FocusStack: Orchestrating Edge Clouds Using Location-Based Focus of Attention

Brian Amento, Bharath Balasubramanian, Robert J. Hall, Kaustubh Joshi, Gueyoung Jung, K. Hal Purdy
AT&T Labs Research, Bedminster, NJ
{brian, bharathb, hall, krj, gjung, khp}@research.att.com

***Abstract*—Allocating and managing resources in the Internet of Things (IoT) presents many new challenges, including massive scale, new security issues, and new resource types that become critical in making orchestration decisions. In this paper, we investigate whether clouds of edge devices can be managed as Infrastructure-as-a-Service clouds. We describe our approach, FocusStack, that uses location based situational awareness, implemented over a multi-tier geographic addressing network, to solve the problems of inefficient awareness messaging and mixed initiative control that IoT device clouds raise for traditional cloud management tools. We provide an extended case study of a shared video application as initial demonstration and evaluation of the work and show that we effectively solve the two key problems above.**

1. Introduction

Since its inception, edge computing has focused on computing facilities associated with the last mile of the Internet. Increasingly , small-form-factor devices that connect to the last mile, such as television set-top boxes, network gateways, cars, and drones, themselves present an interesting target for building a managed computing platform that can serve a rich set of new applications. The applications range from traditional network edge services such as content caches or WAN accelerators, to more novel ones such as privacy-preserving big-data analytics on set-top boxes, connected cars, Internet-of-Things (IoT) sensor sharing applications allowing (e.g.) video sharing from specific locations, and applications allowing users to lease remote-sensing and computation resources in a fleet of drones.

This paper asks whether such a distributed collection of edge devices should be managed just like an Infrastructure-as-a-Service (IaaS) cloud-computing data center by leveraging traditional cloud orchestration tools. A key assumption is that the devices are managed, at-least in part, by a single controlling entity, such as a cable provider with a collection of set-top boxes, an operator deploying a fleet of drones, or an auto manufacturer providing a managed computing platform in its cars. The platform we seek to build considers each mobile edge device the approximate equivalent of a compute server, however with a local stakeholder that may also take critical management and control actions from time to time. Tenant applications, potentially sourced from different developers, may be deployed to these devices via containers. These application instances coordinate with instances on other devices in the system to perform tasks with local or platform-wide scope. Control plane nodes in the cloud orchestrate the management of these "distributed virtual data centers" of edge devices, and allow both tenants and administrators to interact with this edge cloud. Through this interface, tenants deploy new application instances and update existing ones, configure instances to communicate safely over virtual networks, and provide secure access to storage resources.

Despite similarities to the traditional cloud IaaS model, such an approach presents many unique challenges. For starters, the edge devices in question often have very limited compute and memory, and in the case of drones, limited energy as well. Second, the network environment is dramatically different from the typical data center, complete with nodes that may be constantly moving and have intermittent connectivity with variable quality. Third, the ratio of compute to control nodes is dramatically different. With thousands to potentially millions of devices attaching to a small set of cloud controllers, the control plane must achieve very high levels of efficiency. Last but not least, these devices often need *mixed initiative* management that is very distinct from the traditional cloud provider and the tenant separation. Specifically, in addition to the cable or auto company that manages the entire platform, and application providers that manage the apps, each edge device may also be co-managed by the end-user who actually owns it (e.g., the car or set-top box owner). The cloud platform must then not only be able to factor in the user's preferences and actions in any deployment decision making, but it also must be able to protect the overall platform from compromises of individual edge devices.

In this paper, we make the key observation that many of these challenges can be addressed by an intelligent geo- and context-aware messaging bus that allows the "focus of attention" of the cloud control plane to be scoped based on context that includes the device location, edge device health and capabilities, and user authorization preferences. We call this capability *location based situational awareness*. Devices which are not in the current focus of attention are neither tracked by the cloud control plane, nor participate in any control plane protocols. Doing so not only minimizes the resource utilization of the edge devices, since they do not need to provide periodic updates to the cloud, but it also allows the cloud control plane to be more efficient and scalable, since it only needs to handle a small subset of devices at any one time. Finally, such dynamic scoping

978-1-5090-3322-5/16 $31.00 © 2016 IEEE
DOI 10.1109/SEC.2016.22

is essential for handling edge devices that are constantly moving, and may be disconnected from the network at any given time; such devices are simply excluded from the focus of attention of the current orchestration task. The message bus also provides core security features - by only allowing authenticated nodes to communicate over the message bus and revoking their credentials if an edge device is "rooted", it protects the control plane from compromised endpoints.

We have built such a cloud, called *FocusStack* by combining *OpenStack* [1], one of the most popular open source cloud management platforms, with the AT&T Labs Geocast System (ALGS) [2], a multi-tiered geographic addressing (GA) network subsystem that allows packets to be sent to (all devices in) a geographic region instead of a specific set of IP endpoints, as in IP unicast or multicast. FocusStack can be deployed on an unmodified installation of OpenStack, and can deploy applications that are packaged as Docker [3] lightweight OS container instances to "compute nodes" running on small-form factor devices. Each application container can access a full suite of cloud capabilities including the ability to create private and public virtual networks as well as direct access to cloud resources colocated with the controller nodes including cloud storage and VM instances that provide additional compute capabilities. Through some analytical analysis and simple experimental measurements, we show the benefits in efficiency and scalability provided by the use of "focus of attention" as a core primitive. Finally, we demonstrate FocusStack's capabilities through a case study in which we build a cloud of Raspberry Pi's installed in cars and drones, and use it to deploy a video sharing application that combines in-car dashcams with drone-mounted webcams to provide a platform for sharing video streams captured in real time by drones and cars moving in real world environments.

In brief, the novel contributions of our work include

- the FocusStack method for location based awareness and orchestration in an edge cloud;
- a report on an extensive case study of the approach in the domain of video sharing among mobile devices;
- an initial analysis and evaluation of the advantages, limitations, and future work needed of the FocusStack approach.

2. Motivating Examples

We begin by listing a few use cases for FocusStack that illustrate the benefits of situational awareness. We then describe the *Shared DashCam* use case that we use as a running example throughout the paper, and which we have prototyped on FocusStack.

2.1. Use cases

FocusStack can be used to manage clouds that comprise a variety of endpoint types with a range of characteristics: a) Customer premise devices such as set-top boxes, edge routers, or WiFi access points are increasingly built using general purpose CPUs such as Atom or ARM and run commodity OSes such as Linux (e.g., DD-WRT [4]). Providers could call on these devices to provide a number of services ranging from usage analytics to environment sensing. While these devices have limited compute and memory capabilities, they generally have good network connectivity and power, and are not mobile. b) Cars are rich sensor platforms not just for the wealth of data they directly collect on engine performance [5], but also for their ability to measure their environment including factors such as weather, traffic conditions, terrain (potholes, etc.), and driving habits. A number of parties including auto manufacturers, city planners, insurance companies, as well as drivers themselves can benefit from analytics based on car sensor data. In addition to constraints on compute and memory, cars have additional challenges due to mobility and variable network connectivity. c) Drones are the ultimate mobile platforms, with energy constraints and extreme variability in network conditions as they fly in and out of radio range. General purpose drone platforms can be useful as a video-acquisition platform for hire, as well as environmental sensing and tracking. Next, we provide some examples.

