Bringing the Cloud to the Edge

Hyunseok Chang Adiseshu Hari Sarit Mukherjee T.V. Lakshman
Bell Labs, USA
{firstname.lastname}@alcatel-lucent.com

Abstract—**Edge services become increasingly important as the Internet transforms into an Internet of Things (IoT). Edge services require bounded latency, bandwidth reduction between the edge and the core, service resiliency with graceful degradation, and access to resources visible only inside the NATed and secured edge networks. While the data center based cloud excels at providing general purpose computation/storage at scale, it is not suitable for edge services. We present a new model for cloud computing, which we call the Edge Cloud, that addresses edge computing specific issues by augmenting the traditional data center cloud model with service nodes placed at the network edges. We describe the architecture of the Edge Cloud and its implementation as an overlay hybrid cloud using the industry standard OpenStack cloud management framework. We demonstrate the advantages garnered by two new classes of applications enabled by the Edge Cloud — a highly accurate indoor localization that saves on latency, and a scalable and resilient video monitoring that saves on bandwidth.**

I. Introduction

Computation today is dominated by two major trends — cloud computing and mobile computing. The confluence of these two trends means that services that cannot run natively on mobile devices are provided by cloud based backends. As mobile devices improve in storage and processing power in accordance with Moore's Law, services evolve in sophistication and complexity at the same rate, ensuring that the high end applications such as augmented reality and 3D gaming will continue to have a cloud based backend. However, a cloud based backend suffers from lag when the cloud is remote from the end user and is not resilient to network and link failures.

Consider another class of applications that is becoming popular with the advent of various smart devices and the Internet of Things (IoT). We observe that a tremendous amount of data is generated at the edge of the network, for example, video captured by smartphones or tablets, machine-to-machine (M2M) sensor traffic, surveillance video feeds from single-board computers (e.g., Raspberry Pi [1]), etc. Usually this data is transported back to the cloud for storage and processing, which requires a high bandwidth connectivity into the central cloud. However, a majority of the data can be pre-processed and compacted. For example, user video can be compressed, M2M communication can be coded, and surveillance video can be filtered for abnormal incidents. With such pre-processing, the transport cost can be drastically reduced. To achieve this, however, large scale computation is needed at the edge of the network, which is not supported by the centralized data center oriented cloud computing model.

What we observe from the cases above is that although a cloud backend is needed for these types of high performance applications, the end user will benefit from lower latency and bandwidth consumption if the cloud backend is moved either wholly or partially to the edge. This requires a new model in which the centralized data center based cloud is augmented with a presence at the edge to support the emerging edge based applications that are both computation and data intensive.

In this paper, we present such a model of a hybrid cloud architecture we refer to as the *Edge Cloud*, which is designed to deliver low-latency, bandwidth-efficient, and resilient end user services with a global footprint. The Edge Cloud is designed to extend the data cloud all the way to the end user in the home or enterprise by leveraging user and operator contributed compute nodes at the edge. The Edge Cloud facilitates a split cloud application architecture by seamlessly interconnecting edge networks with data center networks, thereby enabling latency-sensitive computation and user interaction components close to end users at the edge nodes, while hosting additional heavy-duty processing and database components in the data center nodes.

In this architecture, end users benefit from the low latency between the users and the Edge Cloud which is now part of their local networks. The latency benefit is particularly pronounced for users far from the data center. Another benefit is the bandwidth reduction provided by the preprocessing at the edge, which reduces the bandwidth requirements between the edge and the core. The Edge Cloud is able to access various edge resources such as sensors, laptops and computers which are not visible outside the edge network. Finally, application resiliency is provided in two ways. In case edge resources become temporarily unavailable, edge components of an application can fall back to the more reliable data center cloud. Conversely, if an edge component can provide an application's basic functionality by itself, the application can continue to function even without data center components (due to lost/flaky Internet connection or data center outage). Either way, the Edge Cloud can provide graceful degradation of service — a particularly valuable feature for any cloud application.

