Design Methodology for Robust, Distributed Time-Sensitive Applications

Aviral Shrivastava, Mohammad Khayatian, and Bob Iannucci

Abstract

Time has become an essential aspect of many computing systems where temporal correctness is as important as functional correctness. Autonomous vehicles, Industry 4.0, and smart grids are a few examples of time-sensitive systems. As time-sensitive applications become large, complex, and distributed, traditional methods fall short of achieving the desired orchestration among components. In this vision article, we first propose a standard to maintain an accurate notion of time among all components of the system, i.e., sensors, computing platforms, and actuators. Then, we propose explicit-time state estimation and closed-loop control algorithms that can tolerate large delays while achieving reasonable performance, and an integrated fail-safe mechanism that achieves a high level of robustness when timing failures happen.

Introduction

Large-scale, distributed, and time-sensitive applications are becoming the backbone of human society. Multiple robots working in sync on a factory floor will optimize production and realize Industry 4.0. Autonomously driven vehicles that communicate with each other to achieve safer and more efficient transportation will realize our dreams of smart transportation. Handling system transients such as weather events for renewable energy sources and sudden insertion/removal of micro-generators will propel us toward a reliable and resilient Smart Grid. All these important applications have performance and safety constraints that are inextricably linked to time.

Since some timing constraints are so crucial for the correct and safe operation of time-sensitive applications, systems have traditionally been designed in a way to avoid timing constraint failures. This is typically achieved by carefully crafting the control algorithm, selecting the hardware and software of the computing components, and tuning the execution mechanism of the software, including describing the application as a list of repeating tasks, specifying the periods, deadlines, and priorities, mapping and schedule of the tasks, etc. — all so that the timing constraints can be guaranteed to be met by design. However, this approach does not scale well for large, distributed time-sensitive applications. As such systems grow in complexity, the use of commercial off-the-shelf hardware together with software libraries, languages, and compilers — not designed with timing in mind, makes the system design challenging. Engineering out every possible timing constraint failure necessitates anticipation of every failure mode (including soft errors, aging effects, ...) that could have timing implications — an impossibly tall order for practical systems. Rather than framing the problem this way, we argue that system designers should *embrace the possibility of occasional timing constraint failures and explicitly-programmed ways to deal with them*.

The traditional approach of designing systems to avoid timing constraint failures has inherent appeal. It stems from a rather logical desire to segregate time-based control systems into two parts: the time part and the control part. When it can be done, the control part can be *time-agnostic*. Sensors can just read values and forward them along. Computing can happen when data are available. Actuators can just do their actuation when commands arrive in blissful ignorance of time because all of the timekeeping machinery is somehow above, beside, and around, assuring that everything remains in a good temporal order. **Indeed there is irony in the fact that time-sensitive systems execute in a time-agnostic manner**. But freedom from thoughts of time within the control logic comes at a substantial implicit cost: the machinery on which it runs needs to be more-or-less deterministic in its temporal behavior. And as systems grow, the cost of making them so becomes prohibitive.

Communication latencies and their variation can be orders of magnitude longer than the update frequency of state estimation or control algorithms. Consider the case of autonomous vehicles sharing a highway — each one with its own navigational agenda but all seeking to maintain safe separation at all times. To do this, they should exchange position information periodically, updating state information in their control algorithms that are adjusting the vehicle's trajectory every few milliseconds. Would it be consequential to a time-agnostic state estimator if one such exchange was delayed by 100 msec (not at all atypical in a cellular network)? At highway speeds (say, 100 kilometers per hour), the "surprise"delay could correspond to an aggregate misestimation of position in excess of five meters — certainly enough to cause concern. Although next generation communication technologies (like 5G) are being deployed, they have limitations and

Digital Object Identifier: 10.1109/IOTM.001.2300048

Aviral Shrivastava is with Arizona State University, USA; Mohammad Khayatian is with Vecna Robotics, USA; Bob Iannucci is with Google, USA.

perhaps we need to go back and re-consider the validity of the assumption of separating the time part of this system from the control part.

This article proposes the design and architecture of an *explicit-time distributed system* to support robust execution of distributed time-sensitive applications. An explicit-time system consists of three parts:

1. Explicit-time State Estimation and Control Algorithms
2. Explicit-time Sensors and Actuators
3. Backup-Routine-based Robust Execution.

