Incident-Supporting Visual Cloud Computing Utilizing Software-Defined Networking

R. Gargees[†][*], B. Morago[†][*], R. Pelapur[†][*], D. Chemodanov[†][*],
P. Calyam[‡][*], Z. Oraibi[*], Y. Duan[*], G. Seetharaman[§] and K. Palaniappan[*]

***Abstract*—In the event of natural or man-made disasters, providing rapid situational awareness through video/image data collected at salient incident scenes is often critical to first responders. However, computer vision techniques that can process the media-rich and data-intensive content obtained from civilian smartphones or surveillance cameras require large amounts of computational resources or ancillary data sources that may not be available at the geographical location of the incident. In this paper, we propose an incident-supporting visual cloud computing solution by defining a collection, computation and consumption (3C) architecture supporting *fog computing* at the network-edge close to the collection/consumption sites, which is coupled with *cloud offloading* to a core computation, utilizing software-defined networking (SDN). We evaluate our 3C architecture and algorithms using realistic virtual environment testbeds. We also describe our insights in preparing the cloud provisioning and thin-client desktop fogs to handle the elasticity and user mobility demands in a *theater-scale* application. In addition, we demonstrate the use of SDN for on-demand compute offload with congestion-avoiding traffic steering to enhance remote user Quality of Experience (QoE) in a *regional-scale* application. The optimization between fog computing at the network-edge with core cloud computing for managing visual analytics reduces latency, congestion and increases throughput.**

***Index Terms*—Visual Cloud Computing, User QoE, Adaptive Resource Management, Software-Defined Networking**

I. Introduction

In the event of natural or man-made disasters, videos and photographs from numerous incident scenes are collected by security cameras, civilian smart phones, and from aerial platforms. This abundance of media-rich video/image data can be extremely helpful for emergency management officials or first responders to provide situational awareness for law enforcement officials, and to inform critical decisions for allocating scarce relief resources (e.g., medical staff/ambulances or search-and-rescue teams). Using computer vision methods to build dynamic 3-dimensional (3D) reconstructions of salient structures in the incident region by fusing crowd-sources and surveillance imagery can increase situational awareness in a *theater-scale* setting of the incident scene [45]. Further, objects of interest can be tracked in aerial video at the *regional-scale* of incident scenes to provide analytics for planning wide-area relief and law enforcement activities [24].

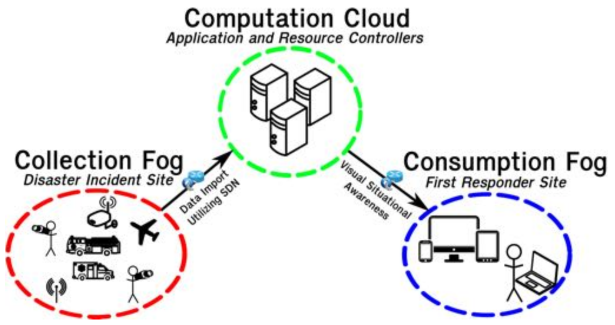

Computation Cloud
Application and Resource Controllers
Collection Fog
Disaster Incident Site
Data Import Utilizing SDN
Visual Situational Awareness
Consumption Fog
First Responder Site

Fig. 1. Overview of the visual collection, computation and consumption (3C) system linking the fog at the network-edge with core cloud computing utilizing SDN which is shown on the links.

However, the computer vision techniques needed to process this media-rich and data-intensive content require large amounts of computational resources that are usually intermittently available, damaged or unavailable within the geographic location of the incident scenes. Emerging techniques in the field of mobile visual cloud computing are well suited for scalable processing of media-rich visual data [2]. Private cloud 'fogs', as well as overlay network paths that are dynamically constructed using software-defined networking (SDN) [33], [46] rely on non-traditional network protocols such as OpenFlow [1]. These can be valuable in the case of damaged or congested network infrastructure within the geographical area of incidents. Fog computing extends cloud computing closer to the network-edge locations of users and data sources (see Figure 1). Coupled with SDN, fog computing at the edge can rapidly compute and organize small instance processes locally and move relevant data from the incident geographical location to core cloud platforms such as Amazon Web Services or NSF Global Environment for Network Innovations (GENI) [4] for on-demand processing. Moreover, the overlay network paths can also be useful for moving cloud-processed data closer to the locations of first responders for content caching at fogs to enable low-latency access via thin-client desktops. Such on-demand computation and integration of thin-clients for visualization can enable large data processing within the cloud and deliver high user Quality of Experience (QoE).

In this paper, we address the incident-supporting visual cloud computing and SDN-based data movement challenges by combining the latest advances in the fields of computer vision, cloud computing, and high-speed networking. Specifically, we define a *collection*, *computation* and *consumption* (3C) architecture shown in Figure 1. Our architecture assumes

[‡]Corresponding and primary author (calyamp@missouri.edu).
[†]These authors contributed equally to this work.
[*]Department of Computer Science, University of Missouri, USA.
[§]Advanced Computing Concepts/ITD, Naval Research Laboratory, USA.

This work was supported by the National Science Foundation under awards: ACI-1440582, NSF OCI-1245795, CMMI-1039433 and CNS-1429294, the U.S. Air Force Research Laboratory grant FA8750-14-2-0072 and FA8750-14-C-0044, and the U.S. National Academies Jefferson Science Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors' and do not necessarily reflect the views of the U.S. Government or any agency thereof.

incident videos/images are collected and pre-processed at a fog near the disaster scene and are transferred utilizing SDN to cloud servers where visual analytics such as 3D geometry, object recognition and tracking can be performed. The 3D visual environment, object and tracking results are subsequently transferred from the core cloud servers to a fog near first responder mobile devices or thin-client desktops for crucial visual data consumption. Based on this 3C architecture, we propose novel computation placement, and SDN control algorithms designed to enable *fog computing* closer to the collection/consumption sites, which is coupled with *cloud offloading* to a public cloud. Our algorithms assume the fogs are capable of handling small instance visual processing functions, and are integrated with a public cloud infrastructure for handling large instance visual processing functions by utilizing SDN. We describe how the 3C provisioning and placement algorithms for fog-cloud compute location selection and small/large instance visual processing can be parameterized in the contexts of: (i) a 'theater-scale' application for reconstructing dynamic visualizations from 3D LIDAR (Light Detection and Ranging) scans, and (ii) a 'regional-scale' application for tracking objects of interest in wide-area motion imagery (WAMI) from airborne platforms. We distinguish between theater-scale and regional-scale applications based on the geographical coverage of the incident and its distributed nature - with theater-scale being small regions distributed across multiple sites, and regional-scale being large regions.

The theater-scale application that we have developed [14], [35] builds 3D models of the environment by registering a set of 2D videos and 3D LIDAR scans to process large collections of videos available from civilian smart phones and surveillance cameras at an incident scene. LIDAR scans, and 3D visualizations have been shown to be useful for assessing damage at disaster sites since highly accurate scans can be obtained quickly to provide awareness of relative locations of activities across multiple viewpoints [28], [45]. This eases the cumbersome task of watching and analyzing numerous videos which are traditionally viewed on a grid of 2D displays. 3D LIDAR scanners that use a laser ranging device to determine distances to surfaces can be used to reconstruct a scene and provide a more intuitive venue for studying video sets. Advancing technology and decreasing costs during recent decades have led to LIDAR data being routinely incorporated for reconstructing, viewing, and understanding real-world scenes with convenience [35], [53]. Commonly used LIDAR data can be large in size (characterized for a typical resolution of about 1 cm for data collected at a range of up to 300m with 6 mm accuracy), and computationally expensive to process in cases of large-scale collection at incident scenes. Moreover, elastic resources must be available to take advantage of this rich source of information because every stage of the processing pipeline requires considerable but variable amounts of resources.

Our regional-scale application uses LOFT (Likelihood of Features Tracking) technology we have developed [38], [41], [42] to track and recognize objects of interest in aerial motion imagery. Such technology has become a vital part of intelligent search and rescue activities in recent years, and also has been proven to be essential in city-wide surveillance during event gatherings where the attendance is large enough to warrant a hawk-eye view in the interest of public safety. With the advent of newer sensor technology, it is now possible to capture high spatial resolution imagery that is data-intensive for wide-area surveillance with resolutions ranging between 10 cm to 1 m. The wide-area motion imagery processed by our LOFT framework is typically high spatial resolution of 25 cm GSD (Ground Sampling Distance), and low temporal resolution of about one to four frames per second [39]. Tracking in such imagery is computationally challenging because the objects of interest are small and have relative large motion displacement due to low frame rate sampling. WAMI data is challenging for automated analytics due to several reasons including: oblique camera viewing angles, occlusions from tall structures, tracking through shadows, variations in illumination, blurring and stabilization artifacts due to inaccurate sensors and atmospheric conditions.

Through detailed experiments in realistic virtual environment testbeds, we implement and evaluate our novel 3C architecture and compute/network resource control algorithms for the theater-scale and regional-scale applications. In our first set of experiments, we explore insights in preparing the cloud provisioning and thin-client desktop delivery within our VMLab platform [6] to handle the elasticity and user mobility demands in the theater-scale application contexts. We study handling LIDAR models and sets of videos in a disaster situation where many videos will be collected remotely, sent to the server for registration, and viewed on a mobile device in real time. We run our theater-scale application system over a regular wireless network and an high-bandwidth overlay network utilizing SDN to identify the network requirements for processing and viewing 3D video and delivering satisfactory user QoE. In the second set of experiments, we demonstrate the use of SDN for on-demand compute offload with congestion-avoiding traffic steering for the regional-scale application configured in the GENI platform [4]. We first consider multiple video resolutions corresponding to different mobile devices and emulate disaster network degradation conditions systematically to characterize the resultant impact on the compute offloading to the cloud. Next, we show user QoE improvements in data throughput and tracking time when remotely analyzing WAMI data, utilizing SDN and by dividing the application into small and large instance processing.

The remainder of this paper is organized as follows. Section II presents related work. Section III describes the theater-scale and regional-scale computer vision based applications. In Section IV, we present details of our 3C architecture. The compute/network resource control algorithms are presented in Section V. Section VI details our experimental setup and performance analysis. Section VII concludes our paper.

II. Literature Review

This section describes the novelty of our approach for incident-supporting visual cloud computing by combining synergies from the fields of computer vision, cloud computing and high-speed networking.

A. Visual Cloud Computing

Computer vision commonly deals with the processing of large data sets, and a typical system in this field usually comprises of several data processing stages such as: (a) acquisition,

(b) pre-processing, (c) analysis, and (d) post-processing. Data requirements change depending on the application in question, and the acquisition step itself usually requires an enormous amount of storage apart from the bandwidth requirements for processing. Separating storage and bandwidth requirements could greatly benefit overall processing time required for a data set. However, the processing time of applications can also have some restrictions based on the location at which they are hosted. In most cases, it is scalable to have data sent over to a cloud-hosted application host have it processed and have the analysis results sent back to the origin. Large-scale visualization and analysis such as NVIDIA's Grid Computing [21] have gained traction in the consumer market. Our work fosters the trend where multimedia cloud computing discussed in [22], [58], [55], [59] can provide high flexibility and mobility to the end user. Demonstrations of similar systems exist in literature and have been shown to work in an environment where hardware resources at data origin are limited [29], [30].

