IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON CYBER-PHYSICAL SYSTEMS

Received December 31, 2018, accepted February 7, 2019, date of publication February 25, 2019, date of current version March 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2901509

On Enabling Technologies for the Internet of Important Things

MARTEN LOHSTROH[1], (Member, IEEE), HOKEUN KIM[1], JOHN C. EIDSON[1], (Life Fellow, IEEE), CHADLIA JERAD[2], BETH OSYK[3], AND EDWARD A. LEE[1], (Fellow, IEEE)

[1]Department of Computer Sciences and Electrical Engineering, University of California at Berkeley, Berkeley, CA 94720, USA
[2]National School of Computer Sciences, University of Manouba, Manouba 2010, Tunisia
[3]Edge Case Research, Pittsburgh, PA 15201, USA

Corresponding author: Marten Lohstroh (marten@eecs.berkeley.edu)

This work was supported in part by the National Science Foundation (NSF), under Award CNS-1836601 (Reconciling Safety with the Internet), in part by the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by Avast, Camozzi Industries, Denso, Ford, Siemens, and Toyota), and in part by the Fulbright Scholar Program, a program of the United States Department of State Bureau of Educational and Cultural Affairs.

ABSTRACT The Internet of Things leverages Internet technology in cyber-physical systems (CPSs), but the protocols and principles of the Internet were designed for interacting with information systems, not cyber-physical systems. For one, timeliness is not a factor in any widespread Internet technology, with quality-of-service features having been routinely omitted for decades. In addition, for things, safety, freedom from physical harm, is even more important than information security, the focus on the Internet. Nevertheless, properties of the Internet are valuable in CPSs, including a global namespace, reliable (eventual) delivery of messages, end-to-end security through asymmetric encryption, certificate-based authentication, and the ability to aggregate data from a multiplicity of sources in the cloud. This paper discusses and surveys architectural approaches, communication protocols, and programming models that promise to bridge the gap, enabling the use of the Internet technologies even in safety-critical, cyber-physical applications such as factory automation and transportation. Specifically, we argue that smart gateways hosted on edge computers complement cloud-based services; they can provide tighter control over timing and security that is robust against network outages, play an active role in managing interactions between things, and isolate safety-critical services from best-effort services. We explain how time sensitive network technology can be leveraged to reliably orchestrate a multiplicity of things, and how augmenting our programming models with a well-defined notion of time can make systems more deterministic and more testable.

INDEX TERMS Cyber-physical systems, edge computing, fault tolerance, Internet of Things, real-time systems, software safety, security, time dissemination.

I. INTRODUCTION

The Internet of Things (IoT) is the class of cyber-physical systems (CPSs) that leverage internet technology for coordination and for the exchange of information about the physical world in real time. The vision embodied by IoT appeals to the imagination of many—our environment and virtually anything in it will turn ''smart'' by having otherwise ordinary things be furnished with sensors, actuators, and networking capability, so that we can patch these things together and have them be orchestrated by sophisticated feedback and control mechanisms. As Wegner argued in [1], *interaction* opens up limitless possibilities for Things to harness their environment and compensate for a lack of self-sufficient cleverness. Sensors aside, a connection to the Internet alone allows a Thing to tap into an exceedingly rich environment, unleashing a real potential for making things smarter.

However, today's IoT solutions are often plagued by problems. Some prove considerably more awkward and inefficient to use compared to the conventional technology that they replace. Setup can be complex and firmware patches sometimes fail. The company providing the cloud side of the service may go out of business or abandon the product, rendering it useless. IoT devices sometimes intend to

The associate editor coordinating the review of this manuscript and approving it for publication was Remigiusz Wisniewski.

replace things that are designed to last decades, but the prospect of these devices working reliably for decades is often remote.

Another problem is that IoT devices often lack adequate security. Particularly manufacturers of seemingly innocuous gadgets tend to prioritize time-to-market and their products' ease of use over steadfast security. Arguably, security may not be a big concern for IoT devices that perform non-critical tasks, but their vulnerabilities can be leveraged to attack higher-value targets or paralyze important network infrastructure. In 2016, for instance, the Mirai botnet, which consisted of compromised IoT devices that allowed access on the basis of default passwords, carried out a distributed denial-of-service (DDoS) attack on Dyn, a major provider of the Internet's domain-name service (DNS). This led to internet connection problems to major websites including Twitter, Netflix, Spotify, and the Financial Times [2].

In this paper, we focus on Things where dependability and safety are extremely important, such as factory robots, trains, and cars. Traditional methods used in engineering to achieve high reliability (e.g., via testing or formal verification) rely on isolation to avoid interference and make the analysis of sought-after safety properties tractable. Hence, the idea of integrating internet technology in such systems is quite a departure from that tradition and poses a formidable challenge. In this paper, we call such safety-critical cyber-physical systems the Internet of *Important* Things, IoIT. The community also refers to the Industrial Internet, Industry 4.0, and Digitalization, each of which we view as a special case of IoIT.

Despite the challenges to safety, networking offers considerable advantages. The proper functioning of a safety-critical CPS relies on the accuracy and dependability of the information it is able to source from its environment. With an increasing degree of autonomy, this reliance becomes more crucial; without a human in the loop, it becomes difficult to avert disaster in the face of errors. Networking can help compensate for a local deficiency of accurate information. For instance, a car equipped with radar or LIDAR cannot identify obstacles past the car in front of it, but it could potentially ask the view-blocking vehicle to report what is in its view.

Ensuring adequate safety, reliability, privacy, and security for the IoIT operating on open networks is extremely difficult. There is precedent, however, for high-confidence systems that use open networks. Today, the world's financial system operates almost entirely electronically and with heavy use of the open internet. No engineered system is perfect, but the benefits appear to outweigh the risks, and losses due to technical failures and malicious actors are simply factored into the cost of operation. Can cyber-physical systems achieve the same balance, where the benefits of open networks outweigh the costs? In the remainder of this paper we discuss and survey several technological innovations that we believe will be instrumental in the development of a safe, secure, and reliable IoIT.

II. EDGE COMPUTING

Building IoT systems using cloud computing has been widely adopted [3], [4]. Services such as Amazon's AWS IoT [5] are primarily cloud services, in this case using Amazon Web Services (AWS). An alternative that we view more promising is to mix cloud services, edge computing, and Things, rather than just cloud services and Things. Edge computing, on the other hand, is an emerging computing paradigm that puts services on devices that are physically much closer to the Things, residing in smart gateways or in networking equipment [6]. Cisco Systems coined the term "fog computing" [7] for such architectures to suggest that it is kind of like the Cloud, but closer to the ground. We will refer to this architecture more generically as "edge computing."

An edge computer is a computing device that can act as an internet gateway or a router, such as Intel's IoT gateway or the SwarmBox developed in the TerraSwarm project.[1] But rather than being just a provider of networking, edge computers also provide services, including, for example, monitoring sensors for anomalies, brokering authentication and authorization [8], filtering data feeds to ensure privacy, and preprocessing data feeds to forward to the Cloud only what the Cloud needs.

Any device with computing and networking capability can function as an edge computer, so it is the role of the device that we focus on. For instance, a smart phone is a *Thing* when used for its sensors or actuators. At the same time, it is an *edge computer* when it acts as an internet gateway for other Things such as wearable devices. A device can perform both roles simultaneously.

Varghese *et al.* [9] classify edge computers as either an "edge node," edge computers running on traditional internet routers, and an "edge device" for edge computers based on user mobile devices including smart phones and laptops. Since "node" and "device" are almost synonymous, we distinguish "immobile" edge computers from "mobile" edge computers. To be an edge computer, it must interact with Things, and mobility is determined by whether it moves with respect to the Things it interacts with. For example, a mobile edge computer can be inside of a car; the car itself is mobile, but the edge computer is stationary relative to Things on the car connected to it. Similarly, mobile phones are typically bound to humans, and they follow their owners around, remaining stationary from the perspective of their owner's wearables. But they are mobile from the perspective of smart building devices, and thus less effective as an edge computer for them.

