Experiences Creating a Framework for Smart Traffic Control using AWS IOT

William Tärneberg  
Department of Electrical and Information Technology  
Lund University  
Lund, Sweden  
william.tarneberg@eit.lth.se

Vishal Chandrasekaran  
Department of Computer Science  
University of Virginia  
Charlottesville, VA, USA  
vc4td@virginia.edu

Marty Humphrey  
Department of Computer Science  
University of Virginia  
Charlottesville, VA, USA  
marty.humphrey@gmail.com

ABSTRACT

Public clouds such as Amazon Web Services (AWS) and Microsoft’s Azure provide excellent capabilities for scalable Web applications and Hadoop-based processing. Recent additions to public clouds to support connected devices and IoT have the potential to similarly disrupt emerging homegrown and/or proprietary approaches. While early public cloud IoT success stories have focused on smaller-scale scenarios such as connected houses, it is unclear to what extent these new public cloud mechanisms and abstractions are suitable and effective for larger-scale and/or scientific scenarios, which often have a different set of constraints or requirements. In this paper, the design and implementation of a representative cloud-based IoT infrastructure in a specific public cloud – AWS – is presented. The system created is for dynamic vehicle traffic control based on vehicle volumes/patterns and public transport punctuality. We find that constructing server-less, stateful, and data driven IoT applications in AWS that can operate in real-time is non-trivial. The primary challenges span application manageability and design, latency performance, asynchronicity, and scalability.

CCS Concepts

- General and reference → Evaluation;

Keywords

Cloud, AWS, IoT, PaaS, Public transport, Traffic, Traffic Signal Priority, Smart Cities, Connected vehicles, Automatic control, Data collection, Data analysis

1. INTRODUCTION

A long-held goal of many scientific projects has been real-time manipulation of sensing devices and actuators. For example, a system to monitor animals could be constructed with a lattice of mobile sensing devices, and the discovery of a particular animal could alert a scientist to dynamically reconfigure both sensor positions/techniques and gating/routing mechanisms to improve data collection and/or improve the animals’ lives. However, in general because of cost and technology constraints, such devices usually lack network capabilities and thus are unable to be read/controlled in real-time. Instead, typically scientists are forced to manually collect the data some time after the fact. This data is then analysed on lab computers, perhaps leading to modifications to the sensing algorithms/positions and control logic used in subsequent re-deployments. These constraints have clearly significantly limited the rate of scientific progress.

Two recent developments have brought such real-time science scenarios closer to reality. First, device manufacturers have begun to believe that there is a market for network-enabled devices in many scenarios (e.g., a “connected” light bulb for the home). Such cost reductions and broadly-available capabilities can be extended or applied to scientific devices, thereby significantly reducing the cost to deploy such devices. Second, public clouds have recently added specific support for Internet of Things (IoT). For example, in October 2015, Amazon Web Services (AWS) introduced AWS IoT, to “support billions of devices and trillions of messages”. While these two developments are promising, there are a number of open issues that must be addressed before large-scale scientific experiments based on real-time sensors and actuators are feasible. Long-lasting power supplies and the availability of networking infrastructure (e.g., cell towers) for devices/actuators must be addressed in general. Additionally, while early public cloud IoT success stories have focused on smaller-scale scenarios such as connected houses, it is unclear to what extent these new public cloud mechanisms and abstractions are suitable and effective for larger-scale and/or scientific scenarios, which often have a different set of constraints or requirements.

This paper addresses the challenge of implementing a scalable IoT infrastructure tested in the public cloud for scientific experimentation. There are two main contributions of this paper. The design and implementation of a representative cloud-based IoT infrastructure in AWS is presented and the evaluation of that system. The system created is for dynamic vehicle traffic control based on vehicle volumes/patterns and public transport punctuality. Specifically, in-road induction sensors, and vehicle GPS positioning comprise the input to a control algorithm to regulate the red-green patterns of traffic lights with the goal of increasing safety and minimizing wait/idle times. The targeted...
system operates in real-time and is both data-driven and stateful. In this first phase, the system has been designed and implemented based on a simulated system and sensors; as the system matures, these simulated sensors can be replaced seamlessly with real sensors that report to the cloud, without needing to modify any of the control logic. We found three primary non-trivial challenges when developing such an application in AWS:

- Designing and defining scalable stateful data-driven IoT services that operate asynchronously with real-time constraints.
- Long tail latency performance barriers.
- Practically managing and extending the infrastructure components in a scalable manner.

