The Design and Implementation of a Real-Time Complex Event Processing Framework for CPS/IoT Systems

Jared Hall
dept. Computer Information Science
University of Oregon
Eugene, Oregon
jhall10@cs.uoregon.edu

***Abstract*—As the field of Cyber-Physical Systems continues to advance, new and interesting changes regarding its capability, adaptability, scalability, and usability [1] have come about. The most notable change has been the aggressive expansion of the variety of entity types that can be deployed in these systems (i.e. the entity eco-system). With this expansion, the complexity of handling vastly different communication protocols, processes, and data becomes a significant challenge since these data sources possess many more characteristics compared to the scalar data acquired by traditional IoT inputs. These additional characteristics severely limit the usefulness of such data sources to a CPS/IoT control system, especially in determining what controls should be enforced, effectively rendering these growing areas of the IoT out of the scope of many current CPS/IoT control systems. New CPS/IoT control systems would therefore need to leverage a great deal of computing power on demand to be able to deal with these devices and their data at scale while still being able to keep up with the core demands of real-time control, adaptability, and usability that users of these systems expect. Typically, the response to these issues has been to rely heavily on cloud computing resources. But this reliance has come at the cost of additional latency that would prove disastrous in application domains such as Industry 4.0, Internet of Healthcare Things, and defense, to name a few. In our previous work [2]–[5], we developed the cloud-based, Command Messaging Policy Enforcement Service (CoMPES). CoMPES is the result of an initiative towards a policy enforcement platform for large-scale, massively distributed Cyber-Physical Systems (CPS) with a core objective to offer an elastic architecture for managing and controlling physical devices using discrete control schemas that are interconnected via our novel CPS/IoT control system. However, CoMPES has a significant performance bottleneck in its policy enforcement cycle as it relies on a collection of processing hubs to forward entity telemetry to the cloud for processing. In this paper, we propose a novel edge computing framework that relies on the established Pub/Sub communication paradigm to build a complex event processing framework that allows users to push critical computations to the edge of the IoT. We then, with supporting results, show that when we increase the utilization of edge computing resources, CoMPES outperforms the cloud-centric variant by a 20-43x when comparing the latency of transitional state changes and supports systems of significantly larger scale (10-100x) while maintaining the same level of service as the cloud-centric version.**

Thanks to Dr. Joe Sventek and the University of Oregon for funding this research.

***Index Terms*—Cyber-Physical Systems, Internet of Things, Edge computing, Complex Event Processing, IoT policy Enforcement, Artificial Intelligence, Cloud Computing**

I. Introduction

The term "Cyber-Physical Systems" (CPS) refers to those systems which seamlessly integrate sensing, computation, control, and networking into physical objects and infrastructure [1]. Formally, a CPS consists of three components: A cyber component that performs computation (e.g., information processing and control), a communication component that handles the communication between the cyber and physical components, and a physical environment that consists of physical entities or processes [1], [6] as shown in Fig. 1 below. In these systems, the cyber component is tightly coupled with the physical component via a feedback loop [1], [7] where the physical environment informs the cyber component of its current state via the communication component, and the cyber component exerts control over the physical environment which alters its state, causing the cycle to repeat.

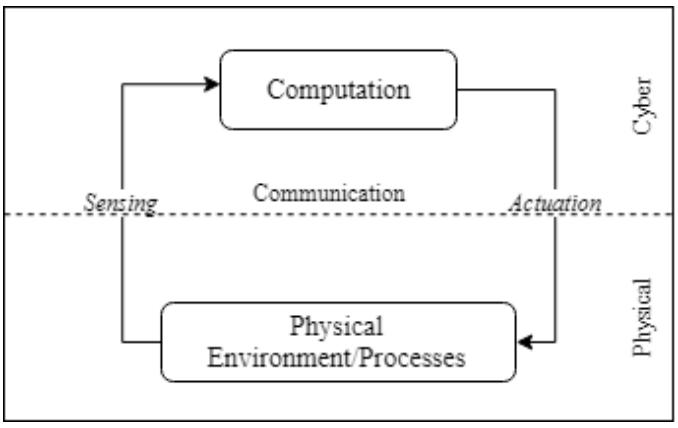

Fig. 1. Cyber-Physical Systems Component Model

Since its introduction in 2006, the field of CPS has evolved with new and interesting advancements concerning its capability, adaptability, scalability, and usability [1]. One such

advancement is the unified field paradigm which merges the fields of CPS with the Internet of Things (IoT), and introduces CPS/IoT systems [1]. The IoT is a concept that enables real-world, everyday objects to connect to the internet and interact with each other. In the unified field paradigm, the two fields are essentially merged with the following changes:

- The IoT takes the place of the communication component in the traditional CPS model.
- The CPS/IoT software platform/middleware is called by the generic term "platform".
- The platform and the physical environment are wrapped together and given the term "CPS/IoT system".

In this paradigm, humans play a central role both as entities in the physical environment and as users of the system (e.g., "human-in-the-loop") via their interactions with both the system and the physical environment. Because of this, humans are added as the fourth component in CPS/IoT systems, as stated by the National Institute for Standards and Technology (NIST) in [1]:

"... reflect the varying roles humans may have in CPS/IoT systems, ranging from user to component, environmental factor, etc. (for example, for a Level 3 automated vehicle a passenger is a user, a safety driver is a component, and a pedestrian is an environmental factor). The interactions of humans with CPS/IoT systems may be limited to the logical realm, to the physical realm, or extend to (and link) both. Because of this diversity of interactional modes, humans are treated as a distinct component in the CPS/IoT Components Model."

In addition to the introduction of the unified field of CPS/IoT, the entity eco-system for CPS/IoT has been rapidly expanding to include a wide array of devices and entities with functionalities that far exceed the simple sensor and actuator networks of the previous generation of IoT devices. This relatively new generation of CPS/IoT entities include industrial robotics, healthcare devices and subsystems, city and community core infrastructure, multimedia systems and entities, vehicles, household appliances, wearables, social networks, etc. – that all come with their own unique challenges and demands as they are integrated into the CPS/IoT ecosystem. These expansions to the traditional view of the CPS/IoT entity ecosystem have been met with new entity paradigms such as the Internet of Multimedia Things (IoMT) [8], The Internet of Healthcare Things (IoHT) [9], and the social IoT [10] among others. This evolution arose naturally because the utility and ubiquity of CPS/IoT systems allows them to be useful in building managed environments for many domain applications from city and residential automation to industrial resource management, healthcare, public safety, and even entertainment.

This expansion of the CPS/IoT ecosystem has resulted in a need for modern CPS/IoT systems to adopt a more expansive approach to what we consider the "sensors" and "actuators", the building blocks of the CPS/IoT ecosystem. We see this shift already with the inclusion of multimedia sensors such as video and audio sources, social networks, or complex industrial mechanisms that generate more data with far greater complexity than the traditional scalar data generated from simple sensors and actuators such as Radio-Frequency Identification (RFID) networks or other traditional sensor and actuator networks.

However, as will be shown in the literature review, most current IoT systems focus rather exclusively on processing scalar data from a constrained set of devices. This is because the type of data acquired by traditional things in the IoT (e.g light, temperature) is by nature scalar and requires less memory and fewer computational resources to analyze. Since this kind of data demands only simple processing capabilities, these systems can enact decisions rapidly because of the lower data rates from the sensing device. But with the expansion of the entity eco-system, the complexity of handling vastly different communication protocols, processes and data becomes a significant challenge since these data sources possess more complex characteristics compared to the scalar data acquired by traditional IoT inputs. These additional characteristics severely limit the usefulness of such data sources to a CPS/IoT control system, especially in determining what controls should be enforced, effectively rendering these growing areas of the IoT out of the scope of many current CPS/IoT control systems. New CPS/IoT systems would therefore need to leverage a great deal of computing power on demand to be able to deal with these devices and their data while still being able to keep up with the core demands of real-time control, data synthesis, and interoperability that users of these systems expect.

A. Core Research Challenges

During the course of our preliminary research, we have identified the following challenges that exist for new CPS/IoT systems that must be addressed:

1) Heterogeneity of CPS/IoT entities and their data: Since the IoT is a fundamentally inclusive networking paradigm CPS/IoT systems must be able to handle communications from a wide variety of sources. These systems must be able to both accept and process this communication as well as being able to communicate with the end device in it's own protocol. With the inclusion of data rich sources such as multimedia devices or smart city infrastructure as members of the IoT, the complexity of handling vastly different communication protocols, processes, and data becomes a significant challenge since these data sources possess more complex characteristics compared to the scalar data acquired by traditional IoT inputs. New CPS/IoT systems must therefore be extensible to these new data sources in order to stay relevant in a rapidly changing CPS/IoT ecosystem.

