AutoVAPS: An IoT-Enabled Public Safety Service on Vehicles

Liangkai Liu
Wayne State University

Xingzhou Zhang
Wayne State University

Qingyang Zhang
Anhui University

Andrew Weinert*
MIT Lincoln Laboratory

Yifan Wang
Wayne State University

Weisong Shi
Wayne State University

ABSTRACT

With the rapid development of Internet-of-Things, sensors and devices are connected enabling a variety of applications. One of the most attractive applications is Video Analysis for Public Safety (VAPS), which has got massive attention from both research community and industry. However, there are still challenges in the system design and implementation of the VAPS service, especially in the mobile environment. For example, law enforcement officers are equipped with body-worn camera when they are on duty, how to connect body-worn cameras with the law enforcement vehicle and enable the law enforcement vehicle to perform near real-time video analysis for the officer are still open questions. Inspired by the promising edge computing technology, we propose an IoT-Enabled public safety service called *AutoVAPS* which integrates body-worn cameras and other sensors on the vehicle for public safety. In *AutoVAPS*, we propose a reference architecture that consists of the data layer for data management, the model layer for edge intelligence, and the access layer for privacy-preserving data sharing and access. Object detection is implemented as a case study of *AutoVAPS*. Early evaluation illustrated the applicability and challenges of *AutoVAPS*.

KEYWORDS

Internet of Things, public safety, video analytics, edge computing

ACM Reference format:
Liangkai Liu, Xingzhou Zhang, Qingyang Zhang, Andrew Weinert, Yifan Wang, and Weisong Shi. 2019. AutoVAPS: An IoT-Enabled Public Safety Service on Vehicles. In *Proceedings of 4th International Science of Smart City Operations and Platforms Engineering, 2019, 4th International Science of Smart City Operations and Platforms EngineeringSCOPE (SCOPE),* 6 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the era of Internet-of-Things (IoT), computing devices are coming increasingly connected. There could be an estimated 50 billion IoT devices by 2020 with a potential yearly economic impact of $11.1 trillion by 2025 [3]. However many IoT use cases, such as Video Analytics for Public Safety (VAPS) [4] face development and operational challenges that limit their potential impact.

Due to small fiscal budgets, size, weight, and power constraints, and the complexity of public safety operations, analytic integration and optimization is a significant challenge for public safety. This is counter to the majority of video research which focuses more on algorithmic design and have a lack of safety or mission critical requirements. An end-to-end architecture that includes representative public safety datasets and standardized performance-based metrics is lacking for VAPS, hindering the development of public safety applications. For example, law enforcement routinely employ IoT devices of body-worn cameras, mobile devices, vehicle dash cameras, and vehicle-based computing. However there are still many architectural and operational challenges hindering the generation of near real-time actionable intelligence.

In response, we propose an IoT enabled service, *AutoVAPS*, that is representative of desired and future public safety vehicles with embedded computer. The *AutoVAPS* reference architecture is an extension of prior work on *OpenVDAP* [17], specifically designed to integrate body-worn cameras and other public safety sensors while leveraging the advancements in edge computing [12]. It consists of three layers: data, model, and access. The data layer automatically aggregates relevant context information from on-vehicle, body-worn, and cloud-based sensors onto the vehicle. The model layer supports edge intelligence, storing many common algorithms and models. The access layer enables privacy-preserving data sharing and access through open developer Restful API interfaces. As part an agile design process, we evaluated an initial prototype of *AutoVAPS* with an object detection use case and measured the latency composition, variance of inter frame, and frame loss and rate. The contributions of this paper are as follows:

- A proposed reference architecture based on *OpenVDAP* but intended for public safety research and development .
- *AutoVAPS* prototype that integrated representative public safety sensors and vehicles.
- Initial performance of *AutoVAPS* that highlights latency challenges of the public safety environment.

The paper is organized as follows. Sections 2 depicts the motivation of *AutoVAPS*. Section 3 discusses the reference architecture of *AutoVAPS*. We present the case study of *AutoVAPS* in Section 4, the evaluation of the proposed service is discussed in Sections 5, and summarize the paper in the last section.