Car diagnostics It's been unseasonably cold in the north east United States. Fast Motors Inc. wishes to understand how the cold temperatures have affected engine performance. While it is not feasible (for volume and privacy reasons) to continuously upload detailed diagnostics data from all cars at all times, it is possible to write a simple one-time app to read specific CAN bus data and run analytics to estimate engine performance. They can use FocusStack to deploy the app to a small sample of cars in New England. When the study is complete, the app is no longer needed, and can be removed from the cars.

Viewership Analytics CableCorp wants to understand differences between TV viewing habits of their LA and NYC viewers. Using FocusStack's geoaddressing primitives, CableCorp can identify and deploy two Hadoop instances in their target regions, each with their own virtual networks. Then, using a simple map-reduce job whose mappers measure channel change events, and whose reducers compute aggregate statistics, CableCorp can compute the aggregate results they need without ever collecting individual users' TV viewing history.

Drones For Hire RentMyEyes flies a fleet of connected drones equipped with cameras and environmental sensors. These drones wait for remote sensing jobs to be submitted over the Internet. Each job is represented by a target area the drone must fly to along with an app that the drone should run once there. Once at the target, the app is authorized to collect and analyze data from the drone's camera and sensors in real time, and potentially adjust the drone's flight plan based on its analysis. On receiving the job, RentMyEyes can use FocusStack to identify a drone close to the target area with sufficient energy left, and deploy the app to it.

2.2. Shared DashCam

From the examples above, it is clear that location awareness can play a meaningful role as a primitive around which to structure cloud orchestration. Next, we describe the *Shared DashCam* service, which we have implemented, and use in the rest of the paper as a running example to describe how FocusStack implements its location and situational awareness capabilities.

The Shared Dashcam service allows subscribers to watch real-time video generated either by dashcams in on-road connected vehicles or in flying drones. Hereafter, the term "vehicle" will refer to both on-road connected vehicles and to flying drones. An application subscriber selects what video source to watch based largely on the geolocation and video camera orientation of other participating vehicles. Such vehicles have a FocusStack device installed and are connected to the Internet using cellular LTE data service. Participating vehicles are owned by drivers who have agreed to share video from the dashcam that is part of the edge computing platform installed in their vehicles.

Avoiding Long Lines at the DMV: Alice needs to get her yearly car inspection at the DMV. But, she's wary of the long lines she's been running into recently. With the shared DashCam app on her tablet, Alice can query for participating vehicles in the vicinity of the closest inspection station that are able and willing to share their video at the current time. Shared Dashcam responds with a map of the area overlayed with available edge nodes, one of which is an edge compute node in Bob's car. Alice can then tap on Bob's node icon to have FocusStack deploy a Shared Dashcam container to Bob's car and send her the video feed, providing her the ability to assess the potential wait time.

Fall colors: Bob wants to go on a road trip to see some fall foliage, but he doesn't know where the peak colors are. Online foliage maps are often out of date. Bob can use Shared DashCam to call up video from cars and drones running FocusStack in a couple of the areas he is considering before he decides whether it is worthwhile to drive there.

Drone Event Watching Many people wish to watch The Pope's speech in New York's Central Park but cannot attend. The NYPD flies several authorized drones over the area, providing video from a selection of viewpoints. They prohibit all other drones from the area. Dashcam remote users can select from among the available device feeds to watch the event, based on desired angle, distance, etc.

2.3. Need for Situational Awareness

In the above scenarios, situational awareness plays a key role in not just the semantics of the service, but also in enabling efficiency and scalability. Assuming that the cloud cannot afford to actively monitor the operations of every edge device at all times, we need an architecture that can focus attention on the devices in an area of interest at a time of interest, extract necessary situational information, and take action on that information. Much can occur while a device is either out of contact with the cloud or while the cloud is not paying attention, including battery failure, operator actions (including application installation and invocation), environmental events affecting device operations, etc. Our approach must handle these possibilities.

For example, Highway I-5 is a long straight road running the length of California along which, most of the time, nothing interesting happens. While it transports thousands of cars per day, only a tiny fraction of them would be tasked to source video by the Dashcam application. The health, computational state, and opt-in state, of the rest is irrelevant to Dashcam and would require significant cellular bandwidth to report at all times. For those that actually *are* of interest, we require a method for gaining fresh intelligence on their computational and resource states at the time of interest.

Motivated by these considerations, our primary requirement for FocusStack's awareness function is to obtain awareness information *when and only when attention is focused on a geographic area*. Consequently, the applications best suited for use with FocusStack are those similar to the ones described in the previous use cases, that is, where the application needs to be deployed and active on edge devices which are within a circumscribed geographical area.

A second key requirement is motivated by the observation that an application is clearly not interested in all IoT devices in a particular area. The Dashcam application is only interested in participating camera carrying devices. A remote sensing application is interested only in the environmental sensors providing information of interest. Thus, FocusStack's awareness component must be capable of limiting the scope of queries so that only a narrow subset of all IoT devices in an area will even reply.

Finally, different applications require different awareness information. The Dashcam application is interested only in the computational, communications, energy, and opt-in state of participating vehicles. It is not interested in information relevant to other applications, such as remote auto maintenance data, sensor data from nearby field sensors, battery level information from nearby smartphones, etc. However, other applications may indeed want these other types of information. Thus, FocusStack must be able to query for custom sets of application-specific awareness information.

In the event that more than one edge device meets the criteria relevant to a particular application, the set of edge devices meeting the application criteria is presented to the application for further application specific selection. It is then up to the individual application to decide whether to invoke and run application elements on one or more of the edge devices in the set presented to the application.

3. FocusStack Architecture

We have built an architecture that supports deploying heterogeneous applications to a diverse set of possible IoT edge devices. These devices are potentially limited in compute power, energy, and connectivity and are frequently mobile. One of our goals is to build a platform that handles the underlying details of discovering an appropriate collection of edge devices (based on location, sensor capabilities, etc)

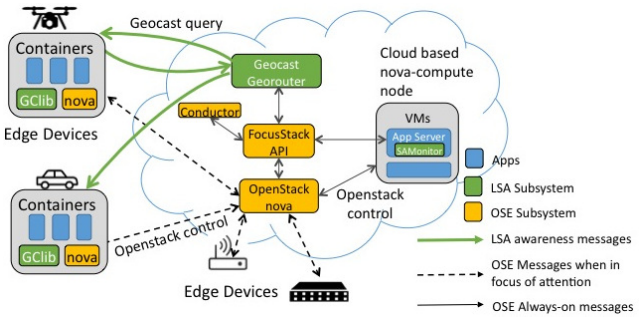

Figure 1. FocusStack Architecture

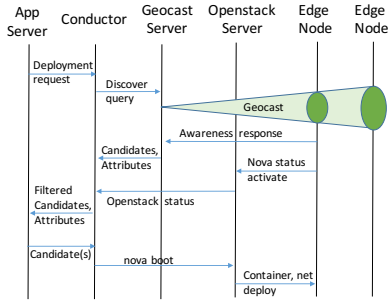

Figure 2. FocusStack Control/Message Flow

with the available resources to run the application, deploy the application code to the devices, and launch the application. Our platform enables developers to focus on their application rather than on finding and tracking the various edge computing devices where they will be deployed.