2) CPS/IoT Platform Elasticity and Reusability: Currently, CPS/IoT systems are designed with a specific domain in mind (e.g., IoHT as seen in [11]). However, for CPS/IoT systems to be truly ubiquitous and useful, new CPS/IoT systems must be designed to be both elastic and reusable. These systems must be able to accept potentially dense and arbitrarily deep network definitions for a variety of domains. For instance, if we consider a city government as an IoT user, then this city

could have one public Network of Things (NoT) for the entire city covering lower level networks for individual boroughs, individual blocks, substations, buildings, etc. This user could also have a private NoT for emergency services that takes on an entirely different architectural design. In order to handle these different application domains, new CPS/IoT systems must allow the users to add, alter, or remove entities and their corresponding control schemas from the CPS/IoT platform, with these changes taking effect in real time. This, therefore, demands that IoT systems be reusable and not hard-coded to a specific use-case as it would severely restrict the applicability of the CPS/IoT system and therefore its usefulness to future users.

3) Interoperability with Middleware: With the advent of Fog/Edge computing, middleware for the IoT is becoming a powerful tool that CPS/IoT systems can use for processing complex data of various formats. Yet there is a dearth of current IoT systems that are interoperable with 3rd party fog/edge computing services. This requires rich data to be processed within the device or system which introduces a heavy burden on the performance of the IoT system as a whole and restricts the ability of the IoT system to adapt to new changes and advancements in the field. For new IoT systems to be able to meet real-time performance constraints and maintain applicability in an ever-changing world they must include some method of interacting with these powerful processing applications.

4) Latency of Environmental Control and Scalability: With the increasing adoption of CPS/IoT systems into the mainstream consciousness, the core performance of these systems has become a pressing issue. During the course of our own research, we have identified two core metrics that can be used to measure the performance of CPS/IoT systems in addition to whether these other systems meet the challenges stated above: 1) The Latency of Transitional State Change (TSC Latency) and, 2) The scalability of the CPS/IoT system. The core concept of TSC latency is similar to Round-Trip Time (RTT): It is the time from when an entity reports its state to the time when that same entity or another receives the corresponding action. With respect to scale, CPS/IoT platforms need to be able to operate arbitrarily deep, potentially densely interconnected, Networks of Things (NoTs). To accomplish this, these new platforms must be able to scale with the network definition given to them while maintaining average TSC latency within the constraints of "real-time" or "near real-time".

In summary, in order to meet these challenges, new High-Performance CPS/IoT systems must be designed that are scalable and elastic with the ability to manage user-defined physical environments composed of a wide variety of entities in real time. To address these challenges, we propose a Complex Event Processing (CEP) edge computing framework for the Command Messaging Policy Enforcement Service (CoMPES). Our core motivation for this research stems from the above challenges in addition to the following research questions:

- Could introducing an event processing framework at the edge of our CPS/IoT operational policy enforcement system significantly increase scale and reduce the overall latency?
- Would the process of "Entity Virtualization" address the challenge of entity heterogeneity and system elasticity/re-usability in an expanded CPS/IoT Eco-system?
- Would altering the policy enforcement cycle to be more reactive lead to significant decreases in processing latency?

The rest of this paper is organized as follows: In Section 2, we take a look at current research trends in CPS/IoT systems with a focus on how they address the core challenges discussed above. In section 3, we present our edge computing framework. In section 4, we present our experimental results, analyze how our new system addresses the core challenges above, and compare this work with our previous architecture. Finally, we end in section 5, with a brief discussion on our contributions and future work.

II. Literature Review

As mentioned in the introduction, the expansion of the entity ecosystem has driven the development of new and successive generations of CPS/IoT systems. Since the field's introduction in the early 2000's we have seen no less than four separate "generations" of CPS/IoT systems. Earlier systems (Gen 1) such as the systems presented in [12] were primarily composed of simple sensor networks arranged around a central "gateway". These early systems mostly focused data synthesis and providing the user with information regarding their individual Network of Things (NoT). After a short period, a second generation of CPS/IoT systems emerged that incorporated traditional sensors and actuators into the IoT such as in [13]. In the last 10 years researchers have begun to expand the scale of CPS/IoT platforms by introducing cloud based IoT systems such as in [2], [3], [5], and [14]. Finally, the most recent generation of CPS/IoT systems has seen a return to the ground with edge/fog computing. As we will see, these changes to the construction of CPS/IoT systems have been primarily driven by two key factors: 1) an inherent drive to expand the "reach" of CPS/IoT systems by incorporating new kinds of "things", and 2) A need to increase the performance of CPS/IoT systems in order to better utilize these new "things". To illustrate this point, we now take a brief look at several archetypal systems and the entity eco-systems for which they were designed.

A. Expansion of CPS/IoT application domains and their entity ecosystems

The system's presented in [12], [13], and [15] are classic examples of many smaller IoT systems that typically featured a mostly hardware-based solution for interacting with a limited set of device types (e.g., transducers, RFID tags, etc) and focused only on presenting the data to the user. These architectures were primarily focused around a central "gateway" that would receive telemetry from it's connected

entities. Typically, this gateway would then send this data to a processing component that would extract useful information from the telemetry before forwarding it to the application layer that would synthesize the data and present it to the user. These simplistic, gateway based, IoT systems make up the first generation of CPS/IoT systems.

However, with the introduction of the field of Cyber-Physical Systems in 2006, as noted in [1], numerous possibilities presented themselves to researchers in the field. We could now include entities that execute actions and, together with the concept of "transitional states", implement discrete feedback loops that exert programmed control over environmental factors in the real world as discussed in [7]. This spawned a second generation of IoT systems that started to gain increasing similarity to traditional CPS systems.

The eventual conjunction of the fields of CPS and IoT would prove to be both a huge boon to the applicability of the field to various application domains but also a significant challenge as discussed in [16]. This research challenge came about because previous CPS/IoT systems simply lacked the processing power necessary to handle the significantly increasing amounts of data.

As an answer to this challenge, researchers in the field began proposing the addition of cloud computing to the processing component of a CPS. The works presented in [17] encapsulate this idea. These systems often feature a distributed controller in some form that is located in the cloud. This cloud-based, distributed controller allowed these systems to do a massive amount of computation without needing the user to invest in additional computing resources for their CPS. As such, third generation Cloud-based CPS/IoT systems have become the preferred solution for CPS/IoT platforms.

With the increase of processing power from the cloud came an increase in the types of entities that are included under the umbrella term "things" in the IoT (i.e. an expansion of the entity eco-system). For instance, [8] introduced the idea of multimedia objects as things, while [10] introduced the idea of "social things".

However, these changes to the entity eco-system do not come without their own challenges to performance and usability. As we increased the reach of the entity eco-system, two major challenges have arisen: 1) The performance of these systems (in terms of TSC) has significantly decreased as noted in [18] and, 2) Systems built for a particular application domain (e.g., IoMT) have limited applicability or usability in other domains. The apparent answer to these challenges has been the introduction of a fourth generation of CPS/IoT systems (cohesive systems) and the integration of Fog/Edge computing to CPS/IoT systems.

B. Fog/Edge Computing Frameworks for CPS/IoT Systems

According to [19], the central goal of Fog/Edge computing frameworks is to move some of the processing that would otherwise take place in the cloud closer to the entity in order to gain the benefits of a ground based system while still having the computing power of the cloud. By moving certain processing closer to the entity, we gain multiple benefits, such as decreased TSC latency which leads to more transitional state changes per second and the decentralization of the IoT. Another benefit of Fog/Edge computing is the ability to abstract CPS/IoT architectures which increases their reusability by allowing developers to build a generic solution and leave specific details regarding the IoT devices to the edge.

Since 2016, there have been a number of papers proposing fog/edge computing frameworks for CPS/IoT systems such as [20] and [21]. However, despite the obvious applicability of edge computing to the policy enforcement cycle of CPS/IoT control systems, we have not been able to find any related work. Of the proposals for fog/edge computing frameworks we have seen, most focus on a different aspect of the CPS/IoT technology stack such as in [22] where the authors introduce a Fog/Edge computing framework for offloading resource management, or [23], where the authors introduce an edge computing method for IoT data analytics.

C. Cohesive CPS/IoT Platforms

A cohesive system addresses many issues, such as those presented by constrained devices, legacy devices without internet connections, and typical bandwidth and processing issues, found in typical cloud-only or ground-only systems. Typically, a cohesive system (e.g. [2], [23]) consists of both cloud and ground components, which enables us to gain the computing power of the cloud while maintaining the speed and adaptability of the ground. With cohesive CPS/IoT systems, the bandwidth and processing issues are usually addressed by means of relegating these tasks to a specially designed application. For example, in our previous work [2], the concept of Compute Pages is introduced to handle both IoT related processing and communications while the hubs offload the processing of data to specially designed applications.