This paper is structured as follows. Section 2 discusses related work and the suspends the research gap. Section 3 details the requirements and properties of the targeted system. Section 4 describes the infrastructure’s components and presents the design. Section 5 presents the evaluation, and Section 6 concludes.

2. RESEARCH GAP

In this paper, a smart city-like traffic control service is used to evaluate the feasibility of deploying a scientific testbed in the cloud. Below we review the state-of-the-art in the field of traffic control and how this paper contributes to that field.

In [4], Djahel et al. present the requirements of a future traffic management system in the IoT era. These requirements include providing real-time road traffic simulation and visualization to help authorities more efficiently manage the road infrastructure and ensuring integration of existing systems and new technologies, and managing the evolution of these systems. In [5], Gradinescu et al. present an adaptive traffic light system based on wireless communication between vehicles and fixed controller nodes deployed at intersections. They prove that total time delay experienced at intersections can be significantly reduced using their system. In [7], Hu et al. propose an intelligent Transport Signal Priority logic based on connected cars. Transit Signal Priority (TSP), also referred to as bus priority, is a collection of techniques that provide preference to transit buses at intersections. By adjusting the traffic signal plan according to bus arrivals, the delay that transit buses experience at intersections is reduced. This system aims to improve the overall transit service quality.

The above findings have to various degrees been evaluated either in a simulator or small real-world systems. This is to our knowledge no work done on how such policies and systems interact with tangent systems in a Smart City or at scale. Additionally, neither the proposed systems nor the evaluations take much consideration to the supporting sensor and orchestration platforms that would be required in a real-world deployment.

Inspired by the past work in smart traffic management, in this paper, we look at a cloud-based infrastructure to support such systems. In particular, we examine the use of contemporary cloud services and platforms to scale experiments and create a hybrid simulated and real-world experiment environment.

Our work aims to leverage AWS IoT services to build the cloud infrastructure necessary for such a forthcoming traffic management system. We evaluate the effectiveness of using AWS IoT, present the challenges we faced and suggest improvements for the future.

3. TARGETED SYSTEM

In smart cities, a Traffic Signal Control (TSC) system incorporated into a general traffic and public transport sub-system will employ a wide range of sensor types with heterogeneous availability, data types and quantities, and outputs. The aggregate system state is represented by the state of the individual connected vehicles and devices as well as historic and real-time data external and internal to the system. Several parallel, event-driven, real-time, and periodic processes will orchestrate devices and vehicles, collate and aggregate data, and provide feedback control based on the system’s objectives. The system’s many objectives, such as actuating traffic lights to meet a certain deadline are in their nature, real-time. Accommodating a data-flow to achieve real-time decisions in a distributed system at scale is a challenge on its own. Ingress data to such a system and its various processes is arguably heterogeneous both in terms of volume, velocity, variety, and veracity.

A scientific IoT TSC testbed will need to be able to orchestrated both real-world as well as virtual objects scalably and in real-time [10]. A system state shall be able to be defined by any subset of the system’s inputs and states. Processes, administrative actions, and system states shall be able to be triggered by devices and vehicles and by observing data flows. Data generated by the system and its constituent components therefore needs to be made available in real-time to those processes and states.

The scale of the set-up and the duration of the experiments varies from experiment to experiment and even during runtime. The system therefore needs to be able to scale to a large number of devices without affecting the real-timeliness of the control processes nor limit the number of concurrent scientific data analysis processes. It is also desirable for a scientific team not to have to commit to, develop, and maintain their infrastructure.

This paper begins to address this challenge by evaluating and exploring the possibility of using an emerging cloud IoT Platform as a Service (PaaS), namely AWS IoT. To provide a platform for developing and evaluating such a system we employ the TSP solution proposed in [6]. In the remainder of this section, the system components are described and then the system properties are enumerated.

3.1 System components

The TSPCV system presented in [6] relies on connected public transit vehicles and wireless TSC sensors [9] to fulfill its objective of reducing mean commuter waiting times. The paper proposes to do so by manipulating the en-route traffic lights to allow the en-route buses to maintain their schedules while considering their current ridership and the impact on peripheral traffic in the affected intersections.