2 MOTIVATION AND SCOPE

Currently there is significant research in autonomy in vehicles which require edge computing based architectures to achieve the vision of a self-driving cars. Like the cloud computing, there are many different ways to architecture compute and storage; there isn't a one size fits all solution. Embedding vehicle compute is challenging due to size, weight, and power constraints. As the public safety community, with unique operational requirements,

*This work was performed under the following financial assistance award 70NANB17Hl69 from U.S. Department of Commerce, National Institute of Standards and Technology.

2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

integrates technology into their vehicles, there is a need to optimize and gracefully degrade performance as resources become degraded.

In the era of cloud computing, technology still fails to provide situational awareness capabilities with the desired reliability, robustness, and compatibility for public safety. As the community transitions to a more edge computing centric network, there is an opportunity to address some of the shortcomings of a cloud centric architecture and enable development of capabilities [15] to (1) improve incident or threat description, (2) to improve two-way situational awareness, and (3) that minimize implementation of new workload or policies.

AutoVAPS will be a reference architecture that will enable this capability development and built upon the lessons learned from years of iterative public safety driven designs [2, 13–16]. This initial development is scoped to enabling VAPS research with dash or body-worn cameras and illustrated by Figure 1. This is driven by the law enforcement operational realities of single vehicle patrols[15] and the widespread adoption of body-worn cameras.

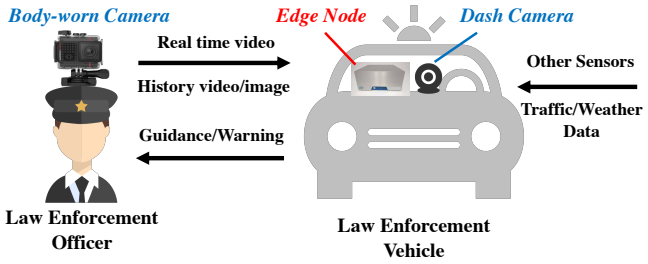

Figure 1: Initial *AutoVAPS* scope.

Specifically consider law enforcement officer is on duty, she/he is equipped with a body-worn camera, which will collect the front video and sends the data to the law enforcement vehicle. On the vehicle there is an edge node which receives various sensors streams, such as video from the body-worn or vehicle dash cameras. Together these sensors capture some perspective of the officer and the surroundings around them. The edge node manages all the data and executes the analytical applications, such as license plate detection, object detection, or situational awareness cues. Guidance or warning will be communication across either the local or back haul network if the officer is in a potentially dangerous situation.

This scenario defined by operational realities also the highlight need for an IoT reference architecture designed for public safety. The consumer passengers of self driving cars likely will not be equipped with body-worn cameras similar to an law enforcement officer. Body-camera support is a foundational requirement for *AutoVAPS* whereas not for consumer-based research. *AutoVAPS* can reduce risk for future VAPS by providing the architecture and interfaces that may be lower priority for consumer focused research.

3 REFERENCE ARCHITECTURE

The proposed *AutoVAPS* three layer reference architecture is based on *OpenVDAP* [17]. The access layer enables privacy-preserving data sharing/access; the data layer enables efficient data storage and management; and the model layer enables edge intelligence. Figure 2 illustrates the reference architecture for object detection.

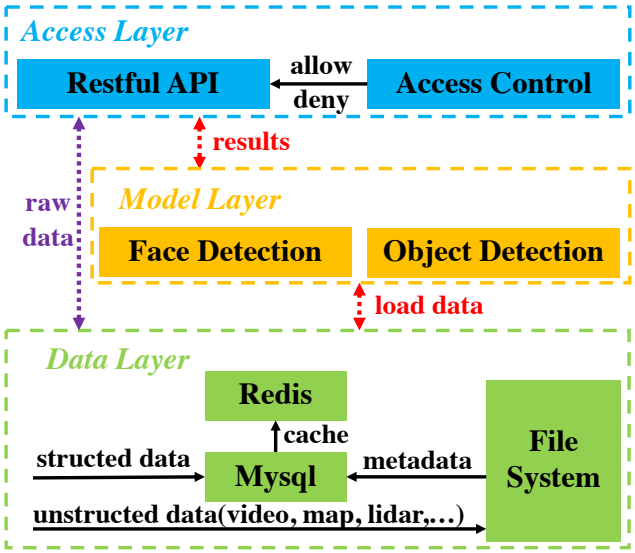

Figure 2: The reference architecture of *AutoVAPS*.

