Time in general-purpose control systems: The Control Time Protocol and an experimental evaluation

Scott Graham and P. R. Kumar

***Abstract*— Since no two clocks generally agree, distributed clocks are inherently unreliable. Distributed control applications however require accurate timing information, to be captured through time stamps throughout the system. This necessitates some form of message passing for interpretation of distributed clocks. We exhibit a fundamental indeterminacy in estimating time when delays are unequal in the two directions, necessitating an assumption of symmetry. The paper then outlines architectural implications of time stamping algorithms and protocols. It presents the Control Time Protocol as an architecturally clean algorithm for providing timing information in the domain of distributed control. The protocol introduces no additional dependencies beyond those already present in the very control loops, thus enhancing the reliability of the systems. It has been implemented on the testbed in the Convergence Laboratory at the University of Illinois. Experimental results demonstrating how time stamping can be used to enhance stability and performance, as well as measurements of network latencies, are provided. The latter may be of independent interest to researchers in control, given the increasing interest in control over networks. We also provide experimental results on estimating plant delay itself using a knowledge of gain, a method which implements CTP.**

I. Introduction

In distributed systems, total ordering of events is important; see Lamport [1], for example, which shows how to causally order events. In control systems, however, it is not just ordering but also the time at which an event took place that is important. The closed loop systems stability and performance can both depend on a knowledge of elapsed time. As we move towards systems where control is exercised over networks of computational devices, it thus becomes important to associate accurate time with information, as for example, sensor measurements.

In digital control, samples are periodically taken, and commands are periodically issued to actuators. In hard real-time control networks, such as the Controller Area Network (CAN), the network protocols are specifically designed such that this clean and predictable periodic behavior can be maintained [2]. However, such predictability is not available with general-purpose communication networks. Our goal in this paper is to investigate the issues arising from distributed time in general-purpose control systems, featuring the convergence of control with communication and computation, which may constitute the next phase of the information technology revolution.

Our contributions are as follows. We show that it is fundamentally impossible to estimate time precisely in a distributed system, unless one makes further assumptions on the symmetricity of delays. Then we discuss the architectural implications of time stamping, particularly with regard to the impact on system reliability, which is increasingly emerging as the most important performance measure. Next we present the Control Time Protocol which introduces no additional dependencies beyond those already present due to the very nature of the control loop, thus enhancing reliability. We present experimental results showing how knowledge of delays in information flows, made possible through the use of CTP, can actually be used to enhance stability and performance, and provide the overall system greater immunity to delays. We also briefly note how time, as a middleware service in general-purpose control systems, can provide additional capabilities to the middleware itself such as migrating computations to balance load over the network, while maintaining and monitoring system delays, so that loop stability is preserved. We provide experimental results on delay histograms in actual networks. These statistics may be of independent interest to researchers in control, since there is much current interest in the topic of exercising control over networks, and it may be useful to know the nature of mean values, thick tailed outliers, etc. Finally, while CTP can measure delay over the controller part of the network where we can exercise the facility of time stamping packets, it cannot estimate plant delay. To complement CTP, we present experimental results on a method to estimate plant delay from knowledge of gain. All the experiments are conducted on a testbed in the Convergence Laboratory at the University of Illinois, described in I-D.

This paper is organized as follows. In the remainder of this section we present an overview of our testbed and the context of general-purpose control systems that is of interest to us. Section II provides an introduction to clocks and clock models, and the inherently unreliable nature of distributed clocks. Section III provides background discussion on measuring clock offsets, and presents a fundamental

This material is based upon work partially supported by DARPA/AFOSR under Contract No. F49620-02-1-0325, AFOSR under Contract No. F49620-02-1-0217, USARO under Contract Nos. DAAD19-00-1-0466 and DAAD19-01010-465, and NSF under Contract Nos. NSF ANI 02-21357 and CCR-0325716, and DARPA under Contact Nos. N00014-0-1-1-0576 and F33615-0-1-C-1905. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views, official policy, or position of the above agencies, the United States Air Force, Department of Defense, or the U.S. Government.

Major Graham is studying under U.S. Air Force sponsorship through the AFIT/CI program in the Department of ECE, University of Illinois, 1308 West Main Street, Urbana, IL 61801, USA. srgraham@uiuc.edu

P.R. Kumar, CSL and Department of ECE, University of Illinois, 1308 West Main St.,Urbana, IL 61801, USA. prkumar@uiuc.edu

result in indeterminacy. Section IV provides an overview of the Network Time Protocol and the architectural differences between synchronization and time translation. Section V experimentally shows how knowledge of delay can be usefully exploited. Section VI then presents the Control Time Protocol and its capabilities, some associated algorithms, and also presents the results of an experimental study of packet delays in networks. Section VII concludes the paper.