A defining feature of an edge computer is its physical proximity to the devices it serves, making it much easier to ensure stable network connections that allow for reliable low-latency communications. In addition, edge computers can leverage their locality to keep data local (for privacy and security); to offload computation from battery-powered devices; to provide temporary storage for memory-constrained devices; to firewall a local network; and to authorize or discover devices

[1] https://www.terraswarm.org/

based on physical proximity. The Cloud, on the other hand, is a better choice for services that require aggregating data from multiple sources or that exceed the compute and/or memory capabilities of edge computers.

We believe that many IoIT applications can benefit from both edge and cloud capabilities, and that services should be carefully partitioned between the Edge and the Cloud based on their requirements. In the IoIT, some services are critical to safety and cannot be beholden to remote services with highly variable latencies and potential network and service outages. Imagine a scenario where a power plant operator is locked out of the local network because a remote authentication service goes offline. A recent example showing that this can happen is the Google OnHub incident [10]. On February 23, 2017, many of Google's smart gateway (router) devices called *OnHub* were unable to authenticate its users for about 45 minutes due to a failure in its remote authentication servers, denying access to any OnHub devices. Users could not even access local resources like printers on their LAN. Edge computers with adequate local authentication and authorization services can avoid this scenario, as shown in [8].

III. SECURITY

Edge computing offers opportunities for enhancing security. An edge computer can mediate access to physical resources such as sensors and actuators, it can isolate a local network from the Internet, and it can authenticate Things. But how? How can it be sure that the data it sees comes from the sensor it expects, for example? And how can it authorize access to resources such as a safety-critical actuator? The IoT has a much wider attack surface compared to traditional computer systems, making it easier to subvert. For example, disruptions of timing that fall far short of denial-of-service attacks can have disastrous consequences. IoT systems may also be vulnerable to energy attacks (draining the battery of a critical device) and physical attacks (tampering with hardware or fabricating sensor input). And wireless communication, which IoT devices often rely upon, is inherently more susceptible to tampering than wired communications.

In the Internet today, authentication infrastructure has become good enough that we conduct our banking online. A public key infrastructure (PKI) specifies policies and procedures of issuing and managing (digital) certificates for authentication of network entities including web servers. For example, given a domain name of a bank (e.g., `bankofamerica.com`) and a certificate provided by a web server of the bank, the browser in our laptop computer can verify that the private key that matches a public key included in that certificate is owned by the same entity that owns the domain name. The browser verifies this by checking a cryptographically signed string of bytes in the certificate with a public key provided by an independent trusted third party, a certificate authority (e.g., VeriSign). However, can this PKI be compromised? Of course, it can; someone could steal the private key from the bank and spoof a domain-name server to provide an alternative IP address for the domain name. A certificate authority could also make mistakes, as shown in the WoSign incident that occurred in 2015 [11], where the Chinese certificate authority WoSign mistakenly issued certificates to incorrect subjects; for instance, if you controlled `foo.github.com`, WoSign would issue a certificate for `*.github.com`. This incident showed that security can collapse when a root of trust, a certificate authority in this example, is broken. Nevertheless, the PKI works well enough that you trust the system with your money.

The Internet and cellular telephony provide reasonably reliable addressing. Specifically, when you launch a packet into the network with a destination address (or phone number), you have some assurance that it will be delivered to the device or person legitimately associated with that address. Again, this is not perfect, to compromise the infrastructure, a hacker would have to break into routers owned by large corporations (internet service providers and cellular telephone providers) whom we trust to take measures to protect their equipment [12]. Many two-factor authentication mechanisms today rely on this reliable addressing, to reset passwords by email delivery, for example.

Many web applications, including online banking, rely on sandboxed execution of proxy code. Specifically, when you direct your browser to your bank's website, the website typically provides your browser with a script, typically written in JavaScript, that executes within the browser under an interpreter that restricts what the script can do. Specifically, the script can access other web pages, but only those from the same source, specified by a URL and port number (''same-origin'' policy), and the script can neither access files on the local disk nor data in local memory belonging to other web pages or programs. These restrictions are enforced by the browser, which you must trust. The ''accessor'' design pattern we discuss in [13] provides similar sandboxing for IoIT applications.

A. AUTHENTICATION AND AUTHORIZATION

The IoT can certainly leverage web techniques for authentication, but while they are sufficient to enable online banking, they are not sufficient for most IoIT applications. One issue is scalability. IoT devices provide services accessible from the network, but they will rarely be associated with a fixed domain name or IP address (although as IPv6 gets more pervasive, they may start to be associated with fixed IP addresses). This creates several problems. One is discovery, because you cannot simply go to `myassemblyrobot.com` to establish a connection with your factory's assembly robot. Second is authentication, because it is difficult to verify that the IP address you connect to indeed points to your robot, as opposed to a device used to stage a man-in-middle attack, for instance. Certificate authorities, as they are operated today, do not scale to the IoT, because the issuance of certificates requires human assessment and approval for each request. While this process may be tenable for the authentication of organizations, there are

many orders of magnitude fewer of those than the number of IoT devices we expect to come online.

Innovative projects like Let's Encrypt[2] provide a major step forward, because the authentication and certificate issuance are free and automated. So, without human intervention, it can scale to many more entities than traditional certificate authorities could possibly serve. However, it is still globally centralized, meaning that it can be a single point of failure, as shown in the Heartbleed vulnerability in 2014 [14] where more than 200,000 certificates from a single certificate authority, GoDaddy, had to be revoked due to a security breach in OpenSSL. Let's Encrypt, however, still requires fixed and global network addresses for certificate issuance, making it a poor match for IoT devices that operate on LANs and/or are assigned a dynamic IP address. Recent work in our group has shown that the principles of Let's Encrypt can be realized in a more robust, scalable way, creating what we call ''locally centralized, globally distributed authentication'' [15]. Exploiting locality can also make such systems more resilient to distributed denial of service attacks.

A problem related to but distinct from authentication, is authorization. You would not want anyone on the Internet who discovers your assembly robot to be able to control it. For human users, passwords are often used for both authentication and authorization. However, it is neither feasible for a human to memorize passwords for a myriad of IoT devices nor is it secure to use passwords that could be guessed or stolen by adversaries. A locally centralized, globally distributed architecture can provide automated and easy-to-manage authorization mechanisms based on physical proximity. In this security model, we assume that a device that can get physically close to another device can be granted access to services. Many applications (such as factory automation) presume constrained physical access (for safety reasons if no other), so these constraints can be leveraged for authorization. For applications that have no constrained physical access, such as roadside infrastructure, other mechanisms will need to be developed.

B. AVAILABILITY THREATS AND RESILIENCY

Availability threats to the IoIT can lead to devastating consequences. For traditional internet services, the damage of distributed denial-of-service (DDoS) attacks will be at most financial loss or privacy breach. However, DDoS attacks on the IoIT, including safety-critical facilities such as hospitals, may result in casualties. DDoS attacks on Boston Children's Hospital [16] in April 2014 disrupted the IT systems of the hospital. Had medical devices been connected to the facility's network (this practice is already becoming a trend [17]) the attacks could have caused medical accidents. Another example is the DDoS attack on building controllers in Finland [18] in the winter of 2016, which raised safety concerns due to a malfunction in buildings' heating systems. Moreover, the frequency and volume of DDoS attacks have been rapidly increasing due to the scale of the IoT [19]. For the IoIT, it is crucial that critical security functions such as authentication and authorization services are resilient to availability threats including DDoS attacks.

Our recent work [20] provides locally centralized, globally distributed authentication and authorization services that are resistant to availability threats and are able to recover from failures. In our proposed approach, authentication and authorization are performed by an entity called *Auth* [21], which is deployed across edge computers and does not rely on remote cloud servers and internet connections. In addition, our approach facilitates recovery in case of availability attacks or failures through a mechanism called *secure migration*. The secure migration technique allows Auths to construct migration policies and back up its Things' credentials to other trusted Auths. When some of Auths become unavailable, their Things can securely migrate to other available Auths and continue authentication and authorization services.