Explicit-Time State Estimation

State estimation is at the heart of most Cyber-Physical Systems (CPS). Kalman filter is a state estimator that is widely used for sensor fusion, Simultaneous Localization and Mapping (SLAM), as well as signal processing. The goal of the state estimation problem is to determine the state of the system — which may or may not be directly observable — using a system model and inaccurate measurements of the system. The vanilla versions of the state estimation algorithms, like the Kalman filter, Luenberger observer, and Particle filter assume that all the measured values are captured simultaneously at the same moment and that the computation and communication times are negligible. When the sampling period of sensors is not the same (e.g., an IMU works at 100 Hz but GPS works at 10 Hz) multi-rate state estimation algorithms [1] are used. Such approaches still assume that the sensor readings are perfectly synchronized and therefore, the actual capture time of each sample can be ignored. However, for large-scale, distributed CPS, where sensing, actuation and computing may be happening on different nodes that are connected by communication protocols that may have high and highly variable latencies, time-agnostic state estimation algorithms will not be accurate. Imagine the scenario where multiple AVs are broadcasting their positional information and performing a joint or distributed state estimation. The packets from a vehicle may be delayed by even seconds, and given the fast dynamics of the system (in the context of the amount of delay), time-unaware state estimation can become inaccurate. **Explicit-time state estimation takes in the timestamps at which the measured values were sensed as well as the inaccuracies in the captured time and value to estimate the state of the system at an explicit time in the future.**

We explain our idea of the design of an explicit-time state estimation algorithm based on the popular Kalman filter algorithm. A unique aspect of our proposed approach is that while existing methods only consider the uncertainty in the measured values (due to sensor noise, ADC resolution, etc.), we will consider the uncertainty in the measured values as well as the uncertainty in the time those measurements were captured at, which is depicted in Fig. 1. In our model, each sensor reading is a quadruple <z, e_z, t, e_t> comprising of the measured value (z), the error in the value e_z, the time of measurement, or the captured timestamp (t) and the error in the captured timestamp (e_t). It should be noted that the measurement times may be inaccurate due to several reasons including the internal clock quality, synchronization source, protocol, and the synchronization frequency of the clock.

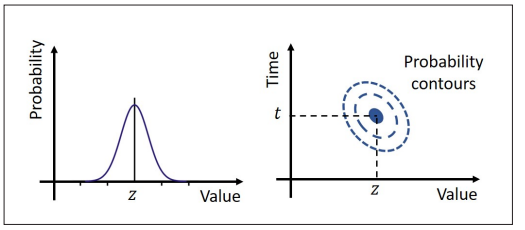

FIGURE 1. Time-agnostic state estimation approaches only consider the inaccuracy in the measurement values (left). Explicit-time state estimation will consider the errors in the measurement values as well as the error in the time at which the measurement was captured.

We propose to model *time* explicitly as a part of the system state i.e., $X = [x_1; x_2; ...; x_n; t]$. Figure 2 shows that since we have made time as a part of the state of the system, i.e., $X = [x_1; x_2; ...; x_n; t]$, the covariance matrix P will have one more row and column to represent the covariance of time with respect to the states. In a distributed system, the measured values from the different parts of the system may end up being captured at different times. Previous approaches wait for all the measurements from the different parts of the system to arrive before they can update the system state. In our approach, we perform the state estimation for each measured value as it arrives. The idea is to repeatedly update the state of the system to the time of the considered measurement value.

When a new measurement $[z_k, t_k]$ is received, the discretized system model (f) — how the system state evolves with time — is re-computed based on $\Delta t = t_k - t_{k-1}$. Similarly, the derivative of the system model F is also recomputed. Then, the predicted system state X and its covariance matrix P are updated based on F and F. Next, the prediction error ($\tilde{y}$) is computed and the Kalman gain (K) is calculated using updated (R_k). R_k is the expected covariance matrix of the measurement noise. Since each measurement has uncertainty in the captured value and time, (R_k) can be updated based on the reported uncertainty of the measured values (e_z) and time (e_t). This way, if a measurement is coming from a node with an inaccurate clock, the update step takes into account the inaccuracy in the time (e_t). Finally, the state of the system (including the time at which the state is estimated) ($\hat{X}$) and its covariance P are updated based on the prediction error ($\tilde{y}$) and the Kalman gain (K). This process is repeated at the arrival of every new measurement in the order of their timestamps.