More specifically, the triggers in such a system are realized by sensors in bus stops reporting the arrival of buses. In parallel, buses report their location, speed, and ridership. The state of the en-route traffic lights can be manipulated and observed in real-time. The punctuality of each bus is monitored by a process that compares the reported arrival of
buses at bus stops and a set time table. A change in the traffic light program will be considered when the bus approaches the intersection and is behind schedule. The evaluation process queries the punctuality process, the state of the affected traffic light, and the prevailing peripheral traffic conditions from the nearby induction loops, cameras, or collaborative connected vehicles [8]. The resulting control decision is then relayed to the affected traffic light(s). The consequences of that decision will be picked up by separate processes that continually produce, analyze, store, and monitor prevailing traffic conditions.

The following components are needed to construct this system (see Figure 1):

**Sensors and actuators** The system will collect data from a large set of sensors. The sensors produce and attempt to report data at a certain rate. The sensors are geographically distributed throughout the evaluation domain. Due to intermittent connectivity, sensors might not be able to successfully report at the desired rate. Additionally, sensors report data with a certain error. The traffic lights are the system’s actuators. A traffic light can be queried for its last reported state and the state can be changed with a control signal.

**Device orchestration** The devices, sensors and actuators are orchestrated in such a manner that enable them to register, validate, and securely communicate with the system.

**Scientific analysis** To determine the effectiveness of a strategy, analysis is performed on the data. The set of running analyses can be changed in runtime to reflect the ongoing experiments. The intent and output of analysis can either be preprocessing for the controller or scientific evaluations of the system.

**Storage** The data collected by the devices, the control decisions, the current state of the systems processes and entities, and the analysed data is recorded and stored indefinitely for concurrent and future analysis.

**Controller** The system operates a set of controllers that act on multiple inputs from both real-time, stored, and preprocessed sensor data to each produce a set of control signal that is then relayed to their constituent traffic lights.

**Monitoring** The performance and availability of the system’s components and devices are monitored in runtime.

---

**Figure 1: Targeted system and application scenario**

**Figure 2: System components of the targeted system**

The components of the targeted system are illustrated in Figure 2.

3.2 System properties

The targeted scientific IoT testbed has the following properties and requirements.

**Real-time** The system shall be able to forward, process and store information as well as run control loops in real time.

**Multiple input** The system shall support Multiple Input Multiple Output (MIMO) by exposing the entire set of sensors to the set of controllers.

**Scalable** To cover large geographic and densely populated areas, the systems needs to scale from 10’s to 1000’s of devices with a proportional number of controllers.

**Survey-able** The system and its component’s states shall be made available to the operator of such a system.

**No Operations (NoOps)** The system shall not require active provisioning, software maintenance, nor extensive software development.

4. IMPLEMENTATION

In this section, the implementation of the targeted system using AWS components is presented. AWS was chosen because it has at this point in time the most comprehensive portfolio to construct a scalable, extensible, and NoOps cloud IoT infrastructure. Additionally, the AWS service offering allows the architecture to bridge real and simulated paradigms by interacting with both virtual and physical devices. The relevant individual AWS components are first described, followed by how they were used to construct the real-time vehicle control system.

4.1 AWS Components

To realise the components and requirements of targeted system, the following AWS are employed.

4.1.1 IoT

AWS IoT is a platform that enables connected devices to securely communicate and relay information to and from the AWS platform using Message Queue Telemetry Transport (MQTT) [2]. Alongside ZigBee [1], MQTT has become one of the prevailing home-IoT messaging protocols. In addition to message passing, AWS IoT also offers data stream endpoint connectivity and message routing from a simple stateless rule engine with an Structured Query Language (SQL)-like syntax. Simple system logics can be achieved with rules, using stateless thresholds and trigonometric functions.
To connect a physical infrastructure of devices to the AWS IoT platform, the accompanying AWS IoT Software Development Kit (SDK) supports a number of embedded systems such as Embedded C and Arduino Yún. Additionally, AWS IoT supports RESTful communication to ensure that virtually any device of any capability can be connected to the system, as long as it is connected to the Internet.

An entity in AWS IoT is referred to as a Thing. AWS IoT maintains the state of each Thing, through what is referred to as a Shadow state. A Shadow state can be queried through services external and internal to AWS IoT. The relationship is maintained regardless whether the physical device connected or not. The targeted system will require a stateful logic that goes beyond the capabilities of the AWS device connected or not. The targeted system will require a relationship is maintained regardless whether the physical device connected or not. The targeted system will require a stateful logic that goes beyond the capabilities of the AWS IoT. Therefore, AWS IoT is in this work used to scalably and securely orchestrate the resident devices’ communication and authentication, in real-time. All external entities are connected to the cloud-hosted infrastructure over AWS IoT.