3.1. Overview

There are two major architectural components that together comprise the FocusStack platform. The first subsystem, based on a *Geocast* primitive, provides location-based situational awareness (LSA) of edge devices to the second component, our OpenStack extension (OSE) that allows deployment, execution and management of containers on small edge computing devices with limited networking capabilities. Figure 1 shows the overall architecture of FocusStack, which forms a hybrid cloud consisting of both edge devices running lightweight Linux containers (based on Docker), and cloud-based compute nodes that can run virtual machines (VMs) like an IaaS cloud.

When a cloud operation (such as deploying a new container instance) is invoked by calling the appropriate FocusStack API, the LSA subsystem based on a Geocast Georouter is first used to scope this request. It does so by sending a geo-addressed message containing details of the desired resource (e.g., what kind of sensors are needed, etc.) to the target area identified by the request, and waiting for responses from edge devices that satisfy the requirements, and are currently healthy and connected to the network. The resulting "focus of attention" list of edge nodes is then used to seed the appropriate OpenStack operation with the help of a component called the *conductor*. Figure 2 shows a representative control message flow between an application and the LSA and OSE components.

3.2. Situational Awareness Subsystem (LSA)

This section describes our implementation of *Location Based Situational Awareness (LSA)* within our FocusStack framework. FocusStack's awareness component is based on the FCOP algorithm [6], which is a distributed algorithm using geographically addressed (GA) messaging. In our implementation, we use the AT&T Labs Geocast System (ALGS) [2] for GA messaging.

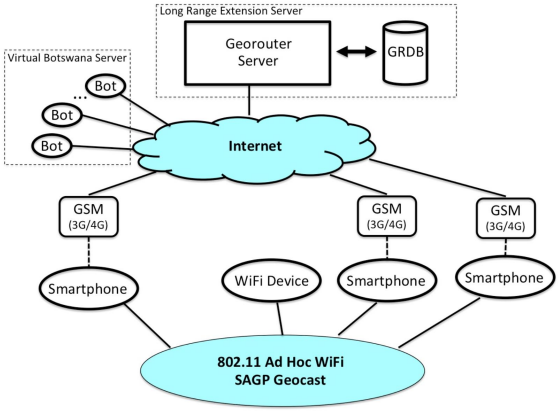

Figure 3. Schematic of the AT&T Labs Geocast System

3.2.1. Background: GA and the ALGS. In *geographic addressing (GA)*, a packet's address consists of a subset of physical space, with the meaning that the packet will be transferred to all devices currently in that space. In the ALGS system, such a subset is defined to be a circle on the surface of Earth, described by the latitude and longitude and its radius. A GA service is implemented in the network and appears to the programmer as an API analogous to (and in parallel with) the IP stack. In some implementations, a GA service can even be used in the absence of IP addressing, which can be of significant advantage in settings, such as mobile ad hoc networks, where the overhead of maintaining IP routing tables is onerous. [7]

The other major benefit of using a network GA service to provide location based packet delivery, is that there is a wide variety of location based applications (with more being invented daily); if each has to implement its own method of determining where clients are physically and routing to them, the overhead would be multiplied accordingly.

The primary use of GA in FocusStack is to transport query and response messages to, from, and between areas of interest, in order to support our awareness component. However, it is also used for command and control of devices in some cases, such as drones, as well as for distributing information on a per location basis. An example of the latter would be to transmit definitions of areas in which video recording is prohibited to all devices near those areas.

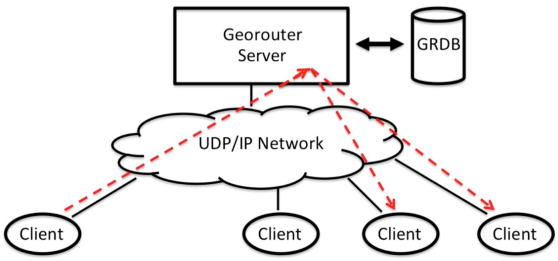

Figure 4. Packet georouting via the ALGS long range tier. The leftmost client sends a GA packet up to the georouter server, which then sends a copy to each client within the addressed region (dashed arrows).

The AT&T Labs Geocast System (ALGS) [2] implements a seamlessly integrated, two-tier network GA service. A packet's address, referred to as its *geocast region*, is defined by a circle, where the packet header contains latitude and longitude of the center of the circle and the radius in meters. Packets sent via the ALGS can transit either an ad hoc WiFi tier or a long range tier mediated by an internet-based georouting service accessed through the 3G/4G/LTE/GSM system. See Figure 3. Packets can be relayed across either tier or both tiers; in some cases, a packet originating in one ad hoc tier can be transferred to a long range capable device, which will relay it over the long range tier to a device near the destination region, where it will be relayed again across the ad hoc WiFi tier to devices in the region.

In our prototype Dashcam system, devices do not have WiFi capability, so the system depends entirely upon ALGS's long range tier. See Figure 4. An originating client sends the packet up into the Georouter server (via LTE and over the Internet), which determines which devices are in the geocast region and routes copies to each of them. Note that a client may be any device running the GCLib code, which includes the ALGS software as a subsystem. This includes edge devices as shown in Figure 1 and also cloud resident components running an SAMonitor instance. Each green arrow (LSA messages) represents GA messaging within the LSA subsystem, as depicted in Figure 4. Location and connectivity information are maintained in the *georouting database (GRDB)*. Reference [2] has further details.

Packets within the ALGS long range tier are transported across IP networks as UDP messages. When a UDP packet traverses a firewall or NATting router, assuming such is not blocking UDP packets to geocast port numbers, the firewall/NAT sets up state (sometimes known as a "UPD connection") that allows reverse-direction packets to go from receiver to source along the same ports. ALGS exploits this to allow bidirectional asynchronous messaging. Note that UDP connections time out typically at between 1 and 3 minutes, so the ALGS system makes sure that an edge device "refreshes" such connections periodically, through sending either normal traffic or a small dummy packet up to the server at a maximum time interval.

3.2.2. Background: The FCOP Algorithm. The *Field Common Operating Picture (FCOP)* algorithm is a GA-based distributed algorithm designed to enable each device to update others on its current awareness information in a scalable manner. More generally, it allows a group of devices M all to monitor the awareness information of a group of devices A. This general case is the *monitoring problem*. We refer to the special case of $M = A$ (when all devices monitor all other devices) as the *common operating picture* problem. This complex special case involves a quadratic number (n^2-n, where n is the number of devices) of logical information flows.

Given a region R and an awareness query spec Q, FCOP operates as follows on each device.

- Whenever the device has neither sent nor heard a query message directed to a region including R within the last P seconds, it sends a GA message addressed to R containing Q.
- Whenever a device receives a query message containing query spec Q, if it has not sent one or more response messages back to a region including the location of the querying device in the last P seconds and containing all the information requested in Q, it formats and sends a query response containing the requested data blocks and directs it to a circle around the querying device. Note that it may have responded multiple times (to multiple queries), each containing part of the requested information and that would count as having responded as well.
- Whenever a device receives a query response message containing information it is requesting, it records the information in its operating picture record for that device.

In our Dashcam prototype, we used $P = 10$ for the awareness monitoring interval.