The foremost advantage of explicit-time state estimation is that it severs the false conflation between the time at which the measurement was taken and the time at which the measurement arrives at the estimator and the state is updated. And this results in accurate state estimation. We implemented a simple version of an explicit-time Kalman filter to estimate the position and orientation of a vehicle in 2 dimensions by sensing its position from a GPS (Global Positioning System) sensor at 2 Hz and its heading from an IMU (Inertial Measurement Unit) at 100 Hz. The inaccuracy in GPS values is about ±10%, and in the IMU values, about ±5%, and the inaccuracy in both the

The foremost advantage of explicit-time state estimation is that it severs the false conflation between the time at which the measurement was taken and the time at which the measurement arrives at the estimator and the state is updated.

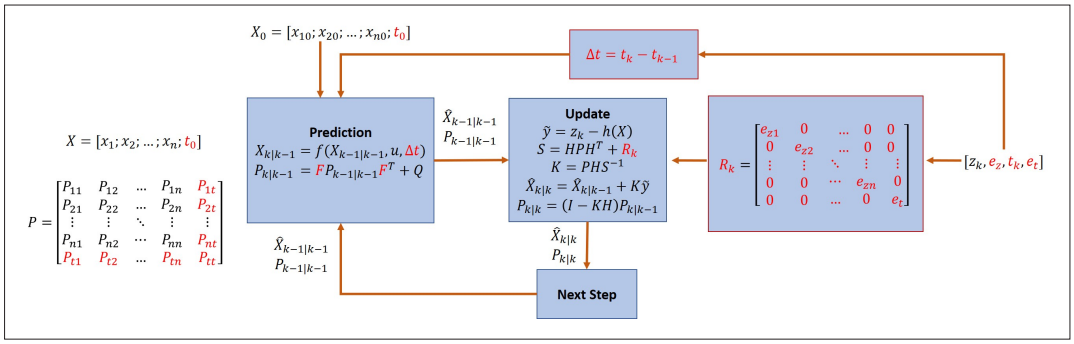

FIGURE 2. Outline of the Explicit-time Kalman filter. Parts highlighted in red show the proposed modifications/extensions.

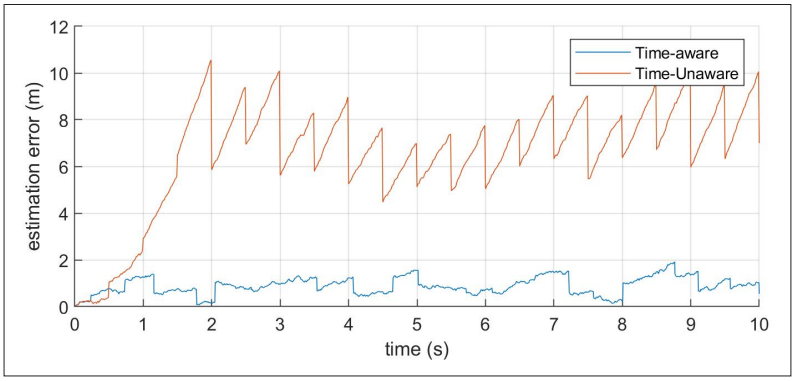

FIGURE 3. Explicit-time Kalman filter can more accurately estimate the state of the system compared to traditional Time-agnostic Kalman filter(Orange).

timestamps is less than ±0.1s. Figure 3 shows that the explicit-time Kalman filter (orange) can estimate the vehicle position (blue) much better than the traditional implementation (yellow). Just like the Kalman filer, other state estimation algorithms, including the Particle filter, Luenberger observer, and different flavors of the Kalman filter such as EKF, UKF, etc. can also be made explicit-time.

Explicit-Time Control Algorithms

Since delays have a more dramatic effect on the quality of control algorithms, more research has been done on the stability of control systems with sensor-to-actuator delay [2]. Hierarchical architectures such as Distributed Control Systems (DCS) and Supervisory Control and Data Acquisition (SCADA) are commonly used in the industry where the fast control loops are implemented locally and only the supervisory control inputs (such as a reference set point) are affected by network delay. When the delay is constant and known, control algorithms like Smith predictor [3], Model Predictive Control (MPC), and delay compensated Proportional–Integral–Derivative (PID) control can be used to compensate for the delay and provide high-performance control. However, for distributed CPS with variable delays, these approaches are unable to provide good-quality control. Figure 4 shows the response time of a Smith predictor when the delay is assumed to be 500 ms but actually it varies randomly between [100, 500] ms. The reference input is a step function changing from 10 to 9 at t = 0.5s. The orange curve shows that the Smith predictor control algorithm cannot properly track the desired reference. There is some work on developing control algorithms for systems with variable time delay [4]. However, the controller design and stability analysis of such systems becomes very complex and results in a conservative design. This is mainly because the stability proof should be provided for all the allowed values of time delays.