3.1 Access Layer

The primary objective of *Access Control* is to enable developers to access data and analytic results through a Python flask microframework *Restful API*, as well as communications and authenticate between different applications. The *RESTful API* is designed as readable URLs which contain several segments indicating the type, identifier, and an Unix timestamp of the data. It can be represented as http://ip-address:port/type/identifier/timestamp. Here *ip-address* and *port* are static, and *type* including *body-camera*, *dash-camera*, *weather*, *OBD*, and *traffic*. *Identifier* can be *video* or *image* or *raw-data*.

The *Access Control* module includes three sub-modules: an application level access sub-module, a system level access sub-module, and an authority management sub-module. The former two sub-modules are used to manage all the data access operations, communications and IO operations, cooperatively. The application level access sub-module is designed to accept popular application level protocols, like HTTP, RTSP, and forward the authenticated queries to service providers, like the *Restful API* module. The prototype access control was registered to `NGX_HTTP_ACCESS_PHASE` in NGINX, which is a free, open-source, high-performance HTTP server and reverse proxy. When NGINX receives an HTTP request to the data layer, the implemented NGINX module will collect the information about the request, including data type and the identity of the request, and sends these information to the authority management sub-module for authentication, which is implemented by `web.py` framework, and provide a white-list based model as the primary access control model.

However, as *AutoVAPS* provides not only raw data but also real file path, extra IO operations for reading a file are also included. Additionally, various Inter-Process Communications (IPCs) are widely used between different applications and a system level access sub-module for IPC management is proposed. All the access operations are determined by the application level across modules.

3.2 Model Layer

With the burgeoning of AI-based applications developed for public safety, *AutoVAPS* should support a variety of artificial intelligence algorithms and models. However, the on-vehicle computing platform is not appropriate for executing large scale models due to large compute and storage requirements.

This layer is designed to contain many common algorithms and models that are frequently used for VAPS. These models are compressed based on the powerful models and can run smoothly on the edge node. After compressing, the models are optimized based on the computing power of the on-vehicle computing platform. Meanwhile, models can be trained based on some personalized data, which makes the model fit for some personalized behavior detection. When the model layer gets requests from the access layer, it first loads the request model and then sends requests to data layer for corresponding data. Based on the model layer, developers can build public safety applications more openly and easily. Additionally one of the challenges we face is deciding to support either Tensorflow or PyTorch, the two most popular open source machine learning frameworks, for the initial prototype.

3.3 Data Layer

The data Layer is designed to automatically collect and manage vehicle data. This includes driving data from the on-vehicle sensors like On-Board Diagnostics (OBD) and LiDAR, videos and images from the camera, High Definition (HD) Map data, and some context information. Context data of driving, such as road condition, weather, traffic information, can play an important role in decision making and support assistant driving, battery cell management, remote diagnostics, and abnormal driving behavior detection.

The database and file system to manage different types of data: structured data including the data from the sensor, weather data, and traffic data is persisted into database, while video, HD map, LiDAR cloud point data, and image data are stored and indexed with additional metadata. Frequently used and some static data like the RTSP URL of the body-worn camera will be cached. Requests from the *RESTful API* module or model layer will be automatically directed. Specifically the Real Time Streaming Protocol (RTSP) is used for video transmission from the body-worn camera to the edge node, with video stored and cached in Redis.

4 CASE STUDY

The vision of *AutoVAPS* is to interface a body-worn camera with the in-vehicle edge node to do near real-time video analytics. To achieve this, two problems should be solved: the real-time transmission of the video stream from body-worn camera to the edge node, and the real-time analysis of the video stream on the edge node.