For full details of FCOP, the reader is referred to reference [6]. When GA messages are transported over the ad hoc wireless tier, a message can in general be delivered in only $O(\lg n)$ transmissions, so the full algorithm's message complexity is $O(n \lg n)$ messages. When using the long range tier, since we are required to use unicast UDP messages for the last link to each device, the worst case complexity is $O(n^2)$ messages. However, even in that case, the FCOP algorithm minimizes the constants involved in two ways. First, by having the device only send queries when it has not already heard one recently to the same area, there is in general only one query message per P seconds. By having devices accept and record information in the responses to queries issued by other devices, the picture is assembled as quickly as if each device sent its own query, but without the need for all the redundant query messages.

3.2.3. The GCLib Framework. GCLib is a software framework providing components access to GA messaging, access to sharing of arbitrary data within the device (car, drone, etc), and automatic support for the query/response awareness function discussed below. GCLib is shown in Figure 5. Components of GCLib talk to each other via TCP

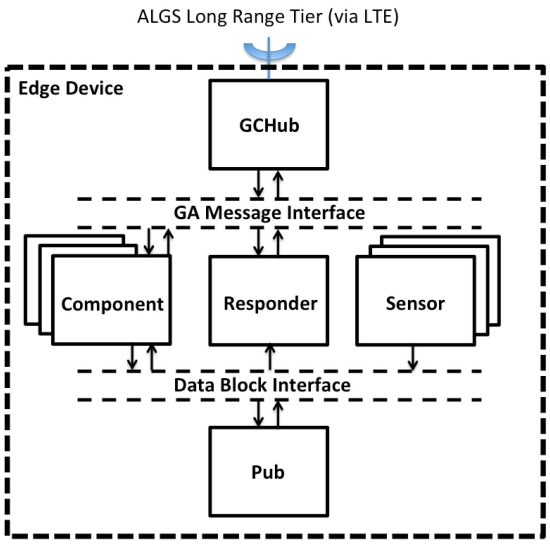

Figure 5. GCLib framework software architecture.

streams over localhost socket connections via defined protocols whose details are suppressed here. These connections are labeled either "GA Message Interface" or "Data Block Interface" in Figure 5.

Referring to Figure 1, GCLib instances run in edge devices and within the cloud application using SAMonitors.

The GCHub mediates access for all other components to the ALGS GA network. To send messages, the payload and address information is sent to the GCHub directly. The GCHub then formats the information into a geocast packet and uses the ALGS tiered geocast protocol to send it out. To receive GA messages, each component registers interest with the GCHub by specifying one or more tags (or prefixes of tags); then, when a GA message is received, all components having at least one tag-prefix matching the start of the GA message payload are sent copies of the message.

The Pub component implements a publish/subscribe system for data blocks, described below. Essentially, it provides the plumbing for data to flow among components in a fully pluggable way. Each component registers interest in data block tags (or prefixes of them) and receives a copy when a component publishes a block update with a matching tag.

The Responder component registers interest in incoming query messages and all data block prefixes (i.e., the empty prefix). It does the matching and formulates and sends the response message in conformance with the FCOP protocol.

Sensors take measurements and periodically publish their data to the Pub, which makes sensed data available to the awareness messaging. Examples include components that determine the current computational state (CPU load averages, memory and storage utilization), energy level (e.g. % of battery left for drones), or kinematic state (velocity, acceleration, etc). Other components can use GA messaging and/or Pub data facilities as desired as well; referring to Figure 1, applications running in containers on the edge devices can publish information into GCLib's Pub so that it can be available for retrieval in awareness messaging. For example, FocusStack can focus on "all devices currently sourcing video streams in a given area" by having the video streaming component (running in a container on the edge device) publish its streaming status into the Pub and then having the SAMonitor awareness queries retrieve this status information. Another usage pattern would be to use GA messages to invoke actions on the edge device, where a container on the edge device registers for particular types of GA messages through GCLib's GCHub and then takes action on receiving them. For example, a location based cloud application could send messages to all devices surrounding a military base to "cease video sourcing in this area".

3.2.4. Tags, Data Blocks, and Query Specifications. To accommodate arbitrary applications, we need to systematize how information is reported over the FCOP algorithm. That is, rather than defining a custom message format for each application, we instead provide a general tagging and reporting mechanism as follows.

A *tag* is a case-insensitive ASCII string consisting of non-space characters enclosed in brackets. For example, the `[Energy]` tag is used to denote a data block transporting the percentage of total energy capacity currently available in a device. Each application will typically use some common tags, like `[Energy]`, and will define its own application-specific tags, such as `[VideoEncoding]`. To minimize semantic conflicts between applications, we anticipate a need for standardized ontologies to define tags' meanings.

A *data block* is a sequence of bytes that starts with a tag and optionally continues with fields representing information. For example, the energy data block would look like

`[Energy]b`

Where `b` is a one byte integer between 0 and 100 representing percentage of capacity. Data blocks can have arbitrarily many fields, including zero. Zero-field blocks can be used to identify simple boolean properties of the device; for example, if a device has registered the `[App.Dashcam]` tag, it indicates this device has opted in to participating in the Dashcam application. Similarly, registering `[Device.Drone]` would label a drone device so that it can be distinguished from cars or other devices.

Data blocks come into being through on-board components, such as sensors, publishing them into GCLib's publish/subscribe blackboard, the *Pub*. (See Figure 5.) Each time a new sensor value is obtained, the component publishes it under the same data block, and the new value replaces the old one. For example, each time the battery energy level is read, the new level replaces the old one under the `[Energy]` tag. This system is not used for all data flows within the edge device, such as video streams, which are managed by the Dashcam application and require more complex buffering schemes.

A *query specification (query spec)* is a sequence of tags preceded by a combinator. Currently, we have implemented the `[Q.AND]` and `[Q.OR]` combinators.

- `[Q.AND]` means that to match the query spec, a device's Pub must contain a data block for *every* tag. For example, to match the `[Q.AND][App.Dashcam][LLA][Energy]` query spec, the Pub must have all three of `[App.Dashcam]`, `[LLA]`, and `[Energy]` data blocks. This would request position and energy information from all Dashcam participants in the area of interest. (Note that "LLA" is a tag representing the latitude, longitude, and altitude of the device.)
- `[Q.OR]` means that to match the query spec, the Pub must contain a data block for *at least one* tag. For example, to match the `[Q.OR][Device.Drone][Device.Car]` query spec, the device must either be a car or drone. This would report all drones and cars in the area of interest.

The FCOP algorithm, on receiving a query spec via the GCHub in the payload of a GA message, interprets it, retrieving values from the Pub. In the case of `[Q.AND]` queries, it must be able to retrieve data blocks for all tags in the spec. For `[Q.OR]`, it only must retrieve at least one such data block. Assuming this matching succeeds, FCOP then formats a query response and sends it out over the GCHub to the region surrounding the querier.

3.2.5. FocusStack Monitoring Component. Each application or service wishing to focus awareness on a region creates an *SAMonitor* component. An SAMonitor is an interface component used to communicate a query to the LSA subsystem and to receive query responses and use them to assemble a real time operating picture of the region in question. It is built using the GCLib library.