A. General-purpose control systems

Katevenis [3] refers to general-purpose computer systems as "systems not biased towards the execution of a particular algorithm, and specifically, systems that execute a mix of word processing, database applications, mail and communications, compilations, CAD, control, and numerical applications."

We similarly define general-purpose control systems as systems with nodes featuring sensing, actuation, computation, and wireless or wireline communication, that are not biased towards a particular control application, but designed to support many applications such as home automation and control, manufacturing, transportation, robotics, etc.

General-purpose computing has proliferated because it has many uses. Proliferation drives down cost, which enables innovation, and encouraging further proliferation. We anticipate that general-purpose control systems will follow a similar trend. One attribute of general-purpose systems is that they use commonly available computer equipment, networks, components and services. Our goal is to endow these with middleware services which makes them suitable for use in control in a routine manner.

A specific issue of importance for control applications is that general-purpose communication networks exhibit random latency, i.e., delay and jitter. Thus we investigate the issue of providing timing as a specific middleware service in such systems. Knowledge of delay can be helpful, i.e., it can be used to stabilize the system and improve performance.

B. Network Time Protocol

The Network Time Protocol (NTP) [9] is a clock synchronization algorithm to keep distributed clocks operating over networks closely aligned. In client-server mode, an NTP client requests the time from an NTP server periodically and adjusts its own clock accordingly. A client may request time from multiple servers and filter responses to protect against failures or malicious behavior. In peer mode, several clocks communicate their view of the current time, establish a consensus as to the correct virtual time, after which each clock endeavors to adhere to this virtual time. In practice, they may simply designate one clock as currently the most accurate, with the designation changing over time.

C. Control Time Protocol

Synchronization has two drawbacks in large control systems. One entails a loss of reliability, and the second, particular to control, involves a possible loss of stability, also an attribute of reliability, though a domain specific one.

Synchronization imposes administrative constraints on the participating nodes. Each node must be running the synchronization protocol and they must all use the same time servers. This results in dependency relations, which need to be avoided for reliability of large systems. A failure of a time server results in a single point of failure to which the entire system is vulnerable. Control loops need to be made autonomous to the extent possible, without making loops themselves dependent on other loops that were created for timing synchronization.

Synchronization controls the clock for a particular node, not for a particular process. The synchronization process is therefore necessarily separate from the processes involved in the control system. Yet the control application critically depends upon correction operation of the synchronization process. This dependency relationship adversely affects reliability by causing many single points of failure.

Furthermore, synchronization may not be possible in an administratively diverse control system. In a city traffic scenario, for example, the city might own and operate the sensors, but the controllers and actuators are in privately owned cars.

Also, synchronization algorithms such as NTP can cause abrupt clock resets. This can lead to instability of the control law.

Thus, we propose an alternative approach to providing a robust timing service which satisfies all the requirements for control, while imposing no administrative constraints and reducing dependencies to a minimum. We call this the Control Time Protocol.

D. Convergence Laboratory

The study of distributed time is part of a larger effort investigating the convergence of control with communication and computation in the Convergence Laboratory at the University of Illinois [6]. We have developed a general-purpose control testbed consisting of remotely controlled cars driving on a small indoor track; see Figure 1. Cameras mounted in the ceiling provide images to vision servers. There are dedicated computers to identify the position and orientation of the small radio-controlled cars on the track below. The data is sent through a wireless ad-hoc network to individual laptops which compute speed and steering commands for respective cars. The controller employs state estimation techniques and a model predictive control law. Each car is driven by a separate laptop.

II. Clocks

The fundamental issue concerning time is that a clock is not perfectly consistent in its tick rate. Environmental conditions such as temperature, pressure, or even humidity, may affect the behavior of the oscillator, causing it to speed up or slow down slightly. The displayed times of clocks will drift with respect to one another.

The ratio of tick rates will be called *skew*, and the difference in displayed time will be called *offset*, which changes over time.

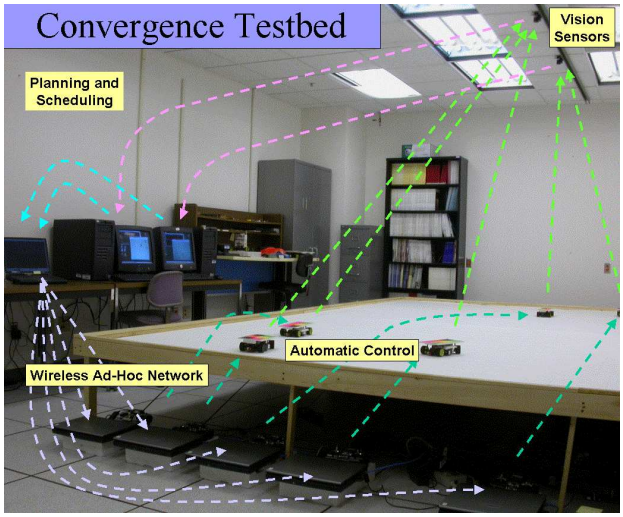

Fig. 1. Testbed in the Convergence Laboratory Testbed at the University of Illinois