4.1.2 Lambda

In addition to its traditional collection of Virtual Machines, AWS is offering a highly scalable serverless microcompute platform called Lambda. A Lambda function is a state-less piece of code, with an input and an output that can be triggered from a wide array sources internal and external to AWS. In contrast to an Elastic Compute Cloud (EC2) instance, a Lambda function has one dedicated purpose and deliberately only runs for up to a few minutes. Arguably, instead of running an entire application in a single Virtual Machine (VM), it can now be broken up into a set of redundant asynchronous sub-functions. Lambda functions scale instantly to hundreds of instances, with almost no platform maintenance. In this work Lambda functions constitute all computational instances used for evaluating bus punctuality, aggregating and maintaining data in databases, sensors fusion, and traffic control loops.

4.1.3 DynamoDB

AWS offer a number of Database (DB) services. AWS DynamoDB is a low-latency No-SQL schema-based DB. In this work AWS DynamoDB is used for storing collected data and maintaining shard states.

4.1.4 Kinesis

AWS Kinesis is a highly scalable aggregating streaming data buffer. Kinesis is scalable in the sense that it can achieve a high throughput by forwarding ingress data to a practically infinite pool of parallel end-points. Its ability to maintain a high throughput and thus ensure that what is beyond the endpoints is updated in a timely manner is what makes it real-time. This property also ensures that additional end-points can be introduced non-intrusively, without interrupting or throttling the existing end-points. In this work AWS Kinesis is used to scalably aggregate and process the flow of reported sensor values. In practice, the Kinesis stream is also used as a means to expose the ingress stream of data to any future end-point or experiment.

4.1.5 CloudWatch

AWS CloudWatch is a platform for monitoring AWS services through logs as well as predefined and custom metrics. AWS CloudWatch is the primary outlet for debugging AWS services in run-time. Additionally, AWS CloudWatch provides a set of primitive plotting capabilities for monitoring existing and user-defined metrics. Cloud watch also allows the setting of time-based triggers.

4.2 Testbed Architecture

The system’s various entities such as traffic lights, buses, and induction loops are connected to the cloud infrastructure as Things via AWS IoT (see Figure 3). The communication between the sensors and the cloud infrastructure is encrypted using Transport Layer Security (TLS). The credentials for each device are stored locally on the device. This implies that each device is registered and trusted by the system. This satisfies the security and reliability requirements. Additionally, entities, or Things, can be added dynamically to the system in runtime.

Data from all traffic and bus monitoring sensors are published to a common data channel for analysis in real time. A rule is deployed as an endpoint to each upstream channel to route the message to a AWS Kinesis stream. The received messages are in their entirety deposited into the Kinesis stream. From this point on any other service can be set-up to access the incoming streaming data without altering the set-up of running AWS IoT flow, provided that they have been granted access. In other words, new data stream endpoints can unobtrusively and trivially be added. Furthermore, a channel in AWS IoT is achieved by an MQTT topic.

Both AWS IoT and Kinesis have a high enough throughput to successfully receive and route hundreds of sensor readings per second. The number of concurrent Lambda functions automatically scales to meet the number of ingress sensor values. In theory, this ensures that the information flow expedited end-to-end in near real-time.

To aggregate and demultiplex the streaming data, AWS Lambda functions are attached as end-points to the streams. To introduce a simple mean value calculation of traffic throughput, a Lambda function is employed to observer the traffic flow data and aggregate the samples from each sensor over an epoch, arguably at the scale of the duration between control loop evaluations. The values are aggregated in a AWS DynamoDB entry for each epoch and used in the analysis and classification of long term behaviours. Moreover, at the arrival of a bus at a bus stop, an event message is sent on the bus_stop_event MQTT topic containing the bus stop id, the bus number and route, and the time of its arrival. The message is intercepted by an AWS IoT rule which internally spawns a Lambda function that assesses the punctuality of that bus by comparing the ingress data with a time table in DynamoDB. The result is stored in a DynamoDB table. Subsequent Lambda functions can be added to evaluate the state of the entire route or the transit system as a whole.