In Explicit-time control algorithm the actuation can be scheduled to be applied at a certain specific time. The explicit-time control algorithm can benefit from the knowledge of actuation time, and achieve better performance. Additionally, the controller design and safety proofs of the explicit-time control algorithms can be simplified, if the upper bound on the sensing-to-actuation latency can be fixed to a constant.

We explain explicit-time control algorithms using the Smith predictor. Consider a system as depicted in Fig. 5. $G(s)$ is the plant and $C(s)$ is the controller. The sensor delay and actuator delay are δ_1 and δ_2, respectively. Figure 5 shows our modifications to the Smith predictor to make it explicit-time. In explicit-time Smith predictor, the controller sets the actuation time and therefore, is aware of the sensing-to-actuation delay ($\delta_1 + \delta_2 = t_A - t_S$) and can compensate for it. The error signal (e) is generated based on the predicted output ($\hat{y}$) and the reference signal (x_r). If the model mismatch ($\Delta m = G(s) - \hat{G}(s)$) is small, the signal y_m cancels the delayed output (y) and with a proper controller, the output can accurately track the reference input.

The time-aware nature of explicit-time Controller will allow it to naturally adapt to the inherently variable computation and communication delays of distributed systems. The blue curve in Fig. 4 shows the output response of our approach, and it is clear that our approach can follow the reference much better. If the actuation time is set to be later than the worst-case delivery time to the actuator, then the sensing-to-actuation delay can be made constant and that makes the controller design and stability analysis much simpler as compared to existing approaches. This is mainly because tighter bounds can be considered for the Lyapunov functions to show that its derivative is negative-definite.

Explicit-Time Sensors, Actuators, and Networks

Ignoring the sensing, computation, networking, and actuation times in DCS can cause performance degradation [5]. As a reference, [6] discusses the network impact on the performance of DCS. Our proposed explicit-time architecture requires the sensors to provide a timestamp along with the value they sense, and actuators have a

mechanism to deliver the actuation at the scheduled timestamp (within the specified tolerance).

Currently, only a few sensors have such capability. For example, the Velodyne VLS-128 LIDAR has a GPS inside the LIDAR to provide timestamps with the point cloud readings. We are not aware of actuators that are capable of scheduled actuation. Depending on the design and modifyability of the sensors and actuators, they can be made explicit-time to varying levels of timing accuracy.

Some sensors and actuators have a microprocessor and therefore an internal clock inside them, and their firmware is modifiable. Either their clocks are already synchronized to an external clock source, e.g., Global Positioning System (GPS). If not, they can be made to synchronize with the clock of the CPS node that it is connected to. For actuators, the firmware of the actuators can be modified so that they set up an internal interrupt to fire at the desired actuation time to perform the actuation. Some other sensors and actuators have an internal processor/microcontroller, but their firmware cannot be modified. Most existing sensors and actuators fall into this category. The best way to capture the sensing time in these kinds of sensors is to capture the timestamp at the CPS node the sensor is connected to. The best way to control the actuation time on these kinds of actuators is to manage the timing and actuation from the CPS node that the actuator is connected to. The driver software for the actuator can be modified so that actuation commands can be sent to the actuator on time. Finally, many analog sensors and actuators have a simple structure and do not even have an internal processor. These sensors and actuators are usually directly connected to the pins of a CPS node. The value sensing is done through the ADC (Analog to digital converter), and the actuation is done through PWM (Pulse Width Modulation). For these sensors and actuators, the sensing/actuation time should be controlled by the CPS node directly.

As mentioned in the previous section, the controller computes an actuation value to be delivered by the actuator at the specified timestamp. Since the timestamp should be set to a later time to account for computation and actuation times and more importantly network delay, the controller's performance may be low for a regular network. Time-Sensitive Network (TSN) [7], on the other hand, provides deterministic delays through traffic shaping. As a result, the controller will set the actuation timestamps as tight as possible to achieve higher performances. Although (Network TIme Protocol) NTP is widely used for clock synchronization, components of a TSN use Precision Time Protocol (PTP) [8] to achieve sub-microseconds clock synchronization.