B. Disaster Management

3D representations of a disaster scenario can be transferred over wired/wireless networks to remote locations for better scene understanding than what a set of disjointed videos and photographs would provide. Several groups have investigated ways in which wireless networks can be set up and utilized for communication in the event of an emergency [27]. Authors in [10] have studied using wireless mesh networks to transfer medical information throughout disaster zones in situations where wired networks are damaged. In addition, authors in [56] set up overlay networks that allow humans to communicate with robots being used to explore the aftermath of a disaster. As these types of studies have become more popular, the speed of message delivery over the on-the-fly networks has been prioritized and explored in works such as [43]. Research on how to set up mobile cloud and overlay wired/wireless networks networks in a disaster scenario, and on how to create 3D models and simulations using LIDAR data have provided strong foundations for accomplishing this, but to the best of our knowledge, our work uniquely studies these topics in a combined manner.

C. 2D-3D Registration

Registering imagery with LIDAR scans has been studied a great deal in the computer vision field. The fact that 2D-3D data fusion allows large scale, photorealistic 3D models to be created very quickly and easily with a high degree of accuracy motivates much of this work. Many groups have focused on performing registration on urban data which generally has an abundance of regularized features that can be matched across dimensions [48]. These include line segments, arcs, and rectangles that can be easily identified. Mutual information can also be used for direct 2D-3D registration. 2D images can be constructed from a LIDAR scan that visualize various properties of the scan such as the reflectivity of the laser [40] or the relative height throughout of a point cloud [32]. The entropy between these types of images and regular photographs is minimized to uncover the relationship between the two. Several groups have explored using LIDAR scanners with built-in cameras that provide an initial set of registered photos to guide registration [16]. Keypoint features can be matched in 2D to obtain an initial camera pose estimate and then 3D information such as normals and edges in the point cloud can guide a refinement stage [35], [57]. Machine learning techniques can be incorporated to obtain helpful information to guide 2D-3D registration as done by authors in [54]. They use learning methods to determine what contours and shapes in the edge data constitute a building outline and match the contours of "regions of interest" across dimensions in aerial views of urban scenes.

D. Object Tracking

Object tracking in standard, full-motion video, and recently, in WAMI is of extreme interest to the computer vision community as a lot of tasks depend on reliable tracking. A host of higher-level tasks such as event analysis and 3D reconstruction depend on object or point feature tracking. Arguably, most of the recent advances have focused on innovative ways of modeling appearance in the case of single target trackers. Multi-target trackers that employ a motion-only approach have also progressed by considering efficient ways of information fusion which typically fall into the category of detect before track approach [44]. Single object trackers such as our LOFT [41], [42] (Likelihood of Features Tracker) focus on effective appearance modeling along with filtering and dynamics. The community has been focused on generalizing the performance of such trackers on a wide range of data sets instead of being restricted or biased to a particular set of scenarios depending on the scale of difficulty. The efforts of which can be seen in some key works that are very comprehensive in terms of performance and evaluation such as the Amsterdam Library of Ordinary Videos [47] (ALOV++) or the Video Object Tracking challenge [26] (VOT). The work in this paper leverages the 3C architecture and demonstrates fog-cloud resource placement algorithms utilizing SDN for LOFT-Lite, which belongs to the class of single object trackers that follows a track-before-detect paradigm and has shown to be robust for several classes of video data [41].

E. Fog Computing

Many distributed computing applications benefit by leveraging fog computing in terms of reduced service latency and operational efficiency. For instance, authors in [23] benefited from the paradigm of fog computing in their efforts to optimize web page performance by caching information at various fog nodes, versus using the traditional content-delivery network platforms. Fog resource management solutions are proposed in [34] to handle resource allocation and pricing based on user application profiles. Interestingly, SDN has been leveraged in context of fog computing recently by authors in [50], where they studied benefits of fog computing in application scenarios such as Smart Grid, and smart traffic lights in vehicular networks. Another notable recent work that leveraged SDN integrated with fog computing is [51], where benefits were shown in the context of vehicular adhoc network cases to enhance resources utilization and decrease service latency. Our work leverages fog computing paradigm in the context of mobile cloud configuration with SDN for disaster incident response scenarios, and shows benefits when handling media-rich and data-intensive visual computing applications for situational awareness of first responders.

F. SDN Management

Several studies have been done in prior works on SDN and cloud computing for overlay network provisioning. Authors in [9] propose a new method to manage Quality of Service (QoS) requirements of applications over SDN-enabled networks based on multi-path routing. Their multi-path routing assumes intermediate hosts to run agents that support their approach to allocate resources effectively by increasing the search space for the idle resources. In the context of multimedia delivery over large-scale SDN paths, the authors in [11] proposed a distributed OpenFlow-based QoS architecture involving co-ordination of multiple controllers. Another related work can be found in [52], where an adaptive routing approach is described to handle QoS requirements of video streaming utilizing SDN. They divide the QoS flows into two levels (base layer packets and enhancement layer packets), and provide highest priority to the base layer to reroute via feasible path in case of the congestion in the shortest path. Lastly, another exemplar related work on using SDN for video flow handling can be seen in [15], where a QoS Controller (Q-Ctrl) system is used to control and allocate bandwidth for the virtual machines supporting video streaming in a cloud infrastructure. This work builds on our earlier methodology [8] on wide-area experimental testbeds such as GENI [4] and extends it for the WAMI data visualization context with OpenFlow based SDN controller implementations for path computation and flow steering to improve user QoE.

III. Visual 3C Applications

We use our visual cloud computing approach to support two-different application types in order to determine the requirements and desired capabilities of a realistic system. Our first application registers ground-level videos with 3D LIDAR range scans obtained on a *theater-scale* so that sets of videos can be viewed in a single, intuitive, 3D virtual environment. Our second application focuses more on the *regional-scale* and identifies and tracks objects of interest in wide-area imagery, allowing officials to study the behaviors of particular vehicles. Table I lists examples of the size of data used by these applications and their runtimes. Data sizes may vary dependent on the resolutions of collected images and videos and the length of video streams. Computation times will vary as well, dependent on the hardware used. Ideally, we assume our environment has the resources to allow user to receive final processed data in real-time. More details on varying data sizes and available resources and how they effect computation and consumption rates are given in Section VI-A.

TABLE I
Application Details

App Scale	Expected Collection Data size	Expected Computation Time	Expected Consumption Rate
Theater	750 MB	4000 ms per frame	25 KB per second
Regional	70 GB	350 ms per frame	100 MB per second

A. Theater-Scale Application

In order to register a video with the LIDAR range scan, we must calculate the camera poses for video frames in relation to the 3D point cloud. This process which is outlined in Figure 2 entails matching a video frame to LIDAR photographs whose 3D correspondences are known, solving for the camera's projection matrix, and identifying and modeling moving objects in the 3D space.

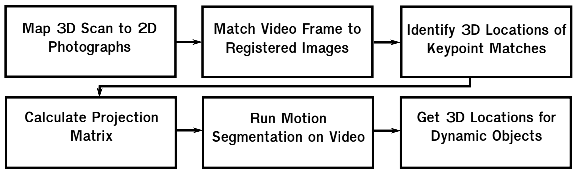

Map 3D Scan to 2D Photographs
Match Video Frame to Registered Images
Identify 3D Locations of Keypoint Matches
Calculate Projection Matrix
Run Motion Segmentation on Video
Get 3D Locations for Dynamic Objects

Fig. 2. Overview of method for registering videos with LIDAR scan and modeling moving objects in 3D space.

Based on the computational needs of the steps required for constructing a dynamic 3D virtualization, we can divide our image processing and computer vision stages into the following classes:

- **Small instance processing**: Camera metadata data processing, static background registration, 3D rendering
- **Large instance processing**: Video camera pose computation, motion segmentation, dynamic object positioning

Our small instance processing functions can optionally be handled in the fogs, where as the large instance functions can benefit from being processed in the cloud.

To initially estimate the 2D-3D relationship between the LIDAR photographs and the LIDAR scan, we have a pre-processing stage during which we map 2D pixels to 3D points. The mapping between the camera and the scan is known from a metadata file by the scanner giving the camera's focal length in pixels and extrinsic parameters (rotation and translation). Using this information during pre-processing, each 3D range scan point is projected onto each image plane to find its corresponding 2D point using standard computer vision techniques described in [20]. The entire point cloud is projected onto each image once and the 2D-3D correspondences are saved on a cloud-hosted server.

Once we have this information, we perform matching between video frames and LIDAR photographs using the technique outlined in [36] and obtain a set of 2D-3D keypoint matches between the video frame and the LIDAR scan. Our 2D-3D matching combines locally defined keypoint matching with contextual regional information to align images of the same scene with different visual properties. These visual discrepancies can be presented by registering images taken with different camera sensors and internal parameters (LIDAR camera vs. video camera) in addition to temporal changes between the sets of videos and LIDAR photographs as an incident unfolds and the scene content begins changing.

Our set of 2D-3D correspondences is finally used to calculate the projection matrix of the camera. We use the six-point algorithm with Direct Linear Transform [20] and Random Sample Consensus [13] to find the set of matches that calculate the most accurate projection matrix P and refine it using Levenberg-Marquardt refinement [31]. The projection matrix maps 3D points (X,Y,Z) to 2D image points (x,y) giving us the information needed to register the video frame with the scan. The color assigned to each 3D point is the color of the pixel it is closest to when projected onto the image plane.

The registration process described so far allows us to register the static background of a video with the 3D point cloud. However, we need one more stage to model the motion of moving people captured on video in 3D. When an object that was not scanned is present in a video, such as a person walking around, it will be projected onto an incorrect location in the 3D space because there is no structure that corresponds to it. Though the visual result of an image's registration may look fine when the scene is viewed from the camera location, these errors are very apparent when the user starts changing perspectives as is demonstrated in Figure 3, Bottom Left. To handle such situations, we segment out the motion in videos and add 3D planes to the virtual environment to "catch" the projection of these new entities.

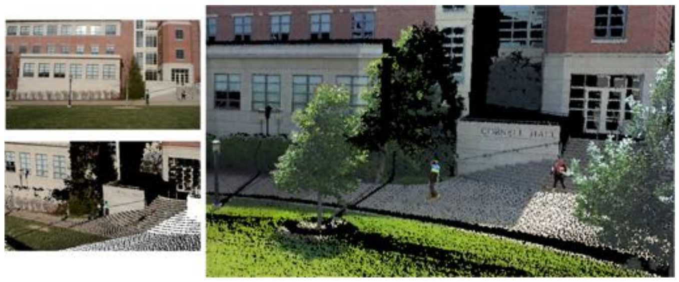


Fig. 3. Creating 3D planes for dynamic objects. *Top Left:* 2D video frame. *Bottom Left:* Video frame projected onto range scan without using our method for modeling moving objects. *Right:* 3D planes constructed for moving objects identified in video using our modeling method.