The key contributions of this paper are as follows:

- A new paradigm for delivering services to the edge which combines the advantages of data center based service delivery with local presence at the edge.
- An instantiation of this paradigm on the OpenStack cloud management platform with OpenStack extensions to support NAT-friendly and secure virtual tenant networks.
- Quantitative evaluation of two Edge Applications (Edge Apps): 3D indoor localization and video surveillance.

The rest of this paper is organized as follows. We describe the Edge Cloud architecture in Section II, and its implementation on OpenStack in Section III. We present two functional Edge Apps in Section IV, and conclude in Section V.

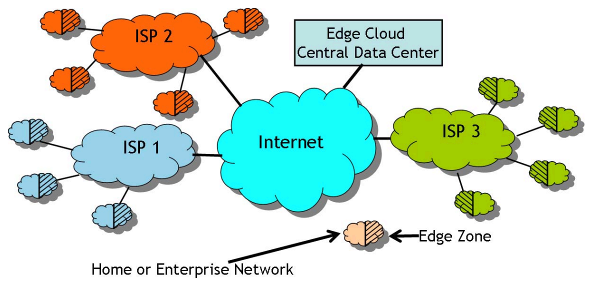


Fig. 1. The distributed Edge Cloud.

II. Edge Cloud Architecture

In this section, we describe the architecture of the Edge Cloud which is designed to deliver low-latency, bandwidth-efficient, and resilient end user services closer to end users by leveraging edge compute nodes.

A. Edge Networks and Edge Cloud

Edge network refers to the part of the Internet to which end users directly connect (e.g., home/enterprise networks or WiFi hotspots). *Edge node* refers to the compute or storage node attached to the edge network and federated to the data center cloud. Edge nodes in the same edge network are grouped together to form an *Edge Zone*. All nodes in the same edge zone are assumed to be part of the same, potentially NATed, network address space managed by the edge owner.

The *Edge Cloud* is the federation of the data center nodes along with all the edge zones. The Edge Cloud operator is assumed to have a pre-existing, traditional IaaS data center cloud. By adding Edge Cloud functionality, the Edge Cloud operator can now extend the cloud's capabilities to deploy applications at the edge networks. Fig. 1 shows the presence of the various edge zones in the Edge Cloud.

B. Edge Apps

We introduce the concept of *Edge Apps* to provide services in the Edge Cloud. An Edge App is a bundled set of IaaS images which are designed to launch and work together in unison in the data center and the edge nodes, to provide a service for mobile devices and smart IoT devices. An Edge App is composed of two types of virtual networks and two sets of compute instances. One set of compute instances run in the data center, and the other set in the edge zone of the edge owner. In each set, there can be different types of IaaS images, and each type of image can be deployed on multiple compute instances (indicated by a replication factor). A virtual network called the *App-Private Network* is instantiated for each Edge App on startup to interconnect all instances belonging to the app. The communication of the Edge App with end users and resources in the edge network is provided by the *Edge-Local Network*, which bridges Edge App instances running in the edge nodes to the local edge network.

Figure 2 shows a sample Edge App layout consisting of four IaaS images, labeled `image1` through `image4`. `image1` and `image2` are used to realize the edge components with a replication factor of one and two respectively, while `image3` and `image4` realize the data center components with a replication factor of three and four respectively. `image1` is connected to the edge-local network, while all components are connected to the app-private network.

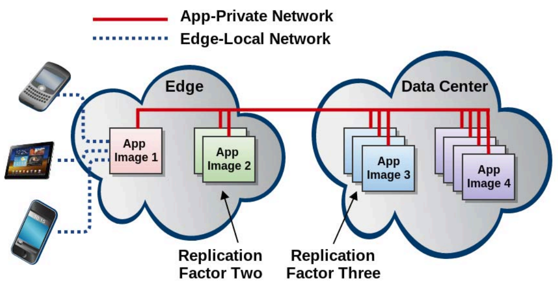


Fig. 2. Anatomy of an Edge App.

III. Edge Cloud Implementation

We implement the Edge Cloud architecture by using the Folsom release of OpenStack, an industry-standard cloud management platform. There are several challenges in building the Edge Cloud on top of OpenStack because OpenStack is originally designed as an IaaS platform for data centers.