We have focused on reasonably well developed and understood security mechanisms, but there are some emerging technologies that may prove useful for the IoIT. Homomorphic encryption, for example, allows certain operations on encrypted messages without decrypting them. Blockchain techniques go further than edge computing in decentralizing critical functions, making them completely distributed. Neither of these is mature enough for us to see exactly what their impact might be, but they are clearly worth watching.

IV. COORDINATION AND TIMING

Specific internet technologies of interest for IoT include networking (e.g. IP, TCP, UDP), the World Wide Web (e.g. HTTP, HTML5), cloud computing, and programming languages for client and server-side functionality. But these technologies were not designed for interaction with Things. An egregious mismatch concerns timing. Timing of internet technologies is strictly a ''best effort'' affair where the goal is simply to be sufficiently responsive that humans do not lose patience. But when talking about Things, timing can matter quite a lot. It matters when a self-driving car applies the brakes or a robot arm on a factory floor moves.

Some kinds of real-time behavior are not realistically achievable with today's internet technology. For example, it is unlikely that the feedback control laws governing a self-driving car can be realized in the Cloud using RESTful interfaces [22], which rely on HTTP and carry all context state information in each exchange of information. The latency of responses from the cloud-based service is likely to be too high and, more important, too variable.

There are market forces that are driving internet technology towards more controllable latencies. Interactive services such as distributed gaming and video teleconferencing demand controllable latencies and are difficult to achieve today with high quality and reliability. To address this demand, a recent industry trend is the emergence of Time Sensitive Networking (TSN) technology. The Time-Sensitive Networking task group of the IEEE 802.1 working group is in the final stages

[2]https://letsencrypt.org/

of issuing standards with promising new capabilities that are compatible with open networks.

Some elements of TSN technologies are already widely deployed, though not yet widely used. For example, the IEEE 1588 [23] standard for clock synchronization, which first appeared in 2002 and was substantially revised in 2008, is supported in essentially all Ethernet PHY chips on the market today. It has been deployed in quite a lot of networking gear, but it has not yet risen to the level of a service for application developers except in niche applications [24]. This technology is capable of synchronizing clocks on a local-area network to nanosecond precision, and it is compatible with legacy Ethernet and TCP/IP equipment. The TSN standards that are expected this year, particularly updates of the 802.1AS standard, could be the galvanizing force that will lead to worldwide high-precision clock synchronization. If we combine this technology with edge computers, which can function as gateways that ensure controlled timing on local area networks, then networks with deterministic latencies and reliable delivery that are compatible with the Internet are within reach. We assume in this paper that such networks will become widely deployed.

Even if these market forces are wildly successful, the Internet is unlikely to ever provide the level of determinism needed for many safety-critical, latency-sensitive services. Its most important defining features are openness, which inevitably will increase variability in quality of service, and its geographical distribution, which inevitably increases latencies. Nevertheless, some temporal properties *are* achievable even with today's internet technology. Some applications can immediately benefit from these properties by becoming more deterministic, more testable, and better able to detect and adapt to failures or degradations of network services. And then as the network infrastructure improves, we will become able to exploit these same temporal properties and more to deliver innovative services.

IoT applications are distributed and communicate over networks, and hence we will require some temporal semantics in the network as well. Fortunately, we can draw on experience and considerable infrastructure developed in conjunction with telecommunications, military, navigation, and industrial automation. However, there are three issues that must be considered and resolved in order to proceed with confidence:

- How can measurements of physical time be coordinated to sufficient accuracy across networks?
- Are the coordination mechanisms and protocols sufficiently robust?
- Can timing aspects of delivery of messages over the Internet be sufficiently improved to enable the IoIT to meet application demands?

A. CLOCK SYNCHRONIZATION

Clock synchronization mechanisms combine local clocks with protocols for aligning local clocks over a network. A local clock is a combination of an oscillator and a counter to count elapsed oscillator cycles since some agreed upon epoch. The required stability and accuracy of the local clock depends on the needs of the applications and the properties of the network. For example, if frequent corrective alignment with a remote stable master clock can be assured, then the local clock can be simple and inexpensive. If on the other hand accurate timing has to be assured over long periods of network outages, then a more expensive local clock may be required.

Fortunately, very stable and accurate clocks have become available at reasonable cost. The workhorse clock of many critical industrial applications is the 5071 cesium atomic clock which has an inherent accuracy of 5×10^{-13} seconds and a long term stability of 1×10^{-14}. These specifications should cover the vast majority of IoIT applications. Such a clock could be provided as part of the edge computing infrastructure, thereby enabling highly accurate timekeeping of many IoIT devices on a local network even if their own local clocks are relatively sloppy. If needed, there are clocks that are at least an order of magnitude better, but they are costly, heavy, and require mains power. Much less expensive oscillators based on quartz crystals or MEMS technology are available with stabilities of 0.5 to a few PPM in a cost and power range accessible to all but the most power challenged IoIT devices.

All clocks are sensitive to some degree to environmental effects, the most important of which is temperature, but include others such as pressure, gravity, and mechanical stress. In certain applications, relativistic effects must be accounted for, e.g. due to elevation and velocity. Nevertheless, the availability of suitable clocks is not likely to be the limiting factor in IoIT applications, particularly because network clock synchronization offers flexibility to trade off clock characteristics, synchronization rates, and synchronization accuracy.

Most applications only require time to be consistent among collaborating devices, for example, within a vehicle, machine, or factory. There are several time transfer protocols suitable for such cases, especially if all devices communicate on the same LAN. Other applications, particularly those spanning wide geographic areas, may also require a timescale consistent with international standards such as UTC or TAI. Again, there are suitable time transfer protocols.

Another common requirement is a federated model with consistent and very precise synchronization within localized collections but with a method of relating local time to international standards among collections, perhaps with reduced accuracy and precision. Note that some applications, particularly in telecommunications, require only a common frequency, i.e. syntonization rather than common time. Depending on the accuracy required, it may be possible to meet such a requirement without the use of a time transfer protocol, instead depending on the inherent frequency accuracy of the clocks involved.

Following is a synopsis of commonly used time transfer protocols suitable for use in IoIT applications:

- GNSS, e.g. GPS, Glonass: Global Navigation Satellite Systems are world-wide time transfer mechanisms maintained by political entities, e.g. the US DoD for GPS, capable of delivering time accuracies of better than a microsecond with little effort and 50 ns with considerably more effort. GNSS operates using a system of satellites each containing an atomic clock and transmitting a frequency signal and coded events to enable terrestrial receivers to compute their location and time. Many GNSS receiver components are available with a range of performance, cost, weight, and power consumption properties.
- Network Time Protocol (NTP): NTP is an internet-wide protocol capable of accuracies to the few millisecond level. The timescale is tied to UTC and is based on a collective agreement among primary NTP servers which in turn can transfer time to clients using a simpler protocol Simple Network Time Protocol (SNTP). SNTP runs on many laptops, workstations, and servers to enable the time-stamping services used by file systems and operating systems.
- IEEE 1588 (also known as the Precision Time Protocol or PTP): Although designed as a LAN protocol, it is widely used in WAN environments by the telecommunications industry. In a LAN environment, PTP enables sub-microsecond synchronization, and with care, perhaps 10 ns. Recent improvements incorporating the White Rabbit technology developed at CERN allow PTP to provide time transfer accuracy to 100ps with 10ps jitter over 40 km of optical fiber [25]. The timescale can either be local or tied to TAI. In the latter case, PTP distributes information to enable the calculation of UTC. Nearly all recent Ethernet PHY chips contain time-stamping hardware to assist PTP or other protocols in capturing very accurate timestamps of synchronization messages. There is considerable hardware (including 802 bridges), software, and consulting available to support PTP.
- IEEE 802.1AS: This is a variant of IEEE 1588 specialized for layer 2 LAN environments and for use in the larger TSN suite of protocols being developed in the 802.1 standards groups. It has the same capabilities as PTP in a LAN environment.
- Industrial protocols including for example TTE (time-triggered ethernet) and PROFINET: Most such protocols originally ran only on proprietary networks but currently have been modified to run using Ethernet technology, though not necessarily seamlessly with conventional 802 networks. Accuracies below a microsecond are the norm.