In order to identify moving objects in the video stream, we use the Mixture of Gaussians (MOG) algorithm [49]. This yields a binary image with the motion segmented from the background. The connected components algorithm is applied to the MOG image to create cohesive segments. We scan this image starting from the bottom row of pixels to find the lowest point in each moving segment. We then identify the 3D point in the range scan that matches this point when the video frame is registered with the range scan. Assuming that the moving object is touching the ground, this 3D point is the correct location for the bottom of the segmented object. New 3D points with the same depth as the bottom point and varying heights are created and projected onto the MOG image. If they fall within the segmented portion of the image, they correspond to a moving object that was not scanned and are added into the 3D space with the corresponding color information from the original video frame. The result of performing these steps is shown in Figure 3, Right.

B. Regional-Scale Application

LOFT (Likelihood of Features Tracking) [38], [41], [42] is an appearance based single object tracker that uses a set of image based features such as gradient orientation information using histogram of oriented gradients, gradient magnitude, intensity maps, median binary patterns [18] and shape indices based on eigenvalues of the Hessian matrix. LOFT robustly tracks vehicles in WAMI video which is airborne imagery characterized by large spatial coverage, high resolution of about 25 cm GSD (Ground Sampling Distance) and low frame rate. WAMI is also known by several other terms including wide-area aerial surveillance (WAAS), wide-area persistent surveillance (WAPS), Large Volume Streaming Data (LVSD) and wide-area large format (WAMI) [38], [39], [19], [5]. LOFT performs feature fusion by comparing a target appearance model within a search region using feature likelihood maps which estimate the likelihood of each pixel with the search window belonging to part of the target [42].

Tracking in WAMI involves several pre-processing steps that have been tested on large-scale aerial data [3], [17] as shown in Figure 4. These steps can be divided into two main classes according to functionality such as:

- **Small instance processing**: Compression, storage, metadata processing, geoprojection, stabilization and tiling
- **Large instance processing**: Initialize objects of interest, detection, tracking and event analysis

Small instance processing classes mainly focus on pure pixel level information. Large instance processing classes however, focus on pixel as well as object level information. Most of the large instance functions are dependent on the pre-processing stages in order to work effectively. As an example, most trackers need the imagery to be stabilized in order to produce the best results and hence registration becomes a key preprocessing step.

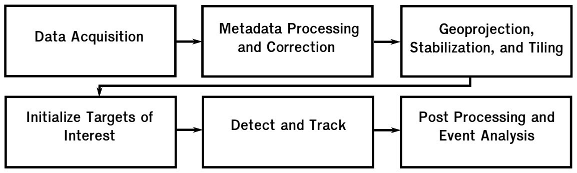

Data Acquisition
Metadata Processing and Correction
Geoprojection, Stabilization, and Tiling
Initialize Targets of Interest
Detect and Track
Post Processing and Event Analysis

Fig. 4. Functional block diagram showing pre-processing and post processing steps in a typical WAMI analysis pipeline.

LOFT-Lite is a version of LOFT [38], [41], which is a software framework for appearance based tracking. It includes all the components of LOFT-Appearance tracking with refactored functionality and an easy plugin based C++ interface that includes several optional modules such as motion dynamics and a tiled image reader. A track-before-detect approach is employed which greatly reduces the search space and is handy especially in large WAMI imagery where objects look similar and have a very small support map. Constraining the search region also results in faster image read throughputs as only multi-threaded partial tiles are read in memory. LOFT-Lite can achieve processing rates of 300 milliseconds per frame (wall-clock time) per target. Large single camera WAMI frames JPEG compressed occupy anywhere near 25-28 Megabits and at 5 frames a second, the minimum throughput required is high enough to consider a large bandwidth streaming pipeline.

LOFT-Lite accepts targets as input in the form of bounding boxes (see Figure 5). For each target, a search window is determined based on prediction and filtering dynamics as described in [41]. A set of features are computed both on the template and the search window, and the resulting likelihood maps are fused using a variance ratio based scheme. Several constraints such as the orientation of the vehicle in space and the prediction are taken into account before reporting a valid match. For details of these algorithms, readers can refer [42].

IV. Architecture for Visual Cloud Computing

In this section, we will first describe the fog-cloud system architecture we use for placing small instance and large

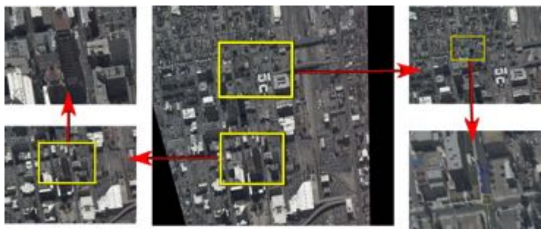


Fig. 5. Illustrative example of data ecosystem: Tiled TIFF aerial image with a resolution of 7800x10600 pixels and ≈80 MB size. The zoomed up insets show the location of the objects that were tracked (right inset) in relation to the Bank of Albuquerque towers (left inset) with zoomed up views.

instance image processing functions for the theater-scale and regional-scale applications. Following this, we will describe the various cloud and SDN technology components that integrate the different application modules.

A. Cloud/Fog System Architecture

Figure 6 shows our architecture, which consists of three layers: Mobile User Layer, Fog Computation Layer, and Cloud Management Layer. The Mobile User Layer is comprised of services that handle both the collection and consumption activities for our system. Incident scene images and video data is collected using security cameras, civilian smart phones, and aerial perspectives and imported into the system for transfer to the Fog Computation Layer. The processed visual information can be accessed at the consumption sites of users via thin-clients such as web browsers with interfaces to explore the outputs, or application client software that downloads the data for local exploration, or appliances that use protocols such as VNC, RDP or PCoIP to access virtual desktops with the exploration software. The consumption fogs could also host caching services to bring the processed data closer to the user thin-clients and reduce the need to have round-trip requests to the cloud. It is possible that the consumption phase involving an expert analyst may result in active use of the caching services that leads to repost of data to the Fog Computation Layer for further processing as part of deep exploration activities.

In the Fog Computation Layer, one service manages the small instance processing in conjunction with directives from the Unified Resource Broker (URB) in the Cloud Management Layer, and another service acts as the gateway to move data from the fog to the cloud via a high-performance network overlay setup with SDN. Thus, the Fog Computation Layer transforms the public cloud infrastructure into a ‘mobile cloud infrastructure’ and allows the management services in the public cloud to seamlessly operate close to the user collection/consumption sites for end-to-end orchestration and dynamic control of data processing locations. At the Cloud Management Layer, the scalable computing services as well as the URB orchestrate the computation placement either in the fog or in the cloud infrastructure. The URB serves as the “brain of the cloud”, and manages the dynamic distribution of the application processing workload to meet application QoS and user QoE requirements.

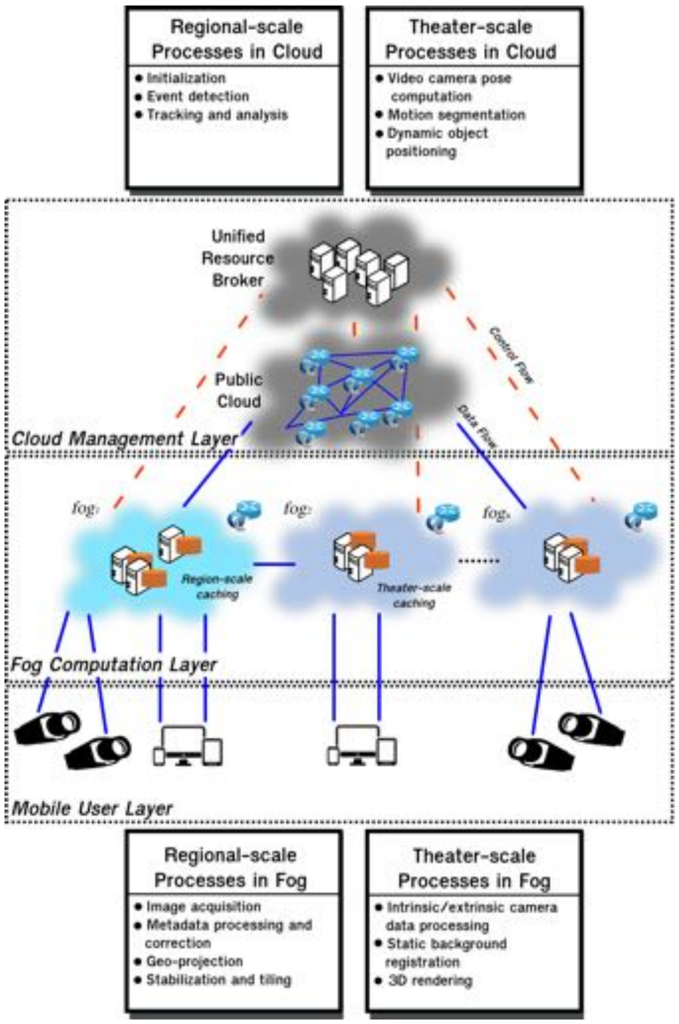


Fig. 6. Illustration of the 3C system showing the relationships between mobile user, fog computation, and cloud management layers. The URB (Unified Resources Broker) controls how resources are provisioned and how data flows are routed with SDN between fogs and the public cloud. The small and large instance processing in the fogs and cloud for theater-scale and regional-scale applications is also shown.

B. Relevance of Cloud and SDN Technologies

In this section, we explain the role of the URB and SDN controller components and how they interface with application modules in the 3C system architecture. Figure 7, which shows our logical architecture with system protocols, demonstrates how the integration of fog-cloud computing with SDN transforms the traditional theater-scale and regional-scale applications. The URB manages the cloud services by coupling the Mobile User, Fog Computation and Cloud Management layers with the application modules using REST (Representational State Transfer) web services [7] that allow services to communicate in a “stateless manner” (i.e., protocols are easily maintainable, lightweight and scalable). The SDN adds the programmable networking ability through use of the OpenFlow protocol by a controller in the URB to create overlay network paths among the collection, computation, and consumption sites. Consequently, SDN enhances the traditional systems that relied only on HTTP and TCP/IP between a remotely-hosted application and user sites. With the REST and

OpenFlow protocols integration, HTTP and TCP/IP are still used to transfer requests between the user interface (UI) and remotely-hosted application, but a control plane is introduced that is orchestrated by the URB for essential services for: data import, computation provisioning and placement, network path provisioning and cache management. Details of the URB's core algorithms for 3C compute provisioning and placement, as well as network path provisioning are presented in subsections V-A and V-B, respectively.

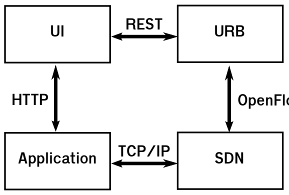


Fig. 7. The logical architecture showing system protocols which integrate the cloud and fog computation with SDN to transform the traditional theater-scale and regional-scale applications.

To understand how we have implemented the REST and OpenFlow protocols interfacing with the application modules and URB controller services, let us consider the illustration in Figure 8. We can see how a RESTful web services schema is used for a POST Request to allocate new links to allow access to the application, and to monitor status of the network path resource allocation. Our implementation of the various cloud services uses web technologies such as HTML, Bootstrap for CSS, JavaScript libraries jQuery and D3.js. We use PHP to handle AJAX requests from user side and mediate the cURL requests to the controller to request information and channel them to the user side.