For example, unlike the traditional data center hosted cloud, the Edge Cloud needs to support low cost, low power platforms which may not support hardware virtualization. Fortunately, OpenStack started to support lightweight Linux Containers (e.g., LXC or Docker). We adopt LXC as a virtualization primitive in the Edge Cloud instead of hypervisor based hardware virtualization.

More importantly, there is no mechanism in OpenStack for managing or deploying Edge Apps on compute nodes which are spread across heterogeneous, potentially NATed, L3 networks. In the Folsom release of OpenStack, Quantum component is dedicated to managing virtual tenant networking. We extend OpenStack Quantum to support the virtual networking in the Edge Cloud. In our extended Quantum implementation, Quantum server and agent communicate via a custom Quantum plugin to manage virtual networks for different Edge Apps on demand. In the rest of the implementation section, we describe this extension in more detail.

A. Edge Cloud Quantum Plugin

Quantum can support various network virtualization technologies by leveraging the notion of "plugins." We design and implement a custom Quantum plugin that allows OpenStack controller to manage edge nodes located behind NATs across different L3 networks, and to instantiate virtual app-private networks spanning these nodes. First, we assume that there is a dedicated management network over VPN between Quantum server (i.e., server component of Quantum) and Quantum agent running on each edge node. Such VPN can easily be created by Quantum agent even behind NAT. Quantum agent is then responsible for configuring app-private networks across edge nodes. Since an app-private network may span across multiple NATed edge nodes over different L3 networks, Quantum agent creates per-app virtual L2 underlay that interconnects edge nodes over physical L3 networks, over which actual app-private networks are maintained. Per-app underlay configuration information is centrally managed by Quantum server, and shared among compute nodes involved to allow connectivity among different Edge App instances over an underlay.

B. L2 Tenant Networking in the Edge Cloud

To create NAT-friendly L2 underlay for app-private networks, we use the open-source `tinc` VPN software [2]. Using VPNs for L2 underlays has several advantages in the Edge Cloud. First, it allows edge nodes residing in different, potentially NATed, L3 networks to be integrated into the address space of the data center. Second, it provides security by encrypting the communications to and from the edge nodes, which is important because the edge nodes are no longer in a trusted zone such as the data center. Third, it provides a single point of control for all Edge Cloud traffic to and from each edge node. This point of control can be used to monitor and optionally throttle Edge Cloud traffic.

Our choice of `tinc` VPN software is driven by the fact that it supports mesh-based VPN topologies, as opposed to hub-and-spoke VPN topologies. In a mesh-based topology, any node can initiate/serve a connection to/from any other node, and form a direct connection in between. Furthermore, `tinc` has STUN [3] functionality built in, and any VPN node with publicly reachable IP address can play the role of a STUN server, bootstrapping nodes behind cone-type NAT gateways to connect with each other directly. This means that traffic from Edge App instances running on different NATed nodes will flow directly between the two nodes without any intermediary node, which greatly improves inter-NAT delay performance and removes any scaling bottleneck in L2 underlays.

C. Extended Quantum Server and Database

As described before, Quantum server stores and manages per-app VPN configuration information including public keys for authentication. To support this, we extend Quantum database schema by adding a new table called ‘app public key’. Each entry in the table stores a public key associated with a specific compute node, generated for an Edge App which has any instance on the node. We then extend Quantum RESTful APIs and RPC communication supported by Quantum server, so that a given app’s public key can be added or removed in Quantum database, and the public key table can be looked up using several search criteria (e.g., App ID, network ID).

D. Public Key Server

To set up a mesh-like per-app L2 underlay among a set of compute nodes over VPN, each participating node must know the public keys of all the other nodes in the underlay. To ensure that app-specific public keys are properly deployed on all participating nodes, we rely on a separate “public key server.” The role of the public key server is to distribute or reclaim public keys to and from the compute nodes involved in the app-specific underlay, upon any change in node membership of the underlay. We develop a simple public key server which deploys or removes public keys on compute nodes over SSH, upon requests from Quantum agent. By design, our public key server is purely stateless; public keys are not stored inside the key server, but rather retrieved from Quantum server via Quantum APIs. That way, there is less administrative overhead in operating the key server, and no inconsistency between Quantum server and the public key server.