The general idea of this kind of system consists of an abstract ground architecture that allows it to be mapped to pre-existing physical IoT infrastructure, a virtual object layer that may be hosted in the cloud or located on the ground, and a cloud-processing layer that may feature some form of virtual object aggregation.

D. Previous work

In our previous work [2]–[5], we developed the cloud-based Command Messaging Policy Enforcement Service (CoMPES). CoMPES is a cohesive CPS/IoT platform that offers decentralized control for individual Networks of Things and, to a greater extent, the Internet of Things. The core architecture is shown in Fig. 2.

1) The "Cloud Side" of CoMPES: The cloud side of the CoMPES is primarily composed of three cloud services: 1) the Policy Enforcement Service (PES), 2) the Compute Service, and 3) the Policy Generation Service. For our brief review of the service we will only be covering the PES since the Compute Service and the Policy Engine have little to do with the actual policy enforcement cycle which is the focus of our contribution in this paper. The PES consists of a scalable set of

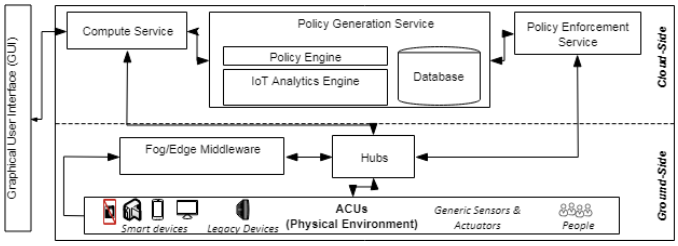

Fig. 2. CoMPES High Level Design

"Compute Pages" hosted on virtual machines. The Compute Page is a software processing object which is given a set amount of the hosting VM's resources. Using these resources, the Compute Page uses the Telemetry Data Transformation (TDT) algorithm first proposed in [2] to enforce the user's policy for their devices. The compute page also operates a web server and performs essential web management, such as load balancing. The Compute Page and the TDT algorithm are the driving forces of the service and were designed with platform independence in mind.

The TDT algorithm largely consists of three parts - a primary cache, a secondary cache, and a rule engine. The primary cache matches the cohesive system state, i.e. the states of every entity in the CPS, to a known operational policy. System-wide caching yields the fastest runtime for the TDT algorithm of O(1) which is also the average case because we only activate the secondary cache or rule engine when unknown states are observed. In the secondary cache, the observed states of each entity are cached to a known behavior as defined by that entity's operational policy. This has a runtime of O(n), since it has to look at every device. The associated algorithm is show in Algorithm 1.

To illustrate how the TDT algorithm works, let us say a user builds their network and a hub is ready to send a State Representational Image (SRI), a textual snapshot of an area's state, to the PES in order to trigger the build cycle. When the Hub sends the SRI to the service, the first thing the SRI encounters is the CoMPES Communication Module (CCM). On the PES side, the CCM will receive the SRI and insert it into a Network-wide State Representational Image (NSRI). This process is done for every Hub with which the client interacts, thus keeping the NSRI up to date. The NSRI is then cached with the primary cache. On cache hit, the resulting network-wide operational policy is retrieved from the cache, broken down into Hub-specific Policy Representational Images (PRI), a textual snapshot of the commands to be executed, and broadcast to each Hub. The Hubs will then either execute the policy themselves or pass on the relevant commands to middleware which will execute them. On a cache miss, the NSRI is used to cache the individual states of each device in the secondary cache using the Operational Policy Definition (OPD), a collection of Associative-Behavioral rules [5]. On a secondary cache hit, the state of each device is matched with an action it should take and these actions are merged into a network wide policy image, which is then used to update the primary cache. This new image is then sent to the communication module to be broadcast to all of the Hubs. A secondary cache miss only occurs if there is no OPD, and as such, the policy engine is called to build one. This perpetuates a feed-forward mechanism where the service would eventually learn every macro state of the user's CPS and be able to, in constant time, match said macro state to a known set of behaviors that the CPS should execute.

Algorithm 1: Telemetry Data Transformation

```
PC = load the primary cache
OPD = load the OPD if it exists

On SRI update from Hub
NSRI = load the NSRI from memory
Update the current NSRI with the new SRI
if NSRI in PC then
    output actions
else
    Policy = ""
    if OPD exists then
        Policy Block
        for each area x in the region do
            for each ACU A in x do
                P = OPD[x][A.ID][A.CS] or NULL
                Area policy += P
            end
        end
        Policy = join all area policies
    else
        Request OPD from Policy Engine
        Trigger policy block
    end
    Update the primary cache with new policy
    Get actions from policy and send to hubs
end
```

2) The "Ground Side" of CoMPES: The ontological IoT network architecture that CoMPES manages consists of a generalized hierarchy of a cluster-based network of things with three categories: Regions, Areas, and Atomic Computational Units (ACUs). At the base of our network hierarchy are the ACUs. An Atomic Computational Unit (ACU) is a discreet, integrated, CPS composed of any sensor, actuator, device, middleware, web service or otherwise unnoted platform with observable states and exactly one representative virtual object. This Virtual Object (VO), at a minimum, must be capable of observing the entity and representing its states but it may also be able to interact with said entity in the form of executing actions. These ACUs are split into two broad categories based on their capabilities: sensors and actuators. For the purposes of our research, a sensor is classified as any entity that only provides information, while an actuator is any entity that both provides information and can execute actions. The ACU serves as our answer to the problem of entity heterogeneity in the

IoT. An ACU's primary purpose is to interpret data received via device specific protocols into a universal protocol and vice versa.

Next in our network hierarchy is the area. An area is a secondary cluster of ACUs that are connected to a single processing hub. The processing hub can consist of any hardware capable of running the VOs. In the hub, an event loop waits until either an update event is triggered for any ACU, the hub receives a PRI from the PES, or a trigger event occurs on an interval to pull ACU states or send an SRI to the PES. Finally, a region is a dependency-based supercluster of areas, similar to a network except that the interconnections consist of semantic-links (i.e. dependencies) instead of physical connections.

A semantic-link is an ontological binding between two ACUs that explicitly determines the behavior of these ACUs. This link can be cataloged as either a one-way or a two-way link. An example of a one-way link would be a dependency of a Wi-Fi enabled light (an actuator) connected to a smart switch (a sensor). An example of two-way semantic link could be explained by means of a master-type smart switch and a joystick. If the master button is pressed then the joystick will be inoperable, whereas if the joystick is in operation then the master button's input will be ignored. This semantic link provides important semantic context to the data received from the device, enriching said data and giving the policy enforcement service needed information so that it can enforce the correct operational policy. Our network architecture uses the "Implicitly Defined Communication (IDC)" concept first proposed in [2]. Since all the devices talk to a controlling agent in the cloud and the controlling agent talks to every other device in the network, we can then say that any device in the network implicitly communicates with any other device in the network even though that communication does not directly happen.

In this brief survey we have shown that while many IoT systems have been proposed for a variety of contexts and applications, a cohesive IoT system which can operate in the expanded entity ecosystem and maximize its use with Fog/Edge computing services has yet to be realized.

III. Proposed Solution

To address the challenges outlined in the introduction, we are proposing a novel edge computing framework for CoMPES; the core idea is to address the inefficiencies of CoMPES while also maintaining the advancements made in our previous works [2-6]. On its own, the CoMPES system is insufficient for domains which require control cycles that process high volumes of data rapidly (in excess of 100 events a second), or infrastructure that requires massive, arbitrarily deep IoT architectures (e.g., smart cities). This is because the CoMPES system was primarily designed for the "Smart and Connected Community" domain that largely focuses on smart homes or offices were the number of entities at a local scope is limited and performance above the capacity for human recognition is not needed. However, for domains such as healthcare, defense, industry, and public infrastructure the speed of the previous system (approx. 5 – 15 events a second) renders the system simply unusable.

The greatest contributing factors to these issues would be our implementation of the Transitional State Change cycle, discussed in the introduction, and the communication framework used in our previous version. To be specific the core issues are:

- The operational policy enforcement cycle necessitated that all information be sent to the Policy Enforcement Service (PES) in the cloud for processing before the system could decide what a particular entity should do. This is unnecessary for entities that are either self-determinate or local in scope.
- The system focused solely on the use of raw states to implement transitional state change. This causes a massive increase in redundant communications over the internet/to the cloud as any entity needs to be constantly polled for its state.
- The communication framework relied largely on HTTPS based WebSockets (which are, admittedly, slow) to facilitate communication because of their portability and asynchronous nature (both client and server can push/pull). However, this creates a communication bottleneck as the extra time is spent communicating over this protocol.