Although not widely implemented at present, eLoran or ''Enhanced Loran'' is potentially a world-wide time transfer mechanism. Currently it is operational only in parts of the UK but is being considered for wider installation by several political entities. eLoran operates using a concept similar to GNSS but with ground transmitters with considerably higher power. The original purpose of Loran was as a navigation aid in coastal waters.

With few exceptions, the power requirements of all of the above time transfer mechanisms rule out straightforward application in power challenged devices, e.g. battery operated sensors that periodically wake up to transmit and receive a few bytes of information. There is considerable literature on the design of wireless sensor networks covering suitable clock technology, energy harvesting schemes, and specialized time transfer protocols. One widely used mechanism is codified in the IEEE802.15.4e standard, which synchronizes nodes in a wireless network to within tens of microseconds. This synchronization is used to coordinate wakeup times in an ad-hoc network and for Time Synchronized Channel Hopping (TSCH) to provide more robust wireless communications. A more extreme example of what can be done is the Seiko ''Astron'' watch. These watches harvest solar energy for power, periodically synchronize to GPS, and are specified to maintain ± 15 seconds of accuracy for a month (about 6 PPM) without access to GPS and over a temperature range of 5 to 35 degrees Celsius.

Because of the richness of these available options and the fact that they are being widely deployed, the availability of a time transfer mechanisms is not likely to be the limiting factor in providing temporal semantics in IoIT systems. However, the robustness of such mechanisms is a serious issue that must be addressed.

B. ROBUST TIME COORDINATION

Designing a robust time delivery mechanism is not completely orthogonal to the robust delivery of data and commands in an IoIT system. However, there are considerations unique to robust time delivery that require modification of or in some cases new remedies compared to techniques for ensuring robust computation or data delivery. As usual, the answer to such problems is application and environment dependent. For example, the Seiko watch shows that in a favorable environment, e.g. ones' wrist and dresser top, and with modest accuracy requirements (e.g., 15 seconds), the application of the well-worn technique of ''holdover'' is adequate. Holdover relies on the stability of a local oscillator in the absence of access to the primary time reference, GPS in the case of the wristwatch.

The initial consideration is the required accuracy and robustness of the reference time for the system. For systems of large spatial extent or if time traceable to international standards is required, a world-wide distribution of time from a national laboratory is the common system. GNSS is by far the dominant technology, but it is quite vulnerable to natural interference, such as solar flares, jamming, and spoofing. Sadly, these later are common occurrences [26]. There is considerable current activity in developing techniques to protect GNSS receivers, but particularly with jamming, the bad guys hold the upper hand. If such interference can be detected, local clocks can operate in holdover mode or switch to another source of time. Protection is also enhanced by

comparing redundant sources of time. Of course, these techniques add considerable cost and complication and have limited accuracy. If the world's political entities can be persuaded to implement eLoran, this will help immensely because eLoran signals are orders of magnitude greater and of lower frequency than GNSS and therefore more difficult to jam, and in addition are accessible indoors where GNSS is problematic.

For systems where only local self-consistent time is required, the other protocols listed earlier are often used. The time sources at a moment in time are either a single clock, in the case of protocols like PTP, or an ensemble of clocks in protocols like NTP. Ensemble protocols typically exclude the outlier clocks, which reduces the danger from a failing or severely out of specification clock. Protocols like PTP select the best clock currently in the system (by some criteria) and, if this clock degrades, fails, or is disconnected, will select the next best clock, again providing a measure of robustness. Both cases are examples of using protocols to protect against certain clock and network failures. As is the case of GNSS, it is common to combine redundant time sources with holdover. For example, the telecommunications industry invariably designs their primary and secondary references to have good holdover properties to protect against inevitable network outages.

Since all time distribution techniques of interest to the IoIT will involve network communication, data corruption, denial of service, etc. are of concern. But the cryptographic techniques used to protect data transfer are not sufficient to protect time transfer. Such techniques can verify that information from a particular clock is actually from that clock (authentication), that a particular clock is actually authorized to participate in the protocol (authorization), and to verify that the timing data has not been modified in transit (integrity). But in the case of time distribution, the propagation latency must be known or measured to correctly compute the time. Intentional or unintentional modification of the latency is not detectable by the protocols themselves and is a source of vulnerability. For example, the mistaken measurements that neutrinos travel faster than light in 2011 by the OPERA experiment were traced to a faulty connector which changed the electrical length of a cable carrying synchronization measurements. This type of attack or failure is particularly difficult to detect or prevent. The best solution to date is the comparison with redundant time distribution channels using, for example, triple modular redundancy.

To the extent possible, it is essential that applications themselves be designed to be robust in the face of time errors. For example, the use of a design and execution environment such as Ptides [27] can potentially enable the prediction of a future timing failure or detection that a timing failure has occurred, thus allowing programmers to take remedial measures.

C. TIMELINESS OF MESSAGE DELIVERY

Clock synchronization accuracy depends on the variability of the latency of delivery of synchronization messages. The higher the variability, the coarser the accuracy. IoIT applications with closed-loop physical control also depend on the latency of message delivery. Stability of feedback control systems is harder to ensure when latency is higher and when latency is highly variable. There are trends in the Internet that affect such latency.

The first is the introduction of software defined networking (SDN). SDN separates the data and control planes in a network to provide greater and quicker capabilities to manage traffic flow to maximize capacity. To date there is not much experience on what this will mean for accurate time distribution, but if not done carefully, SDN will surely degrade accuracy. For interesting discussions of the possible relationship of SDN to network timing, see Stein [28] and Mizrahi and Moses [29]. SDN will result in much shorter duration of routing patterns, and this will be enhanced by the ability to utilize a system-wide timescale to coordinate the exchange of one routing pattern to another.

A second trend is the ongoing TSN standardization effort in the IEEE 802.1 standards committees designed to control latency of message delivery in a LAN environment. As noted, TSN depends on having a robust notion of LAN-wide time. The IEEE 802.1AS protocol, a variant of IEEE 1588, serves this purpose. TSN is in part an adaption of techniques long used in the industrial automation community. It involves centralized allocation of bandwidth, and time-slots to privileged applications, while the remaining bandwidth is used for general traffic. TSN potentially can provide much better bounds on latency and more robust time distribution services. However, applications must be adapted to limit communication to assigned time-slots. This clearly has limits of scale and in any case cannot overcome the limits imposed by the available bandwidth.

V. PROGRAMMING MODELS

A recent trend in cloud computing is to focus on *real-time* data analytics. The emerging IoT promises a flood of sensor data that many organizations already are collecting but not effectively using. Consulting and market research company Gartner calls ''dark data'' the ''information assets that organizations collect, process and store in the course of their regular business activity, but generally fail to use for other purposes.'' The subtext is that those same businesses are missing an opportunity. They should be mining the data. The data has value. The research and consulting firm Forrester defines ''perishable insights'' as ''urgent business situations (risks and opportunities) that firms can only detect and act on at a moment's notice.'' Fraud detection for credit cards is one example of such perishable insights. This has a real-time constraint in the sense that once a fraudulent transaction is allowed, the damage is done. In CPSs, a perishable insight may be, for example, a determination of whether to apply the brakes on a car, where a wrong or late decision can be quite destructive.

Real-time data analytics implies both timing constraints and streaming data. Computing on streaming data fundamentally means that you don't have all the data, but you have to

deliver results. It differs from standard computation in that the data sets are unbounded, not just big, and you can't do random access on input data, which constrains the types of algorithms you can use. Major research efforts, such as the industry-funded RISELab (Real-time Intelligent Secure Explainable Systems[3]) launched at Berkeley in 2016, are getting a lot of attention. Examples of algorithmic innovations for real-time streaming data include adaptations of machine learning and optimization algorithms [30], [31] and adaptations of formal methods [32] to operate on streams.