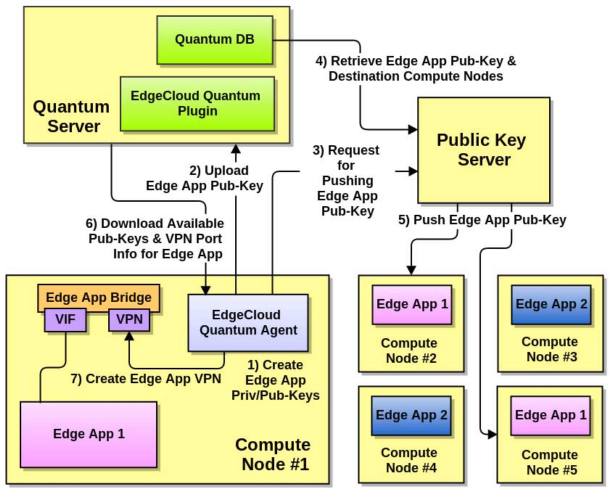


Fig. 3. Quantum extension in the Edge Cloud.

E. Quantum Agent

Quantum agent is a daemon process running on each compute node. The agent communicates with Quantum server via Quantum APIs over a RPC connection, and uploads/downloads app-specific public keys to/from Quantum database. Upon request from Quantum server to create a new app-private network and to add a particular Edge App instance to the network, Quantum agent proceeds as follows.

- If there is no L2 underlay provisioned for the app-private network on this compute node, generate a private/public key pair for the app-private network on the node.
- Upload the created public key to Quantum server via RPC, which is stored in Quantum database.
- Request a public key server to distribute the public key to other compute nodes connected to the app-private network.
- Retrieve from Quantum database all available public keys that have been deployed on other compute nodes for the same app-private network, and VPN port number to use for the app-private network.
- Using collected public keys and port number, generate necessary underlay configuration, and launch an L2 underlay associated with the app-private network.
- Finally, attach the underlay to the app-private network (i.e., add the underlay’s VPN interface to app’s Linux bridge).

The step-by-step process of provisioning an app-private network via the Quantum plugin/agent is illustrated in Fig. 3.

IV. Edge App Implementation and Experiments

In this section, we describe the implementation of two Edge Apps, and present experimental results. In the experiments, we focus on illustrating the benefits of the Edge Cloud as an end user service delivery platform, e.g., reducing latency and bandwidth of application traffic and providing graceful degradation of service.

A. 3D indoor Localization Application

The first Edge App we present is 3D indoor localization, where the goal is to find the 3D position and orientation of a

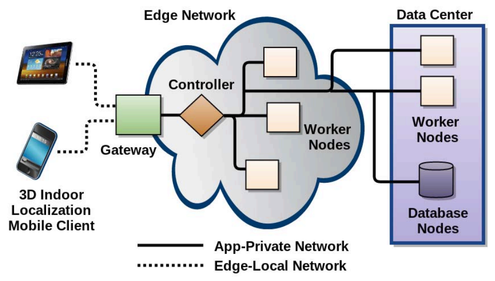


Fig. 4. 3D indoor localization app.

user's mobile device in a particular indoor environment. The high-fidelity 3D localization increasingly finds practical usage in augmented reality and interactive guidance applications.

In [4], we present a vision-based 3D indoor localization system for smart devices such as smartphones or tablets, which can achieve sub-meter level 3D localization accuracy. In this approach, an indoor map that captures the 3D geometry of a given indoor environment is pre-built by using depth-sensing camera [5]. In order to find the 3D position and orientation of a smart device, we first capture a 2D photo image and accelerometer measurement from the device. Based on this data, we then compute the translation vector and rotation matrix between 3D indoor map and the viewpoint captured in 2D photo, by solving a nonlinear optimization problem with six variables (i.e., 3-axis position/orientation of a rigid body) iteratively. It turns out that this optimization problem is too computation-intensive to run natively on the current generation of smart devices, not to mention the storage and bandwidth needed to download and store 3D indoor maps. This makes 3D indoor localization an ideal Edge App that can leverage low-latency compute resources at the edge.