To address these limitations, we are presenting a new CPS-IoT architecture that makes the following changes:

- The central policy enforcement cycle will be split amongst the different layers of the architecture enabling lower level entities (e.g., self-determinate entities) to make their own decisions.
- The introduction of real-time Complex Event Processing (CEP) to significantly reduce communication over the network and increase processing speeds.
- The introduction of a real-time Complex Event Processing cache [25] along with a Pub/Sub communication module that uses SRPC [24] to significantly reduce the communication bottleneck.
- The Atomic Computational Unit (ACU) has been completely redesigned with the ability to make its own decisions in accordance with the user's policy without needing to push data upstream.
- The CoMPES processing hub was also redesigned to include the addition of an AI algorithm that, we hypothesize, will significantly reduce the processing time needed to make decisions as well as completely remove the build and secondary cache cases from our previous version of CoMPES.

Our central hypothesis is that the introduction of these core changes will significantly increase the performance of the system relative to the task of transitional state change and as well as increase its scalability. These alterations make up the core of our new edge computing framework. The rest of this section will explain each of these in depth along with an overview of the new edge computing framework.

A. An overview of the Edge-Computing Framework

As can be seen in Fig. 3 below, the new edge-computing framework is similar component-wise to the old CoMPES framework with the notable addition of a "virtual cache". However, the individual components on the "Ground-Side" were altered significantly. We will detail each of the new components starting from the outermost edge of the new edge computing architecture.

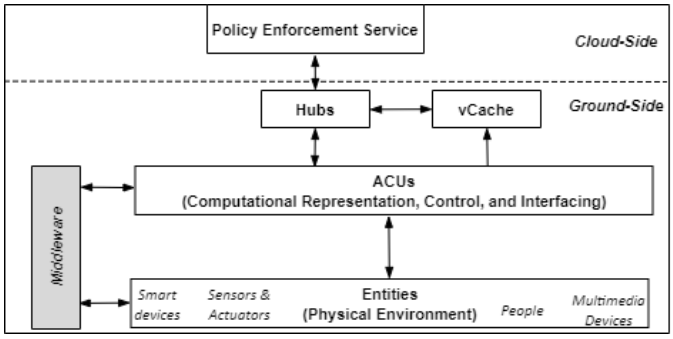

Fig. 3. The new Edge computing architecture for CoMPES

B. The Physical Environment

The first half of our new edge computing framework is composed of two primary components: a) the "physical entity", and b) the Atomic Computational Unit (ACU). While the nascent concept of these components existed in our previous work, for this work, one of our major questions is if restructuring these components will increase the performance of the CPS has a whole. Another major question was whether making the physical entities and their associated ACU's components on their own will increase the scale-ability, elasticity, and re-usability of the system.

1) Agnostic Entity Design: A "physical entity" consists of anything in the real-world that can be either observed or controlled by the user's provisioned CPS with the entity's communication being handled either directly or via a processing middleware framework such as the multimedia middleware we proposed in [3]. These physical entities include traditional sensors and actuators, both legacy and smart devices, computational resources, appliances, environmental factors such as temperature or humidity, multi-media representations of things, etc. Fig. 4 shows the general layout of how our system would interact with real-world entities. How the entity interfaces with the CPS is up to the user as well as the format of its communication.

In general, the user has two generic options for establishing communications between the entity and the system: a) the entity can communicate with the system directly or, b) it can do so via a 3rd party middleware framework. Acceptable middleware for our framework consists of any fog or edge computing middleware that exists to process raw data from the physical entity into telemetry that our system can use. This includes rich media data or any other form of complex data that is too costly to process in the core architecture. Whether

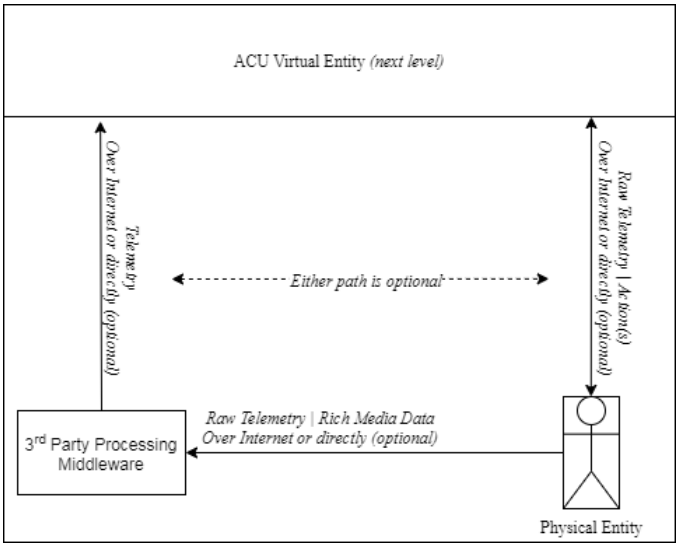

Fig. 4. Physical Entity Design

the communication happens over a direct wire to the ACU or over the internet is also left to the user. The general idea in our design of the communication framework is to be as flexible as physically possible in order to accommodate the user's choice of application domain and/or physical infrastructure. It is at the physical entity that we address the challenge of "middleware interoprability" discussed in the introduction.

2) The Atomic Computational Unit: For the field of CPS/IoT in general, the concept of a "thing", a sensor, or an actuator is borrowed from other fields (e.g., the concept of sensor/actuators from computer engineering), however, in our previous work, we introduced a different principal for classifying things in relation to our system: the Atomic Computational Unit. As discussed in our previous work, an Atomic Computational Unit (ACU) is a discrete, integrated, cyber-physical subsystem composed of any physical entity and exactly one representative virtual entity. This Virtual Entity (VE), at a minimum, must be capable of observing the physical entity and representing its states but it may also be able to interact with said entity in the form of executing actions. For this work we further expand on our previous definition by adding two distinct classes of ACU's:

- **Sensors** – A sensor ACU's sole purpose is to provide information, via telemetry/events, into the CPS. In this case the VE will not have a write routine because it cannot execute actions and it will also not receive events from the upper levels. Some good examples of a "sensor" ACU would be a temperature sensor or the video representation of a human user.
- **Actuators** – This type of ACU can execute actions in addition to providing telemetry/events. It possesses all of the components seen in Fig. 5. An actuator's main task is to control the physical entity as well as provide a computational interface with said entity/physical process. Actuators are, in and of themselves, cyber-physical

systems. This is an important distinction and is one of our main contributions in our new framework (the other being the processing hub which will be discussed later).

We refer to the process of creating an ACU as "Entity Virtualization" since from our systems perspective the physical entity *is* the virtual entity. In order to build the IoT, many systems try to interface their core components directly with the physical entities. This has spawned the problem of entity heterogeneity in CPS/IoT and has resulted in a boom of IoT-centric communication protocols. To avoid this, we utilize entity virtualization, effectively solving the problem of entity heterogeneity in the IoT since an ACU may communicate with an entity in its own native format but communicates with the rest of the framework using our communication protocol (which will be discussed later).

In our previous version of CoMPES [2]–[5], the VE was simply a virtual object that was utilized by the processing hub. The hub would use the VE to get a device's state and to execute action on the device. However, for ACUs that are self-determinate (i.e., the ACU's own state determines what action it should take), the Hub still had to perform an entire PES cycle just to tell that device to execute an action. This is also true for ACU's who are dependent upon other ACUs under the same processing hub. Since everything is locked into the Hub's PES cycle this meant that there would be some delay before an ACU could execute an action as the hub would need to talk with the cloud before telling the device what to do. This delay also applied to states that affect other entities. In order to remedy this issue, we are proposing an ACU redesign that allows the user to close the loop closer to or even at the ACU level along with an operational policy schema that supports this.

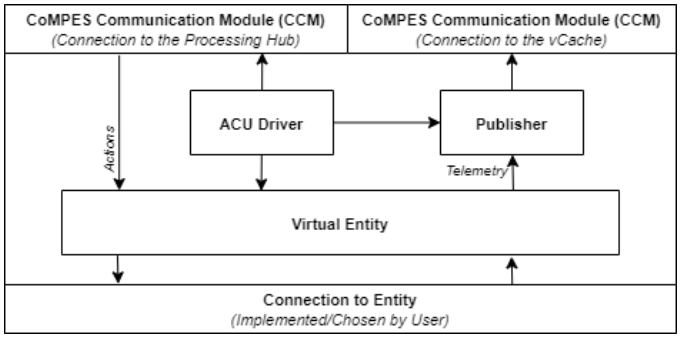

Fig. 5. The Atomic Computational Unit Design

As can be seen in Fig. 5, in our new ACU design, we decoupled the ACU from the processing Hub and gave it the ability to run on its own with three central components: a) a driver, b) a Virtual Entity and, c) a publisher. The driver is primarily responsible for setting up and running the components. For the virtual entity we chose an object-oriented approach in its design. As such, the virtual entity is a class that contains the following:

- An abstract base class that contains the base functionalities for data translation and communicating with the publisher/processing hub.
- An open reader method that is left to the entity manufacturer or user to implement.
- A writer method that is also left to the user/manufacturer to implement.
- A method for starting the VE and running it in its own process or thread.