4.1 System Overview

The initial prototype design of *AutoVAPS* is shown in Figure 3 where the requests from the user trigger the pipeline. Here we assume the user requests for video analytic results. Two cameras are connected with the edge node: body-worn camera is connected via *Wi-Fi* and video transmission via RTSP protocol, while dash camera is connected via USB port. OpenCV is used to open either the RTSP URL or the USB camera to get the video stream. For body-worn camera, to decrease the influence of *Wi-Fi* communication delay, a Multi-Process(MP) scheme is used to average the speed of frame reading and model inference. A Convolutional Neural Network (CNN) [10] trained for object detection receives the newest frame from the buffer and do inference. For USB camera, as it is wired connected with the edge node, the communication delay is stable. Therefore, the frame is directly loaded into the CNN model to do inference. After model inference, the results will be appended into the video and be sent to the user through *Restful API* module.

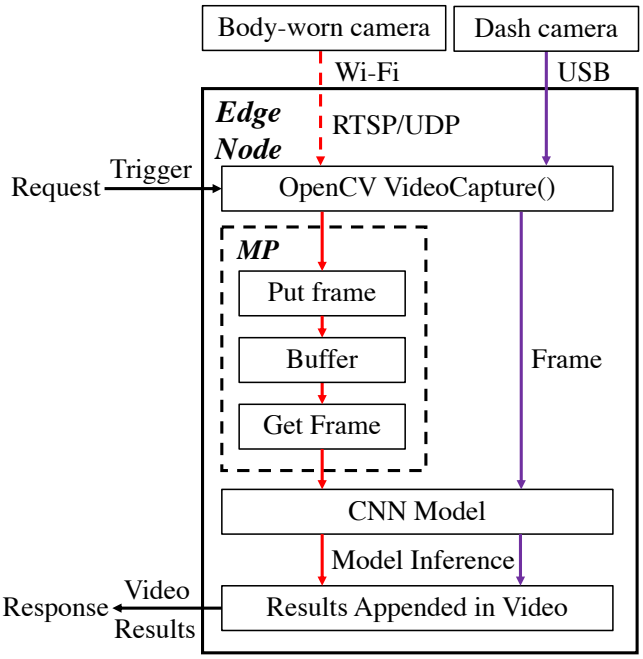

Figure 3: The system design of object detection.

4.2 Real Time Video Transmission

For real-time video analytics, the video transmission is essential when determining the overall performance. *AutoVAPS* implements the Real Time Streaming Protocol (RTSP), an application-level protocol, for video stream transmission [11]. RTSP provides a variety of delivery channels like UDP, multicast UDP and TCP, and it allows the user to choose delivery mechanisms using RTP.

For body-worn camera, the initial prototype *AutoVAPS* RTSP server only supported UDP, subsequently introducing potential frame loss in transmitting video frames from body-worn camera the edge node. Frame loss increases the transmission delay, and stifles the the object detection analytic due to a lack of input video. Meanwhile, it only takes several milliseconds to read a frame from the body-worn camera, but it takes tens of milliseconds to do model inference. Therefore, the reading frame process is much faster than the model inference resulting in increasingly latent data for interference. This reading and processing issue can be addressed by a proposed Multi-Process Scheme, Algorithm 1, which uses a buffer to decrease the difference of frame reading and processing speeds.

Algorithm 1 Pseudocode of Multi-Process Scheme.

```
function Frame-Reading(url)
    f ← VideoCapture(url) // Get the newest frame
    if (buffer.length < buffer.size) then
        Add f into the buffer
        buffer.length = buffer.length + 1
    else
        Dequeue the oldest frame from buffer
        Add f into the buffer
    end if
end function

function Frame-Processing
    while True do
        f ← buffer.dequeue() // Get frame from the buffer
        Load the CNN model M
        resutls ← M.Inference(f) //Do model inference on f
        f ← f.append(resutls) Add results on f
    end while
end function
```