Backup Routine-Based Robust Execution

Failure of timing constraints is an unavoidable concern in the design of time-sensitive systems, and different solutions have been suggested at various levels of CPS design abstraction. At the runtime level, Medhat *et al.* proposed to monitor the end-to-end timing constraints [9], and on failure, the event can be logged, the user can be warned about the event, or the program just terminated. At the task scheduling level, various approaches for budget replenishment have been

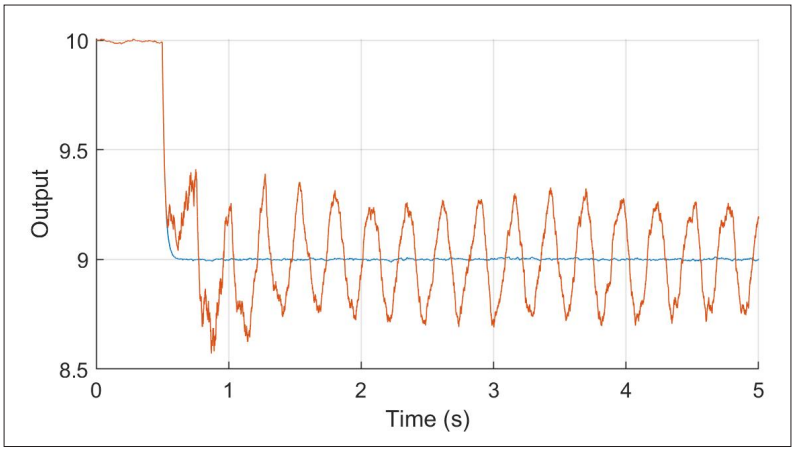

FIGURE 4. The Smith predictor (Orange) is not able to track the reference when the delay is variable. However, Explict-time Smith Predictor can track it pretty accurately (blue).

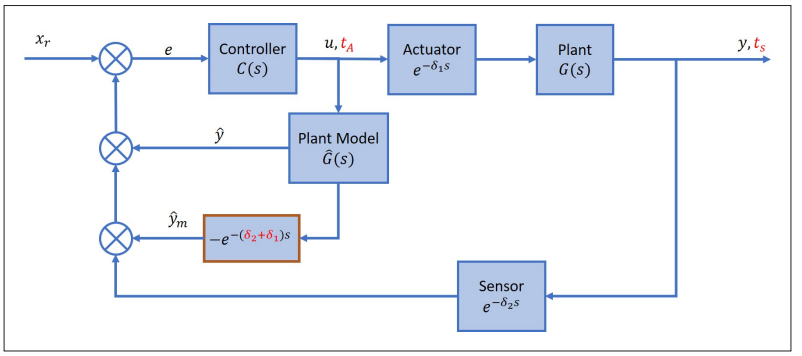

FIGURE 5. Explicit-time Smith Predictor can compensate for the sensing-to-actuation delay since it explicitly knows the sensing time and determines the actuation time. The modifications are highlighted in red

proposed where the scheduler decides to abort (kill the task for the current iteration and not generate an output), ignore (continue and eventually generate the output), or queue (which is the default when timing failures are not considered) an overrun execution. A more complicated way to handle timing failure is to continuously adjust the period of tasks based on the best-case and worst-case delays that have been observed till the current moment. However, providing safety guarantees for such approaches is hard and a rare long execution time can make the design very conservative. In the control systems domain, researchers have developed adaptive control approaches that work for variable time-delay systems by updating the parameters of the controller. However, the controller design and stability analysis of such controllers does not account for the exceeded time delays and timing failure.

The main idea of Backup Routine-based execution is to design a CPS such that it will meet the timing constraints "most of the time," but if the timing constraint is not going to be met, a "backup routine"will be fired up in time to keep the system in a safe state [10]. This approach does result in a conservative design that is developed based on the worst-case timing and at the same time guarantees safety by relying on a fail-safe backup routine that is triggered on time. Since the designed system may switch between "normal"and "backup"modes, system safety and stability should be verified similar to switched and hybrid systems.