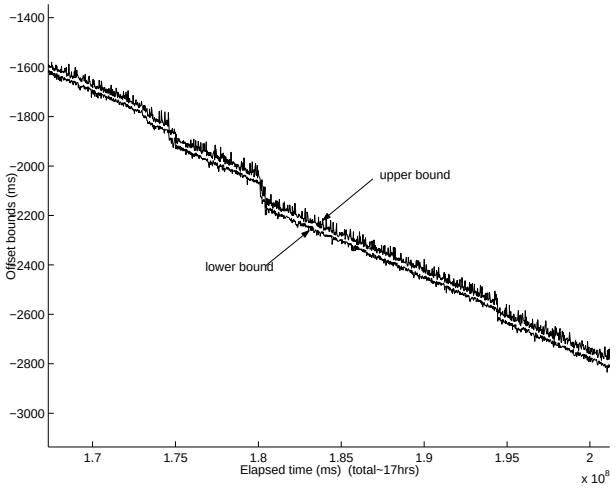

Fig. 2. Offset between two clocks changing over time

To determine the *offset* at some point in time, we require communication between the clocks in the form of time stamped messages. If the communication medium itself had no delay, we could instantaneously know the time of one clock at the other. (This is also true when the delay is known and fixed.) In communication networks, however, delay is not entirely predictable, and precision is therefore inherently limited by the extent of the delay uncertainty.

A. Clock models

If we assume that clocks are linear, i.e., *skew* α is constant,

$$t_{\text{remote}}(t) \quad = \quad \alpha t_{\text{local}}(t) + \textit{offset}(0), \qquad (1)$$

where $t_{\text{local}}(t)$ is the time as measured at a reference clock, and t_{remote} is the time at the other clock. This simple clock model is a useful starting point. If *skew* is time-varying, the model itself varies over time. Then, a more appropriate model is

$$t_{\text{remote}}(t) \quad = \quad t_{\text{local}}(t) + \textit{offset}(t), \qquad (2)$$

indicating the changing offset over time.

To illustrate, we plot the results of an experiment to show the relative offset between two clocks over time. To create these plots, one clock sends a message to a remote clock, which replies. Both clocks attach time stamps to the message. From causality, we know that each message was received after it was sent, so the time stamps provide upper and lower bounds on the current offset of the two clocks. The difference between the sent and reply time stamps is the upper bound, while the lower bound is the difference between the reply and returned time stamps; see Figure 2.

The important point is that although clocks are close to linear, they can vary slightly and unpredictably. In some cases, the variation can be very large and abrupt, for instance, when a user changes the time directly.

III. Estimating offset

As noted in the previous section, from causality, we can bound the current *offset* above and below, where the difference in the bounds is the round trip time of the message. We now investigate a fundamental limit on how precisely one can estimate the offset within these bounds. We will see that, in fact, this depends upon additional information, such as delay symmetry, or knowledge of minimum delays. Without some additional information, it is impossible to determine the current *offset* precisely, as we will now show.

Suppose, as in Equation 1, that we have two perfectly linear clocks. Assuming fixed noiseless communication delays, we wish to characterize the *skew* (α) and *offset*, where for brevity we denote *offset*(0) by *offset*.

We collect data in a series of short pings. These are messages which are time stamped immediately before being sent, and immediately time stamped when received. We will number these messages in order. Thus message one is sent from the local clock to the remote clock, message two is the reply to message one, including the time stamps from the first message. Messages three and four form a similar pair and so on, as shown in Figure 3.

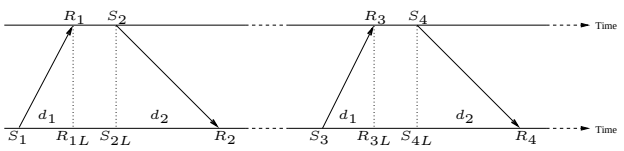

Fig. 3. Message exchange for *offset* determination.

A. *Indeterminate* offset *in the presence of asymmetric delay*

First we show that the estimation problem in general has no solution, unless we make further assumptions detailed below.

Theorem 1: Even with perfect clocks and fixed noiseless delay, it is impossible to uniquely determine the *offset*

between two clocks when unknown delays d_1 and d_2 in the two directions are asymmetrical, i.e., unequal.

Proof: Consider a set of time stamped messages such as in Figure 3. The time stamps S_1, R_2, S_3, R_4, etc., are in terms of the local clock, while R_1, S_2, R_3, S_4, etc., are in terms of the remote clock.