Real-time data analytics benefits from being hosted in the Cloud because it benefits from aggregating data from multiple sources. Machine learning algorithms, for example, get better given more data. Edge computers can benefit from a partnership with cloud services if the learning that is done in the Cloud can be transported to the Edge. A hybrid edge/cloud architecture can, for example, learn parameters for a model in the Cloud, and use those parameters to provide quick local responses at the Edge. In such an approach, even time-critical services can benefit from big-data machine learning in the cloud.

But even at the Edge, providing ''quick local responses'' remains challenging. While edge computers can, in principle, provide lower and less variable latencies than cloud computers, software and networking support for controlling timing remains weak [33]. It is difficult, for example, to ensure temporal isolation between services, where one service cannot disrupt the timing of another. The most common solution to this problem, using priorities, is inadequate for IoIT because of the highly dynamic nature of the applications. Priorities work best in situations where the task set competing for resources is well understood *a priori*. Edge computers, unlike embedded computers, will not be programmed once and then deployed to operate for years with a fixed program. Instead, they will be service providers for Things that come and go. They will need to perform admission control, rejecting a new Thing if providing service to it will disrupt the timing of a preexisting service. But this requires developing quite a bit of new technology. It does not fit well the classical task models of real-time computing.

A. ASYNCHRONOUS ATOMIC CALLBACKS

Since an edge computer will be interacting with both Things and cloud-based services, it will need to use the mechanisms and APIs that those Things and services define. It is very common today for both Things and services to provide APIs (endpoints that respond to HTTP or MQTT commands), and the prevailing mechanism for interacting with them is via a concurrency pattern that we call *asynchronous atomic callbacks* (AAC). AAC is very different from the most common task models for real-time computing. In AAC, when a service request is made, the requester does not block to wait for an answer but instead provides a callback function to be invoked asynchronously when the service has been completed. These callbacks are invoked atomically in that each callback executes to completion before any other callback begins executing (or at least, it must appear that this is the case). This atomicity distinguishes the AAC concurrency model from interrupt-driven I/O, threads, and many asynchronous remote procedure call mechanisms. The same benefits can, in principle, be accomplished with threads, but the resulting programs are much less scalable, more difficult to understand, and vulnerable to the many nefarious bugs that multithreaded programs inevitably have [34].

The AAC pattern is central to the JavaScript programming language, used extensively in web programming, both on the server side (using for example Node.js[4]) and in browsers. AAC is also central to Vert.x,[5] a Java-based framework. Vert.x supports islands of atomicity called ''verticles'' (think ''particles'') where callbacks are atomic with respect to all other callbacks in the same verticle, but verticles can execute in parallel. Vert.x therefore scales to very large numbers of servers. To provide a sound concurrency model, communication between verticles is restricted to only occur via a publish-and-subscribe data bus or using immutable data structures. AAC has also been used in much older frameworks such as Active Messages [35] and in embedded systems, particularly in TinyOS [36].

The AAC pattern is almost universally used when making a request for data from a URL on the Internet. By providing a callback function instead of blocking to wait for a response, an application avoids becoming unresponsive during the long and highly variable latencies of an HTTP request on the Internet.

AAC comes with costs, however. First, it becomes essential to write code carefully to consist only of quick, small function invocations. A long-running function will block other callback functions, reducing the responsiveness of applications. Second, AAC accentuates the chaos of asynchrony, where achieving coordinated action can become challenging. For example, if you make multiple requests in sequence to a service, each time passing a callback function, there is no assurance that the callbacks will be invoked in the same order as the requests. Both problems are important for IoIT, where heavy computation may be required to analyze sensor data, and coordinated physical actions may be dependent on the order in which things occur.

Because of these limitations, several alternatives mix AAC with other concurrency models. Many JavaScript implementations realize a thread-like mechanism called a Web Worker, which runs tasks in the background concurrently with the main AAC function invocations. Unlike threads, these Web Workers cannot share data with the main application. Instead, they send messages to the main application, which, if it is listening, will invoke a callback to handle the message. ECMAScript 6, a recent version of JavaScript, enriches AAC with a cooperative multitasking model, which allows a

[3]https://rise.cs.berkeley.edu/

[4]http://nodejs.org/

[5]http://vertx.io/

function to suspend execution at well-defined points, allowing other functions to be invoked while it waits for some event. The Vert.x framework enriches AAC with concurrent verticles that interact with one another through a publish-and-subscribe bus. Calvin [37], Node-RED,[6] and NoFlo[7] use a dataflow concurrency model for interactions between services that are using AAC.

Our own IoT composition platform, CapeCode [13], uses the discrete events simulation engine from Ptolemy II [38] to coordinate the interaction between components that are written in JavaScript. The discrete events semantics makes system behaviors repeatable and testable, and consequently makes the interactions between cyber services and Things more reliable than is possible with JavaScript's non-deterministic concurrency model, which uses an event loop to schedule the execution of AACs [39]. A formal description of this particular embedding of AACs in a discrete event system is given in [40].

But even using pure JavaScript it is possible to mitigate the non-determinism of the event loop by enforcing a temporal semantics; in [41], implementations (and extensions) of standard JavaScript functions `setTimeout` and `setInterval` are given, customized so that they schedule periodic events according to a discrete events semantics, but without the help of an external coordinator such as CapeCode. This temporal semantics includes a notion of simultaneity, enforcement of causal data dependencies, and logical timelines that can be bound to real time (approximately) to achieve real-time behaviors with much better determinism than typical best-effort methods. But before we explain this, it is helpful to understand what we mean by "real time."

B. WHAT IS REAL TIME?

In practice, when engineers talk about "real time," they may mean[8]

1. fast computation,
2. prioritized scheduling,
3. computation on streaming data,
4. bounded execution time,
5. temporal semantics in programs, or
6. temporal semantics in networks.

These are very different views, and which view dominates has a strong effect on the choice of technical approaches to the problem.

The first, fast computation, is useful in all computation, and does not deserve our attention here. Nothing about fast computation distinguishes real-time problems from non-real-time problems. In fact, many real-time systems execute on decidedly slow computers, such as microcontrollers, and timing precision, predictability, and repeatability is more important than speed.

[6]https://nodered.org/

[7]https://noflojs.org/

[8]In 1988, Stankovic cataloged quite a few more possible (mis)interpretations of the term "real time" and laid out a research agenda that is dishearteningly valid today [42].

The second, prioritized scheduling, is the centerpiece of much work in the real-time systems community [43]. In this approach, the requirements of the physical world are reduced to deadlines and periods for periodic tasks, and the temporal properties of software are reduced to execution times for tasks. Prioritized scheduling is often paired with a predefined global periodic task schedule to remove the possibility of starvation for low-priority periodic tasks. Priorities, however, do not work as well when the tasks to be executed are not known about ahead of time. A priority is a global property in that it has meaning only relative to all other priorities in the system. In an edge computer, tasks will depend on Things, mobiles, and internet requests that come and go.

The third meaning, computation on streaming data, demands different software architectures, using for example actors as components rather than objects, threads, or tasks. Many subtle timing questions arise. For example, in the absence of time stamps, when streams converge, what is the meaning of the interleaving of their elements? Are their elements simply nondeterministically interleaved, or is there more meaning to the relationship between an element of one stream and that of another? Is there any notion of time associated with the elements of the stream, and what is the semantics of that notion of time?

The fourth meaning, bounded execution time, assumes that some deadline exists for a software execution, and that ensuring that the execution never oversteps that deadline is sufficient. The typical purpose is to meet control system stability requirements, where the generated control signal is only useful for some duration of physical time. This meaning is central to the sense-process-actuate programming models and is usually a basic assumption of the second meaning, prioritized scheduling. But bounding the execution time of software is particularly problematic, particularly on an edge computer that is providing a multiplicity of services. A bound can only be determined for a particular implementation of the computer, where every detail of not only the software, but also the hardware on which it runs and the execution context are known. Edge computers are likely to be high diverse, and services will need to be designed to run on many of them.