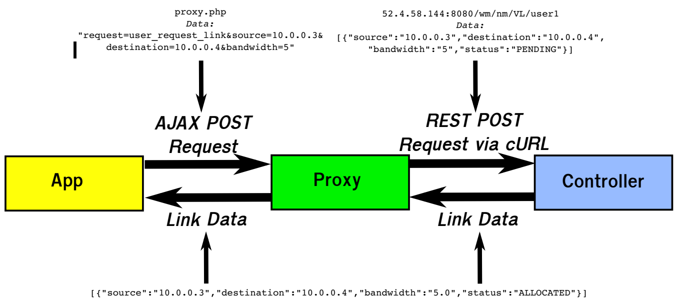

proxy.php
Data:
"request=user_request_link&source=10.0.0.3&
destination=10.0.0.4&bandwidth=5"
52.4.58.144:8080/wm/nm/VL/user1
Data:
[{"source":"10.0.0.3","destination":"10.0.0.4",
"bandwidth":"5","status":"PENDING"}]
AJAX POST
Request
REST POST
Request via cURL
App
Proxy
Controller
Link Data
Link Data
[{"source":"10.0.0.3","destination":"10.0.0.4","bandwidth":"5.0","status":"ALLOCATED"}]

Fig. 8. Web services schema for a POST Request to allocate new links and monitor status of the allocation.

V. 3C Control Algorithms

In this section, we present algorithms for the *optimal resource allocation* of the visual cloud computing infrastructure for the theater-scale and regional-scale applications. The algorithms handle the orthogonal requirements of applications demanding *minimum service time* (*i.e.*, the time taken to transfer and process application workload at a cloud/fog site, and receive visual results back at the consumption user side), and the need of the cloud service provider to minimize expensive resource over provisioning. We first describe the *3C Provisioning and Placement Algorithm* (CPP) that decides on a computation location to find *the minimum service time that is less than a specified constraint of the application process in order to deliver satisfactory user QoE*. Following this, we describe the *Network Path Provisioning Algorithm* (NPP) that finds *the optimal path* in the programmable network supporting SDN. We assume that *the optimal path is the shortest possible path that satisfies application QoS constraints*, since it has been proved that shorter routing paths result in higher overall throughput, and which in turn increases the overall network utilization [37]. The CPP runs in the Compute Provisioning Service, and the NPP runs in the Path Provisioning Service that are part of the URB in Figure 6.

A. 3C Provisioning and Placement (CPP) Algorithm

Algorithm 1 outlines the CPP workflow that assumes application context to be of two types based on the temporal sensitivity contexts of the application image processing functions: (i) real-time application RT, and (ii) non real-time application NRT. If the application processing falls under RT context, then besides computational resources requirements and demands in bandwidth, QoS requirements will be sensitive to the propagation delay. We consider only propagation delay because the end-to-end delay for service time is a function of both network bandwidth and propagation delay. The inputs for the CPP algorithm are: the set of available resources $\{R1, R2, R3, ..., Rn\}$ for each fog and the cloud; ps process of application App_i to be placed, process specific resources requirement R_{ps}, process service time (ST) upper bound ST_{ps}^{ub}, lower bound of bandwidth for process BW_{ps}, and process upper bound of propagation delay D_{ps} (∞ for NRT application type). The output of the CPP algorithm is the computation location Cl for specified process ps in the application App_i.

1) Search for computation location candidates: First, the algorithm finds the computation location candidates, *i.e.*, the fogs that have enough resources for ps (Algorithm 1, line 5), as well as both fogs and cloud that have *the optimal paths* between them and ps (Algorithm 1, lines 5 and 10). We find *the optimal path* by using Algorithm 2 (Algorithm 1, lines 4 and 9).

2) Estimation of Service Time (ST) for all candidates: In the second step, we estimate for each candidate (fog or cloud) its service time (ST) based on Equation 1 (Algorithm 1, line 14):

$$ST = TR(NPD) + TR(PD) + CT, \qquad (1)$$

where TR is the transfer time for NPD (non-processed or raw data) and PD (processed data), and CT is the computation time. Note how the time for transfer of NPD from the application to the visual computing cloud differs from the time for transfer of the PD from the visual computing cloud back to the application. The computation time can vary for the same process in the different fogs and in the cloud depending on the resource availability (*e.g.*, number of CPUs), as well as transfer time varies due to a different size of data and different propagation delays.

3) Finding the best candidate to allocate resources: Finally, we find the best candidate with the minimum ST (Algorithm 1, line 16). Then we check if the candidate ST is less than the specified ST upper bound for the current

Algorithm 1: 3C Provisioning and Placement (CPP)

```
   Input: $\{R_1, R_2, R_3, ..., R_n\}$:= set of fogs and cloud resources, $ps$:=
          process of $App_i$, $R_{ps}$:= resources constraint, $ST^{ub}_{ps}$:= $ST$
          constraint, $BW_{ps}$:= bandwidth constraint, $D_{ps}$:= propagation
          delay constraint ($\infty$ for $NRT$)
   Output: $Cl$:= computation location, $R^{new}$:= new resources of $Cl$
1  begin
       /* Computation location candidates search    */
2      Initialize a list of the fogs
3      foreach $fog_i \in fogs$ do
           /* Find the optimal path to the $fog_i$     */
4          $Path_{fog_i} \leftarrow NPP(location(ps), fog_i, BW_{ps}, D_{ps})$
5          if $R_{ps} \leq R_{fog_i}$ and $Path_{fog_i}$ exist then
6              Add $fog_i$ to the candidate set
7          end
8      end
       /* Find the optimal path to the cloud          */
9      $Path_{cloud} \leftarrow NPP(location(ps), cloud, BW_{ps}, D_{ps})$
10     if $Path_{cloud}$ exist then
11         Add cloud to the candidate set
12     end
       /* Estimate ST for all candidates              */
13     foreach candidate do
14         $ST_{candidate_i} \leftarrow$ estimate $ST(ps)$
15     end
       /* Find the best candidate to allocate
          resources                                   */
16     $ST^{best} = \min_{i=1,N} (ST_{candidate_1}, \dots, ST_{candidate_N})$
17     if $ST^{best} \leq ST^{ub}_{ps}$ then
18         Allocate the resources at $Cl$
19         Allocate $Path^{best}$ to selected candidate with $ST^{best}$
20         Push $ps$ to the resources scheduler
           /* Update the resources at the selected
              location                                */
21         $R^{new}_{Cl} = R^{old}_{Cl} - R_{ps}$
22     else
23         $ST^{ub}_{ps}$ cannot be satisfied.
24     end
25 end
```

process ps (Algorithm 1, line 17). If so, we allocate resources (Algorithm 1, lines 18-20), update the remaining available resources R_{new} (Algorithm 1, lines 21) and finally terminate the algorithm. Otherwise, we have to terminate algorithm without ps placement as there is no candidate fog or cloud which can satisfy its ST upper bound (Algorithm 1, line 23).

4) Simple example of a CPP run: For the sake of illustration of the CPP algorithm, we assume that there are four fogs, one cloud, and four processes from different applications (either RT or NRT nature) as shown in Figure 9. Table II shows the flow of our example. The algorithm decides the location of each of those processes according to the ST for this process in that location, where each process has different ST in each *fog* and *cloud*. For example, the ST for process 1 (ps_1) is: 10ms at fog_1, 20ms at fog_2, 6ms at fog_3, 9ms at fog_4, and 17ms at *cloud*. According to that, the best location to execute the ps_1 is at fog_3, which has the minimum ST. Similarly, the best computation location for the process (ps_2) is the fog_3, for the process (ps_3) is the *cloud*, and for the process (ps_4) is the fog_4. On the other hand, the fog_1 and fog_2 do not have enough resources for the ps_2 and ps_3, respectively. This dynamic allocation of the resources in various locations ensures suitable distribution of the workload between number of fogs and the cloud to deliver the best user QoE, and ensures application usage of the network resources with necessary QoS.

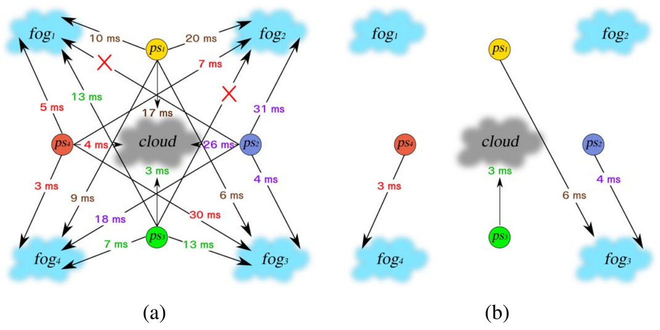


Fig. 9. Illustrative example of a CPP run: (a) CPP calculates Service Time (ST) for the each process based on estimated time for process to be serviced in the each *fog* or in the *cloud*; (b) based on estimated ST, the CPP places ps_1 and ps_2 process to be serviced in the fog_3, ps_3 to be serviced in the *cloud* and ps_4 to be serviced in the fog_4.

B. Network Path Provisioning (NPP) Algorithm

The *Network Path Provisioning Algorithm* (NPP) finds *the optimal path*, which we assume is *the Restricted Shortest Path* (RSP), *i.e.*, the shortest possible path which satisfies QoS constraints (bandwidth BW and delay D). The RSP problem is known to be NP-complete [25]. To make its solving feasible (in polynomial time) for the 3C visual computing cloud, we use notion of “neighborhoods” NHs in NPP, *i.e*, a set of nodes that can be reached from the source node with the same number of hops, with each “neighborhood” NH set containing only unique elements. That allows us to estimate the length of *the optimal path* before its finding, and hence we are able to provide a path solution in polynomial time.

Algorithm 2 shows the main workflow of NPP, whereas Algorithms 3 and 4 detail intermediate steps. Algorithm 2 accepts source node X, destination node Y, bandwidth BW and delay D constraints as an input and outputs *the optimal path*. We perform two main steps: build neighborhoods NHs by using Algorithm 3 to estimate the length of *the optimal path* (Algorithm 2, line 2) and do backward pass by using Algorithm 4 to find this *optimal path* solution (Algorithm 2, line 4).

Algorithm 2: Network Path Provisioning (NPP)

```
   Input: $X$:= src $Y$:= dest, $BW$:= bandwidth constraint, $D$:= delay
          constraint
   Output: Shortest path between $X$ and $Y$ satisfying $BW$ and $D$
1  begin
       /* Build $NHs$ with $BW$ and $D$ from $X$ to $Y$   */
2      $NHs \leftarrow Build\ Neighborhoods(X, Y, BW, D)$
       /* Remove last $NH$                              */
3      $NHs.remove(NHs.size)$
       /* Backward Pass to find path between $X$ and $Y$
          satisfying $BW$ and $D$                       */
4      $path \leftarrow Backward\ Pass(Y, NHs, BW)$
5  end
```

1) Neighborhoods Building: Algorithm 3 describes the neighborhood building step. The neighborhoods data structure NHs contains not only information about nodes but also the minimum path metric m_p for each node, *e.g.*, (Algorithm 3, line 2). Note how we save information about delay to estimate D constraint in line with the length of the solution. Further, we exclude all neighbors that do not satisfy the constraints, or the neighborhood NH already contains the same node with a

TABLE II
CPP ILLUSTRATIVE EXAMPLE FLOW.