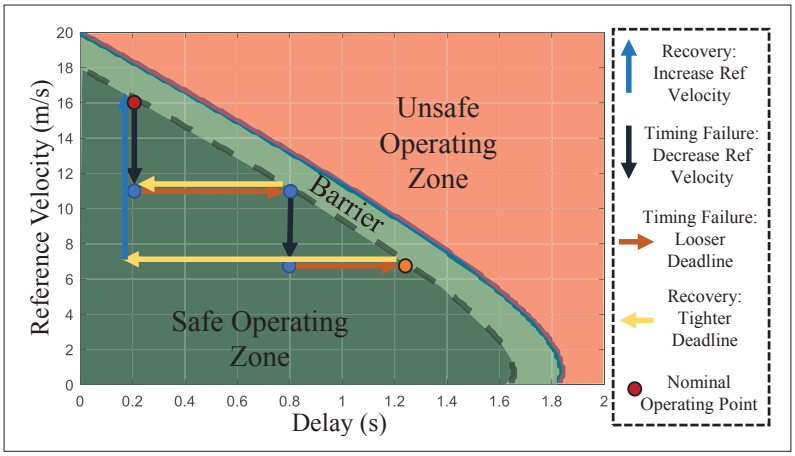

FIGURE 6. Backup routine-based execution method enables CPS designers to optimize system performance while meeting the safety requirements. As sensor-to-actuation delay increases, the system can adapt itself and reduce the maximum or reference velocity (black vertical arrows), and when it is met again, the system can go back to operating at a higher maximum velocity (blue vertical arrows).

One of the most important safety-critical timing constraints in an AV is that the delay from the sensing (using cameras, LiDAR, or RADAR) to applying the brake (if needed) should be less than a maximum value. If this timing constraint is not going to be met, a simple backup routine is to apply the "brake"before the deadline expires. This early execution of the backup routine — before the deadline expires — will in some cases result in false positives — i.e., executing the backup routine even though the timing constraint was actually going to be met. However, even with that, the backup routine mechanism will ensure system safety, and therefore convert a safety requirement (meeting the timing constraint) into an optimization metric — The safety of the system now actually depends on making sure that the backup routine completes in time. This can be achieved since the backup routine can be really small, fast, and local, and it can even be permanently cached so as to make its execution really fast. This way executing the backup routine will only require a small fraction of the original deadline, and thereby reduce the rate of false-positive firings of the backup routine. It should be noted that the AV will stop if its velocity is less than 2 m/s.

As mentioned, the main advantage of the backup routine-based execution mechanism is that it makes the original timing constraint a performance issue and not a correctness issue. The correctness is dependent on the timely execution of the backup routines. This flexibility gives system designers freedom to explore various methods for continuously adapt the system execution to optimize system performance while ensuring safety, e.g., operating the system at lower performance when timing constraints are met with a small margin or violated, and then shift back towards operating at a higher performance point when timing constraints are met with large margins. Consider an AV (Autonomous Vehicle) that can safely operate at 16 m/s. This requires the sensing-to-actuation latency to be less than 200 ms. Figure 6 shows the relationship between the maximum velocity of an AV and its sensing-to-actuation delay requirement. The red area (top right) shows unsafe operating points and the green area (bottom left) shows the safe operating points. The AV was supposed to drive at 16 m/s and suppose a timing failure happens. The backup routine can then reduce the maximum velocity of the AV by 20% and adjust the sensing-to-actuation timing requirement to 12.8 m/s. After the execution of the backup routine, the delay requirement is relaxed, and the AV can still safely operate, albeit at a lower performance level. If the delay becomes greater than 600 ms, another backup routine can be executed, reducing the maximum speed of AV to 10.2 m/s, and setting the sensor-to-actuator timing requirement to be less than 950 ms. When the end-to-end delay returns back to normal (200 ms), the AV can re-adjust its operating point and drive at its nominal velocity (16 m/s). If the delay is beyond the acceptable tolerance of the AV (1.6 s), the fail-safe backup routine (applying full brake) can be executed.

This backup routine-based execution mechanism can become the central point for tackling several robustness issues of the CPS. This is because a lot of other system failures also show up as timing failures. For example, if a sensor goes bad and stops working, then that manifests itself as a timing failure at the controller that is supposed to use the sensed value. Diagnosis and recovery (startup of a backup sensor) in such scenarios can be done as a part of backup routines.

Future Research Directions

Control-theoretic Aspects: Just like the Smith predictor, explicit-time versions of MPC, delay-compensated PID, state feedback control, etc. can be developed. Their stability can be proved using Lyapunov-Krasovskii and Lyapunov-Razumikhin methods. In these approaches, a positive definite energy (Lyapunov) function can be considered and with the help of time-invariant functions, it should be possible to show that the derivative of this function is negative definite.