The fifth meaning, temporal semantics in programs, has a long history with little practical impact. This history includes programming languages with timing constructs such as Modula [44], [45], PEARL [46], Ada [47], Occam [48], Real-Time Euclid [49], and Erlang [50]. These improve things by including in the language some of the mechanisms of a real-time operating system (RTOS), which means that a model (a program) is more self-contained. One does not need to combine the semantics of the language with the semantics of a separate and distinct RTOS to interpret the model. Few of these languages survive, however, and all fall short of yielding a deterministic modeling paradigm. This is, in part, for very practical reasons. The underlying computers do not provide in their instruction-set architectures mechanisms for controlling timing, except weak and disruptive ones like timer interrupts. In fact, the trend has been relentlessly towards

less predictable and controllable timing. Either the timing specified by programs in such languages will be coarsely approximated by the physical implementation, or the physical implementation will have to be over-provisioned so that the variability it exhibits in timing is sufficiently small.

The final meaning, temporal semantics in networks, is present today only in specialized networks, such as those in safety-critical systems like factory automation, avionics, and automotive. As we explain below in section IV, however, temporal semantics is not entirely incompatible with commodity networks [27].

Providing control over timing in edge computers, we believe, can only be achieved under the last two interpretations, temporal semantics in programs and networks. But what do we mean by ''temporal semantics''? Fundamentally, what we mean is that certain temporal properties should be elevated from *quality metrics* to *correctness criteria*. In programs, we are accustomed to being able to assume that every execution of a program will (with very high probability) perform exactly the logical functions specified by the program. These logical functions are correctness criteria in that failing to perform them correctly is treated as a fault condition. We believe that certain timing properties should similarly be correctness criteria. What timing properties?

Consider a program that wishes to take two distinct orchestrated actions A and B at 100ms intervals. We can argue that it is physically impossible for these actions to be simultaneous, but that would be missing the point. It may be very useful to have these actions be *logically* simultaneous. What does this mean? It could mean that any observer of these actions will at all times have counted the same number of actions A and B that have occurred. That is, if the observer has seen n A actions, then it has also seen n B actions. Note that this requirement is independent of timing precision and is most certainly physically realizable (with certain assumptions about what an ''observer'' is). It gives a clean semantic notion to simultaneity. This is an example of a temporal property that can be a correctness criterion.

A useful semantic notion of time has to provide a clear ordering of events. Specifically, each component in a system must be able to distinguish past, present, and future. The *state* of a component at a ''present'' is a summary of the past, and it contains everything the component needs to react to further stimulus in the future. A component changes state as time advances, and every observer of this component should see state changes in the same order. We also require a semantic notion of time to respect an intuitive notion of causality. If one event A causes another B, then every observer should see A ordered before B. We also require a semantic notion of simultaneity. Under such a notion, two events are simultaneous if all observers see them occurring at the same time. We may also want to avoid models where one observer deems two events to be simultaneous and another does not. These requirements are much easier to meet with an abstract semantic notion of time than one that is more closely tied to physics. But we are interested in cyber-*physical* systems, so we cannot ignore physics.

On the physical side of a CPS, it is natural to be tempted to adopt a notion of time directly from physics. However, this puts us squarely in a minefield, since time is a poorly understood physical phenomenon [51], and models of time differ significantly between Newtonian, quantum, and relativistic physics. However, our goal is not to understand the physics of time, and the usefulness of our models is not determined by how well they match physics. Instead, the usefulness of our models depends on whether we can build physical systems that match the behavior of our models with high confidence [52]. This leads to significantly different choices for modeling time [53].

The most common choice for modeling physical time is Newtonian time. But ironically, Newtonian time proves not so practical for cyber-physical systems. The most obvious reason is that digital computers do not work with real numbers. Computer programs typically approximate real numbers using floating-point numbers, which can create problems. For example, real numbers can be compared for equality (e.g. to define ''simultaneity''), but it rarely makes sense to do so for floating point numbers. In fact, some software bug finders, such as Coverity, report equality tests of floating point numbers as bugs. Moreover, addition of floating point numbers is not associative. This precludes any clean notion of simultaneity. Fortunately, these problems have been solved, and we can simply adopt the quantized superdense model of time that has been shown effective for both *cyber* and *physical* models [54].

C. LOGICAL CLOCKS AND REAL TIME

A semantic notion of simultaneity and control over ordering of events are useful but not sufficient for many IoIT applications. More explicit quantitative control over timing may be required, for example, to specify periodic actions with specified temporal periods or to control latency in a feedback control system. To get approximate timed behavior in an internet setting, most AAC frameworks support *delayed* callbacks. For example, JavaScript environments typically provide a `setInterval`(F, T) function, where function F is to be invoked after T milliseconds and then again periodically with intervals of T milliseconds. Of course, the actual time of the function invocations cannot be *exactly* every T milliseconds, since that would require a perfect timekeeper, which does not exist, and it would require that the JavaScript engine be idle at every multiple of T milliseconds, since the AAC model requires that the function invocation be atomic. We expect (and get) some jitter in the actual timing of the function invocations. Such jitter is unavoidable in any software platform.

But the situation is worse because the time T actually has very little meaning at all. It is at best a suggestive guideline to invoke the function at some time near the multiples of T milliseconds. When there are multiple delayed callbacks, there are no guarantees on the order of invocation of the

callbacks even if the time intervals are identical or related by integer multiples. In the IoT, it is common to build programs that interact periodically with actuators. Since actuators affect the physical world and can do damage, we believe that we need stronger temporal semantics.

Indeed, Jerad and Lee [41] achieve a stronger temporal semantics by defining labeled logical clock domains (LLCDs) within which such time values have specific meanings with respect to one another. For example, two periodic function invocations with the same label and period that are started within the same atomic function invocation will be perceived as simultaneous by all observers. Moreover, the order in which logically simultaneous callback functions are invoked is well defined and deterministic. In addition, periodic actions can join or leave LLCDs dynamically, at the request of some asynchronous callback. With some care, such asynchronously introduced actions can align synchronously with previously specified actions in the same clock domain, thereby supporting a high level of adaptability in a controlled way.

The clocks in each LLCD are *logical* clocks, which means they track logical time, not wall-clock time. Logical time advances in a computational world, in discrete jumps. Physical time, according to Newton's model, advances smoothly and uniformly. An execution platform will make every effort to make logical time track physical time, but perfect tracking is not possible (nor even definable). The stronger semantics of LLCDs, nevertheless, offers a more deterministic concurrency model, and hence IoIT designs will be easier to test and their behaviors will be more predictable. The *correctness* criteria enforced by LLCDs concern atomicity and ordering, not precise timing. For soft real-time applications, the latter can be treated as a matter of *performance*, not correctness.

Of course, for hard real-time applications, such as the control systems in an autonomous vehicle, timely execution *is* a correctness criterion; applying the brakes after a collision has zero utility. This idea is formalized by Time/Utility Functions, originally proposed by Jensen *et al.* [55] and expanded upon by Wu *et al.* [56]. TUFs express constraints on the concurrence between logical and physical time. To achieve precise timing predictibility for time-critical functions, hardware support is required. No other state (e.g., caches) or activity (e.g., other threads, interrupts) must be able to interfere with the execution of a time-critical task, or else its timing could be affected. Full temporal isolation can be achieved with the hard-real-time threads of PRET [57]. Predictable memory access time can be guaranteed through a DRAM controller with hardware-supported command-level prioritization [58].