The reader and writer methods are the primary customizable methods of communicating with the entity itself. Since they are left open for implementation, they can be customized to utilize any given entity's communication framework. So long as the reader method dumps the data into the publishers' channel in the abstract base class and the writer is able to send/execute actions on the entity the system will work.

This customizability in the ACU gives users an incredible amount of freedom in designing and building their CPS. Should the ACU be self-determinate then the user can simply add an operational policy directly to the ACU and on reading telemetry from the entity the ACU will cache the telemetry with the policy and immediately execute the relevant action. This process gives our system it's fastest run time since there is only one step needed to resolve the loop.

We chose to use the Simple Remote Procedure Call (SRPC) [24] to facilitate communication between the ACU and the rest of the system since it is a reliable protocol based on UDP. SRPC is an extremely fast and scalable protocol built for pub/sub architectures which enables us to rapidly publish events up-stream without the prohibitive overhead imposed by using TCP.

Let's take a WiFi light bulb as an example to illustrate how the new ACU design works. For this smart light bulb, we have a reader method that is capable of requesting the current state of the light once a second. We also have a writer method that can execute commands such as "turn off" or "turn on" by sending the corresponding commands to the light bulb in its own language. The VE, in this case. would be in charge of collecting the entity's state and executing any action it receives. Under our event-based framework, on receiving any telemetry from the entity, the VE's reader method sends that telemetry to the publisher. The publisher then publishes this telemetry up-stream to the vCache and waits for new telemetry. For actions, when the ACU receives an action from the hub, it sends the action to the writer method that executes the action (e.g., telling the light to turn off). Thus, with just the ACU we have established a discrete CPS with the ability to control the light. Since the communication channels are dedicated to either publishing telemetry or receiving actions, we can also take advantage of the speed of SRPC in order to build a near real-time networked control system.

C. The Processing Hub

In this work, the processing hub is the second main contribution to our new edge computing framework and contains two key technological improvements; namely, the "Virtual Cache" (vCache) and our novel event processing module. In

this section, we will be presenting both of these new modules (the vCache and the CEP architecture) in greater detail.

1) The Virtual Cache (vCache): The Virtual Cache (vCache) consists of a Homework system cache [25] whose central purpose concerns the translation of entity telemetry into discrete events via automatons. The vCache serves as the central location in an individual Network of Things(NoT) where ACUs publish their telemetry in order for it to be translated into discrete events for the processing hub. These publications take the form of a simple SQL insert call made by the ACU's publisher to insert the telemetry into an ephemeral table registered with the vCache for that specific ACU. This telemetry is then translated into discrete events by an automaton programmed using the Glasgow Automaton Programming Language supported by the Homework Cache [22-23].

In order to give the user a wide range of flexible options in how they design their CPS, the cache is configured so that the user can either select a pre-built automaton or program one themselves. This difference in the automaton's internal programming allows it to handle ACU telemetry in vastly different manners. For instance, consider our WiFi light bulb from earlier. The light bulb ACU can publish two kinds of telemetry: a)current power consumed, or b) a discrete state such as 'on' or 'off'. For the first kind, we could use an automaton that scans over the power telemetry and synthesizes an event by utilizing a sliding window over the telemetry. This event can be acted on by the hub. In the second case, if the ACU already reports a discrete state, then the automaton can either forward this state to the hub or it can also run a window over the state to synthesize higher level events.

These automatons are registered with the vCache before the policy enforcement cycle starts and are subscribed to an ephemeral table created for that ACU. So, when an ACU receives telemetry from its entity, the ACU's publisher will publish that telemetry into the vCache under the ephemeral table for that particular ACU. Then the automaton subscribed to that table will then do the following: 1) grab the data from the table, 2) process it into a discrete event according to its internal programming (decided or selected by the user), and 3) send said event to the event processing module.

2) The Event Processing Hub: Adjacent to the vCache is our event processing module, that is responsible for making decisions on what actions should be executed as well as forwarding events from the cache to the cloud. The overall architecture for the hub is shown in Fig. 6.

The hub itself consists of two primary components, the first of which is the device manager – the module responsible for performing routine administrative tasks for the NoT it manages. These routine administrative tasks include tasks such as adding or removing ACU's, alterations to an ACU's policy, and setting up the ACU with the event processing server. On startup the device manager registers the hub with the Compute Service, then starts the event processor's main loop. From here, the device manager will then start the cache instance, create all of the ephemeral tables for each ACU registered with it, load the automatons given by the user and send them to the

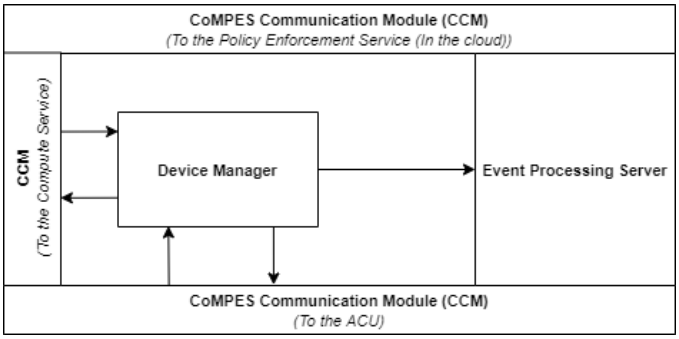

Fig. 6. The Event Processing Hub's Architecture

event processing server to be registered with the cache. When a new ACU connects to the hub the device manager will load the configuration file for that ACU and send it to the ACU as well as setup an SRPC connection between the ACU and the event processing server.

The second component of our event processing module is the event processing server – the module responsible for implementing and maintaining the policy enforcement cycle. The event processor's main job is to receive events from the vCache, send them to either its own on-board AI algorithm or to the cloud to obtain the corresponding action that should be executed, then forward that action to the corresponding actuator.

One of the main contributions of this paper to the hub, aside from the ACU redesign, lies in the addition of a reactive AI algorithm and the vCache to the hub to facilitate fast decision making. As stated in the previous section, the central goal of the CoMPES platform is to offer users a way to integrate ambient intelligence into physical environments (e.g., a smart home or smart city). To accomplish this, we are proposing the use of AI in the form of a Dynamic Reactive Agent (DRA) to solve the core problem of operational policy enforcement discussed in great detail in our previous work [5].

The DRA is a novel take on the classic reactive agent [26], with changes to its composition and focus to make it useful to CoMPES. As such, the basic idea of the DRA is similar in that it takes in some input data, matches it against a set of known behaviors, and then outputs the actions our actuators would need to take. The key differences are in its composition. The DRA relies primarily on a special table called the Operational Policy Definition (OPD) instead of the typical if-else chain. This allows the DRA to be programmable to fit the specific environment defined by the user as well as maintain constant time complexity for triggering a rule [6]. The benefit of this is significant compared to the linear or quadratic run times that typical reactive agents take to trigger a rule in rule-based CPS [5].

In Algorithm 2, we show the main algorithm the DRA uses to make decisions. The DRA starts by grabbing an update event from the vCache. Then it processes the event to pull out useful information such as the ACU's name and state. Once this is completed, the DRA then updates the State

Algorithm 2: Dynamic Reactive Agent (DRA)

Result: Set of actions **Act**
SRI = A lookup table containing the current states of all entities under this hub
ACUs = An Associative array containing information for each ACU
OPD = The operational policy for all actuators under this hub
while *event loop is active* **do**
  On update from any entity E:
  A ← Null
  SRI[E.name] ← E.state
  foreach *Actuator A in* ***ACUs*** *with a Link to E* **do**
    cohesiveState ← **SRI**[A.name]
    foreach *Entity* E_2 *with a Link to A* **do**
      cohesiveState += **SRI**[E_2 .name]
    end
    Act += (A.name, **OPD**[**cohesiveState**])
  end
end

Representational Image (SRI), an ordered textual snapshot of every ACU's last known state, using the entity's name and proceeds to the main decision-making loop. In this loop, the DRA needs to cycle through every actuator in the network in order to find out if it needs to send actions to that actuator. The process for making this decision is as follows:

1) Check if the updating ACU is contained in the actuators map of semantic links. If not, move to the next actuator. If it is then move to step 2.
2) Loop through every entity this actuator is linked to and do the following:
   a) Build an array of current states (the Cohesive state) for every linked ACU using the inner array from the links map.
   b) Join this array using commas as the delimiter to create a key for the OPD.
   c) Hash it and grab the action from the OPD.
   d) Send this action back to the processing server to be forwarded to the actuator.
3) Repeat until the exit flag is given.

This process allows the DRA to respond more quickly to events than the old TDT algorithm. This is primarily because the DRA does not need the "learning period" that TDT needs since events from the vCache will always be directly hashed with the OPD and on misses we simply ignore the event since the user has not specified an action to be taken. In this way, the DRA is a central part of our CEP framework since it makes decisions based on the events synthesised in the vCache.