In the following, we describe how we deploy the 3D indoor localization app in the Edge Cloud, and then present experimental results relevant to the Edge Cloud environment.

1) 3D indoor localization in the Edge Cloud: The cloud portion of the 3D indoor localization is powered by JPPF parallel computing framework [6], and is spread across the Edge Cloud and an external data center. When the connectivity to the data center goes down, the indoor localization app can continue to function on the Edge Cloud, possibly with degraded performance — the type of application resiliency mentioned before. The cloud infrastructure needed for the app consists of four components as shown in Fig. 4.

Gateway: It is the front end of the 3D indoor localization app which exposes a web service interface to end users. The gateway and end users communicate with each other via an edge-local network. The localization client software running on user's device captures a 2D photo and accelerometer measurement, and sends a localization request to the gateway. When the gateway receives the request, it delegates localization computation to a compute cluster via an app-private network, and sends the result back to end user after computation is done.

Controller: The controller is a job scheduler component of the compute cluster, responsible for coordinating the overall execution of localization computation in the compute cluster. Controller receives a job from the gateway, splits the job into multiple tasks, and assigns the tasks to individual compute nodes in the cluster for parallel execution. Each task is then run independently, and its output is sent back to the controller. Controller aggregates the output of individual tasks, and sends a final result to the gateway.

Worker node: A worker node executes tasks assigned by the controller. It can be deployed at the edge (for latency) or in the data center (for reliability). Different worker nodes and the controller are interconnected with an app-private network.

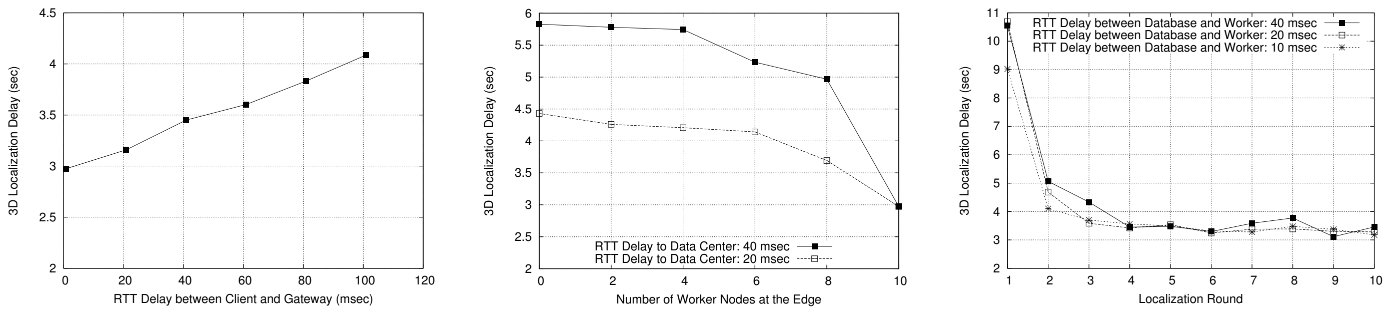
Database node: A database node is an optional node in the data center that physically hosts indoor map database files for worker nodes to access during localization computation. Alternatively, map database files can be co-located on each worker node, in which case the database node is not necessary.

The deployment of 3D indoor localization app in the Edge Cloud involves creating an app-private network dedicated to the app, and installing three app images and one database on the aforementioned four components. The app image for the gateway contains a Java Servlet for indoor localization, while the app images for controller and worker nodes contain the JPPF framework components. A database node houses and exports (e.g., via NFS) indoor database files.