Process	Type	fog_1 ST	fog_2 ST	fog_3 ST	fog_4 ST	$cloud$ ST	Selected Computation Location
ps_1	RT	10	20	6	9	17	fog_3
ps_2	NRT	**X**	31	4	18	26	fog_3
ps_3	RT	13	**X**	13	7	3	$cloud$
ps_4	RT	5	7	30	3	4	fog_4

D metric m_D less than a new one m_D^{new} (Algorithm 3, line 9). A new minimum D metric m_D^{new} for neighbor nh can be calculated as sum of the D metric m_D for predecessor node n and a weight w_D of a link between node n and nh (Algorithm 3, line 8), *i.e.*, w_D is a delay of this link. Note how some of the nodes may now appear in several neighborhoods. In this case, the first step ends as soon as the destination node Y appears in the current neighborhood cNH (Algorithm 3, line 4). If the new neighborhood NH is empty or a number of neighborhoods NHs is higher or equal than to the nodes number (Algorithm 3, line 14), we terminate the algorithm concluding that the node Y is unreachable.

Algorithm 3: Build Neighborhoods

```
/* Returns neighborhoods list $NHs$ from $X$ to $Y$ if reachable */
Input: $X$:= src, $Y$:= dest, $BW$:= bandwidth constraint, $D$:= delay constraint
Output: $NHs$ from $X$ to $Y$
1  begin
       /* Initialize $NHs$ and put therein the current neighborhood $cNH$ with $X$ and 0 as $D$ metric */
2      $cNH \leftarrow (X, 0)$
3      $NHs \leftarrow NHs \cup cNH$
4      while $Y \notin cNH$ do
5          $NH \leftarrow \emptyset$
6          foreach Node $n \in cNH$ do
7              foreach Neighbor $nh \in n$ do
8                  $m_D^{new}(nh) = m_D(n) + w_D(n, nh)$
9                  if link between $n$ and $nh$ satisfies $BW$ and $m_D^{new}(nh) \leq D$ and $m_D^{new}(nh) < m_D(nh)$ then
10                     $NH \longleftarrow NH \cup (nh, m_D^{new}(nh))$
11                 end
12             end
13         end
14         if $NH \notin \emptyset$ and $NHs.size \leq$ number of nodes then
15             $NHs \leftarrow NHs \cup NH$
16             $cNH \leftarrow NH$
17         else
18             $Y$ is unreachable
19         end
20     end
21 end
```

2) Backward Pass: Algorithm 4 describes the second step of NPP. In this case, we need to ensure the satisfaction of the BW constraint on the backward pass, and find a path with the minimum D metric m_D. To do so, we subtract a weight w_D of a link between node n and its neighbor nh from the path D metric m_D for n and select a node in the previous neighborhood NH whose path D metric m_D^{prev} matches this difference (Algorithm 4, line 9). Because at least one solution will be found, we do not need to build all the possible paths (Algorithm 4, line 13). The second step ends as soon as we hit the zero neighborhood $NHs[0]$ (Algorithm 4, line 5).

3) Simple example of a NPP run: To illustrate how our NPP algorithm works, consider an example network consisting of 4 nodes X, Y, A and B (Figure $10(a)$). Each link has

Algorithm 4: Perform Backward Pass

```
/* Input list of neighborhoods $NHs$ does not contain the last $NH$ */
Input: $Y$:= dest, $NHs$ - list of the sets of nodes, $BW$:= bandwidth constraint
Output: The shortest $path$ between $X$ and $Y$ which satisfies $BW$ and $D$
1  begin
       /* Place $Y$ in the $path$ */
2      $path \longleftarrow Y$
3      $k \longleftarrow 1$
       /* take previous neighborhood $NH$ for $Y$ */
4      $NH \longleftarrow NHs[size - k]$
5      while $NH \neq NHs[0]$ do
6          Node $n \longleftarrow path[1]$
7          foreach Node $nh \in$ neighbors of $n \cap NH$ do
8              $m_D^{prev}(nh) = m_D(n) - w_D(nh, n)$
9              if link between $n$ and $nh$ satisfies $BW$ and $m_D^{prev}(nh) = m_D(nh)$ then
10                 $path \longleftarrow neighbor \cup path$
11                 $k \longleftarrow k + 1$
12                 $NH \longleftarrow NHs[size - k]$
13                 break
14             end
15         end
16     end
17 end
```

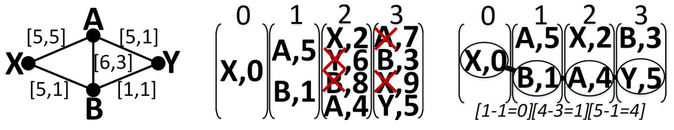


Fig. 10. Illustrative example of a NPP run: (a) simple network configuration with [bandwidth, delay] constraints for each link; (b) forward pass finds the best length and the minimum delay to Y; (c) backward pass identifies the valid path X, B, A, Y (shown with the ellipses and line annotation) by matching the subtraction result (shown below neighborhoods) for each previous node among the path.

two metrics: bandwidth BW and propagation delay D. Our aim is to find a path from X to Y which satisfies two constraints: $BW \geq 5$ and $D \leq 5$. Figure $10(b)$ shows the forward pass, where each neighborhood now contains pairs of nodes and their minimum D metrics. During this step, we exclude all invalid neighbors, *i.e.*, those whose links do not satisfy the BW constraint, as well as those violating the D constraint. Note that during this step the same nodes may appear in several neighborhoods and could be part of multiple valid paths. Figure $10(c)$ illustrates the backward pass: in this case, Y cannot appear in the second neighborhood due to the violation of either the D constraint (through node A) or the BW constraint (through node B). However, Y now appears in the third neighborhood. Owing to the successful backward pass subtraction, we find a solution with the minimum D metric for this length. Finally, NPP returns $X \rightarrow B \rightarrow A \rightarrow Y$ solution which satisfies BW and D constraints.

VI. Experimental Methodology and Results

A. Theater-Scale Evaluation

1) Experimental Setup: Our testbed setup consists of clients connected to a wireless overlay network that represents a standardly available campus enterprise network and a compute manager VMware Horizon View© connected over a higher-speed campus research network managed with SDN. We emulate a network made available in a disaster scenario in which, these two networks can be used in parallel for public safety purposes during disaster incident response. The campus research network (with fiber connections between buildings to support data-intensive science application traffic) becomes the disaster overlay network, and the wireless overlay network (typically used for enterprise traffic such as e-mail and web browsing) can be used for QoS priorities to support disaster-supporting cloud services. By pooling resources in disaster-supporting cloud services, the 3C (collection, computation, and consumption) steps can be employed more effectively with our algorithms for first responders to gain visual situational awareness to potentially save lives. Our wireless overlay network has a bandwidth of $\approx$10 Mbps upload/download and our disaster overlay network has a bandwidth of $\approx$600 Mbps upload/download. We have a virtual server setup with 6 virtual CPUs (12GHz) and 16 GB of memory. Our physical server has 2 processors Intel Xeon Processor E5-2640 v2 with 8 cores each for a total of 16 cores. The clients have a Windows 7 Enterprise 64 bits operating system installed and the server uses Windows 2008R2 64 bits. Our clients are able to stream data to the server by using cURL functionality that is authenticated by the FTP server in the virtual server.

To obtain a 3D model for our location of interest, we use a Leica C10 HDS LIDAR scanner. This scanner provides a high-resolution point cloud of a scene and 2D images of the scanned subject using a built-in camera. The output data from the scanner also consists of files containing the internal and external camera parameters for each image. We also capture multiple video streams of people walking around the university campus with HD video cameras. This video data needs to be transferred in real-time to the server over the high-speed campus research network to conduct visual data processing.

2) Design of Experiments: We perform tests on our university campus after obtaining the LIDAR scan and several HD videos. We separately evaluate the performance for the three stages of our system shown in Figure 1, i.e. collection and transferring the 3D scan and video files over the network, computing the 2D-3D data fusion, and consumption by the user to receive 3D scenes and multiple videos for virtual navigation and video analysis. The goal is to obtain real-time (or near real-time) responses for all of these tasks. We also experiment with scaling up the amount of data transferred to see how many videos we can handle and how large the 3D model data can be, depending on the hardware used.

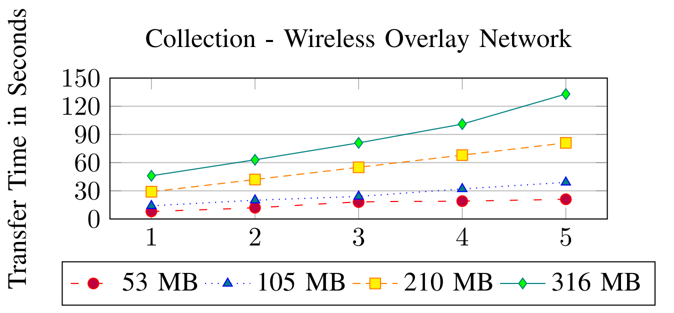
To simulate the collecting and transferring of any number of real-time video streams, we first obtain several HD videos on campus. These videos are stored on a laptop and a varying number of duplicates are sent over the network simultaneously to tax the system. Our goal is to observe what happens to the system when we have either one or many videos available that need to be viewed. Individual video frames are transferred sequentially to mimic real-time video capture. The 2D-3D registration and video motion analysis stages are performed on the server. The final rendering of the 3D environment with dynamic objects is transferred to a mobile device (e.g., laptop, tablet, smart phone) in a remote location where it is viewed and manipulated with a thin-client protocol.

For the final consumption stage, the large 3D model only needs to be transferred over the network to the remote device one time when it is first requested. If the user wishes to view a different location, a new model will need to be sent to the mobile device. The 3D data corresponding to dynamic objects will need to be continuously computed on the server as new video frames arrive, and also updated by sending to the user's device, therefore we test how long it takes to transfer data for varying numbers of dynamic objects to the remote location. Depending on how many people and vehicles are present in the scene and captured on video, this number can change drastically. Through a series of experiments, we find out how many dynamic objects our system and networks can handle, and thus informs a cloud/fog infrastructure design.

3) Study Results: We studied the collection, computation, and consumption sections of our pipeline individually. All of our tests were performed three times, and in this section we report the averages of these tests as our final results.

Our collection transfer test consisted of streaming files to the server from a client connected to the wireless overlay network and from a client connected to the disaster overlay network. We tested sending varying file sizes over the server (52, 105, 210, and 316 MB) to account for situations where low-resolution, black and white security camera footage may be utilized compared to high definition video captured by a smartphone or hand held camera. We experimented with transferring between one and five videos simultaneously over both the wireless overlay and disaster overlay networks. The transfer times in seconds for these tests are shown in Figure 11. The maximum number of videos we tested sending at once is five because our server has six cores and cannot process more than that number of videos at once. Sending more videos to the server together will not improve our overall computation time. Despite the fact that these are relatively small-scale tests, we still get a good sense from the charts how communication time will increase as the number of videos rises. These tests also show that the disaster overlay network is able to transfer data about 10 times faster than our wireless overlay network and would be extremely beneficial in a disaster scenario where timely information sharing is key.