IV. Experimental Setup and Results

In this section we will discuss our experimental setup and provide an analysis of the results from a series of tests in both the real world and in simulation. The application domain for our experiments is the smart home/home automaton domain with an eye toward applicability in other domains. The real-world and simulation experiments consist of arranging a collection of ACUs into a smart room and measuring the performance of the CoMPES system according to our performance criteria. For our test case, a multi-area environment is unnecessary since communication between area's is handled by the cloud. As such we would see no difference in performance over our previous work. The performance criteria for these experiments are as follows:

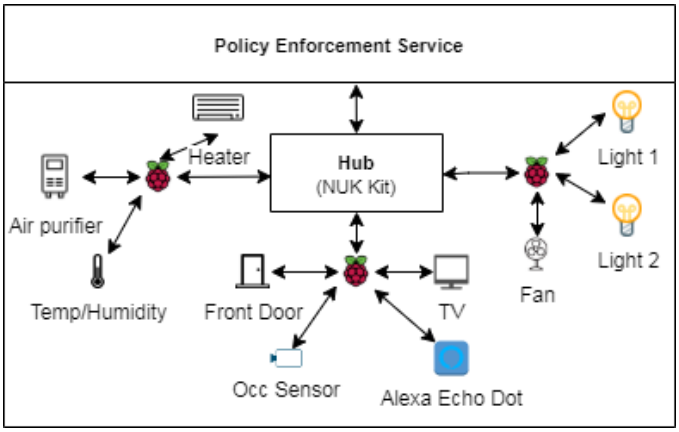

Fig. 7. The layout of the devices used in our real-world experiment. The devices are not connected to each other but because of IDC [2] they behave like they are.

- *Latency of Transitional State Change (TSC Latency)* - The CoMPES platform was designed to enable users to build autonomous, intelligent, physical environments. As such one the primary metrics with which we are concerned is how quickly the system responds to environmental stimulus. A transitional state consists of some stimulus or event that serves as an impetus for the physical environment to change its state from E_i to E_{i+1}. For instance, in a smart home setup, the temperature rising from 65 to 70 could serve as a stimulus for a CPS to turn on the air conditioner. The latency of a TSC is measured from the time the virtual entity receives telemetry to the time that the corresponding actuator's virtual entity receives an action. We chose this interval because we have no control over how long it takes to send or receive communications with the entity itself as this is decided by the manufacturer.
- *Scalability* - This metric concerns the number of transitional states that can be processed per second by the hub. This allows us to gauge the number of ACUs a user could connect to a single hub before it incurs significant increases in latency caused by congestion over shared resources.
- *Connection type* - We also tested different methods of connecting to the hub. For this we connected the ACUs to the hub using either Ethernet or WiFi.

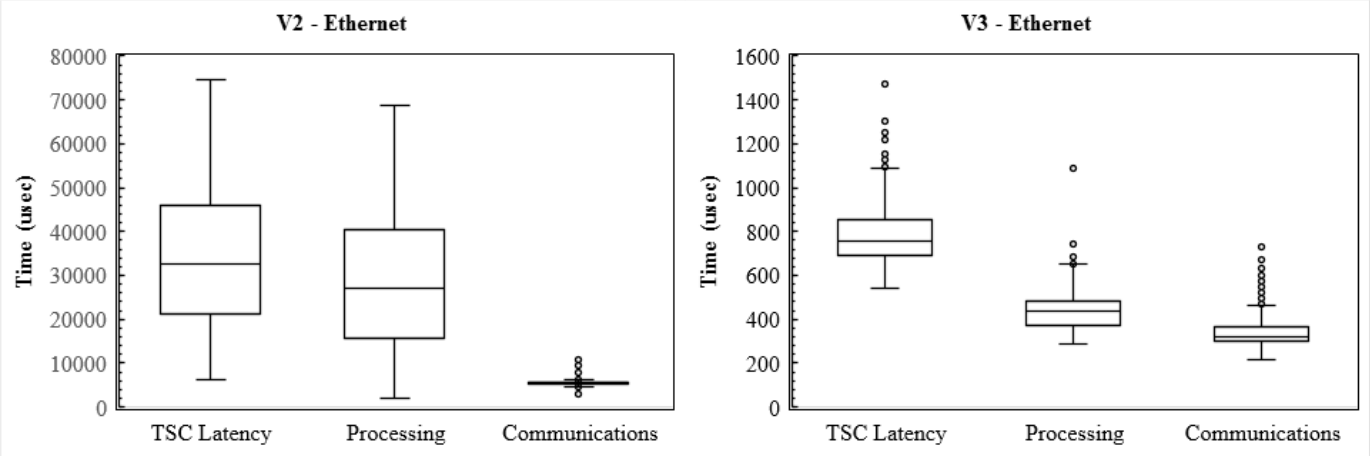

Fig. 8. This figure shows the real-world performance results for CoMPES v2 and v3 in microseconds over Ethernet. The TSC Latency shows the distribution of latency's for each of the 1000 Transitional State Change's (TSC) performed in the experiment. Processing shows the distribution of processing times per TSC, likewise for communications.

In order to perform a consistent comparison between our original work and our improved version, we performed the same experiments with both versions using CoMPES v2.0's cloud architecture and on the same physical/simulated environments.

A. Real-world experimental design

The the design of our real-world experiment is as follows: The processing hub was implemented on an Intel® NUC Kit NUC6i7KYK (2.6 Ghz, 4-core CPU), with 16 GB RAM, running Linux Mint 20. Connected to the processing hub (either by WiFi or Ethernet) were 3 Raspberry Pi 4's. Each Raspberry Pi ran a collection of ACUs based on a entity connection archetype: one for WiFi, Bluetooth, and X-10, one for Zigbee, and one for Z-Wave. The entities for this experiment are a mix of typical smart things currently available on the market, DIY projects that can be built by a user, and legacy devices with no IoT compatibility. The ten devices chosen for this experiment are as follows:

- Lamp 1 - A table lamp connected to an X10 lamp module, which is then connected to the raspberry pi via a Firecracker PC interface.
- Lamp 2 - A lamp with a Kasa/TP-Link LB130 WiFi light bulb.
- Front Door - An Aeotec Door/Window contact sensor.
- Desk Fan - A Honeywell desk fan connected to a Kasa WiFi smart plug.
- Occupancy Sensor - a Raspberry Pi with a camera and a motion detector running our own person detection/gesture recognition software [4]; when the pi detects motion it runs the images through a neural network that detects human shaped objects.
- Air Purifier - A Levoit LV-PUR131S air purifier.
- Voice control - Alexa Echo Dot; with skills that send recognized commands to the pi.
- Heater - Off brand Ceramic indoor Heater.
- Temp/Humidity - Aqara Temperature/Humidity Sensor
- Samsung Smart TV - Controlled with a Samsung universal remote library for Python.

In addition, we made a policy focused around device interactivity. For instance, for our Kasa WiFi light bulb, we linked it to the occupancy sensor with a semantic link and gave it a policy such that if the user were to enter the room then the light would turn on. If the user left the room the policy stated that the light would turn off. To implement this the hub was given an Operational Policy Definition (OPD) with the following associative-behavioural rules:

- "Off,1" : "Turn_On"
- "On,0" : "Turn_Off"

This OPD would turn on the lights when a person entered the room and off when they left. The first word in the rule is the state of the actuator (the light bulb) and the second is the number of people detected by the occupancy sensor (1 in this case). A policy like the one above was constructed for every actuator and given to the hub. Fig. 7 shows a graphical representation of our setup.

1) Real-World Experimental Results: For the real-world experiment, we let the setup run until 1000 Transitional State Change (TSC) cycles per actuator was observed while we interacted with the devices in a manner typical to home use (i.e. we would do things like enter the room and watch tv, etc). While the experiment was running we would passively measure the TSC latency and collect the telemetry for the simulation experiment. After the devices reach 100 TSC cycles, we changed the connection types for all the Raspberry Pi's and ran it again.