2) Experiments: The full-functional 3D indoor localization app is deployed on the Edge Cloud, which consists of 12 node instances (one gateway, one controller and ten worker nodes). The 45MB indoor map, which captures the 3D geometry of an office room of $3m \times 3m \times 3m$ dimension, is deployed on each worker node. We use pre-recorded 35 camera shots of the office room and accelerometer measurements, both taken from Galaxy tablet PC, to conduct 3D localization. We use `netem`, kernel support for network emulation in Linux, to emulate different network latencies between nodes. The metric used in our evaluation is the user-perceived end-to-end delay of 3D localization (i.e., time taken by client to receive a result after requesting for localization) In our experiments, we consider several deployment scenarios for 3D indoor localization app.

Deployment in the data center. We first examine the deployment scenario where the app runs purely at the edge or in the data center. To emulate the data center deployment, we vary the network latency between client and the gateway, and examine what impact it has on the end-to-end 3D localization delay. The result is shown in Fig. 5 (a). The x-axis represents the average network latency to the data center. The special case of zero RTT delay corresponds to the deployment purely at the edge. The figure shows that the 3D localization delay increases linearly with network latency to the data center. An increase of 100 msec latency leads to 25% higher 3D localization delay. This illustrates the importance of cloud proximity in latency-sensitive, computation-intensive cloud apps such as 3D localization. More generally, the curves in Fig. 5 (a) could be pushed up or down depending on the complexity of computations needed for a given app, but the slope of the curve would remain the same due to network latency constraints.

Hybrid deployment. Next we consider a hybrid deployment where 3D indoor localization is deployed partially at the edge as well as in the data center. For the experiment, we first configure the network latency between the controller and ten worker nodes to a small value (e.g., 0.5 msec), as if the worker nodes were located in the local edge network. Then we choose some worker nodes out of ten, and artificially increase



(a) Latencies between client and gateway. (b) Latencies between controller and workers. (c) Latencies between workers and storage.

Fig. 5. The effect of network latencies on 3D indoor localization delay.

their network latency from the controller, to a set value (e.g., 20 or 40 msec), as if the close-by edge nodes were replaced by distant data center nodes. As we move more worker nodes from the edge to an external data center like this, we check how that affects user-perceived 3D indoor localization delay.

In Fig. 5 (b), we plot the 3D localization delay as a function of the number of worker nodes at the edge. The left extreme on x-axis corresponds to a case where all ten worker nodes are in the data center, while the right extreme means all ten worker nodes at the edge. The figure demonstrates the degraded performance of indoor localization computation (in terms of delay) as edge nodes start to disappear. In this particular app, the increase of 3D localization delay with fewer edge nodes is substantial. The reason for such sensitivity is that the 3D localization computation involves *multiple* rounds of interactions between the controller and worker nodes to solve smaller problems (e.g., feature point matching, iterative optimization for localization). Thus, even a small increase in network latency between the controller and worker nodes leads to a much longer end-to-end localization delay. While this observation only applies to a particular implementation and setup of our 3D indoor localization app, the benefit of edge processing and graceful degradation is still valid.

Deployment at the edge with storage in the data center. Next we examine the effect of the storage location in the deployment. For 3D indoor localization, storage is needed to store indoor database map files. While indoor map could be deployed on individual worker nodes along with application binaries, multiple compute nodes could share storage for space efficiency or low app provisioning time. In this experiment, we provision a separate database node which houses and exports locally stored indoor map files to other worker nodes via NFS. To emulate a database node in the data center, we vary network latencies between the database node and worker nodes. On each worker node, we enable file caching for remote NFS share, so that it can locally cache previously accessed indoor map files. In this setup, client continues to issue localization requests, a batch of ten requests in each round. We examine average 3D localization delay in each round as the number of rounds increases under varying network latencies, and plot the result in Fig. 5 (c). The figure shows the dramatic performance difference between "cold" cache (e.g., round 1–2) and "hot" cache (e.g., round 4–), across all network latencies. The dramatic increase in localization delay with cold cache demonstrates the importance of storage proximity for compute nodes. For the 3D localization application which typically runs over a long term with the same data sets, the storage latency issue can be mitigated by file caching as shown, when the database node is provisioned in a distant data center.