In this algorithm, the work is divided into two processes. One process *Frame-Reading*, which reads frame using OpenCV and puts the newest frame into the buffer. The other process receives a buffered frame and conducts model inference. Through this design of reading and processing, the latency between the final shown up video with the body-worn camera's video can be greatly decreased. Because in the reading process, when a new frame comes and the buffer is full, we choose to abandon the oldest frame and store the new frame. As *Frame-Processing* only takes frame from the buffer, which guarantees that the frame to do model inference is always the newest. The MP scheme can be seen as a trade-off between the frame per second (FPS) and real-time performance. Many frames are abandoned to prioritize processing the most recent frames.

4.3 Algorithmic Design

The prototype object detection can be divided into three states: loading, detection, and visualization. 1) In the loading state, the image is captured from the body-worn camera. The trained CNN model from the TensorFlow Model Zoo is loaded into memory. We did not optimize the network architecture of the CNN [1]. 2) Detection is the core state of the whole process. In this state, we run the MobileNet-SSD [5] model to detect the objects, including vehicles, signs, and human being. As a state-of-the-art deep learning model, MobileNet-SSD can be regarded as a combination of MobileNet and Single Shot MultiBox Detector (SSD) [9]. MobileNet replaces the lightweight depthwise separable convolution layer with standard convolution layer to reduce the number of computations, which is more suitable for the resource-constrained vehicle computing platform. SSD is a widely used deep learning model for objection detection. 3) In the visualization state, the images are composited into the video and the results of the detection will be visualized to users. For each frame of the video, outputs are boxes, classes, and scores. Each box represents an image segment where a particular object was detected and each score represents the confidence for the class. The output is the number of detections.

For the prototype, OpenCV *VideoCapture()* and *read()* functions are used to capture the video, split it into each frame, and read in the memory. TensorFlow Object Detection API[6] 1.10 is leveraged to run the deep learning model for object detection. The MobileNet-SSD downloaded from the TensorFlow detection model zoo is pre-trained on the MS-COCO dataset[7].

5 EVALUATION

For public safety applications, the most significant part is the availability and stability of real-time analytic performance. To evaluate the performance of *AutoVAPS*, we implemented the system based on Garmin VIRB Ultra 30 body-worn camera, Intel Real Sense SR300, and Intel Fog Reference Design. And we measure the latency breakdown, time variance of frame, and frame loss rate.

5.1 Experimental Design

The hardware configuration of the body-worn camera and dash camera is shown in Table 1. The video format of both cameras is H.264. The Intel Fog Reference has 8 Intel Xeon(R) CPU with 3.60GHz frequency. And it has 32GB memory. We use the Intel Fog Reference as the on-vehicle computing device as the edge node.

Table 1: The hardware configuration of body-worn camera and dash camera.

Name	Model	Resolution	FPS	Connection & Bandwidth
Body-worn Camera	Garmin VIRB Ultra 30	1920x1080	60	802.11 b/g/n 450Mbps
Dash Camera	Intel Real Sense SR300	640x480	30	USB 2.0 480Mbps

For experiments, a Wi-Fi hotspot is set up in Garmin body-worn camera and the edge node connects to body-worn camera via Wi-Fi. We measure the latency of the whole process including reading frame, model inference, and results output. Body-worn camera is measured in two cases: one is the distance from body-worn camera to edge node is 1 meter, the other is 20 meters. Dash camera is also measured in two cases: using Multi-Process(MP) scheme and not using Multi-Process(MP) scheme. These four cases are shown as *Body-Camera-1M*, *Body-Camera-20M*, *Dash-Camera-MP*, and *Dash-Camera-No-MP*. And the buffer size in the MP scheme was the two most recent frames.

5.2 Latency Breakdown

Figure 4 shows the time sequence of *AutoVAPS*. For each frame, t_0 and t_1 represents the start time and end time of the read frame process, so t_1-t_0 is the read frame delay. t_2 and t_3 represent the start time and end time of model inference respectively, so the model inference delay is t_3-t_2. Result output is done exactly after model inference and t_4 is the end time for result output, so t_4-t_3 is the result output delay.