We will use the local clock as a reference; hence we use the notation R_{1L}, S_{2L}, R_{3L}, S_{4L} for the virtual local reference times for the remote times R_1, S_2, R_3, S_4, respectively. We assume fixed but asymmetric delay, so

$$\begin{aligned} R_{1L} &= S_1 + d_1, \\ R_{3L} &= S_3 + d_1, \\ &\dots \\ R_2 &= S_{2L} + d_2, \\ R_4 &= S_{4L} + d_2, \\ &\dots \end{aligned}$$

and so on. Note that both delays are measured in terms of the reference clock. Translating the local times into remote times gives

$$\begin{aligned} R_1 &= \alpha R_{1L} + \textit{offset} = \alpha S_1 + \alpha d_1 + \textit{offset}, \\ R_3 &= \alpha R_{3L} + \textit{offset} = \alpha S_3 + \alpha d_1 + \textit{offset}, \\ &\dots \\ S_2 &= \alpha S_{2L} + \textit{offset} = \alpha R_2 - \alpha d_2 + \textit{offset}, \\ S_4 &= \alpha S_{4L} + \textit{offset} = \alpha R_4 - \alpha d_2 + \textit{offset}, \\ &\dots \end{aligned}$$

and so on. These relationships can be captured in matrix form as follows:

$$\begin{bmatrix} R_1 \\ S_2 \\ R_3 \\ S_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} S_1 & 1 & 0 & 1 \\ R_2 & 0 & -1 & 1 \\ S_3 & 1 & 0 & 1 \\ R_4 & 0 & -1 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha d_1 \\ \alpha d_2 \\ \textit{offset} \end{bmatrix}.$$

The leftmost vector and middle matrix each contain known information, assuming the time stamps of all transactions are pooled between the nodes. This could be done, for example, by exchanging them in later packets. Thus, this system of equations represents all the information available to be used to solve for the four values: α, αd_1, αd_2, and *offset*. However, the middle matrix has rank 3, since the fourth column is the difference between the second and third columns, regardless of how many additional messages are exchanged. ■

Since it is impossible to determine *offset* under asymmetric delays we need to make some further assumption to proceed. Because it yields the minimum error, we will assume the delays in each direction in the round trip message are equal. Therefore, for the sample containing sent (S_1), received (R_1), replied (S_2), and returned (R_2) time stamps respectively, we have:

$$\textit{offset}(t) = \frac{R_1 + S_2}{2} - \frac{S_1 + R_2}{2}. \tag{3}$$

The history of computed offsets can then be used to determine skew. With continuously updated estimates for skew and offset, a local clock can effectively characterize a remote clock, whether to synchronize with it, or to simply translate to and from its reference time.

IV. Timing mechanisms

A. Synchronization with the Network Time Protocol

Synchronization involves two or more clocks acting as peers, wherein they share each other's view of the current time and collectively decide on a correct virtual clock, after which they each attempt to synchronize with that virtual clock. (Of course, an actual clock may serve as the virtual clock as well.) A useful simplification of this approach is a simple client-server configuration wherein one clock adjusts itself to its estimate of the time on the remote machine. The widespread Network Time Protocol (NTP) [9] utilizes both of these approaches, depending on the configuration.

In the NTP algorithm, clocks are arranged in a hierarchy, with a few precise clocks at the top, typically operated by national or international agencies, called Stratum 0 devices. Stratum 1 servers are directly connected to Stratum 0 devices, synchronizing the clock of the server to the Stratum 0 device, and subsequently serving the time to a slightly larger number of downstream Stratum 2 servers. The hierarchy continues, with most of the hundreds of thousands of NTP clients using the Stratum 2 and 3 servers.

For correct operation of a synchronization service, the NTP client obviously must have the ability to modify the local clock. While this is reasonable for most computer networks which are administered in a centralized fashion, it should noted that while the owner or administrator of the machine is privileged to modify the clock, a client of the machine is not. That is, if the machine serves as a sensor for a remotely located control system client, the control system does not have the authority to modify the clock. In other words, the control system cannot expect to install and run NTP in order to have all the components in its control loop synchronized.

Even if all the computers use NTP, different computers may receive the time from different time servers. Therefore, even "synchronized" clocks may differ in time. This is a direct result of the degradation in timing performance resulting from communication delays.

In summary, synchronization imposes unnecessary functional dependence which degrades reliability, which is increasingly regarded as the most important performance measure in large systems.

B. Time translation

In *time translation*, one clock creates an estimate of the current offset between itself and a particular remote clock of

interest as discussed in Section III, by requesting the current time of the remote clock. Upon receiving the reply, the local machine estimates the offset between the two clocks based on the sent, replied, and returned time stamps, as well as perhaps past exchanges. With this estimate, the time stamps of all future messages from the remote sender are simply translated into the time reference of the local machine. In this fashion, the only dependence is that the remote machine must provide the data and time stamps. As the system is already dependent on the sensor for the data, there is no additional dependence introduced.