The application gives the SAMonitor a circular geographic region, R, defined by a center latitude/longitude pair and radius in meters. It also gives the SAMonitor a query spec, Q, defining the information to be returned from the devices in the area. In accord with FCOP, the SAMonitor periodically sends a GA message containing Q to (all devices in) R; each device in possession of information satisfying Q responds by sending a GA message back to a circle containing the querier. In accord with FCOP, by sending replies to a circle around the querier, not only do we allow for possible mobility of the querier, but in the case of cloud-resident services, this allows all services wishing to monitor R to share both queries and responses, thereby reducing traffic to and from the area.

Due to the dynamic nature of mobile applications, the SAMonitor uses response messages to assemble an *operating picture* of the area of interest. This is a continually updated data set recording the set of devices reporting from the area, the information received from each device, and the *age* of the information. A client application can make decisions based on information recency, which can improve service quality. For example, if a car has left the monitored area, so that its information is relatively old, then the Dashcam application can avoid selecting it when there are other cars known that have more recent information.

Once the service determines that its task in region R is complete, it deactivates the SAMonitor, removing the focus of attention and stopping the query and response messages.

3.2.6. Use of Monitoring Within Orchestration. An application needing to perform a task within a region sets up an SAMonitor for the region with a query such as

```
[Q.AND][App.Dashcam][LLA][Energy][CompState]
```

In our Dashcam prototype, the SAMonitor is part of the Dashcam App Server (see Figure 1) in the cloud.

To match this query, a device must first have opted in to the Dashcam service so that it has the `[App.Dashcam]` tag. Next, it must have position information, in the form of latitude, longitude, and altitude, in order to match the `[LLA]` tag. It must also have onboard sensors reporting energy (`[Energy]`) and computational state (`[CompState]`). Note that energy, while not too critical for cars, which have plentiful and renewable battery capacity, is extremely important for drones, which are strongly energy limited. Dashcam can run in cars, drones, or other vehicles or venues; to restrict attention only to a particular device type, one can add a tag, such as `[Device.Car]`, to the above query spec. Computational state includes such dynamic measures as CPU load average(s) and amounts of memory and storage.

The query response will have not only the tags, but the sensed values for each tag type:

- `[LLA]` will include fields reporting the sensed lat/long/alt values;
- `[Energy]` will include device energy level, as a percentage; and
- `[CompState]` will include CPU load average(s) and amount of free memory and storage.

The specific query spec used will be different for different applications. For example, a remote sensing application may request sensor data values other than those above, an auto maintenance application might seek alarm log information, engine temperature data, etc. Our general awareness framework based on arbitrary tags and query specs accommodates a wide range of domains and applications.

Once the operating picture is assembled, the information is handed to the orchestration subsystem within OSE (Conductor) to check that the device is capable of executing the task and satisfies all pre-defined policy rules, such as authorization by the device's owner for executing the task.

Passing these checks, OSE then invokes nova to carry out the management task. In our system, a message is sent to invoke OpenStack mechanisms to accomplish the management action by transporting OpenStack's message bus across a VPN tunnel set up from device to cloud.

Once the management actions are complete, the involved devices need not stay in focus of attention, whether or not the invoked edge device applications continue running on the edge devices. The SAMonitor and orchestration mechanisms

can be deactivated until the next time the device comes within focus of attention for some operation, while the edge application(s) can continue their operations while outside of FocusStack's focus of attention if appropriate.

3.2.7. Mixed Initiative Control. The concept of FocusStack is to acquire awareness information about a device only when needed. If the user or other stakeholder of a device has taken management actions between times when FocusStack is paying attention to it, our query mechanism allows the control plane to reacquire a current picture of relevant measures, such as computational state and the apps and hardware resources are available on the device. The operating picture (awareness information gathered using LSA subystem) acquired at time T2 may have little or no relationship to the operating picture acquired at T1, due to management actions by other stakeholders. This contrasts with the traditional cloud control model, where if a stakeholder activates a process on a host, OpenStack will not be aware of it. An example of this mixed initiative control in a Dashcam system would be if the onboard device were activated to run navigation or entertainment functions in response to the driver or passenger, while FocusStack operated the Dashcam application.

Note that this approach allows *multiple* FocusStack control planes to operate concurrently upon the same sea of devices, though we have not as yet demonstrated this capability. Each control plane independently focuses its attention, acquires the operating picture, takes action, and then moves on. We expect this multiple-cloud mixed initiative model to become prevalent when dealing with systems at the scale of the Internet of Things.

3.3. OpenStack Extensions (OSE) Subsystem

In a standard OpenStack environment, virtual machines are deployed and managed on compute nodes that are full-fledged, heavy weight server machines. This approach isn't feasible with limited edge device compute nodes. We have opted to integrate lightweight Docker containers with the OpenStack management platform. With this combination, we benefit from the portability, security and application isolation of Docker containers while still sharing the rich set of orchestration and management tools available in OpenStack with other typical datacenter applications.

3.3.1. Edge Compute Nodes. Compute nodes require several components to interact with our architecture. Nodes run a custom version of Nova Compute that interacts with a local Docker instance to launch and manage containers. When active, Nova reports back to the OpenStack instance over an IPSec tunnel instantiated at bootup. Location awareness and messaging are provided by the AT&T Labs Geocast System (Section 3.2). Devices use a GPS receiver to track their location and report it to the server to allow geographic addressing over the ALGS long range tier over LTE.

Containers running on the edge nodes are provided full OpenStack services, including access to virtual networks, configurable on a per-application basis. These virtual networks can be configured to provide container instances with private connectivity both with other container instances as well as with virtual machines running on regular compute nodes in the cloud that also belong to the application. The virtual networks are implemented using OpenStack's standard LinuxBridge neutron plugin which supports both VLAN as well as VXLAN overlays. All edge node communications, including those between edge nodes, occur over an IPSec tunnel running over the LTE cellular network that connects the edge node to an OpenStack L3 (layer 3) network node in the cloud. Such an L3 node runs virtual routers that enable IP routing to occur between different virtual networks. The architecture of LTE networks, which forces traffic to be aggregated through a packet core aggregated in regional sites [8] precludes local communication between edge nodes over LTE. However, if the edge nodes have direct connectivity via WiFi, they can leverage Geocast's adhoc networking capabilities for direct communication.

Using these facilities, its easy to start up applications requiring multiple nodes, e.g., a Hadoop micro-cluster running on several edge nodes, as well as hybrid applications requiring access to some cloud resources. E.g., the Shared DashCam application instantiates a streaming video server VM in the cloud to allow dashcam video from the edge apps to be broadcast to multiple subscribers.

3.3.2. Cloud components. There are several cloud components provided by the FocusStack architecture. The Geocast georouter server is part of the LSA subsystem and tracks the location and other metadata about each edge device, enabling geographic addressing. An application server makes requests through a FocusStack API to the Geocast SAMonitor and receives a list of available edge nodes in a given geographic area. The decision to include a node within a geographic area is made based on location, speed, heading, altitude or other factors. This list is then sent to the Conductor, a constraint solving algorithm described below, for filtering. The Conductor chooses nodes based on their capabilities and resource availability. For example, if an application requires aggregated data from an accelerometer and gyroscope, the Conductor can ensure that only nodes with the correct sensors and adequate storage and cpu for the computations are returned. Once the list of matching nodes is generated, the application server can then select the desired node(s) to deploy the application or present the list to a user for final selection. The OpenStack Nova API is responsible for managing and deploying applications on the selected edge devices.