The latency breakdown of read frame delay, model inference delay, and result output delay in four cases are shown in Figure 5. From

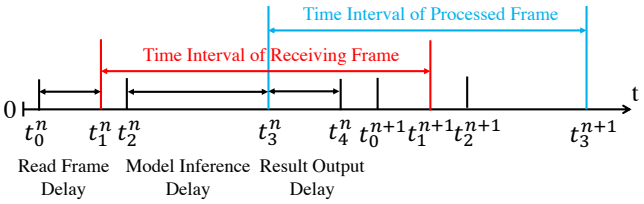

Figure 4: The time sequence of *AutoVAPS*.

Figure 5, we can see that the model inference makes up the majority of the total delay and MP scheme makes the read frame delay longer. The resulting delay is the smallest and similar across cases. Also, when the distance of Wi-Fi connection increases, reduced signal strength makes the reliability of video data transmission decreases. From Figure 5, the read frame delay of *Body-Camera-20M* is higher than that of *Body-Camera-1M* by 9 ms. The MP scheme makes the read frame delay higher to decrease the difference between read frame delay and processing delay. For body-worn camera without MP scheme, the processing program crashes frequently because the frame processing is much slower than frame reading. And we can see the overall delay of these four cases are less than 150ms, which can be seen as near real-time.

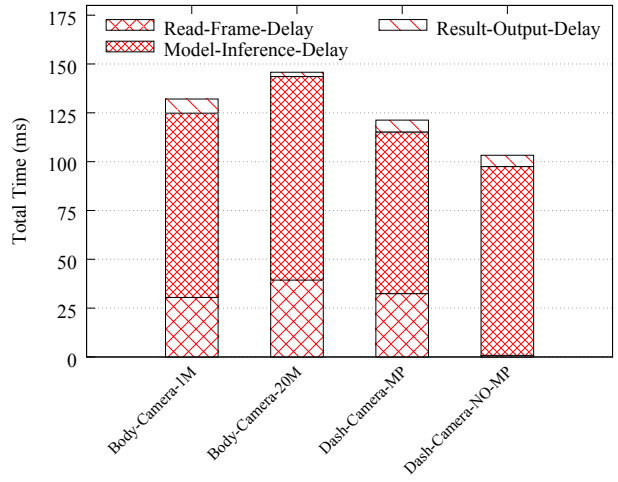

Figure 5: The latency breakdown of *AutoVAPS*.

Observation 1: The delay of model inference still takes a large portion of the overall delay. So designing appropriate models to speed up model inference on edge is still a challenge.

5.3 Variance of Inter Frame

In order to evaluate the variance of inter frame, we calculate the Δt_1, which is the difference of $(n+1)^{\text{th}}$ frame's t_1 and that of the n^{th} frame's, and it equals to $t_1^{n+1} - t_1^n$. Δt_3 represents the difference of processing time of nearby two frames. Δt_1 and Δt_3 are also marked in Figure 4. And we make the Cumulative Distribution Function(CDF) of Δt_1 and Δt_3 in Figure 6 and Figure 7.

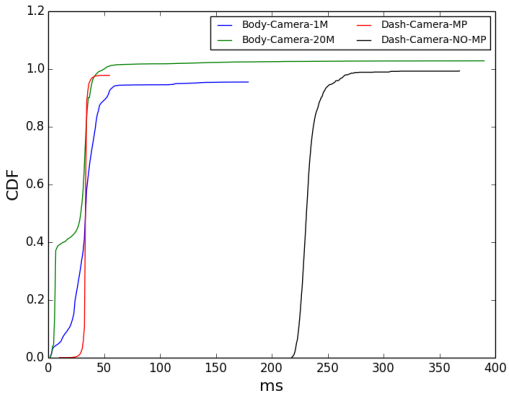

Figure 6: The variance of receiving inter frame.

