. *Computer Networks*. Morgan Kaufmann, 2000.

[21] D. Schmidt, A. Gokhale, T. Harrison, D. Levine, and C. Cleeland. TAO: A high-performance endsystem architecture for real-time CORBA. *IEEE Communications Magazine*, February 1997.

[22] W. Richard Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley.

[23] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems. Principles and Paradigms*. Prentice-Hall, Upper Saddle River, NJ, 2002.

[24] A. Tzes, G. Nikolakopoulos, and I. Koutroulis. Development and experimental verification of a mobile client-centric networked controlled system. To appear.

[25] Manel Velasco, Josep M. Fuertes, Caixue Lin, Pau Marti, and Scott Brandt. A control approach to bandwidth management in networked control systems. In *30th Annual Conference of the IEEE Industrial Electronics Society (IECON04)*, 2004.