Once applications are deployed on the selected edge devices, they need not remain in focus for the entire time that the application is running on the devices. Even if OpenStack Extensions components such as Nova Compute continue to run on an edge node where an application is deployed, these components may transition into a semi-quiescent state in order to minimize communications and computational burden on the cloud management system. Design details

for how this transition takes place and the nature of the quiescent state are both subjects for future work.

3.3.3. OpenStack Messaging. OpenStack uses the Advanced Message Queue Protocol (AMQP) as implemented by RabbitMQ for its messaging platform. Many OpenStack components create component specific message queues at initialization time. For example, an OpenStack compute node would create a message queue whose node type is "compute." OpenStack messages directed to a topic exchange are delivered by the messaging system to all node message queues whose node type corresponds to the topic. This means that the majority of OpenStack messages are multicast to every compute node managed by the system. For a traditional cloud data center with high speed networking links from OpenStack controller components to tens or at most hundreds of compute nodes, such topic-based message multicasting provides flexibility and decoupling between message publisher and consumer.

With IoT, there are potentially millions of addressable edge compute nodes, with limited and intermittent network connectivity. It is infeasible to use the currnet OpenStack messaging architecture in IoT. The LSA subsystem provides us efficiency and scalability, by restricting the number of OpenStack compute nodes to only those within the focus of attention. The set of compute nodes in an OpenStack instance becomes a dynamic, small subset of the universe of edge devices. In OpenStack terms, the nova-compute service is only started on devices which are in the focus of attention. A FocusStack compute node ceases being an OpenStack compute node either when it leaves or is no longer interested in the geographic area of focus. With this key change to the OpenStack architecture, the multicast nature of OpenStack messaging becomes efficient and scalable for IoT applications while preserving the elegant, loosely coupled properties of the existing OpenStack messaging design.

3.3.4. Conductor. Aside from the importance of their location and their mobility, edge compute nodes incorporated in FocusStack differ in other important ways from traditional compute resources in a cloud environment. First of all, such edge nodes are very restricted in their available resources (e.g., CPU, memory, and disk) as compared with compute nodes in a traditional cloud. Additionally, other unique characteristics of IoT devices should be considered including sensor functions of edge nodes, battery or energy levels, security-awareness, etc. To address the need for these additional filters, we introduce a constraint solving component, Conductor, which was originally designed as a scalable deployment decision maker for cloud services in large-scale cloud data centers [9]. Conductor efficiently searches for the optimal deployment of cloud resources that meets a given set of constraints (e.g., resource and energy availabilities) and application requirements (e.g., sensor types and security policies for deployment). In our current approach, we integrate the constraint solving component of Conductor and leave the final decision among possible candidate nodes for the application. In order to incorporate the constraint solving component of Conductor into the IoT application space, e.g., the Shared Dashcam service, Conductor selects among edge compute nodes that meet all constraints described above.

Figure 6. Shared Dashcam Hardware

4. Evaluation I: Shared Dashcam Case Study

As previously described, the Shared Dashcam service allows subscribers to watch real-time video generated either by dashcams in connected cars or in flying drones.

4.1. Challenges of Node Mobility and Networking

Applications to support Shared Dashcam service use cases are interesting edge computing systems because their design and implementation require that desired resources be located in the face of edge nodes that are mobile and have limitations in their network connectivity.

Unlike the compute nodes in a cloud data center, the resource of primary interest in the Shared Dashcam service is video sources; in particular, active video sources near a particular geolocation pointed at interesting things. Which video sources are in which location right now is a dynamic property of the service because nodes move around. In fact, it is almost exclusively video sources which are right now or have recently been mobile that are of the most interest. A parked car is usually turned off and thus its dashcam is also powered off, and, even if the dashcam were on and the video available, a static view from a garage or parking spot is unlikely to be of much interest to other subscribers. The fact that nodes move around independently of one another also stresses the networking design. Since the video sources (vehicles) for the Shared Dashcam service are outside most of the time, accurate geolocation is reliably implemented using GPS and networking is based on cellular LTE data service. However, even in today's smartphone focused world, reliable, robust cellular data service is not always present, and when it is, connectivity to the Internet is characterized by Network Address Translation (NAT) and changing public IP addresses.

As is discussed in Section 3.2 ALGS implements geographic addressing and is therefore specifically designed to accommodate computing nodes that are mobile, that are ephemerally connected and whose underlying network addresses change. However, the video streaming feature of

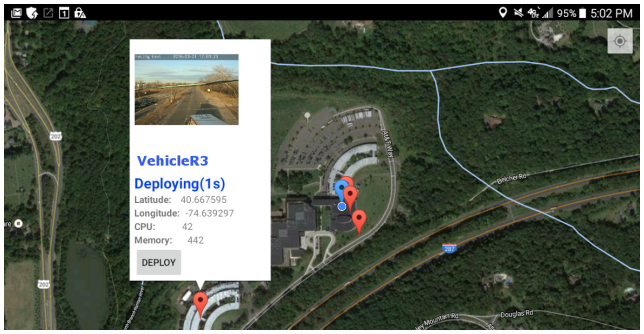

Figure 7. Shared DashCam Map Interface

the Shared Dashcam system is built upon standard TCP networking and is quite fragile with respect to ephemeral node network presence and changes of underlying public IP addresses. Because of the network address translation present in cellular data carrier networks, the problem of edge nodes' changing IP addresses is even harder since the edge device itself is unaware of the IP address change. It uses the private IP address that was given to it via the carrier operated DHCP service when the device initially started up.

4.2. DashCam Application Implementation

In order to evaluate our FocusStack architecture we built a prototype version of the Shared Dashcam service. The edge compute device we chose is the Raspberry Pi 2 Model B running Ubuntu Mate 15.10. We augmented the Raspberry Pis with a 5MP, 1080p camera, GPS receiver and LTE dongle. The hardware was installed in the target vehicles, running off of the car battery, with the camera mounted on the rearview mirror to provide a view of the road ahead. (Figure 6) The system powers on when the car is started and shuts down after the key is removed from the ignition. The base software on the edge device is minimal, a docker instance, the GCLib software (Section 3.2) architecture implemented in Java, and a paired down version of Nova compute. Initially, there is limited network traffic between the device and the cloud. Infrequent awareness updates are sent through Geocast to keep track of the location and availability of the device. In a full deployment of this system there could be thousands or millions of these devices, but none of them interact with the cloud infrastructure until they are required by another user of the application.

The user interface consists of an Android application running on a tablet mounted in the vehicle. We have intentionally separated the user interface from the rest of the DashCam hardware to allow the flexibility of viewing shared video from outside the vehicle, for example, at home waiting for traffic to clear. There are a number of privacy concerns that arise with any shared video system, but in order to simplify the application description, we have omitted the discussion of any user level privacy controls.

When the tablet application is started, a request is sent to the application server to build an awareness picture of the area of interest and retrieve a list of potential vehicles from the LSA SAMonitor component. In addition to being located in the area, these are devices whose current awareness information shows they are willing to share their video and are within a 10km radius. As described above, this list is further filtered by the Conductor to return only nodes that have the required capabilities and resources. Vehicles reported by the LSA SAMonitor picture that survive this filtering phase are displayed on a map interface to the user with a thumbnail image of the current camera view, as shown in Figure 7. The user then selects the desired vehicle to obtain its video feed. Once a selection is made, the application server contacts Openstack and triggers a download to the edge device in the target vehicle. A docker container is then launched and the application is deployed. When the video stream is requested from the docker container, it begins streaming the live feed over UDP back up to the application server. The feed is then broadcast back down to any interested tablets. In a traffic heavy area, we assume that there will be numerous vehicles requesting the same video feed and hope to preserve bandwidth by re-broadcasting the video rather than using point to point connections.