The benefit of a Lambda function as compared to an EC2 instance in made clear in this scenario. The evaluation of a bus punctuality is event-driven, and does not happen continuously, but multiple evaluations might run concurrently for multiple buses. An equivalent EC2 instance would have to be run continuously, maintained, and the resident software would have to be able to scale to multiple evaluations while guaranteeing real-timeness on one machine and one socket. Lambda functions and Kinesis thus contribute to the system meeting its scalability, real-timeness, and NoOps requirements.
At this point in the system, data is accumulated in a set of DynamoDB tables that are structured in such a way that they are suitable for both long-term storage and for use by the controller. There are several ways to access the data from both AWS-internal and external services. Access privileges can be established to regulate who and what services can access to the DB.

When a bus approaches an intersection, the adjacent induction loops register the presence of a bus and report the intersection id, bus number, and the time of the event to a dedicated AWS IoT topic, *intersection_event*. The rule performs two operations – the data is forwarded to Kinesis for record keeping and further analysis. Primarily, the rule triggers a Lambda function intended to evaluate the state of the traffic lights in the affected intersection. The resulting Lambda function begins by querying the punctuality DB entry for that specific bus to determine the extent of the need to assist the bus by altering the traffic pattern. If so, then the AWS IoT *Thing* shadow states of the induction loops in the affected intersection are queried for their last reported state. In addition to forwarding the relevant data to the control loop, the state update is intercepted by an AWS IoT rule that forwards the data aggregation and processing to the Kinesis stream. The control output is acted on by sending the new states the affected traffic lights, this is accomplished by setting the Shadow State. The process is repeated for each such incident. Multiple such processes can run concurrently. Any interactions with AWS services are done through Python Boto3.

If the controller was designed to be triggered on multiple events, an intermediate state would have to be created in for example Dynamo DB. Although, Lambda function can be triggered from a variety of data source, such as an update in a table, and another Lambda function via an intermediary AWS Simple Notification Service (SNS) message, it cannot be atomically triggered based on a state composed of multiple inputs. This is because data is submitted and processed asynchronously in the system, there is no active entity in the information flow that can make the atomic decision to iterate the control loop. Founded in the fundamental concept of the Kinesis and Lambda, AWS cannot be ensured that a Kinesis entry will only be processed once. Multiple entries can be due to an error in the input or an error in the Lambda code. If AWS were able to guarantee that a table entry was atomic then multiple controller Lambda functions that each trigger on an update for a subset of reported sensors over the past epoch could be deployed. As a result the controller is therefore activated asynchronously and independent of the data source using a CloudWatch Scheduled Event. This ensures that only one instance of the controller Lambda function is called, and allows the controller to act independently of the data source.

### 4.3 Simulated testbed architecture

To validate the infrastructure and to provide a portable proof-of-concept, a simulated environment was developed based on the scenario presented in Section 3. Evaluating and experimenting with the infrastructure with actual sensors and real traffic at a real-world rate is neither safe nor allows the stressing of the limitations of the infrastructure. The simulated environment therefore employs a simulated traffic environment with virtual entities. The primary architectural discrepancies between the real-world testbed and the simulated environment is illustrated in Figures 3 and 4. The information flow and control logics in the simulator are maintained with a few additional states and signals.

The simulated traffic environment is supplied by Simulation of Urban MOBility (SUMO) [3]. SUMO is a microscopic, inter- and multi-modal, space-continuous, and time-discrete traffic flow simulation platform. A SUMO simulation scenario is at minimum specified by a network of roads, traffic-monitoring induction loops, traffic lights, and vehicle arrival rates, speeds, and entry points. SUMO is accompanied by a visualisation tool that renders the environment and the changes within it. The tool also allows enables real-time interaction with the simulation.

SUMO provides an extensible interface, Traffic Control Interface (TraCI), that allows researchers and developers to interact with the simulator, environment, and the visualisation tool over a socket in real-time. This decoupling enables external entities to control the simulator’s clock, traffic lights, and extract the prevailing traffic conditions. In this work the TraCI module is used to expose the induction loops, buses, cars, bus-stops, and traffic lights to the information flow as *Things* in AWS IoT. This is achieved by running the SUMO simulator core and an MQTT enabled TraCI python script on a Windows AWS EC2 instance. Because Boto3 cannot act as an AWS IoT *Thing* a virtual *Thing* module was developed to connect SUMO entities with AWS IoT. As a result, SUMO entities are managed by AWS IoT, have a
shadow state, and can be queried in the same manner as their real-world counterparts.