Technological Realizability: One challenge related to the proposed Kalman filter is a determination of covariance matrices (R_k and Q_k). Researchers have proposed techniques to overcome this challenge [11]. Another challenge associated with the explicit-time Kalman filter is that more computation is needed since the algorithm computes the inverse of the matrix S with every received measurement. However, the good news is that the computation overhead is not too much since the sensor model h and its derivative are sparse (they only consider one measurement at a time) matrices and there is scope for further reducing the computation overhead. In addition, the state estimation for samples that end up being captured at times that are very close to each other can be performed in the same iteration. As another challenge, it may not be possible to determine strict ordering among the measurements since the timestamps of the sensed values can be inaccurate (e.g., due to the limits of clock synchronization). Those measurements may be good candidates to club together for a single-shot update. As a suggestion, the sensitivity of the sensed values can be used as a metric to determine how much we can delay the update for a sensed value or combine its update with the later ones, or just drop them. In networked systems, it is even possible that the sensed values may

arrive at the compute node in a different order than the one in which the measurements were taken. Clearly, we will need a buffer in front of the compute node to store and reorder the measurement values. However, if that is not possible (for example because one packet was delayed a lot), several solutions are possible, including, just ignoring the incoming value if it is within a small margin of the expected value at its timestamp. If the measurement is coming too late, then rollback to a previous state, and then recompute from there is a possibility. It may be possible to develop analytical solutions to apply the effects of the late-coming value without doing the rollback — if that happens often.

Platform-related Constraints: An important issue in non-explicit-time sensors and actuators is that there is always a bias in the captured timestamp (i.e., the captured timestamp is always late), and in the actuation time (i.e., the actuation time is always later than the time when the system node applied the actuation) that is not compensated for. This is because of the intervening microcontroller inside the sensor or actuator that takes a non-zero time to provide the sensed value or perform the actuation. Research is needed to develop ways to estimate these systematic biases and compensate for them. The good news is that this systematic bias should have low variation and therefore it should be possible to estimate it using existing time delay estimation methods [12].

Timely execution of backup routines: The most straightforward approach to execute the backup routine at the right time will be to set up a timer at the start event of the timing constraint and fire it at the right time (before the deadline of the timing constraint) so that the system safety can be ensured. However, one challenge associated with using timers for firing up the backup routines is that many platforms have a limited number of hardware timers while an application may have multiple timing constraints. To address this issue, the concept of virtual timers can be used to accommodate multiple timing constraints using a single hardware timer. A virtual timer is basically the combination of a hardware timer and a software queue of backup routines in sorted order by their firing times. The timer always verifies and fires the backup routine at the head of the queue, and removes them after that. In case of repeating timing constraints, the backup routine just moves to the back of the queue at its right place (by firing time). Since the firing of backup routines on time is safety-critical, they can be pre-cached and will have a high priority for execution. They can also be executed on a separate and dedicated processor so that the WCET of the backup routines can be computed more accurately. Also, although rarely, there can be cases when multiple timing constraints fail at the same time or are very close to each other so that the execution of the corresponding backup routines overlap each other. In such cases, the maximum block time for a backup routine — by other backup routines — should be taken into account which can be done by assigning a priority value to each backup routine and performing an analysis similar to the WCRT (Worst Case Response Time) analysis [13].

Determining safe backup routines: Given a backup routine and a "firing time," it should be possible to test/check if the backup routine can keep the system in a safe state by modeling the physics of the system with differential equations and software using a state machine. From a high-level perspective, the software part of the proposed architecture operates in two modes:

1. Normal operation, where no timing violations happen
2. Backup mode when a timing failure happens. In the CPS domain, such systems are commonly modeled as switched systems and more commonly hybrid systems

A few tools such as Breach [14] and Flow* [15] can determine the set of the state that a hybrid system may reach in the future and verify if it reaches a set of unsafe states.

Security: Networked systems are prone to cyber attacks such as spoofing and Denial of Service (DoS). An attacker can change the sensed timestamp or commanded actuation timestamp or prevent the actuator from receiving the actuation command and timestamp. As a result, sensors, computation nodes, and actuators should have a mechanism to encode/decode the data that is sent over the network and be able to detect spoofed packets. In addition, spoofing can happen if the components of the system (e.g. Phase Measurement Unit or PMU) synchronize its clock via GPS. However, GPS with a precise internal clock can detect considerable GPS spoofing and have deterministic time accuracy.

The most straightforward approach to execute the backup routine at the right time will be to set up a timer at the start event of the timing constraint and fire it at the right time (before the deadline of the timing constraint) so that the system safety can be ensured.