VI. CONCLUSION

The Internet is highly asynchronous whereas physical Things can be highly timing dependent. If the IoT is to be applied to important and safety-critical applications, this contradiction needs to be reconciled. We have argued that fully embracing edge computing may enable the Internet of Important Things, and we have highlighted some of the problems that need to be solved and some promising partial solutions. (1) We have shown that our work on hosting critical security infrastructure, such as authentication and authorization services, on edge computers can enhance reliability and resilience as well as improve scalability. (2) We have highlighted clock synchronization and time-aware communication protocols as a key enabler for the realization of the Internet of *Important* Things. (3) We have discussed the importance of having a well-defined temporal semantics embedded in the programming languages we use to define system behavior. Ultimately, we envision edge computers to serve as integration platforms that can orchestrate secure communications between Things under a discrete events semantics, allow reliable low-latency connections, and achieve real-time responsiveness when integrated with our hardware-supported approaches.

REFERENCES

[1] P. Wegner, "Why interaction is more powerful than algorithms," *Commun. ACM*, vol. 40, no. 5, pp. 80–91, May 1997.
[2] S. Hilton. (Oct. 2016). *Dyn Analysis Summary of Friday October 21 Attack|Dyn Blog*. Accessed: Jul. 26, 2018. [Online]. Available: http://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/
[3] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "On the integration of cloud computing and Internet of things," in *Proc. Int. Conf. Future Internet Things Cloud*, Aug. 2014, pp. 23–30.
[4] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.
[5] Amazon Web Services. *How the AWS IoT Platform Works—Amazon Web Services*. Accessed: Mar. 21, 2017. [Online]. Available: http://aws.amazon.com/iot-platform/how-it-works/
[6] P. G. Lopez *et al.*, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.
[7] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments* (Studies in Computational Intelligence), N. Bessis and C. Dobre, Eds. Cham, Switzerland: Springer, 2014, no. 546, pp. 169–186.
[8] H. Kim, E. Kang, E. A. Lee, and D. Broman, "A toolkit for construction of authorization service infrastructure for the Internet of things," in *Proc. IEEE/ACM 2nd Int. Conf. Internet-of-Things Design Implement. (IoTDI)*, Apr. 2017, pp. 147–158.
[9] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Nov. 2016, pp. 20–26.
[10] I. Morris. (Feb. 2017). *Google's Latest Failure Shows How Immature its Hardware IS*. [Online]. Available: http://www.forbes.com/sites/ianmorris/2017/02/24/googles-latest-failure-shows-how-immature-its-hardware-is/
[11] L. Tung. (Sep. 2016). *Mozilla to China's WoSign: We'll Kill Firefox Trust in You After Mis-Issued GitHub Certs*. Accessed: Jul. 26, 2018. [Online]. Available: http://www.zdnet.com/article/mozilla-to-chinas-wosign-well-kill-firefox-trust-in-you-after-mis-issued-github-certs/
[12] L. Seltzer. (2013). *BGP Spoofing—Why Nothing on the Internet is Actually Secure*. Accessed: Feb. 6, 2017. [Online]. Available: http://www.zdnet.com/article/bgp-spoofing-why-nothing-on-the-internet-is-actually-secure/
[13] C. Brooks *et al.*, "A component architecture for the Internet of things," *Proc. IEEE*, vol. 106, no. 9, pp. 1527–1542, Sep. 2018.
[14] Z. Durumeric *et al.*, "The matter of heartbleed," in *Proc. Conf. Internet Meas. Conf.*, New York, NY, USA, Nov. 2014, pp. 475–488.
[15] H. Kim and E. A. Lee, "Authentication and authorization for the Internet of things," *IT Prof.*, vol. 19, no. 5, pp. 27–33, Oct. 2017.
[16] Cyber Security Intelligence. (Aug. 2016). *Easy: Hackers Take Down a Hospital*. Accessed: Jul. 26, 2018. [Online]. Available: https://www.cybersecurityintelligence.com/blog/easy-hackers-take-down-a-hospital-1566.html

[17] B. Marr. (Jan. 2018). *Why the Internet of Medical Things (IoMT) Will Start to Transform Healthcare in 2018*. Accessed: Jul. 26, 2018. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2018/01/25/why-the-internet-of-medical-things-iomt-will-start-to-transform-healthcare-in-2018/

[18] L. Mathews. (Nov. 2016). *Hackers use DDoS Attack to Cut Heat to Apartments*. Accessed: Jul. 26, 2018. [Online]. Available: https://www.forbes.com/sites/leemathews/2016/11/07/ddos-attack-leaves-finnish-apartments-without-heat/

[19] "Corero DDoS trends report|Q2–Q3 2017," Corero Netw. Secur., Marlborough, MA, USA, Tech. Rep., 2017. [Online]. Available: http://info.corero.com/rs/258-JCF-941/images/2017-q2q3-ddos-trends-report.pdf

[20] H. Kim, E. Kang, D. Broman, and E. A. Lee, "An architectural mechanism for resilient IoT services," in *Proc. 1st ACM Workshop Internet Safe Things*, Delft, The Netherlands, Nov. 2017, pp. 8–13.

[21] H. Kim, A. Wasicek, B. Mehne, and E. A. Lee, "A secure network architecture for the Internet of things based on local authorization entities," in *Proc. IEEE 4th Int. Conf. Future Internet Things Cloud (FiCloud)*, Vienna, Austria, Aug. 2016, pp. 114–122.

[22] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002.

[23] J. C. Eidson, *Measurement, Control, and Communication Using IEEE 1588*. London, U.K.: Springer-Verlag, 2006.

[24] J. C. Eidson and K. B. Stanton, "Timing in cyber-physical systems: The last inch problem," in *Proc. IEEE Int. Symp. Precis. Clock Synchronization Meas., Control, Commun. (ISPCS)*, Oct. 2015, pp. 19–24.

[25] J. Serrano *et al.*, "The white rabbit project," in *Proc. 12th Int. Conf. Accel. Large Exp. Phys. Control Syst.*, Oct. 2009, pp. 936–942.

[26] C. Curry, "Sentinel project—Report on GNSS vulnerabilitiies," Chronos Technol., London, U.K., Tech. Rep. 001, Apr. 2014.

[27] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou, "Distributed real-time software for cyber–physical systems," *Proc. IEEE*, vol. 100, no. 1, pp. 45–59, Jan. 2012.

[28] Y. Stein, "New opportunities for timing with SDN and NFV," in *Proc. ITSF*, 2015.

[29] T. Mizrahi and Y. Moses, "Time4: Time for SDN," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 3, pp. 433–446, Sep. 2016.

[30] I. Akkaya, "Data-driven cyber-physical systems via real-time stream analytics and machine learning," EECS Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2016-159, 2016

[31] I. Akkaya, S. Emoto, and E. A. Lee, "PILOT: An actor-oriented learning and optimization toolkit," in *Proc. Int. Workshop Robotic Sensor Netw.*, Apr. 2015.

[32] R. Alur, D. Fisman, and M. Raghothaman, "Regular programming for quantitative properties of data streams," in *Proc. Eur. Symp. Program. (ESOP)* (Lecture Notes in Computer Science), vol. 9632. Berlin, Germany: Springer-Verlag, 2016, pp. 15–40.

[33] E. A. Lee, "Computing needs time," *Commun. ACM*, vol. 52, no. 5, pp. 70–79, May 2009.

[34] E. A. Lee, "The problem with threads," *Computer*, vol. 39, no. 5, pp. 33–42, May 2006.

[35] T. V. Eicken, D. E. Culler, S. Goldstein, and K. E. Schauser, "Active messages: A mechanism for integrated communication and computation," in *Proc. 19th Annu. Int. Symp. Comput. Archit.*, May 1992, pp. 256–266.

[36] P. Levis *et al.*, "The emergence of networking abstractions and techniques in TinyOS," in *Proc. 1st USENIX/ACM Symp. Netw. Syst. Design Implement. (NSDI)*, Mar. 2004, p. 1.

[37] P. Persson and O. Angelsmark, "Calvin—Merging cloud and IoT," in *Proc. 6th Int. Conf. Ambient Syst. Netw. (ANT)*, Jun. 2015, pp. 210–217.

[38] E. A. Lee, J. Liu, L. Muliadi, and H. Zheng, "Discrete-event models," in *System Design, Modeling, and Simulation using Ptolemy II*. C. Ptolemaeus, Ed. Ptolemy.org, 2014.