SUMO can run in real time with real world input, but to be able to scale any experiments the simulator clock needs to be centrally controlled. In this implementation, the simulator clock has therefore been made to drive time progression in the entire system. At each instance that the simulator progresses the time horizon, it sends a time-stamp on a dedicated AWS IoT MQTT topic, \textit{time\_tick}. To realize a pseudo-real-time environment all external data-producing entities subscribe to the \textit{time\_tick} topic. At each time update, any data producing entity updates its shadow state. This structure enables the system to execute at almost any rate. Being able to vary both the input rate and the rate of execution allows one to scale the experiments to load the information flow in both time and volume.

5. EVALUATION

In this section, the designed architecture is evaluated using a representative scenario based on the proposed TSP-CV from [6] detailed in Section 3. Based on this set-up, a basic cost and a latency analysis is presented.

5.1 Representative Scenario

To evaluate the designed architecture, a representative scenario was developed in the simulated environment. The scenario is based on an area of roads and intersections in Charlottesville, VA. The area incorporates two intersecting bus routes, namely the Free Trolley and Route 7. The buses follow their regular schedule and are along with all the entities connected to AWS IoT. Furthermore, the area features four intersections, each intersection implements the TSP-CV policy proposed [6]. The incorporated components and the layout of the scenario is illustrated in Figure 5.

5.2 Performance

5.2.1 Throughput and scalability

AWS IoT provides an interface to add and manage Things (sensors) deployed in the testbed. The developed infrastructure can support thousands of sensors, as AWS can scale on demand. However, the credentials required for sensors to communicate with AWS IoT have to be stored locally in each sensor. This means that an overhead is incurred each time a sensor is added to the system. Further, when the number of sensors in the system is very high, managing the sensors via the interface provided by AWS is tedious and error-prone. Thus, the infrastructure can scale to support thousands of sensors, but manually managing the sensors via the AWS IoT interface is challenging.

5.2.2 Latency

As the control loop is decoupled from the simulator, it operates asynchronously with the simulation clock. The control loop is the most latency sensitive component in the system and determines the maximum rate of the system. Although AWS claims to offer their services in real time, running the control loop in a separate entity subjects it to the AWS-internal latencies.

To find the upper bound for the simulation rate, the individual latencies for the operations involved in the control loop, as specified in Section 4, were measured. Figure 6 presents the distribution of latencies for AWS IoT Shadow State get, and DynamoDB reads from a Lambda function using Boto3. The bimodal distributions for each operation are attributed to the first such operation each time an instance of a control Lambda function is instantiated. This phenomena is independent of the way the Boto3 instance is initialized and reused.

Section 4 also reveals the total round trip time for the...
control action, measured from the time an event is sent on bus_stop_event until the affected traffic light’s Shadow State is updated on the virtual entity. This finding suggests that it will take on average 1.2103 seconds for the control action to take effect. This implies that either the simulation or the real world will need to accommodate 1.2 second delay from the time a bus is detected until the first instance a traffic light can be changed. And the system, at some point, will need to be aware and compensate for this additive stochastic delay. Furthermore, running with real devices would subject the system to an additional Wide Area Network (WAN) latency, in the range of 30 to 150 ms, depending on your physical proximity to an AWS Data Center (DC).

5.2.3 Cost

The cost of deployment is an essential factor in determining the effectiveness of a cloud service provider in terms of deploying a large scale scientific experiment. The cost of the developed infrastructure for varying data sampling rate and varying number of deployed sensors is depicted in Figure 7. It is assumed that the number of shards used is 1 and the data sampling rate is the data collected per minute from each deployed sensor.

6. CONCLUSIONS

In this paper, the emerging IoT support in public clouds was investigated and evaluated for scientific experimentation. The system created provided dynamic vehicle traffic control based on vehicle volumes/patterns and weather conditions. It was found that while AWS IoT performance and performance scalability often do not meet requirements of many next-generation scientific IoT use-cases. Additionally, manageability/modifications of a scientific IoT scenario can be challenging for moderate- to large-scale deployments. We are currently investigating techniques to replace the simulated sensors and actuators with their corresponding real devices. In addition, new control algorithms are being pursued to efficiently control traffic. Such application-specific developments will synergize with any new IoT support from public clouds.

7. ACKNOWLEDGEMENTS

The research was in part funded by the Swedish Research Council (VR) for the project Cloud Control under contract C0590801, Lund Center for Control of Complex Engineering Systems (LCCC), the Excellence center ELLIIT, and Mobile and Pervasive Computing Institute Lund University (MAPCI). The work was done while William Tärneberg was hosted by the University of Virginia.

8. REFERENCES