There is no requirement for administrative privileges, or that the remote clock be synchronized with some universal time. The only requirement is that the remote clock respond to time requests and time stamp any time sensitive data. This then represents a necessary and sufficient dependency condition for reliability and proper operation.

V. The influence of delay on stability

Delayed samples lose their value over time for two reasons. The first reason is well known in control; delay that is uncompensated because it is unknown can drive a control system unstable. Second, the delay reduces the responsiveness of the system, thus deteriorating performance.

However, knowledge of per-packet delay, can improve both stability and performance. In particular, since time is itself an important part of the data, knowing when some sampled information was taken is important. We illustrate the difference between known and unknown delay by an experiment run on the Convergence Testbed.

A. Effect of unknown delay on overall system performance

For comparison, we first plot the trajectory of a car on the testbed under normal operation. The controller in this case is tuned to the known delay of the loop, which includes the time to send commands through the radio control transmitter, reaction time of a car, exposure time of the camera, processing time at the vision server, and the time to compute a new control, totaling approximately 240 ms. (We will see in Section V-D how one may estimate such delay.) Its performance in following a rounded rectangular trajectory is adequate, as can be seen from Figure 4.

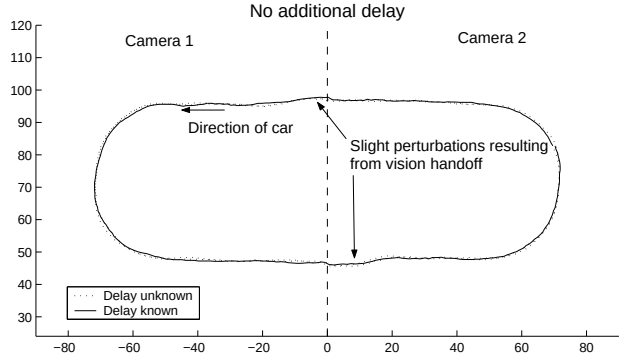

Fig. 4. Trajectory of car under normal operation.

To demonstrate the effect of unknown and therefore uncompensated delay on stability, we now intercept feedback packets received at the controller, hold them in a buffer, and pass them on after a fixed delay. This delay is in addition to the inherent and unavoidable overall system delay of 240 ms noted above. The experiment was conducted ten times, with varying values of additional delay, from 0 - 900 ms in 100 ms increments. We show the plot corresponding to 300 ms additional loop delay as it is the smallest additional delay which exhibits instability in the unknown delay case; see Figure 5. Note that with 300 ms of unknown delay (plus $\sim$ 240 ms of inherent delay), the system becomes unstable.

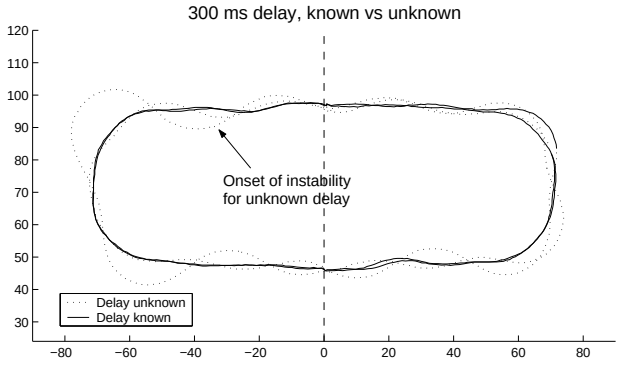

Fig. 5. Instability with 300 ms unknown additional delay.

By using time stamps, we can recover knowledge of the delay, and compensate it using a state estimator, as we now demonstrate.

B. State estimation

Although a closed loop system relies on sensor feedback, it is possible to compute an estimate of the system state even in the temporary absence of such sensory feedback, i.e., open loop, since the controller issued the controls to the plant, and has an estimate of the plant model. This is similar to the time update step of the Kalman Filter.

We employ such a state estimator in our testbed. If the time of a sample is known, the estimator can update the position estimate for that time, then predict the current state using the controls commanded from that point in time. Thus, we always have an estimate of the current state, even when vision is lost or delayed for some time. When new packets are received containing feedback data, the filter can use them to update the estimate of the system state, regardless of how old they are as, for example, in the measurement update step of the Kalman Filter.

To use feedback data, the filter thus needs to know the time at which the measurement was taken. In the testbed, this is the time at which the vision system captured the image of the track used to produce a particular position estimate. We create a time stamp at the moment when the vision server receives the image from the frame grabber.