4.2.1. DroneCam extension. Dashcam can run on a Raspberry Pi mounted on a drone. Of course, there is no onboard consumer of video, but the same approach to dynamic sourcing of its onboard camera video applies as in the car case. The LSA awareness picture now includes the altitude in addition to position, and the filtering by Conductor may include reasoning about available `[Energy]` by extending LSA's query spec to include that tag.

5. Evaluation II: Benefits of LSA

In this section, we compare the network and processing costs of a full-time active monitoring system, essentially similar to how OpenStack today monitors hosts within a cluster, with FocusStack's focus of awareness based approach using a simple calculation based on a single IoT scenario: Dashcam at scale.

A full-time active monitoring OpenStack communicates awareness messages at the rate of B bytes per P seconds from each host. We use $P = 10$ as in our prototype. We have measured $B = 17509$ bytes per 10 second interval generated by full-time active OpenStack in our prototype. For a device managed by FocusStack using the LSA subsystem, on the other hand, we see awareness messages (one query plus one response) totaling 358 bytes per P seconds as long as the device is within focus of attention.

In addition, the operation of the ALGS long range system requires that any device that has not sent a packet to the long range server within the past L seconds send a *dummy* GA packet to the long range server so that it may record the current position and IP address information of the device. Such packets in our current prototype are 122 bytes (including headers), and we use $L = 30$, though this number was chosen for convenience in debugging and should be tuned to a larger value for large scale use. Therefore, to

summarize, devices within focus of attention will be actively communicating , so do not send dummy packets, while devices *not* in focus will only send dummy packets.

For our large scale Dashcam example, let us assume that in the future all cars on U.S. roads become Dashcam devices, and ignore all other IoT devices. Further assume that the number of cars on U.S. roads stays the same as today, approximately 253 million [10]. Wolfram Alpha reports 5100 accidents per day in the U.S. as of 2005; assume that Dashcam users are interested in all and only these accidents. Next assume the area of interest for an accident covers 1000 cars and the time of interest lasts 2 hours per accident.

In this scenario, OpenStack with full-time active awareness monitoring would transfer approximately 38.2 quadrillion ($3.82e16$) bytes over LTE *per day*, and all of these must be processed by the OpenStack control plane. However, using instead FocusStack's LSA awareness subsystem, by contrast, would result in 88.7 trillion ($8.87e13$) bytes of dummy packets sent up to the long range servers of ALGS, plus only 1.31 trillion ($1.31e12$) bytes of awareness messages for processing by the control plane.

Just viewing this Dashcam scenario in isolation, using OpenStack with full-time active monitoring would cost 431 *times* as many bytes transmitted over LTE as our prototype FocusStack. In terms of processing load, the OpenStack control plane would have to process 23,875 *times* as many bytes of awareness data as the control plane of FocusStack.

While these savings are substantial, it is important to realize that the GA subsystem underlying FocusStack is a shared resource, so that once we consider a future in which there are many tens or hundreds of IoT applications using it, the relative savings are much larger. In particular, all dummy traffic is shared among all applications; i.e., a device needs only to send one dummy packet no matter how many FocusStack instances or FocusStack-based applications are using the system. Furthermore, as more applications use the system concurrently, more devices will be in some focus of attention, so the dummy traffic will decrease, leaving only the awareness traffic. We expect that further tuning (e.g. of the L parameter) will reduce the dummy load even further. Finally, in a multiple-control-plane scenario, each traditional control plane must do its own active monitoring, effectively multiplying the cost. In FocusStack, the awareness traffic is proportional only to the number of applications and the sizes of their attention focus regions, independent of how many FocusStack control planes may be running.

6. Related Work

There has been a proliferation of work in recent times in the area of edge computing, exploring the idea of computation and storage being performed closer to the end user. While the seminal work in [11] mainly envisaged the use of stable compute resources near the edge, such as a set of local multi-core computers, this has been extended to other edge devices such as wireless gateways and set-top boxes in more recent work [12], [13]. We too focus on performing computations on edge devices forming virtual clouds. The work in [12], which is thematically closest to our work, presents a platform that allows central entities like Netflix or a security-based camera vendor to utilize the home wireless gateway to perform local computations relevant to the user residing in that home. Their concept of "chute" is very similar to our use of containers and is used to share resources in a virtualized platform. The fundamental difference between the work in these papers and our work, is that their notion of the edge is very much defined by proximity to the user while our notion of the edge is defined by the focus of attention (such as the cameras near an accident site). We dynamically instantiate virtual clouds in these areas to serve the service need for a particular request. Further, we integrate our service with the OpenStack cloud service, making it much more viable for wide-spread use.

The scope of edge computing has been extended to the mobile devices carried by the users which include smart phones and tablets [14], [15]. These works make the claim that mobile phones/devices now are strong enough to actually host computation and hence mobile devices can offload to any other mobile devices that are willing to share free resources. We too forage for resources in every focus of attention and handling device mobility is an important part of our work. However, in these prior works, resource discovery and orchestration is performed in a fully local manner by the requesting device, and is limited in its choices of forming the mobile cloud. In our work, a central cloud, which has a global view of the available resources orchestrates virtual clouds anywhere required, and hence a user can request a service pertaining to any area as long as it contains registered devices that are available and equipped to perform the service.

There has also been several works in the area of cloud-offloading [16], [17], [18], where resource constrained mobile devices off-load some of their computation to a central cloud. In our work too, when a task is being performed by a bunch of devices hosting containers in a focus of attention, the orchestrating cloud is also part of this computation. In other words, rather than the edge devices off-loading their work to the cloud, our model is always a hybrid model that uses both the cloud and edge resources to perform tasks.

The concept of situational-awareness has been explored in the literature [19], [20]. These works address the challenges in collecting and disseminating information from edge devices (including mobile devices) that are present in a disaster area. The main challenge they deal with is the adhoc nature of the networks and the inherent fallibility of the devices. In our work, we assume that the devices are always connected to the LTE network and further, information dissemination is performed based on geographic-addressing and attribute tagging. Also, a central cloud and the edge cloud controller (within an instantiated edge cloud) manage the devices and the flow of information, leading to a much more robust architecture, albeit dependent on connectivity.

There has been prior work on work-offloading to volunteer machines [21], [22]. Even in our work, there can be voluntary edge devices willing to participate in a computation. However, in the referenced prior work, the machines

are usually fixed machines with powerful computational capabilities and the only criterion for their participation is the availability of resources. On the other hand, in our work, the devices are selected based on focus of attention and availability. Further, the devices that may perform a certain task need not be fixed and could vary significantly based on mobility, failure and so on.

7. Limitations and Future Work

Extending the cloud model of orchestration to edge devices raises new issues in security.

Edge devices cannot be trusted. In a normal data center, all hosts are securely contained within physical security boundaries and are accessible only to authorized personnel. While this may not always be strictly true, security designs based on this idealization have worked well for the most part. However, edge devices are frequently *not* secured by the same stakeholder who runs the cloud. Instead, they are inside a car, drone, or home of a normal end user, or even just outside in the natural world (sensors, edge caches).