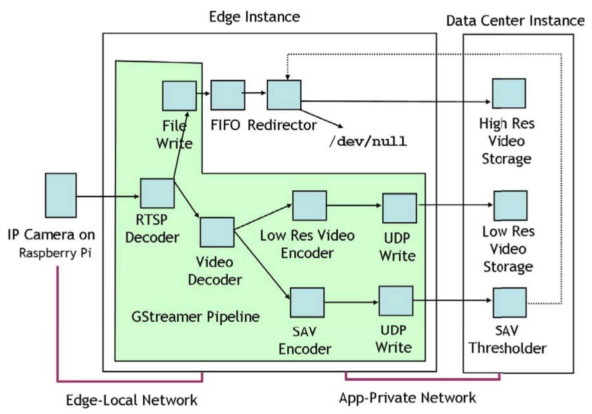


Fig. 6. SAV based video surveillance app.

B. Video Surveillance Application

The second Edge App we present is Scene Activity Vector (SAV) based video monitoring and surveillance [7], which is based on a highly-compressible video feature and motion detection and filtering algorithm. Due to the popularity of single-board computers (e.g., Raspberry Pi) nowadays, IP camera based surveillance becomes increasingly popular. Video surveillance applications are usually based on monitoring and archiving video feeds collected from IP cameras at a central data center. In comparison, the SAV based video surveillance application can optimize the bandwidth consumption of video surveillance feeds by highly compressing the feeds between the edge and the central site. The input to SAV is a sequence of uncompressed motion video images obtained from a video stream, and its output is a series of SAVs. Compared to the original video stream from the IP camera, SAV is highly compressed, with compression ratios exceeding 100 or more.

Our Edge App implementation of the SAV based video surveillance App is as shown in Fig. 6. There are two instances — one in the edge and the other in the data center. The IP camera is mounted on Raspberry Pi. The SAV module is implemented as a `gstreamer` [8] module. The `gstreamer` framework is used to create a three-way pipeline as shown in the figure. The first pipeline takes the original high-resolution video stream and sends it to a FIFO pipe. The subsequent Redirector pipe is configured to discard it by default, but can also send it to a data center instance on demand. The second pipeline decompresses the video, and sends it to SAV

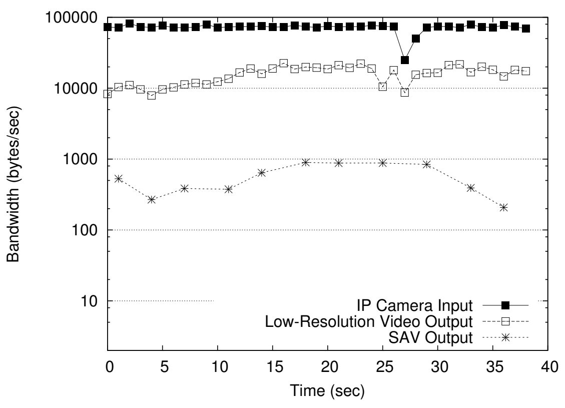


Fig. 7. Bandwidth usage: SAV vs. IP camera.

processing before sending it upstream. The third pipeline decompresses the video before re-compressing it to a lower resolution and framerate, and then sends it upstream. The data center instance consists of an archiving module which takes the low resolution video for archiving, and a SAV thresholding module which generates a trigger when the SAV activity exceeds a threshold. When a trigger is generated, the Redirector pipe of the edge node is re-configured to send the data upstream for additional processing. After a specified time period, the Redirector pipe reverts to its original setting.

Fig. 7 shows the bandwidth consumption of the incoming RTSP video stream from IP camera, the output SAV stream and the output low-resolution video stream over a representative interval. The IP camera is configured to generate a H.264 stream at 15 fps with 640×480 resolution. The low-resolution video is a 340×240 FLV stream. Over the time interval shown in the graph, the bandwidth of SAV was merely 0.2% of the IP camera, and the low resolution video consumed 21% bandwidth needed by the IP camera, which illustrates the fact that Edge Apps can be used to provide bandwidth optimized services at the edge by leveraging edge resources.