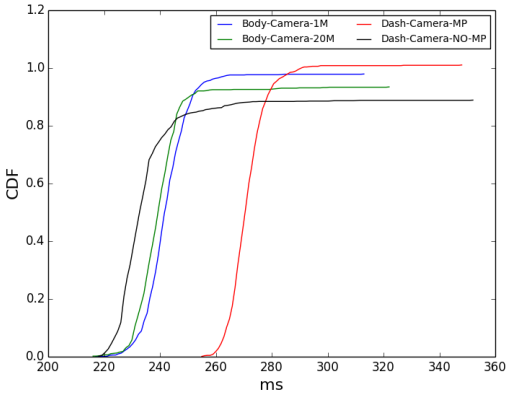

Figure 7: The variance of processing inter frame.

In Figure 6, the majority of variance of receiving inter frame for *Body-Camera-1M*, *Body-Camera-20M*, and *Dash-Camera-MP* is under 50ms, while that for *Dash-Camera-No-MP* is about 250ms. It reveals that the frame reading through MP scheme is more real-time. In Figure 7, over 80 percent of Δt_3 for all four cases are in the range of 230-270ms. For *Body-Camera-1M*, *Body-Camera-20M*, and *Dash-Camera-MP*, the difference of Δt_3 and Δt_1 is about 190ms. And the difference for *Dash-Camera-No-MP* is about 20ms. The additional 170ms is the overhead of applying the MP scheme. However, without the MP scheme to abandon the old frame and model inter on the latest frame, the result will always be out-of-date. In addition, the performance of *Body-camera-20M* and *Body-camera-1M* is almost the same. Although 1m case receives frames faster, the processing time is much longer than the reading time. Thus the inter frame is determined by the processing time.

Observation 2: Multi-Process scheme can decrease the difference of frame reading speed and processing speed. But how to optimally determine the trade-off between the buffer size and timeliness of processing frames remains a challenge.

Table 2: The frame loss and rate in four cases.

Type	Total Frame	Received Frame	Receiving Frame Loss	Receiving Frame Loss Rate	Processed Frame	Processing Frame Loss	Process Frame Loss Rate	Total Frame Loss Rate
Body Camera(1M)	5093	2691	2402	47.16%	641	2050	76.18	87.41%
Body Camera(20M)	3907	1599	2308	59.07%	321	1278	**79.92%**	**91.78%**
Dash Camera(MP)	2743	2723	20	**0.73%**	721	2002	73.52%	**73.71%**
Dash Camera(No-MP)	1171	300	871	**74.38%**	300	0	**0**	74.38%

5.4 Frame Loss and Rate

The frame loss and rate for those four cases are shown in Table 2. Since RTSP/UDP is used to communicate between the body-worn camera end edge node, there is a potential for frame loss. We observed a receiving frame loss of body-worn camera of 50–60 percent. As the transmission distance increases, the receiving frame loss also increases. It can be seen from Table 2 that the reading frame loss of 20M case is higher than that of the 1M case by over 12 percent. The difference of the reading frame loss is owing to the signal strength of Wi-Fi connection degrades when the distance increases. Meanwhile, there is processing frame loss for body-worn camera because of the MP scheme, with total frame loss rate of about 74–92 percent. When total frame loss rate is 90 percent, only 10 percent of the original video can be shown resulting in an output of 6 FPS. For dash camera, the receiving frame loss rate for *Dash-Camera-MP* is less than 1 percent. But that for *Dash-Camera-No-MP* is 74.38 percent because the frame reading needs to wait for the processing of the former frame. And it doesn't lose any frame in processing, while that for *Dash-Camera-MP* is 73.25 percent. The overall frame loss is almost the same for these two cases. Therefore, using MP or not depends on the reliability of the communication. For body-worn cameras using Wi-Fi, MP scheme can be a better choice.

Observation 3: The frame loss is unavoidable because frame reading is faster than processing. Discarding out-of-date frames can make the analysis more real-time but it decreases the FPS. But how to achieve real-time performance with low overhead is a challenge.