For the computation stage of our system, we performed tests on a virtual server. We modified the virtual CPU capacity with 2, 4, 8 and 12 GHz, testing the processing times in seconds for videos containing 1-28 moving objects. Each dynamic object in every video needs to be identified, segmented from the static background, and modeled in 3D so we are interested in what happens to our overall performance as more and more objects are recorded. We stopped at 28 objects because this seems to be a reasonable limit on the maximum number of people that will be captured in a typical video camera's field of view and be able to be separately identified (without people overlapping in the video) and modeled as individual objects in 3D. We also tested the system's performance when processing 1 to 4 videos simultaneously and looked at the CPU percentage utilization. The video files used in this test are all 185 MB



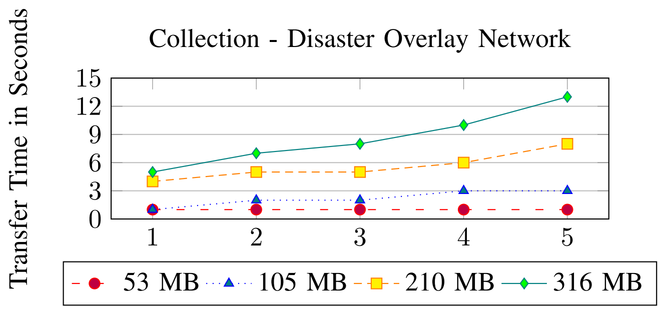
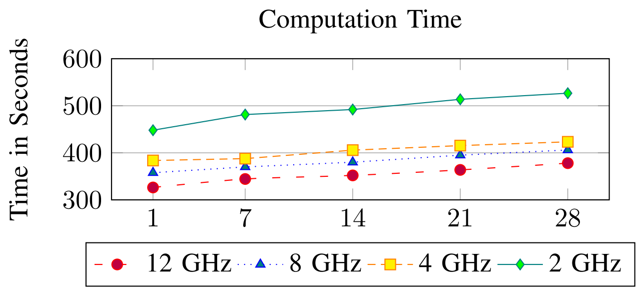


Fig. 11. Collection stage transfer times for varying video sizes. *Left:* Transfer times for the wireless overlay network. *Right:* Transfer times for the disaster overlay network.



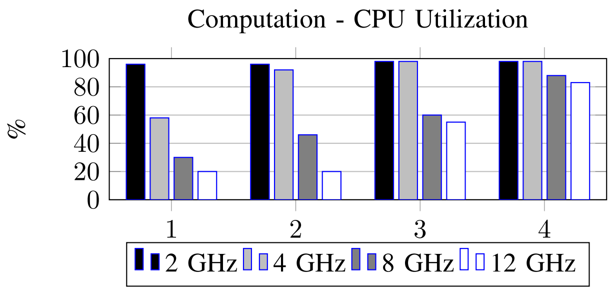


Fig. 12. Performance during the computation stage. *Left:* Time required to compute 3D pose for increasing number of dynamic objects in videos on different server configurations. *Right:* CPU utilization on server during computation stage for different configurations.

and have the same content. These results are all shown in Figure 12. We observe here that the system becomes saturated when processing four videos and can see what will happen as more and more videos are added to the system. We gain the greatest boost in performance when increasing from two to four videos.

During the consumption stage, the clients need to download the files containing 3D information for dynamic virtual objects from the server. In the case that a client is connected to the server via the wireless overlay network, this process is time consuming compared with a virtual desktop accessed from a thin client (hardware) or from Horizon View Client (software). For both cases, Teradici PCoIP protocol© was used. A comparison of file transfer times in seconds between a physical client connected to the wireless overlay network and using a virtual deskop setup on a server connected to the disaster overlay network is shown in Figure 13. We tested transferring 3D data files for 377-3,496 individual moving objects simultaneously to really tax the system and to find out how much information can be processed in a timely manner if the disaster site is very congested with people and cameras. We can see that using the disaster overlay network, thousands of moving objects can be transferred and displayed in a matter of seconds, making this setup great for first responders needing to sift through lots of information quickly.

B. Regional-Scale Evaluation

Herein, we first consider characterize the resultant impact on the LOFT-Lite application compute offloading to the cloud when using multiple video resolutions corresponding to different mobile devices and under disaster network degradation conditions. Next, we show user QoE improvements in data throughput and tracking time when using our URB implementation that utilizes SDN and divides the LOFT-Lite application into small and large instance processing for cloud/fog computation, versus complete compute offloading to a core cloud over best-effort IP networks.

1) Disaster Network Experiments Setup and Results: Multiple video resolutions in practice need to be processed because the input source imagery in surveillance typically spans a wide variety sensor technologies found in mobile devices. In our experiments, we consider common resolutions in surveillance video belonging to the broad categories of: (a) Full-resolution WAMI (7800 x 10600) (see Figure 5), (b) Large-scale aerial video (2560 x 1900), and (c) Ground surveillance video (640 x 480) (see Figure 14). To consider disaster network scenarios systematically that impact data transfer, we assume a 4G-LTE network configuration with an initial bandwidth of 100 Mbps (best case) and apply a bandwidth degradation profile during compute offloading test cases with different resolutions. For experimental purposes, the profile degrades the bandwidth at a rate of 20 Mbps per minute due to heavy cross-traffic load or candidate network path failures till it falls to zero (i.e., worst case disconnection scenario).

Our visual cloud computing setup for the disaster network experiments includes two virtual machines (VMs) for the data collection and computation sites, respectively each with a single core CPU and 1GB of main memory in a GENI platform testbed connected through an OpenFlow switch. Several performance metrics such as estimated throughout, tracking time, waiting time and total time are measured to characterize Quality of Application (QoA) of LOFT-Lite application computation as well as SCP (standard secure copy utility) data movement under the bandwidth degradation profile.

Table III shows measurement results averaged over ten trials

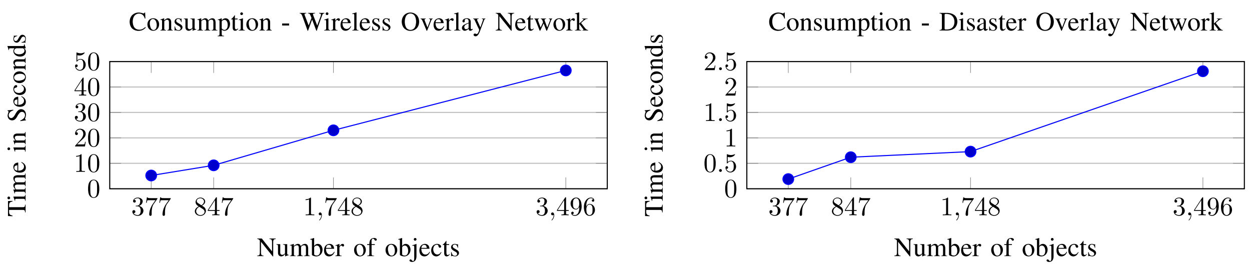


Fig. 13. Transfer times for dynamic 3D objects captured in videos to be sent to remote device for viewing. *Left:* Transfer times for our wireless overlay network. *Right:* Transfer times for our disaster overlay network.

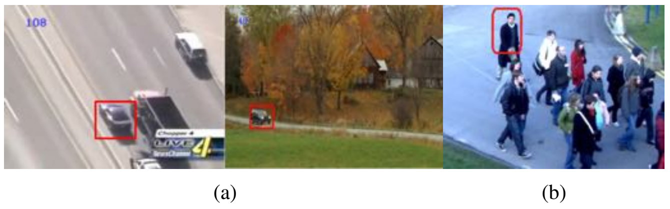


Fig. 14. LOFT-Lite results on (a) standard and (b) Full-Motion surveillance video. Each frame in these video datasets is about 2MB compressed.

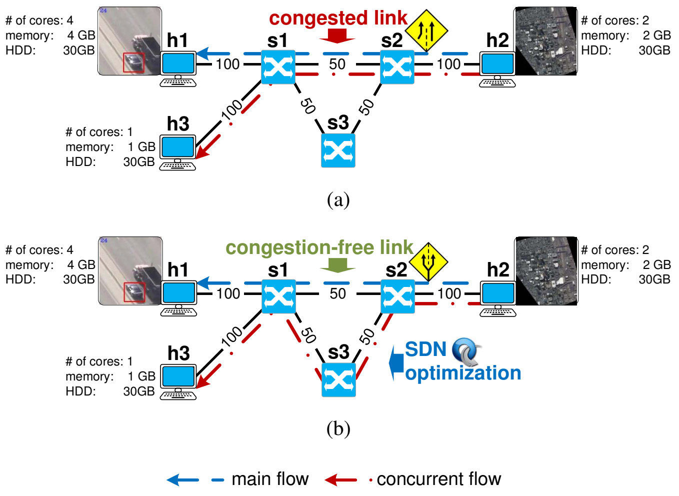


Fig. 15. Data flows in the allocated GENI topology: (a) Standard video data flow interferes with concurrent flow on the $s2 \rightarrow s1$ link as regular network sends data through the best (the shortest) path; (b) Using SDN and the NPP algorithm, we optimize network resources usage and redirect concurrent flow through the longer path $s2 \rightarrow s3 \rightarrow s1$ which avoids congestion. Further, moving image pre-processing to the fog ($h2$ instead of $h1$) enables real-time tracking using LOFT-Lite.

with 95% confidence intervals. Our full-resolution WAMI and large-scale aerial video processing pipelines are non real-time and suffer relatively long wait times in comparison with the lower resolution ground-based FMV pipeline that runs in real-time. These results quantify system scalability and the benefits of reducing video resolution under disaster network conditions to support single target real-time tracking for multiple instances of LOFT-Lite. Standard video resolution results in the highest throughput over 3G/4G networks.

2) Cloud/Fog Computation Experiments Setup and Results: Standard (VGA) video resolution was used for the cloud/fog experiments to track pedestrians [12] in a crowd (see Figure 14(b)). An adaptive contrast enhancement global image pre-processing operation is applied as needed in the cloud/fog (using Imagemagick) before images are sent to the core cloud for object tracking. All images are pyramidal tiled TIFF (Tagged Image File Format) and the pre-processing retains the tile geometry.

Our setup for the cloud/fog computation experiments includes six virtual machines (VMs) in the GENI platform testbed as shown in Figure 15, where three of these VMs emulate OpenFlow switches ($s1$, $s2$ and $s3$) and others are regular hosts ($h1$, $h2$ and $h3$). Each *host-to-switch* link has 100 Mbps bandwidth, and each *switch-to-switch* link has only 50 Mbps bandwidth to emulate congested and damaged network infrastructure in a disaster scenario. Our LOFT-Lite application runs on $h1$ (quad-core CPU, 4GB of RAM and 30GB of HDD) which acts as a *computation cloud* site, whereas $h2$ (double-core CPU, 2GB of RAM and 30GB of HDD) acts as a *collection fog* site, and $h3$ (single-core CPU, 1GB of RAM and 30GB of HDD) consumes raw data from $h2$ by acting as a storage *consumption fog* site. Node $h3$ is configured with cross-traffic flow consumption such that it interferes with the main data traffic for the LOFT-Lite application. We call this cross-traffic as the 'concurrent flow', and the application traffic for LOFT-Lite as the 'main flow'. Finally, the thin-client (local PC) acts as a data consumer at the user end. LOFT-Lite runs on a thread with a backoff timer which sleeps for a specified delay while querying the local folder for the image stream. To transfer data between hosts, we use the SCP utility.