C. The advantage of knowing per-packet delay for overall system stability and performance

To show that it is indeed valuable to know the delay, even when it is large, we show the results of the experiment again for even larger additional delay. These plots were created by first operating the state estimator with delay unknown for two complete loops all the way around the track, then switching to known delay for two additional loops. Note that in the 800 ms delay case, the dotted line in Figure 6, illustrating this unstability in the unknown delay

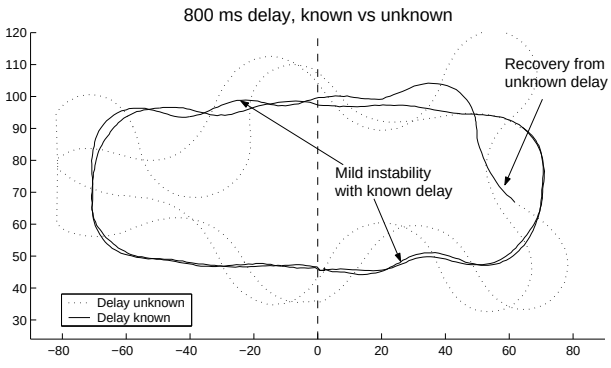

Fig. 6. Trajectory of car with 800 ms additional delay.

case, converts into a solid line on the right side of the plot, and thus shows the recovery from instability. Whereas the unknown delay produced instability at 300 ms of additional delay, the car remains stable even with 800 ms of additional delay when the delay is made known by time stamping, and is thus compensated for by the state estimator.

Comparing these two plots we can witness the very real advantage of knowing delay, or equivalently, knowing the sample time of the data. The system can remain stable for an additional 500 ms of delay beyond the delay which produced instability in the unknown delay case, roughly doubling the tolerable delay.

D. A method for determining delay in a control system

The CTP protocol can be used to estimate the delays in information flows only through the "controller" part of the network, where the nodes are at our disposal and can thus be commanded to time stamp information, however, it cannot be used for measuring actual delay in the plant dynamics itself. A different approach is therefore needed for estimating that component of overall loop delay. The basic idea is to tradeoff knowledge of plant gain for delay.[1] By using the known relationship between gain and delay, we can determine an unknown plant delay if we know the smallest gain which destabilizes it. The key point here is that even though time, and thus delay, cannot be measured accurately, if gains can be measured accurately in the plant itself, then we can actually determine the delay.

[1]We are grateful to Professor Raffaello D'Andrea (Cornell) for suggesting this approach.

We have implemented this procedure in the testbed, subject to the discreteness of control values, by building a simple controller resulting in $\dot{x}(t) = -\alpha x(t-\tau)$ as the system. For this test, we need only operate the car in one dimension; we chose the x-axis. The discrete nature of the controls, as well as the quantized position information, limit the precision to which we can determine plant delay.

Implementing this simple control law, we introduce periodic perturbations by changing the set point, as seen in Figure 7, every eight seconds, which is enough time for a

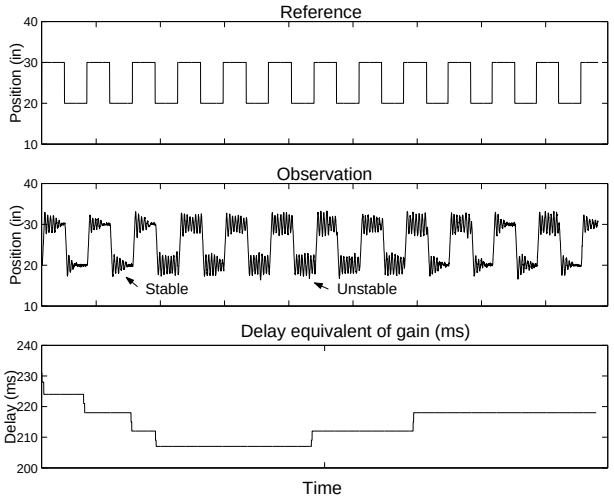

Fig. 7. Determining delay by observing the onset of instability as the gain is progressively increased.

stable delay-gain pair to stabilize.

In conducting this experiment, we start with a gain α for which the system is stable, then increase the gain in steps until the system demonstrates instability. We then reduce the gain again until the system stabilizes. These two gains then bound (above and below) the precise gain for which the onset of instability occurs. The top plot in Figure 7 shows the changing set point. The middle plot shows the actual position of the car as it attempts to reach the set point.

In the third graph we plot $\frac{\pi}{2\alpha}$, which is derived from the delay vs. gain relationship, and which we call the "delay tolerance," of the gain. It represents the amount of delay the system can tolerate at that gain. When the actual closed loop system delay exceeds the delay tolerance, the system becomes unstable.

Is is also possible to estimate plant delay online using temporal alignment, as discussed in [5].

VI. The Control Time Protocol

The Control Time Protocol uses time stamping to determine delays of all loops of information flows in the network. It avoids the adverse dependencies created when synchronization alone is used for determining delays. In fact the only dependency is on nodes in the closed loop, on which the system is already dependent anyway. It uses the estimate of the current offset to translate all messages

from the remote clock into the time reference of the local clock. Thus the overall system is an architectural solution. As important as the actual algorithms and filters is the clean approach to reducing unnecessary dependence.