6 SUMMARY

In this paper, we propose an IoT-Enabled public safety service called *AutoVAPS* which can integrate body-worn camera and other cameras for public safety. It is designed as an reference architecture which includes a data layer for data management, a model layer for edge intelligence, and an access layer for privacy-preserving data sharing and access. This preliminary work illustrates *AutoVAPS* applicability to public safety and prioritize development for the next iteration of *AutoVAPS*. Future work will leverage publicly available representative public safety datasets to further refine *AutoVAPS* and experiment with semantic-based analytics [8] whose more primitive models may address some of the challenges with processing speed.

REFERENCES

[1] Michael Chan, Daniel Scarafoni, Ronald Duarte, Jason Thornton, and Luke Skelly. 2018. Learning Network Architectures of Deep CNNs Under Resource Constraints. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Salt Lake City, UT, 1784–17847. https://doi.org/10.1109/CVPRW.2018.00222

[2] Peter Erickson, Andrew Weinert, Paul Breimyer, Matt Samperi, Jason Huff, Carlos Parra, and Scarlett Miller. 2013. Designing public safety mobile applications for disconnected, interrupted, and low bandwidth communication environments. In *2013 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, Waltham,MA, 790–796. https://doi.org/10.1109/THS.2013.6699028

[3] Paula Fraga-Lamas, Tiago M Fernández-Caramés, Manuel Suárez-Albela, Luis Castedo, and Miguel González-López. 2016. A review on internet of things for defense and public safety. *Sensors* 16, 10 (2016), 1644.

[4] John S Garofolo, Simson L Garfinkel, and Reva B Schwartz. 2017. *First workshop on video analytics in public safety*. NIST Pubs NISTR-8164. NIST. https://www.nist.gov/publications/first-workshop-video-analytics-public-safety

[5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017), 9. arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[6] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. 2017. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Honolulu, HI, 3296–3297. https://doi.org/10.1109/CVPR.2017.351

[7] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Zurich, Switzerland, 740–755.

[8] Jeffrey Liu, Andrew Weinert, and Saurabh Amin. 2018. Semantic Topic Analysis of Traffic Camera Images. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, Maui, HI, 568–574. https://doi.org/10.1109/ITSC.2018.8569449

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Amsterdam, The Netherlands, 21–37.

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 6 (June 2017), 1137–1149. https://doi.org/10.1109/TPAMI.2016.2577031

[11] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and Ed M. Stiemerling. 2016. *Real-Time Streaming Protocol Version 2.0*. Technical Report. Internet Engineering Task Force (IETF). https://doi.org/10.17487/RFC7826

[12] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[13] W. Smith, G. Kuperman, M. Chan, E. Morgan, H. Nguyen, N. Schear, B. Vu, A. Weinert, M. Weyant, and D. Whisman. 2017. Cloud computing in tactical environments. In *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*. IEEE, Baltimore,MD, 882–887. https://doi.org/10.1109/MILCOM.2017.8170823

[14] Andy Vidan and Gregory Hogan. 2010. Integrated sensing and command and control system for disaster response. In *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, Waltham,MA, 185–189. https://doi.org/10.1109/THS.2010.5655066

[15] Andrew Weinert and Chris Budny. 2018. Outreach to Define a Public Safety Communications Model For Broadband Cellular Video. In *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*. IEEE, Woburn,MA, 1–4. https://doi.org/10.1109/THS.2018.8574193

[16] Andrew Weinert, Hongyi Hu, Chad Spensky, and Benjamin Bullough. 2015. Using open-source hardware to support disadvantaged communications. In *2015 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE, Seattle,WA, 79–86. https://doi.org/10.1109/GHTC.2015.7343958

[17] Qingyang Zhang, Yifan Wang, Xingzhou Zhang, Liangkai Liu, Xiaopei Wu, Weisong Shi, and Hong Zhong. 2018. OpenVDAP: An Open Vehicular Data Analytics Platform for CAVs. In *Distributed Computing Systems (ICDCS), 2018 IEEE 38th International Conference on*. IEEE, Vienna, Austria. https://doi.org/10.1109/ICDCS.2018.00131