As can be seen in Figs. 8 and 9, the benefits of using our new edge computing framework, in regards to performance, cannot be understated. The best case for both versions is over Ethernet where the new version of CoMPES performs a whole cycle at around 799 micro-seconds while the old version did the same thing in approximately 34 milli-seconds. This is

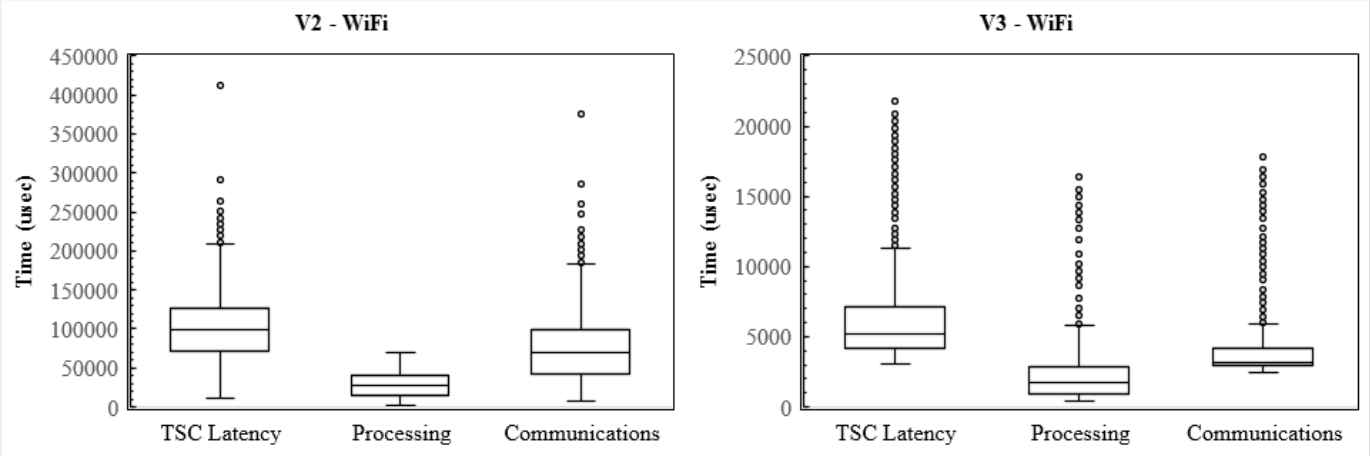

Fig. 9. This figure shows the real-world performance results for CoMPES v2 and v3 in microseconds over WiFi. The TSC Latency shows the distribution of latency's for each of the 1000 Transitional State Change's (TSC) performed in the experiment. Processing shows the distribution of processing times per TSC, likewise for communications.

around a 43x speedup over the previous version. We see even greater performance increases when you compare the other performance cases for the old version. However, even over WiFi we observed a roughly 10-20x speedup when using the edge computing framework.

In Figs. 8 and 9, we can see the distributions of the performance results for the real-world experiment for CoMPES v2 (without the CEP framework) and CoMPES v3 (with the CEP framework) by connection type. The performance was measured in micro-seconds for both trials. In both figures, the "TSC Latency" column shows the distribution of the observed latency's for each of the 1000 TSC cycles performed in the experiment. The "Processing" and "Communications" columns show the distributions of time spent processing or passing messages per TSC. These two columns give us an explanation as to what CoMPES is spending it's time on during the TSC.

As can be seen in Fig. 8, a majority of the TSC latency in v2 is caused by the "Processing" column. This processing is a combination of the time spent doing pre-processing for the Telemetry Data Transformation (TDT) algorithm, TDT itself, as well as post-processing for the policies generated by TDT. Here we can clearly see the benefit of the new AI, as the average processing times were reduced from roughly 30,000 usec to under 500 usec.

In v2, there is a large amount of variance as well as significant number outliers which is primarily due to v2's costly "learning period". During this period, the system needs to build the rule base as well as construct the hash table for it's operational policy. It does this by waiting until it encounters a TSC it has not seen before, then it looks for an appropriate rule to trigger, and finishes by building a hash entry for that TSC based on the rule. This process takes a long time (around 100-300k usec for the cache miss case with up to 4 seconds for the build case). The AI in v3 completely gets rid of this entire process by relying on the Homework vCache to synthesize only the events that it knows (i.e. the events that it has an operational policy for). So it only encounters the best run case from v2.

Another thing we can see from the figures, is the benefit of the new event system shown in the "Communications" column. In Figs. 8 and 9, part of the reason v3 is much faster than v2 due to the Pub/Sub nature of ACU communications in v3. In v2, the hub ran on a cycle and would poll each ACU for it's current state before building an image of these states to send to the cloud. This incurred additional latency since even if the ACU had reported it's state at time *t*, the hub may not report it to the cloud until time *t+1* when it sends up the next image. With our new Pub/Sub system and ACU's telemetry is immediately sent upstream to be processed by either the hub or the cloud with no polling cycle necessary. This difference is even more apparent when we ran the experiment over WiFi. When using WiFi the time spent in message passing clearly outpaces the time spent in processing. In v2, this makes up the majority of TSC latency seen in Fig. 9. Of course, both versions suffer latency spikes when we switch to WiFi. However the chief difference between the two is that at approximately 100 milliseconds the old version cannot handle more than 10 messages a second. Whereas the new version can handle up to 100 messages a second on WiFi.

One thing to note on the distribution of latency's in v3 as seen in Fig. 9; we would see a rare jump in latency of around 20 milli-seconds that would happen once a minute or so. We are currently investigating the reason for these odd jumps in latency, but is is our current belief that this can be attributed to issues with the way the underlying Linux system manages WiFi communications. However, at 20 milliseconds this extra latency is far beyond the human capacity to recognize and well within the constraints of "near-real time". The majority of the distribution was around 5 milliseconds.

B. Simulation Experiment

For the simulation experiment we followed the general layout of the real world experiment with a notable few changes. In the real world experiment we collected the states recorded from the entities into CSV files that could be read by virtual entities. We arranged the virtual entities in the same layout as the real-world ones with the same policies except we added a mass client to the environment. The mass client provides a multi-threaded environment so that we can run many ACUs from a single computer. We then scaled the number of ACUs provisioned and running in the system. This allowed us to do our scaling experiment without having to purchase thousands of entities. And since we do not measure the time to get the message from the virtual entity to the physical entity (we have no control over that) the two are basically the same performance-wise. The mass-client starts by telling each ACU under it to send telemetry in a round robin fashion. After the telemetry is sent it waits for one second and then sends the next round. This process is done for 1000 rounds with the number of ACUs scaling up to 1000 in groups of 100. In Fig. 10, we show the results of this experiment for each version.

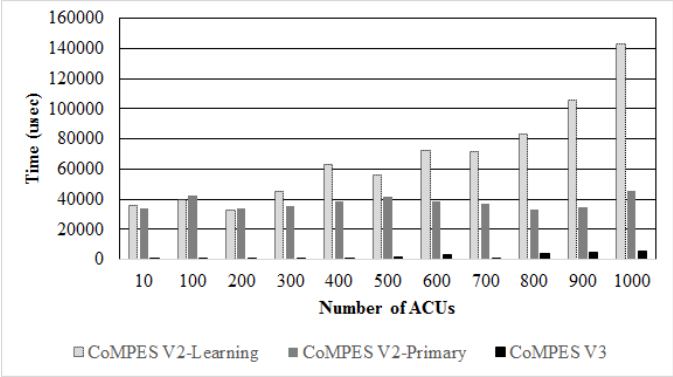

Fig. 10. The performance results of our scale experiment for CoMPES V2 and V3. The y-axis lists the time measured in micro-seconds and the x-axis is the number of ACUs in the trial.

In Fig. 10, we show the average performance of both versions as the number of ACUs under one hub is scaled from 10 to 1000. The new version honestly surprised us. We scaled it up to 1000 devices and it still maintained average latency's under 20 milliseconds. However there is a caveat to the higher scaling values; In v2, the performance was primarily lowered by the secondary (cache miss) case and an increasing component of v2's performance could be attributed to the effects of congestion. In a home setting, the poorer performance of v2 is of no real consequence since even at 1000 devices the average TSC latency was under 150 milli-seconds (roughly at the lower bound of human recognition). As such, the user would perceive any action taken by the CPS as or near instantaneous. However, for other application domains such as industry, healthcare, or defense this would be unacceptable. V3 maintains latency's in the low 10's of milliseconds with no noticeable jump in latency's. Of course, if the CPS had a hard requirement to service events the instant they are received, v3 would be able to handle 100 devices over WiFi or 1000 devices over Ethernet per hub while v2 could only handle roughly 2-10 devices in either case. If the user allows some queuing or additional latency then v3 would ensure that each device's telemetry was utilized within 20 milli-seconds over WiFi.

V. Conclusion

In this paper, we presented our new edge computing framework for our CPS/IoT platform CoMPES. For this research, we hypothesized that by integrating our new edge computing/CEP framework into the CoMPES platform, we would address the following research questions:

- Could introducing an event processing framework at the edge of our CPS/IoT system significantly increase scale and reduce the overall latency?
- Would "Entity Virtualization" address the challenge of entity heterogeneity in an expanded CPS/IoT Ecosystem?
- Would altering the policy enforcement cycle to be more reactive lead to significant decreases in latency?