We now present some aspects of the estimation methods for precision enhancement of CTP while operating in the presence of noise. We will then explore some additional uses for CTP beyond determining per-packet delay.

A. Improving the precision of CTP

As real systems have noise and varying delays, we wish to use algorithms which filter the noise through some sort of averaging or filtering. A reasonable choice for estimation is linear regression implemented with a linear least squares estimation algorithm.

However, we will see in Section VI-F that delays in networks have heavy tails. Thus there is a significant proportion of excessively delayed packets. In conjunction with the fact that in delay estimation we make the assumption of symmetry to overcome indeterminacy of Theorem 1, these packets have detrimental effects. Thus we need to make the estimation procedure robust to outliers [7]. If a packet's delay exceeds a threshold, say a multiple of the average delay, we do not use it in the computations. As the *skew* does not change very quickly, we can easily afford to ignore such packets.

1) Linear least squares: Unfortunately, the basic linear least squares algorithm weights all samples equally. If the clocks experience abrupt changes, the algorithm will not adapt to the change for a very long time. We thus wish to weight recent data more than earlier data.

2) Recursive linear least squares: Using the recursive form of the linear least squares algorithm (RLS) we can apply an exponential forgetting factor to the offset samples. This algorithm essentially stores all past data in a small matrix. Unfortunately, the representation of time is often very large and ever increasing. A commonly used representation uses Jan 1, 1970 as a reference time. Thus, current representations in milliseconds are on the order of 10^{13}, leading to an ill-conditioned matrix since one element is large and growing while the other is small and relatively constant. We adopt the method described next.

3) Windowed linear least squares: Instead of fitting the entire history of observations, we only fit a moving window of the samples. This will allow for rapid adaptation, while providing a smooth estimate under normal conditions. We have used a history of 1000 data points with good success, although the size of the history can easily be reduced. In fact, a very small history can be kept using the techniques of [10].

This algorithm is further augmented by a few practical filters to improve its robustness to outliers in delay and sudden shifts in *offset*. To cope with random large shifts in offset, we utilize the earlier observation that the actual offset during any exchange of packets can be bounded above and below. Therefore, when the windowed least squares estimate of the offset at the current time exceeds these bounds, we hold it to the bound rather than using the estimate.

B. Translation algorithm for the Control Time Protocol

CTP thus continuously monitors the offset between two clocks. To perform proper translation, the current estimate of offset need only be subtracted from the time stamp of an incoming data sample, to yield a local time stamp. The control algorithms can thus function with a common local time reference for all time related computations. If multiple sensors are used, CTP maintains an offset for each one of them and performs appropriate translation.

A further simplified version of the Control Time Protocol is often sufficient. This version stores no history and computes no drift. It periodically sends and receives a time stamped ping, assumes symmetric delay, and computes the offset accordingly. This offset is assumed to be true, and all incoming time stamps are adjusted accordingly. The computational overhead of this approach is very small, involving only a few subtraction operations. The drawback is that offset drift between pings is not accounted for.

C. Ping intervals

The frequency of pings in CTP must be appropriately chosen with respect to the system. We do this adaptively by changing the ping interval the following manner. When an incoming packet has an unusually large delay, or appears to have a negative delay due to an offset shift, we initiate an extra exchange of messages to verify the current offset. When the message exchange is complete, the local node re-translates the time stamp. This provides robustness to clock resets and other shifts in offset, allowing the nominal interval between message exchanges to be large.

D. CTP scaling

CTP clearly does not scale well as each consumer is generating ping messages for a sensor, resulting in a ping explosion as the number of sensors and controllers grows. However, this issue is not of concern for a few reasons. First, the frequency of pings required to maintain an accurate offset is quite low, depending on the precision of the clocks. Moreover, CTP can easily be modified to piggyback on existing data packets exchanges, thus rendering the overhead negligible. Second, the numbers of sensors useful to a particular controller is limited, thus scaling laws may not be particularly relevant. Third, the number of controllers to which a sensor may be able to send data is also fairly limited. The communication bandwidth consumed by the data packets far outweighs any bandwidth consumed by the protocol, and its percentage overhead is thus low. Thus scaling is limited more by the sensor and controller configurations rather than by the timing service.

E. CTP alongside NTP

There is no interaction between a time synchronization protocol and a translation protocol. In fact, using both

simultaneously allows one to enhance the performance of each. If a system running NTP also runs CTP, the offset determination for CTP should always be close to zero. If it is not, CTP can alert the system about the problem. CTP can always be used for control, regardless of the presence or absence of NTP, or any other synchronization method.