To differentiate between the cloud/fog and the core cloud computation, our experiment workflow is as follows: (i) start sending concurrent traffic from $h2$ to $h3$; (ii) start sending main traffic (video) from $h2$ to $h1$; (ii.a) while performing cloud/fog computing, start pre-processing concurrently with step (ii) (we assume here that pre-processing is faster than data transfer); (iii) wait till at least the first frame has been transferred; (iii.b) in case of core cloud computing, start pre-processing before step (iv) (in this case LOFT-Lite has to wait for each frame when its pre-processing ends); (iv) start LOFT-Lite; (v) wait until all main traffic has been transferred; and (vi) terminate both the applications and data transfers.

Table IV shows the final timing results averaged over ten trials to estimate 95% confidence intervals for the cloud/fog and core cloud computation cases. For each trial, we used a 500 frame video sequence and measured several QoA performance metrics such as estimated throughput, tracking time, waiting time and total time. We can pre-process frames faster in the core cloud computation case in comparison to cloud/fog computation. Due to congestion in best-effort IP

TABLE III
QoA impact for compute offloading of multiple video resolutions for a systematic network degradation profile.

Performance Metrics	Full-resolution WAMI (7800 x 10600)	Large-scale aerial video (2560 x 1900)	Ground-based standard video (640 x 480)
(SCP QoA) Number of transferred frames	25.80 ± 0.26	180.9 ± 0.9	892 ± 9
(LOFT-Lite QoA) Estimated throughput (Mbps)	66.5 ± 0.9	76 ± 0.9	43.9 ± 0.4
(LOFT-Lite QoA) Tracking time (sec/fr)	0.4035 ± 0.005	0.368 ± 0.002	0.403 ± 0.004
(LOFT-Lite QoA) Waiting time (sec/fr)	9.03 ± 0.13	0.845 ± 0.014	0 ± 0
(LOFT-Lite QoA) Total time (sec/fr)	9.46 ± 0.13	1.214 ± 0.014	0.403 ± 0.004

TABLE IV
QoA impact results comparison for core cloud computing over IP network versus utilizing SDN and cloud/fog computing by dividing the application into small and large instance processing.

Performance Metrics	Core Cloud Computing over IP network	Cloud/Fog Computing utilizing SDN	Perceived Benefits
(SCP QoA) Storage transfer time (sec/fr)	0.564 ± 0.007	0.402 ± 0.006	Avoiding congestion with SDN traffic steering results in lower transfer time
(Imagemagick QoA) Pre-processing time (sec/fr)	0.1955 ± 0.0011	0.292 ± 0.023	No significant difference
(LOFT-Lite QoA) Estimated throughput (Mbps)	13.50 ± 0.34	41.85 ± 0.24	Lower transfer time and fog computation maximizes application throughput
(LOFT-Lite QoA) Tracking time (sec/fr)	0.4097 ± 0.0022	0.4229 ± 0.0024	No significant difference
(LOFT-Lite QoA) Waiting time (sec/fr)	0.902 ± 0.032	0 ± 0	Achieving maximum application throughput avoids waiting time and supports real-time computation
(LOFT-Lite QoA) Total time (sec/fr)	1.312 ± 0.034	0.4229 ± 0.0024	Cloud/fog computation of small and large instances can produce 3X speedup over core cloud computation

network and the unavailability of video at the *computation cloud* site, we cannot track with LOFT-Lite application in real-time (with 0 waiting time) in the core cloud computation case. Whereas in the cloud/fog computation utilizing SDN, LOFT-Lite can be run in real time at $3-4$ Hz.

VII. Conclusions

In this paper, we have proposed a novel visual cloud computing architecture for media-rich scalable data movement and computation utilizing the benefits of SDN for collection, computation and consumption (3C) in handling incidents due to natural or man-made disasters. We have shown how our 3C architecture and cloud resource provisioning and placement algorithms (CPP and NPP) can be used in combination with computer vision based applications to: (a) create 3D visualizations of disaster scenarios, and (b) track objects of interest for automated scene understanding. Thus, our approach enables situational awareness for emergency management and law enforcement officials during disaster incidents. Our algorithm novelty was to propose a parameterization of application resource requirements in the form of small and large instance visual processing, which enables optimization of fog and cloud computation location selection utilizing SDN to connect the network-edges in the fog with the cloud core. We developed a realistic virtual testbed environment for experimentally validating that the proposed optimization tradeoffs between fog and core cloud computing reduces latency and congestion while increasing application responsiveness.

Our computer vision algorithms for disaster incident response are not fundamentally different from visual analytics algorithms for other applications such as autonomous systems. Our work can be extended to use special variants of occlusion-aware algorithms in disaster scene imagery with limited visibility due to smoke, fire, etc. obtained from smart devices e.g., thermal cameras. Thus, our work lays the foundation for adaptive resource management to handle incident-supporting visual cloud computing that can foster effective disaster relief co-ordination to save lives.

References

[1] Openflow switch specification. https://www.opennetworking.org/sdn-resources/openflow/57-sdn-resources/onf-specifications/openflow.

[2] H. Agrawal, C. Mathialagan, Y. Goyal, N. Chavali, P. Banik, A. Mohapatra, A. Osman, and D. Batra. CloudCV: Large scale distributed computer vision as a cloud service. *arXiv:1506.04130*, 2015.

[3] H. Aliakbarpour, K. Palaniappan, and G. Seetharaman. Robust camera pose refinement and rapid SfM for multiview aerial imagery Without RANSAC. *IEEE Geoscience and Remote Sensing Letters (GRSL)*, pages 2203–2207, 2015.

[4] M. Berman, J. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, pages 5–23, 2014.

[5] E. Blasch, P. Deignan, S. Dockstader, M. Pellechia, K. Palaniappan, and G. Seetharaman. Contemporary concerns in geographical/geospatial information systems (GIS) processing. *IEEE National Aerospace and Electronics Conference (NAECON)*, pages 183–190, 2011.

[6] P. Calyam, A. Berryman, A Lai, and M. Honigford. VMLab: Infrastructure to Support Desktop Virtualization Experiments for Research and Education. *VMware Technical Journal*, pages 2–8, 2012.

[7] P. Calyam, A. Berryman, E. Saule, H. Subramoni, P. Schopis, G. Springer, U. Catalyurek, and D. Panda. Wide-area overlay networking to manage science dmz accelerated flows. *International Conference on Computing, Networking and Communications (ICNC)*, 2014.

[8] P. Calyam, S. Rajagopalan, A. Selvadhurai, S. Mohan, A. Venkataraman, A. Berryman, and R. Ramnath. Leveraging OpenFlow for resource placement of virtual desktop cloud applications. *IEEE Symposium on Integrated Network Management (IM)*, pages 311–319, 2013.

[9] E. Chemeritskiy and R. Smeliansky. On QoS management in SDN by multipath routing. *Conference on Science and Technology (MoNeTeC)*, pages 1–6, 2014.

[10] E. Chissungo, H. Le, and E. Blake. An electronic health application for disaster recovery. *Symposium on Information and Communication Technology (SoICT)*, pages 134–138, 2010.

[11] H. Egilmez and A. Tekalp. Distributed QoS architectures for multimedia streaming over software defined networks. *IEEE Transactions on Multimedia (MM)*, pages 1597–1609, 2014.

[12] A. Ellis, A. Shahrokni, and J. Ferryman. Pets2009 and winter-pets 2009 results: A combined evaluation. In *IEEE Workshop on Performance Evaluation of Tracking and Surveillance (PETS-Winter)*, pages 1–8, 2009.

[13] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM (CACM)*, pages 381–395, 1981.

[14] B. Giang, P. Calyam, B. Morago, R. Antequera, T. Nguyen, and Y. Duan. LiDAR-based virtual environment study for disaster response scenarios. *IEEE Symposium on Integrated Network Management (IM)*, pages 790–793, 2015.

[15] K. Govindarajan, K. Meng, H. Ong, M. Wong, S. Sivanand, and S. Leong. Realizing the quality of service (QoS) in software-defined networking (SDN) based cloud infrastructure. *Conference on Information and Communication Technology (ICoICT)*, pages 505–510, 2014.

[16] W. Guan, S. You, and G. Pang. Estimation of camera pose with respect to terrestrial LiDAR data. *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 391–398, 2013.

[17] A. Hafiane, K. Palaniappan, and G. Seetharaman. UAV-Video registration using block-based features. In *IEEE International Symposium on Geoscience and Remote Sensing (IGARSS)*, pages 1104–1107, 2008.

[18] A. Hafiane, G. Seetharaman, K. Palaniappan, and B. Zavidovique. Rotationally invariant hashing of median patterns for texture classification. *Lecture Notes in Computer Science (LNCS)*, pages 619–629, 2008.

[19] A. Haridas, R. Pelapur, J. Fraser, F. Bunyak, and K. Palaniappan. Visualization of automated and manual trajectories in wide-area motion imagery. *IEEE Conference on Information Visualization (IV)*, pages 288–293, 2011.

[20] R. Hartley and A. Zisserman. *Multiple View Geometry*. Cambridge University Press, 2010.

[21] A. Herrera. NVIDIA GRID: Graphics accelerated VDI with the visual performance of a workstation. *NVIDIA Corp*, 2014.

[22] A. Hossain. Framework for a cloud-based multimedia surveillance system. *International Journal of Distributed Sensor Networks*, 2014.

[23] Z. Jiang, D. Chan, M. Prabhu, P. Natarajan, H. Hao, and F. Bonomi. Improving web sites performance using edge servers in fog computing architecture. *IEEE Symposium on Service Oriented System Engineering (SOSE)*, pages 320–323, 2013.

[24] J. Klontz and A. Jain. A case study of automated face recognition: The Boston marathon bombings suspects. *IEEE Computer*, pages 91–94, 2013.

[25] T. Korkmaz and M. Krunz. Multi-constrained optimal path selection. *IEEE Conference on Computer Communications (INFOCOM)*, 2001.

[26] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, L. Čehovin, G. Nebehay, T. Vojíř, G. Fernandez, A. Lukež, A. Dimitriev, et al. The visual object tracking VOT2014 challenge results. *IEEE European Conference on Computer Vision Workshops (ECCVW)*, pages 191–217, 2014.

[27] S. Kumar, R. Rathy, and D. Pandey. Design of an ad-hoc network model for disaster recovery scenario using various routing protocols. *ACM Conference on Advances in Computing, Communication and Control (ICAC3)*, pages 100–105, 2009.