Another benefit to edge owners is graceful degradation or service resiliency. It is possible to run the SAV thresholding program in the edge instance itself, rather than in the data center, at the cost of reduced accuracy, as the data center instance can run on faster hardware, and access historical databases for more intelligent surveillance. In case of network outage, video surveillance can still continue to run by switching to the thresholding module running in the edge instance.

V. Conclusions and Related Work

In this paper, we present a computational model called the Edge Cloud for bringing the cloud all the way to the edge to enable a new type of hybrid applications called Edge Apps. We describe its implementation over OpenStack, as well as two functional Edge Apps we implemented and deployed in the Edge Cloud prototype.

Numerous efforts have been made to cluster unused edge computing resources into a globally available compute pool, e.g. BOINC [9], BOINC based SETI [10], SEATTLE [11]. While the goal of these efforts is to harness compute cycles at the edge using a parallel computational workflow, the Edge Cloud and Edge App models are designed for general application services. In addition, the Edge Cloud allows Edge Apps to execute in a coordinated manner in the edge as well as the data center. Such hybrid execution distinguishes the Edge Cloud from other cloud computing models like Cloudlets [12] and Nanodatacenters [13], which focus solely on bringing the data center to the edge. Fog computing [14] presents a paradigm for a cloud extension called the "fog" to interface with IoT platforms like connected vehicles and smart grids. RACE [15] is a framework and system for specifying and executing a class of distributed real-time applications in the cloud based on real-time feeds collected from a large number of edge-devices. The Edge Cloud is a more general model which can work with all types of edge services including IoT platforms and regular enterprise/home applications, and leverages a standard cloud platform rather than a specialized application level framework.

The Edge Cloud is a natural extension to ISPs seeking to provide enhanced edge services to their customers. For example, Comcast is planning to partition WiFi access on home gateways into separate private and public wireless networks, with the public network acting as a WiFi hotspot to nearby Comcast customers [16]. While this is not a full-blown Edge Cloud functionality, it illustrates the ability of ISPs to manage black boxes at the edge.

The concept of the distributed Edge Cloud raises several interesting open issues, e.g., application deployment strategies (i.e., where to place application instances), failure recovery (i.e., how to recover from failed edge nodes), distributed bandwidth resource management (i.e., how to share limited bandwidth resources at the edge), edge node security (i.e., how to protect Edge App components hosted at the edge). We plan to explore these issues as we refine the Edge Cloud model.

References

[1] "Raspberry Pi," http://www.raspberrypi.org.

[2] "Tinc VPN," http://www.tinc-vpn.org.

[3] J. Rosenberg *et al.*, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489, Mar. 2003.

[4] D. Yun *et al.*, "Sub-meter Accuracy 3D Indoor Positioning Algorithm by Matching Feature Points of 2D Smartphone Photo," in *Pacific PNT*, Apr. 2013.

[5] "Kinect for Xbox 360," http://www.xbox.com/kinect/.

[6] "JPPF," http://www.jppf.org.

[7] L. O'Gorman, Y. Yin, and T. K. Ho, "Motion Feature Filtering for Event Detection in Crowded Scenes," in *PRCA*, 2012.

[8] "GStreamer," http://gstreamer.freedesktop.org.

[9] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage," in *5th IEEE/ACM International Workshop on Grid Computing*, 2004.

[10] D. Anderson *et al.*, "SETI@home: An Experiment in Public-Resource Computing," *Communications of the ACM*, vol. 45, 2002.

[11] C. Kim *et al.*, "Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises," in *ACM Sigcomm*, 2008.

[12] M. Satyanarayanan *et al.*, "The Case for VM-based Cloudlets in Mobile Computing," in *IEEE Pervasive Computing*, 2012.

[13] "Nanodatacenters," http://www.nanodatacenters.eu.

[14] F. Bonomi *et al.*, "Fog Computing and Its Role in the Internet of Things," in *ACM Sigcomm MCC*, 2012.

[15] B. Chandramouli *et al.*, "RACE: Real-time Applications over Cloud-Edge," in *ACM Sigmod*, 2012.

[16] "Comcast Integrated Private-Public Wifi," http://goo.gl/gWTMS.