To address the first and third question, in our experiments we achieved a 20-43x speedup over the version of CoMPES that was not using the edge computing framework along with a roughly 10-100x increase in scale. This performance increase can be attributed to three factors: 1) in v3, we are closing the loop closer to the end devices, 2) we altered the operational policy enforcement algorithm to include a reactive agent that makes decisions more rapidly, and 3) we introduced a new Pub/Sub CEP architecture that processes telemetry into events more quickly than the old interpretation scheme did. Over Ethernet, a significant portion of the TSC was spent processing. By adding our Dynamic Reactive agent in addition to the CEP framework, we reduced the time spent processing telemetry from 30,000 usec on average to 500 usec on average over Ethernet. We can thus say with confidence that adding an edge computing framework to CPS/IoT control systems should result in decreased latency and increased scale.

We addressed our second research question by the addition of a new ACU design. The ACU is now it's own entity with it's own processing capacity. This allows it to more easily "map" to physical entities in the real world. The ACU was specifically re-designed to offer users and manufacturers greater control over how their devices communicate with the CPS/IoT system. This was achieved by relying on programmable automatons [24] that scan over entity telemetry and synthesize events. Together with the new ACU design users can now either select an appropriately programmed ACU or build their own for their entities. These ACUs can be made for virtually any kind of entity so long as it can either interact and/or observe said entity.

These alterations have multiple benefits. First is the concern over privacy: since the hub can be running over the users own hardware at their location, data no longer needs to be shipped to the cloud unless it concerns entities in another area. This offers the user an increased degree of control over what

information is actually sent out over the internet. Second is that with tighter loops users are free to add critical systems or other features (such as additional security) to the TSC cycle and still meet their real-time bounds.

For future work, we will be investigating whether similar changes to way we process telemetry in the cloud side of CoMPES would have similar effects. We are also interested in investigating the security and privacy aspects of CPS/IoT systems in general and CoMPES in particular. There is also a need to see how to apply CPS/IoT control systems like CoMPES to other application domains such as agriculture, healthcare, and the "smart city".

REFERENCES

[1] D. W. C. Greer, M. Burns and E. Grior, "Cyber-physical systems and internet of things," Tech. Rep. NIST.SP.1900-202, National Institute for Standards and Technology, 2019.

[2] J. Hall and R. Iqbal. 2017. "CoMPES: A Command Messaging Service for IoT Policy Enforcement in a Heterogeneous Network," In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation (IoTDI '17). Association for Computing Machinery, New York, NY, USA, 37–43. DOI:https://doi.org/10.1145/3054977.3054988

[3] R. Iqbal, J. Lee and J. Hall, "A Cloud Middleware Enabling Natural Speech Analysis for IoT Policy Enforcement in Smart Home Environments," 2018 IEEE International Congress on Internet of Things (ICIOT), San Francisco, CA, 2018, pp. 184-187, doi: 10.1109/ICIOT.2018.00035.

[4] R. Iqbal, J. Hall, J. H. Lee, A. Islam, "Enabling real-time audio-video inputs for Internet of Things operational policy enforcement," Internet of Things, Volume 6, 2019, 100041, ISSN 2542-6605, https://doi.org/10.1016/j.iot.2019.02.001.

[5] Hall, J, "The Generation of Operational Policy for Cyber-Physical Systems in Smart Homes" (2019). MSU Graduate Theses. 3441. https://bearworks.missouristate.edu/theses/3441

[6] M. C. V. Y. Tan and S. Goddard, "Spatio-temporal event model for cyber-physical systems," in 29th IEEE International Conference on Distributed Computing Systems Workshops, pp. 44-50, IEEE, 2009.

[7] R. Baheti and H. Gill, "Cyber-physical systems," tech. rep., IEEE Control Systems Society, February 2011.

[8] Sheeraz A. Alvi, Bilal Afzal, Ghalib A. Shah, Luigi Atzori, Waqar Mahmood, Internet of multimedia things: Vision and challenges, Ad Hoc Networks, Volume 33, 2015, Pages 87-111, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2015.04.006.

[9] S. M. R. Islam, D. Kwak, M. H. Kabir, M. Hossain and K. Kwak, "The Internet of Things for Health Care: A Comprehensive Survey," in IEEE Access, vol. 3, pp. 678-708, 2015, doi: 10.1109/ACCESS.2015.2437951.

[10] Luigi Atzori, Antonio Iera, Giacomo Morabito, Michele Nitti, The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization, Computer Networks, Volume 56, Issue 16, 2012, Pages 3594-3608, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2012.07.010.

[11] M. Asif-Ur-Rahman et al., "Toward a Heterogeneous Mist, Fog, and Cloud-Based Framework for the Internet of Healthcare Things," in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4049-4062, June 2019, doi: 10.1109/JIOT.2018.2876088.

[12] Q. Zhu, R. Wang, Q. Chen, Y. Liu and W. Qin, "IOT Gateway: Bridging Wireless Sensor Networks into Internet of Things," 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Hong Kong, 2010, pp. 347-352, doi: 10.1109/EUC.2010.58.

[13] Datta SK, Bonnet C, Nikaein N (2014) An IoT gateway centric architecture to provide novel M2M services. In: IEEE World Forum on Internet of Things (WF-IoT) pp. 514-519. doi: 10.1109/WF-IoT.2014.6803221

[14] V. C. Emeakaroha, N. Cafferkey, P. Healy and J. P. Morrison, "A Cloud-Based IoT Data Gathering and Processing Platform," 2015 3rd International Conference on Future Internet of Things and Cloud, Rome, 2015, pp. 50-57, doi: 10.1109/FiCloud.2015.53.

[15] C. Tang, Z. Tang, Y. Yang and Y. Zhan, "WSID identification platform of heterogeneous networks based on RFID and WSN," 2010 IEEE International Conference on RFID-Technology and Applications, Guangzhou, 2010, pp. 217-221, doi: 10.1109/RFID-TA.2010.5529935.

[16] Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M (2015) Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. In: IEEE Communications Surveys & Tutorials vol. 17 4:2347-2376. doi: 10.1109/COMST.2015.2444095

[17] V. K. Sehgal, A. Patrick and L. Rajpoot, "A comparative study of cyber physical cloud, cloud of sensors and internet of things: Their ideology, similarities and differences," 2014 IEEE International Advance Computing Conference (IACC), Gurgaon, 2014, pp. 708-716, doi: 10.1109/IAdCC.2014.6779411.

[18] M. Olson and K. M. Chandy, "Performance Issues in Cloud Computing for Cyber-physical Applications," 2011 IEEE 4th International Conference on Cloud Computing, Washington, DC, 2011, pp. 742-743, doi: 10.1109/CLOUD.2011.118.

[19] Bonomi F., Milito R., Natarajan P., Zhu J. (2014) Fog Computing: A Platform for Internet of Things and Analytics. In: Bessis N., Dobre C. (eds) Big Data and Internet of Things: A Roadmap for Smart Environments. Studies in Computational Intelligence, vol 546. Springer, Cham. https://doi.org/10.1007/978-3-319-05029-4_7

[20] M. A. López Peña and I. Muñoz Fernández, "SAT-IoT: An Architectural Model for a High-Performance Fog/Edge/Cloud IoT Platform," 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 2019, pp. 633-638, doi: 10.1109/WF-IoT.2019.8767282.

[21] A. Mijuskovic, R. Bemthuis, A. Aldea and P. Havinga, "An Enterprise Architecture based on Cloud, Fog and Edge Computing for an Airfield Lighting Management System," 2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW), Eindhoven, Netherlands, 2020, pp. 63-73, doi: 10.1109/EDOCW49879.2020.00021.

[22] D. Wang, N. Zhao, B. Song, P. Lin and F. R. Yu, "Resource Management for Secure Computation Offloading in Softwarized Cyber-Physical Systems," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2021.3057594.

[23] Jun Kim, Ju Yeon Lee, Server-Edge dualized closed-loop data analytics system for cyber-physical system application, Robotics and Computer-Integrated Manufacturing, Volume 67, 2021, 102040, ISSN 0736-5845, https://doi.org/10.1016/j.rcim.2020.102040.

[24] Sventek J., Koliousis A. (2012) Unification of Publish/Subscribe Systems and Stream Databases. In: Narasimhan P., Triantafillou P. (eds) Middleware 2012. Middleware 2012. Lecture Notes in Computer Science, vol 7662. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35170-9_15

[25] Richard Mortier, Tom Rodden, Peter Tolmie, Tom Lodge, Robert Spencer, Andy crabtree, Joe Sventek, and Alexandros Koliousis. 2012. Homework: putting interaction into the infrastructure. In Proceedings of the 25th annual ACM symposium on User interface software and technology (UIST '12). Association for Computing Machinery, New York, NY, USA, 197–206. DOI:https://doi.org/10.1145/2380116.2380143

[26] Nils J. Nilsson, "Agents That Plan," Artificial Intelligence: A New Synthesis, Morgan Kaufmann, 1998, Pages 117-127, ISBN 9781558604674, https://doi.org/10.1016/B978-0-08-049945-1.50013-7.