[28] M. Kwan and D. Ransberger. LiDAR assisted emergency response: Detection of transport network obstructions caused by major disasters. *Computers, Environment and Urban Systems*, pages 179–188, 2010.

[29] B. Liu, Y. Chen, R. Blasch, K. Pham, D. Shen, and G. Chen. A holistic cloud-enabled robotics system for real-time video tracking application. *Lecture Notes in Electrical Engineering*, pages 455–468, 2014.

[30] B. Liu, Y. Chen, A. Hadiks, E. Blasch, A. Aved, D. Shen, and G. Chen. Information fusion in a cloud computing era: a systems-level perspective. *IEEE Aerospace and Electronic Systems Magazine (AES-M)*, pages 16–24, 2014.

[31] Y. Ma, S. Soatta, J. Kosecka, and S. Sastry. *An Invitation to 3-D Vision*. Springer, 2004.

[32] A. Mastin, J. Kepner, and J. Fisher. Automatic registration of LiDAR and optical images of urban scenes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2639–2646, 2009.

[33] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM Computer Communication Review (CCR)*, pages 69–74, 2008.

[34] A. Mohammad. and H. Eui-Nam. Fog Computing micro datacenter based dynamic resource estimation and pricing model for IoT. *IEEE Conference on Advanced Information Networking and Applications (AINA)*, pages 687–694, March 2015.

[35] B. Morago, G. Bui, and Y. Duan. Integrating LiDAR range scans and photographs with temporal changes. *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 732–737, 2014.

[36] B. Morago, G. Bui, and Y. Duan. An ensemble approach to image matching using contextual features. *IEEE Transactions on Image Processing (TIP)*, 2015.

[37] K. Nahrstedt and S. Chen. Coexistence of QoS and best-effort flows. *Multimedia Communications*, pages 175–188, 1999.

[38] K. Palaniappan, F. Bunyak, P. Kumar, I. Ersoy, S. Jaeger, K. Ganguli, A. Haridas, J. Fraser, R. Rao, and G. Seetharaman. Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video. *IEEE Conference on Information Fusion (FUSION)*, pages 1–8, 2010.

[39] K. Palaniappan, R. Rao, and G. Seetharaman. Wide-area persistent airborne video: Architecture and challenges. *Distributed Video Sensor Networks*, pages 349–371, 2011.

[40] G. Pandey, J. McBride, S. Savarese, and R. Eustice. Automatic targetless extrinsic calibration of a 3D LiDAR and camera by maximizing mutual information. *AAAI Conference on Artifical Intelligence*, pages 2053–2059, 2012.

[41] R. Pelapur, S. Candemir, F. Bunyak, M. Poostchi, G. Seetharaman, and K. Palaniappan. Persistent target tracking using likelihood fusion in wide-area and full motion video sequences. *IEEE Conference on Information Fusion (FUSION)*, pages 2420–2427, 2012.

[42] R. Pelapur, K. Palaniappan, and G. Seetharaman. Robust orientation and appearance adaptation for wide-area large format video object tracking. *IEEE Conference on Advanced Video and Signal based Surveillance (AVSS)*, pages 337–342, 2012.

[43] N. Ray and A. Turuk. A framework for disaster management using wireless ad hoc networks. *ACM Conference on Communication, Computing and Security (CCS)*, pages 138–141, 2011.

[44] V. Reilly, H. Idrees, and M. Shah. Detection and tracking of large number of targets in wide area surveillance. *IEEE European Conference on Computer Vision (ECCV)*, pages 186–199, 2010.

[45] N. Schurr, J. Marecki, M. Tambe, P. Scerri, N. Kasinadhuni, and J. P. Lewis. The future of disaster response: Humans working with multiagent teams using DEFACTO. *AAAI Symposium: AI Technologies for Homeland Security*, pages 9–16, 2005.

[46] S. Seetharam, P. Calyam, and T. Beyene. ADON: Application-driven overlay network-as-a-service for data-intensive science. *IEEE Conference on Cloud Networking (CloudNet)*, pages 313–319, 2014.

[47] A. Smeulders, D. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 1442–1468, 2014.

[48] I. Stamos, L. Liu, C. Chen, G. Wolberg, G. Yu, and S. Zokai. Integrating automated range registration with multiview geometry for the photorealistic modeling of large-scale scenes. *International Journal of Computer Vision (IJCV)*, pages 237–260, 2008.

[49] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 246–253, 1999.

[50] I. Stojmenovic and W. Sheng. The Fog computing paradigm: Scenarios and security issues. *IEEE Conference on Computer Science and Information Systems (FedCSIS)*, pages 1–8, 2014.

[51] N. Truong, L. Gyu, and Y. Ghamri-Doudane. Software defined networking-based vehicular adhoc network with Fog computing. *IEEE Symposium on Integrated Network Management (IM)*, pages 1202–1207, 2015.

[52] Y. Tsung-Feng, K. Wang, and Y. Hsu. Adaptive routing for video streaming with QoS support over SDN networks. *Conference on Information Networking (ICOIN)*, pages 318–323, 2015.

[53] R. Vázquez-Martín, P. Núñez, A. Bandera, and F. Sandoval. Curvature-based environment description for robot navigation using laser range sensors. *Sensors*, pages 5894–5918, 2009.

[54] Q. Wang and S. You. A vision-based 2D-3D registration system. *IEEE Workshop on Applications of Computer Vision (WACV)*, pages 1–8, 2009.

[55] Y. Wen, X. Zhu, J. Rodrigues, and C. Chen. Cloud mobile media: reflections and outlook. *IEEE Transactions on Multimedia*, pages 885–902, 2014.

[56] U. Witkowski, E. Habbal, M. A. Mostafa, S. Herbrechtsmeier, A. Tanoto, J. Penders, L. Alboul, and V. Gazi. Ad-hoc network communication infrastructure for multi-robot systems in disaster scenarios. *EURON Workshop on Robotics for Risky Interventions and Environmental Surveillance (RISE)*, 2008.

[57] G. Yang, J. Becker, and C. Stewart. Estimating the location of a camera with respect to a 3D model. *IEEE Conference on 3-D Digital Imaging and Modeling (3DIM)*, pages 159–166, 2007.

[58] L. Zhou and H. Wang. Toward blind scheduling in mobile media cloud: Fairness, simplicity, and asymptotic optimality. *IEEE Transactions on Multimedia*, pages 735–746, 2013.

[59] W. Zhu, C. Luo, J. Wang, and S. Li. Multimedia cloud computing. *IEEE Signal Processing Magazine*, pages 59–69, 2011.

Rasha Gargees received her MS degree in Computer Science from University of Mosul, Iraq in 2011. She is currently pursuing her PhD degree in Computer Science at University of Missouri-Columbia. Her current research interests include computer networking, distributed and cloud computing, and big data.

Brittany Morago received a BS degree in Digital Arts and Sciences from the University of Florida in 2010 and her PhD in Computer Science at the University of Missouri-Columbia in 2016. She is currently an Assistant Professor in the Department of Computer Science at the University of North Carolina-Wilmington. Her research interests include computer vision and graphics. She is a recipient of NSFGRF and GAANN Fellowships.

Rengarajan Pelapur received his BS degree in Computer Engineering from University of Pune, India in 2010. He is currently a PhD student in the Department of Computer Science at University of Missouri-Columbia. His current research interests include appearance-based target tracking in imagery, biomedical image analysis, and multiscale theory.

Dmitrii Chemodanov received his MS degree from the Department of Computer Science at Samara State Aerospace University, Russia in 2014. He is currently a PhD student in the Department of Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, network and service management, and peer-to-peer networks.

Prasad Calyam received his MS and PhD degrees from the Department of Electrical and Computer Engineering at The Ohio State University in 2002 and 2007, respectively. He is currently an Assistant Professor in the Department of Computer Science at University of Missouri-Columbia. His current research interests include distributed and cloud computing, computer networking, and cyber security. He is a Senior Member of IEEE.

Zakariya Oraibi received his BS and MS degrees from the University of Basrah, Iraq in 2007 and 2010, respectively. He is currently pursuing his PhD degree in Computer Science at the University of Missouri-Columbia, and is a recipient of HCED scholarship. His research interests include image processing and machine learning.

Ye Duan received his BA degree in Mathematics from Peking University in 1991. He received his MS degree in Mathematics from Utah State University in 1996. He received his MS and PhD degree in Computer Science from the State University of New York at Stony Brook in 1998 and 2003. He is currently an Associate Professor of Computer Science at University of Missouri-Columbia. His research interests include computer graphics and visualization, biomedical imaging and computer vision.

Guna Seetharaman received the PhD degree in electrical and computer engineering from the University of Miami, in 1988. He served as a Principal Engineer for Computer Vision and Video Exploitation with the Information Directorate, Air Force Research Laboratory, Rome, NY. He served as an Associate Professor of Computer Science and Engineering with the Air Force Institute of Technology (AFIT) (2003?2008) and the University of Louisiana at Lafayette (1988?2003). He is currently a Senior Scientist with the Senior Executive Service for advanced computing concepts and the Chief Scientist for computational sciences with the Information Technology Division, Systems Directorate, U.S. Naval Research Laboratory, DC. His research interests include high performance computing for video exploitation, such as computer vision, machine learning, content-based image retrieval, persistent surveillance, and computational science and engineering. He guest edited the IEEE Computer Special Issue (2006) on Unmanned Intelligent Autonomous Vehicles and the Special Issue of the EURASIP Journal on Embedded Computing on Intelligent Vehicles. He was the General Chair of the IEEE Workshop on Computer Architecture for Machine Perception (2003), and the Co-Chair of the Technical Program Committee of the IEEE AIPR (2014). He served as the Section Chair of the IEEE Mohawk Valley Section, NY, in 2013 and 2014. He serves as an Associate Editor of the journal ACM Computing Surveys. He has been recognized as the Fellow of the IEEE as of January 2015.

Kannappan Palaniappan received his PhD from the University of Illinois at Urbana-Champaign, and MS and BS degrees in Systems Design Engineering from the University of Waterloo, Canada. He is a faculty member in Computer Science at the University of Missouri where he directs the Computational Imaging and VisAnalysis Lab and helped establish the NASA Center of Excellence in Remote Sensing. At NASA Goddard Space Flight Center he co-invented the Interactive Image SpreadSheet for visualizing large multispectral imagery and deformable cloud motion analysis. His research is at the synergistic intersection of image and video big data, computer vision, high performance computing and artificial intelligence to understand, quantify and model physical processes with applications to biomedical, space and defense imaging. Recent multi-disciplinary contributions range across orders of scale from sub-cellular microscopy at the molecular level to aerial and satellite remote sensing imaging at the macro level. In 2014 his team won first place at the IEEE Computer Vision and Pattern Recognition Change Detection Workshop video analytics challenge. He has received several notable awards including the William T. Kemper Fellowship for Teaching Excellence at the University of Missouri, ASEE Air Force and Boeing Welliver Summer Faculty Fellowships, the NASA Public Service Medal for pioneering contributions in data science for understanding petabyte-sized archives, and the first US National Academies Jefferson Science Fellowship from the state of Missouri. He is a member of the Editorial Board of the IEEE Transactions on Image Processing. He is a Senior Member of IEEE.