F. Studies of network delay

In recent years there has been much interest in developing control laws appropriate for use over communication networks. To aid in this research, we present results of experiments of actual delays experienced over networks.

Three two important characteristics may be noted. The first is the heavy tailed nature of packet delays. For systems which depend on low latency, the infrequent but large occasional delays can disrupt control loops. This can be seen in Figure 8. Packets with round trip times greater than 100 ms were rejected in this experiment. A second

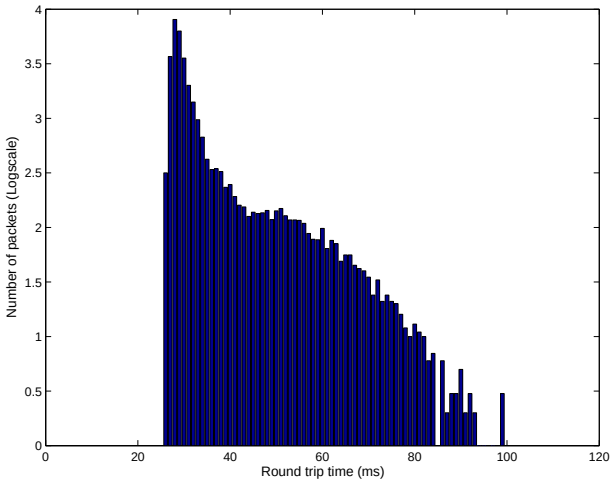

Fig. 8. Histogram of round trip times

observation relates to the growth in minimum and average latencies as a function of the number of hops a packet must traverse, and the communication medium used. Figure 9 exhibits an experiment where two unloaded networks, one wireless and the other wireline, were used to relay packets from one through six hops. Note that the minimum round trip times increase linearly in this simple setup, and that the slope of the increase is different for the two communication mediums.

A final point is that network delay is inherently unbounded.

VII. Conclusion

Knowledge of time is fundamental to control in general, and distributed control in particular. It is becoming increasingly more important as one develops control laws for systems running over general-purpose communication and computation networks. Time stamping provides valuable timing information since it makes available knowledge of

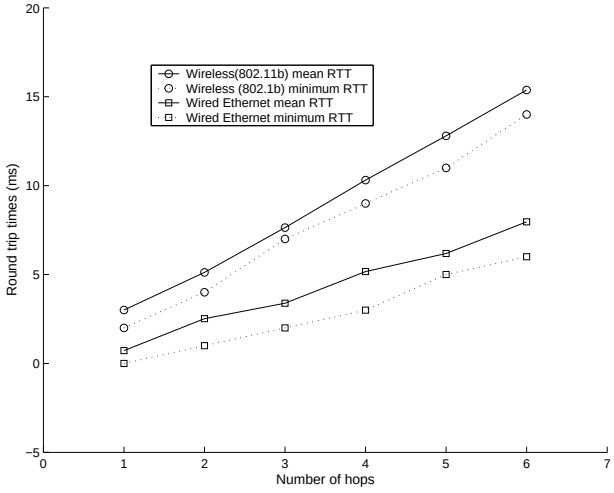

Fig. 9. Mean and minimum round trip times by hops

the per-packet delay, and thus allows the design of superior controllers to stabilize systems and improve performance. Yet it potentially introduces new dependencies in the overall system which adversely affect reliability. The Control Time Protocol is a solution to the need for time stamping, without creating unnecessary dependencies in the system. It provides robustness of an overall system to widely varying distributed clock dynamics.

References

[1] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System" *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, July 1978.

[2] G. Leen, D. Heffernan, "Expanding Automotive Electronic Systems" *IEEE Computer*, vol. 35, issue 1, pp. 88–93, Jan 2002.

[3] M. Katevenis, "Reduced Instruction Set Computer Architectures for VLSI", *ACM Doctoral Dissertation Award*, MIT Press, 1984.

[4] S. Graham, G. Baliga, and P. R. Kumar, "The Convergence of control with communication and computation: proliferation, architecture, design, and middleware." Submitted to IEEE Conference on Decision and Control, Dec 2004.

[5] K. Huang, S. Graham, and P. R. Kumar, "Temporal alignment of distributed sensors with an application to characterization of plant delay." Submitted to IEEE Conference on Decision and Control, Dec 2004.

[6] "The Convergence Laboratory testbed," University of Illinois, http://black1.csl.uiuc.edu/∼prkumar/convergencelab/.

[7] P. J. Huber, "Robust Statistics" Wiley, New York, 1981

[8] JavaTM 2 Platform Std. Ed. v1.4.2 documentation, Sun

[9] Internet time synchronization: the network time protocol, *IEEE Trans. Communications*, vol. 39 no. 10 pp. 1482–1493, Oct 1991.

[10] M. L. Sichitiu, C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks", *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC 2003)*, New Orleans, LA, March 2003.