[39] K. Gallaba, A. Mesbah, and I. Beschastnikh, "Don't call us, We'll call you: Characterizing callbacks in javascript," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Oct. 2015, pp. 1–10.

[40] M. Lohstroh and E. A. Lee, "An interface theory for the Internet of things," in *Proc. Int. Conf. Softw. Eng. Formal Methods (SEFM)* (Lecture Notes in Computer Science), vol. 9276. Cham, Switzerland: Springer, 2015, pp. 20–34.

[41] C. Jerad and E. A. Lee, "A javascript extension providing deterministic temporal semantics for the Internet of things," EECS Dept., Univ. California, Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2017-136, 2017.

[42] J. A. Stankovic, "Misconceptions about real-time computing: A serious problem for next-generation systems," *Computer*, vol. 21, no. 10, pp. 10–19, Oct. 1988.

[43] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 2nd ed. New York, NY, USA: Springer, 2005.

[44] N. Wirth, "Toward a discipline of real-time programming," *Commun. ACM*, vol. 20, no. 8, pp. 577–583, Aug. 1977.

[45] N. Wirth, *Programming in Modula-2*. Berlin, Germany: Springer-Verlag, 1983.

[46] T. Mirtin, "Realtime programming language PEARL—Concept and characteristics," in *Proc. IEEE Comput. Soc. 2nd Int. Comput. Softw. Appl. Conf., (COMPSAC)*, Nov. 1978, pp. 301–306.

[47] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*, 3rd ed. Boston, MA, USA: Addison-Wesley, 2001.

[48] J. Galletly, *Occam-2*, 2nd ed. London, U.K.: UCL Press, 1996.

[49] E. Kligerman and A. D. Stoyenko, "Real-time Euclid: A language for reliable real-time systems," *IEEE Trans. Softw. Eng.*, vol. SE-12, no. 9, pp. 941–949, Sep. 1986.

[50] R. Virding, C. Wikström, M. Williams, and J. Armstrong, *Concurrent Programming in Erlang*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[51] R. A. Müller, *Now: The Physics of Time*. New York, NY, USA: W. W. Norton and Company, 2016.

[52] E. A. Lee, *Plato and the Nerd: The Creative Partnership of Humans and Technology*. Cambridge, MA, USA: MIT Press, 2017.

[53] E. A. Lee, "The past, present and future of cyber-physical systems: A focus on models," *Sensors*, vol. 15, no. 3, pp. 4837–4869, Feb. 2015.

[54] D. Broman, L. Greenberg, E. A. Lee, M. Masin, S. Tripakis, and M. Wetter, "Requirements for hybrid cosimulation standards," in *Proc. 18th Int. Conf. Hybrid Syst., Comput. Control (HSCC)*, Apr. 2015, pp. 179–188.

[55] E. D. Jensen, C. D. Locke, and H. Tokuda, "A time-driven scheduling model for real-time systems," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 1985, pp. 112–122.

[56] H. Wu, B. Ravindran, E. D. Jensen, and P. Li, "Time/utility function decomposition techniques for utility accrual scheduling algorithms in real-time distributed systems," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1138–1153, Sep. 2005.

[57] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *Proc. IEEE 19th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2014, pp. 101–110.

[58] H. Kim, D. Broman, E. A. Lee, M. Zimmer, A. Shrivastava, and J. Oh, "A predictable and command-level priority-based DRAM controller for mixed-criticality systems," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2015,pp. 317–326.

MARTEN LOHSTROH received his B.S. degree in computer science and M.S. degree in grid computing from the University of Amsterdam. He is currently pursuing a Ph.D. degree in computer science with the University of California, Berkeley, under the supervision of Prof. E. A. Lee. He studies models of computation, programming languages, and systems design. From 2012 to 2014, he was an Associate Specialist with UC Berkeley, mostly dedicated to the development of Ptolemy II, an open-source software framework that supports experimentation with actor-oriented design. He was a recipient of the Systems and Software Modeling Best Paper Award, in 2018.

HOKEUN KIM received his Ph.D. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 2017. He was a Researcher with UC Berkeley, where he was involved in the Ptolemy Project. He was a Research Associate with HP Labs. He is currently with Google, where he focuses on the Internet security research for phishing detection and prevention with the Safe Browsing Team. His research interests include computer security, the Internet of Things, and cyber-physical systems. He was a recipient of the ACM/IEEE Best Paper Award and the IEEE Honorable Mention, in 2017.

JOHN C. EIDSON (LF'06) received a B.S. and M.S. degree from Michigan State University and a Ph.D. degree in electrical engineering from Stanford University. He was with the Central Research Laboratories, Varian Associates; Hewlett-Packard Company; and Agilent Technologies. He is currently a Visiting Scholar with the University of California at Berkeley. He was involved in a variety of projects including analytic instrumentation, electron beam lithography, and instrumentation system architectures and infrastructure. He received the 2007 Technical Award from the IEEE I&M Society. He was a co-recipient of the 2007 Agilent Laboratories Barney Oliver Award for Innovation. He was heavily involved in the IEEE 1451.2 and IEEE 1451.1 standards and was an Active Participant in the standards work of the LXI Consortium. He is the Chairperson of the IEEE 1588 Standards Committee.

CHADLIA JERAD was born in Tunis, Tunisia, in 1979. She received her Dipl.Ing and Ph.D. degree in electrical engineering from ENIT, University of Tunis El Manar, in 2002 and 2008, respectively. She was a Coordinator of the embedded systems and software specialization for four academic years. She was the Tunisian Partner of several DAAD funded projects. From 2016 to 2017, she visited the Research Group of Prof. E. A. Lee with the University of California at Berkeley as a Fulbright Visiting Scholar, where she contributed to the Accessors project and to CapeCode: a Ptolemy II-based design tool for the Internet of Things. Since 2009, she has been an Associate Professor with ENSI, University of Manouba, Tunisia. Her research interests include embedded systems, the Internet of Things, and rewriting logic. She was selected as the Portrait of the Month, in 2017, by DAAD Office, Tunis.

BETH OSYK Osyk received a B.S. degree in computer engineering from Case Western Reserve University and a M.S. and Ph.D. (with a focus on a methodology for assessing the reliability of safety-critical in-vehicle networks) degrees in electrical and computer engineering from Carnegie Mellon University. She was a Senior Engineer with Robert Bosch LLC, focused mainly on analysis of automotive engine controllers. She was a Research Staff Member with UC Berkeley, helping to build and test a disciplined, multihost Internet of Things development and execution environment as part of the TerraSwarm Center. She is currently an Engineer with Edge Case Research, LLC, specializing in safety assessment for embedded systems.

EDWARD A. LEE (F'94) received a B.S. degree from Yale University, an S.M. from MIT, and a Ph.D degree from UC Berkeley. From 1979 to 1982, he was a Technical Staff Member with Bell Labs, Holmdel, NJ, USA. From 2005 to 2008, he was the Chair of the EE Division at UC Berkeley. Then, he was the Chair of the EECS Department at UC Berkeley. He is currently a Professor of the Graduate School in EECS at UC Berkeley. He is the Director of iCyPhy, the Berkeley Industrial Cyber-Physical Systems Research Center. He is a Co-Founder of BDTI, Inc. He has consulted for a number of other companies. He has authored several books and more than 300 papers and has delivered more than 180 keynote and other invited talks at venues worldwide. His research focuses on cyber-physical systems, which integrate physical dynamics with software and networks. His focus is on the use of deterministic models as a central part of the engineering toolkit for such systems. He led the development of several influential open-source software packages, notably Ptolemy and its spinoffs. He was a recipient of the 1997 Frederick Emmons Terman Award for Engineering Education and the Berkeley Citation, in 2018. He received the 2016 Outstanding Technical Achievement and Leadership Award from the IEEE Technical Committee on Real-Time Systems. He was an NSF Presidential Young Investigator.