Conclusion

In this vision article, we study the challenges of meeting timing constraints in complex, large-scale, and distributed time-sensitive applications. We propose a set of standards to build explicit-time sensors and actuators as a prerequisite to perform explicit-time state estimation and closed-loop control. Our explicit-time state estimation and control algorithms can tolerate large and variable delays while achieving high performance. Finally, we introduce an integrated fail-safe mechanism that kicks in when timing constraints are not met. Our fail-safe mechanism ensures the safety of the system when timing failures happen.

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grants No. 1646235 and 1645578.

References

[1] Y. Liang, T. Chen, and Q. Pan, "Multirate Optimal State Estimation," *Int'l. J. Control*, vol. 82, no. 11, 2009, pp. 2059–76.
[2] E. Fridman, "Tutorial on Lyapunov-Based Methods for Time-Delay Systems," *European Journal of Control*, vol. 20, no. 6, 2014, pp. 271–83.
[3] C.-L. Lai and P.-L. Hsu, "Design the Remote Control System with the Time-Delay Estimator and the Adaptive Smith Predictor," *IEEE Trans. Industrial Informatics*, vol. 6, no. 1, 2009, pp. 73–80.
[4] W. Zhang *et al.*, "A Double Disturbance Observer Design for Compensation of Unknown Time Delay in A Wireless Motion Control System," *IEEE Trans. Control Systems Technology*, vol. 26, no. 2, 2017, pp. 675–83.
[5] J. K. Yook, D. M. Tilbury, and N. R. Soparkar, "A Design Methodology for Distributed Control Systems to Optimize Performance in the Presence of Time Delays," *Proc. 2000 American Control Conf. ACC (IEEE Cat. No. 00CH36334)*, vol. 3, 2000, pp. 1959–64.
[6] F.-L. Lian, J. Moyne, and D. Tilbury, "Network design consideration for Distributed Control Systems," *IEEE Trans. Control Systems Technology*, vol. 10, no. 2, 2002, pp. 297–307.
[7] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, 2018, pp. 22–28.

[8] J. Kannisto *et al.*, "Software and Hardware Prototypes of the ieee 1588 Precision Time Protocol on Wireless LAN," *2005 14th IEEE Wksp. Local & Metropolitan Area Networks*, 2005, pp. 6.
[9] R. Medhat *et al.*, "Runtime Monitoring of Cyber-Physical Systems Under Timing and Memory Constraints," *ACM Trans. Embedded Computing Systems (TECS)*, vol. 14, no. 4, 2015, pp. 1–29.
[10] M. Khayatian *et al.*, "Plan B-Design Methodology for Cyber-Physical Systems Robust to Timing Failures," *ACM Trans. Cyber-Physical Systems*, 2022.
[11] Y. Huang *et al.*, "A Novel Adaptive Kalman Filter with Inaccurate process and Measurement Noise Covariance Matrices," *IEEE Trans. Automatic Control*, vol. 63, no. 2, 2017, pp. 594–601.
[12] L. Chunmao and X. Jian, "Adaptive Delay Estimation and Control of Networked Control Systems," *2006 Int'l. Symp. Communications and Information Technologies*, 2006, pp. 707–10.
[13] R. J Bril, J. J. Lukkien, and W. F. J. Verhaegh, "Worst-Case Response Time Analysis of Real-Time Tasks Under Fixed-Priority Scheduling with Deferred Preemption Revisited," *19th Euromicro Conf. Real-Time Systems (ECRTS'07)*, 2007, pp. 269–79.
[14] A. Donzé, "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems," *Int'l. Conf. Computer Aided Verification*, Springer, 2010, pp. 167–70.
[15] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Flow*: An Analyzer for Non-Linear Hybrid Systems," *Int'l. Conf. Computer Aided Verification*, Springer, 2013, pp. 258–63.

Biographies

Aviral Shrivastava is a full Professor in the School of Computing and AI at Arizona State University, where he established and heads the Make Programming Simple Lab (https://labs.engineering.asu.edu/mps-lab/). He completed his Ph.D. in Information and Computer Science and from the University of California, Irvine, and bachelor's in Computer Science and Engineering from IIT Delhi.

Mohammad Khayatian is a Senior Robotics Software Engineer at Vecna Robotics, working on autonomous mobile robots for wearhouses. He received his Ph.D. in Computer Engineering from the Arizona State University and his M.Sc. and B.Sc. in Electrical Engineering — Control Systems from Shiraz University.

Bob Iannucci is a Distinguished Engineer at Google. Before this he was a Distinguished Service Professor at Carnegie Mellon University, and Senior VP and CTO at Nokia. He recieved his Ph.D. from MIT in Electrical Engineering and Computer Science.