It is a strong limitation of OpenStack that it trusts its compute nodes and, if any compute node is compromised, the entire cloud infrastructure may be brought down [23]. As we are extending OpenStack to model edge devices as compute nodes, this trust of compute nodes becomes even more unwarranted. Future work must address this fact and attempt to limit the trust granted to edge device compute nodes to a scope that is manageable.

In our current implementation, if a device's misbehavior is detected, we can revoke its ALGS credential, the cryptographic key needed for packets to transit the GA system. This removes the device from any further communication with either the cloud or other devices, at least via the GA system (so it won't become a candidate host for future orchestration actions, for example). Revocation of VPN certificates could close off packets into the cloud as well.

Security is a trio, not a duet. In a typical cloud, orchestration actions need only be authorized by an application authenticating to the cloud. Once the cloud is convinced that a request comes from a known, authorized application, it goes ahead with the action.

However, in our scenario, there is a third party involved, namely the owner of the edge device. That is, we are *not* proposing a model where one user's application can just force another's device to execute arbitrary code. Instead, there are three authentications that must be strong and enforced: user to application, application to cloud, and user to cloud. That is, for an orchestration action to proceed, first a user must take an authentic and authorized action within the application; next, the cloud must authenticate the identity and authorization of the requesting application; and third, the cloud must securely verify that the owner of the target device(s) has authorized the requested action under some known and current policy specification that has been securely authenticated with the owner.

OpenStack has no concept of authentication of the owner of a host to the cloud, because it implicitly assumes the data center is under the control of the same entity who deployed OpenStack into it. This is a limitation that must be studied and addressed in future work.

Privacy of stakeholders also becomes a more complicated issue when edge devices and their owners are considered. This is another area where additional work is needed.

Issues in Mixed Initiative Control We have only begun to explore the issues in mixed initiative control (see Section 3.2.7). In particular, we need a general policy framework that allows a device owner/operator to express conditions under which applications are authorized to execute orchestration actions on the owned device. Such policies may reasonably be expected to depend upon location, time, identities of the parties involved, resource state, energy state, and many others.

Secondly, we need to explore policies to allow safe and understandable methods for determining when one computation can usurp resources from another. For example, safety critical applications must be given priority over entertainment and other low priority applications. Further, the owner/operator must have an appropriate priority relative to the platform operator; for example, who decides when software is upgraded? For which applications? There are many interesting issues raised by mixed initiative control.

8. Conclusion

In this work we have addressed the question of how to treat huge numbers of real world IoT devices as members of a cloud, specifically attempting to make feasible the communications and processing load by limiting awareness messaging to only during times and at locations where a transaction will take place, whether said transaction is a management action, an invocation action, or another type.

Our primary contribution is the FocusStack approach, based on location based situational awareness implemented using geographic addressing feeding into an otherwise typical OpenStack installation. We have shown that this can reduce the awareness traffic by astronomical factors, bringing the approach into the realm of feasibility. We have demonstrated the approach through an extended and implemented case study of a shared video application.

FocusStack solves two major problems standing in the way of treating mass numbers of IoT devices as cloud hosts. First, we show that we can reduce management awareness traffic by huge factors through location based situational awareness. The second problem, of dealing with mixed initiative management control, which is not present in a traditional cloud, is addressed by this same location based awareness mechanism: even though arbitrary user actions can take place during times when the cloud is not paying attention, by building a current awareness picture when needed, the cloud can still make informed and timely management decisions when the cloud needs to do so.

We feel this work represents positive first steps toward applying cloud management to large scale IoT device clouds and should be pursued.

References

[1] "OpenStack." http://www.openstack.org. Accessed: 2016-02-29.

[2] R. Hall, J. Auzins, J. Chapin, and B. Fell, "Scaling up a geographic addressing system," in *Proc. of the 2013 IEEE Military Communications Conference*, 2013.

[3] "Docker." https://www.docker.com. Accessed: 2016-02-29.

[4] "DD-WRT." http://www.dd-wrt.com/site/index. Accessed: 2016-02-29.

[5] "The OpenXC Platform." http://openxcplatform.com. Accessed: 2016-02-29.

[6] R. Hall, "A geocast based algorithm for a field common operating picture," in *Proc. of the 2012 IEEE Military Communications Conference*, 2012.

[7] R. Hall, "An improved geocast for mobile ad hoc networks," *IEEE Transactions on Mobile Computing 10(2)*, 2011.

[8] Q. Xu, J. Huang, Z. Wang, F. Qian, A. Gerber, and Z. M. Mao, "Cellular data network infrastructure characterization and implication on mobile content placement," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pp. 317–328, ACM, 2011.

[9] G. Jung, M. Hiltunen, K. Joshi, R. Panta, and R. Schlichting, "Ostro: Scalable placement optimization of complex application topologies in large-scale data centers," in *Proc. IEEE International Conference on Distributed Computing Systems*, (Columbus, OH), pp. 143–152, June 2015.

[10] J. Hirsch, "253 million cars on u.s. roads; average age is 11.4 years," L.A. Times, June 2014.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, Oct. 2009.

[12] D. Willis, A. Dasgupta, and S. Banerjee, "Paradrop: A multi-tenant platform to dynamically install third party services on wireless gateways," in *Proceedings of the 9th ACM Workshop on Mobility in the Evolving Internet Architecture*, MobiArch '14, (New York, NY, USA), pp. 43–48, ACM, 2014.

[13] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the Third ACM Workshop on Mobile Cloud Computing and Services*, MCS '12, (New York, NY, USA), pp. 29–36, ACM, 2012.

[14] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: Enabling remote computing among intermittently connected mobile devices," in *Proceedings of the Thirteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '12, (New York, NY, USA), pp. 145–154, ACM, 2012.

[15] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, "Femtoclouds: Leveraging mobile devices to provide cloud service at the edge," in *IEEE International Conference on Cloud Computing (IEEE CLOUD)*, 2015.

[16] Y. Igarashi, K. Joshi, M. Hiltunen, and R. Schlichting, "Vision: Towards an extensible app ecosystem for home automation through cloud-offload," in *Proceedings of the Fifth International Workshop on Mobile Cloud Computing & Services*, MCS '14, (New York, NY, USA), pp. 35–39, ACM, 2014.

[17] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), pp. 49–62, ACM, 2010.

[18] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), pp. 301–314, ACM, 2011.

[19] K. Fall, G. Iannaccone, J. Kannan, F. Silveira, and N. Taft, "A disruption-tolerant architecture for secure and efficient disaster response communications," in *In ISCRAM*, 2010.

[20] K. M. Hanna, B. N. Levine, and R. Manmatha, "Mobile distributed information retrieval for highly-partitioned networks," in *IN IEEE ICNP*, pp. 38–49, 2003.

[21] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, IPDPS '09, (Washington, DC, USA), pp. 1–8, IEEE Computer Society, 2009.

[22] D. P. Anderson, "Boinc: A system for public-resource computing and storage," in *5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004.

[23] W. K. Sze, A. Srivastava, and R. Sekar, "Hardening openstack cloud platforms against compute node compromises," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, (New York, NY, USA), pp. 341–352, ACM, 2016.