This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

Resource Efficient Edge Computing Infrastructure for Video Surveillance

Pavana Pradeep, Amitangshu Pal, Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, USA
Email: {pavana.pradeep, amitangshu.pal, kkant}@temple.edu

***Abstract*—The emerging edge computing applications often use high definition cameras as edge devices to capture video streams that need to be analyzed in real-time for situational understanding and answering queries. However, such devices suffer from limited energy (and hence limited computing power) and limited bandwidth available to stream the data to the edge controllers that provide much higher computing capacities. In this paper, we address these issues in the context of vehicular traffic monitoring and develop a scheme that has two components: YLLO and BATS. YLLO is a lightweight object recognition algorithm that runs on the edge device itself and substantially reduces the frame rate sent to the edge controller without removing the important information. BATS adapts the transmissions to the available bandwidth by taking advantage of further redundancy in the video stream in both single and multi-camera scenarios. We show that these mechanisms together can maintain object identification accuracy of above 95%, while transmitting just ~5-10% of all the frames recorded by the cameras.**

***Index Terms*—Edge Computing, Energy Efficiency, Video Analytics, Bandwidth Management**

I. Introduction

Edge computing is a rapidly developing field with numerous applications in the management of cyber-physical systems as discussed in [1]. Some of the prominent applications include intelligent management of road traffic, health-care operations in health-care facilities, industrial manufacturing, packaging and inventory control, automated logistics operations, etc. Most of these applications are increasingly driven by video streams generated by the monitoring cameras. Fig. 1 shows the well-accepted 3-tier model of edge computing where the individual IoT devices form the bottom layer and wirelessly connect to edge controllers (ECs) which are largely responsible for heavy-duty real-time analytics on the data streams. Many traditional IoT devices that do not necessarily have any computing capabilities can be brought into this ecosystem by connecting them to a local node with compute, storage, and wireless access capabilities through a regular wired interface such as USB. Such an arrangement allows every IoT device to be considered "smart" (i.e., have local compute, storage and wireless access capabilities).

The top layer is usually a cloud that provides both large scale storage and large offline analytics capabilities. An EC may receive data streams from many cameras and other IoT devices, usually from a specific physical region. Some common characteristics of the data streams directed to an EC include the following: they are generated by devices that are limited in processing power, provide overlapping coverage of the activities in the area, and must compete for the limited transmission bandwidth to the EC.

Although the IoT (or edge computing) devices continue to advance in their processing and storage capabilities, they are still limited by the restrictions of cost, size, energy consumption, radio bandwidth, etc. The ECs themselves are typically deployed in a hostile environment (e.g., in a small enclosure without cooling) and generally need to be designed inexpensively; and hence, an EC may also be constrained by energy consumption, computational resources, and cost. The energy consumption of IoT systems remains a significant challenge and various approaches have tried to reduce it. For example, the IoT devices may use duty cycling, opportunistic sleep, or DVFS (dynamic voltage frequency scaling), as appropriate.

In this paper, we focus on the application of edge computing to intelligent transportation systems (ITS), where it may be used for a range of functionalities including (a) tracking of suspicious or abnormally behaving vehicles, (b) management of vehicles belonging to an organization (for example, city buses, ambulances, delivery vehicles, and refuse collection trucks), (c) monitoring congested areas or areas that are prone to accidents, and (d) monitoring impacts of abnormal events such as flooding, downed power lines, fallen trees, etc. This environment has all of the characteristics mentioned above involving deployment in outdoor environment (e.g., cameras installed on light poles and an EC also hanging off a light pole, possibly every few Km, receiving camera streams from all the cameras in its region). Other environments also have similar issues and the techniques described here are applicable to them as well; however, for concreteness, we henceforth only speak of the ITS environment. In our road traffic application, there is generally no privacy concern so long as the monitoring is limited to public roads and other public spaces. In applications where privacy is an issue, it is possible to filter the frames and send only a subset or part of frames or just the metadata; however, addressing the privacy issues is beyond the scope of this paper.

The goal of this paper is to devise an ITS monitoring solution that addresses the above limitations. It consists of two components: (a) design of a lightweight and energy-efficient video-stream analytics algorithm called **YLLO** that runs on individual edge devices (EDs) and substantially reduces the video frames sent to the EC and the processing needs at the EC itself, and (b) design of another algorithm called **BATS**, that exploits the overlaps between the views of multiple cameras and dynamically adapts to the available transmission

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

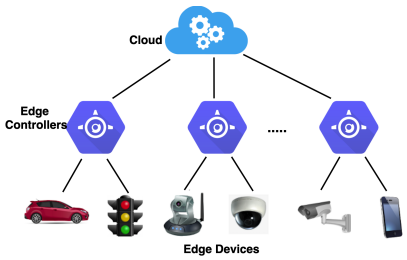


Fig. 1: Hierarchical architecture of edge computing.

bandwidth to the EC. We assume that for each important local spot to be monitored, there are a few cameras mounted on light poles on both sides of the road (typically 2-6 cameras in total), that provide an overlapping field of view of the spot. These cameras may connect to the EC via an already deployed cellular technology, including 4G-LTE and the emerging 5G, and in that case, the ECs could potentially be bundled with base stations. Other techniques include TV whitespace (i.e., the unused gap between TV channels in 470-790MHz range) and even WiFi. We do not assume a specific technology, but only consider the impact of bandwidth limitations that are likely to be present in typical deployments.

YLLO stands for "**Y**ou **L**ook **L**ess Than **O**nce" since it avoids full processing of all frames by exploiting the redundancy in a video stream. YLLO borrows basic functions from other popular single-step algorithms such as variants of the popular YOLO series of algorithms [2], where YOLO stands for "You Only Look Once". BATS stand for "**B**andwidth **A**daptive **T**ransmissions of **S**treams" and adapts the frame transmission rate to EC according to the available wireless channel capacity. BATS also assume that there are multiple cameras located in a place (e.g., same or nearby electric poles) to provide a good coverage of the entire traffic area.

For evaluation purposes, we captured High Definition Video Streams (HDVS) at 30 FPS both for real road traffic and toy cars imitating traffic patterns on the road. As discussed above, a real EC in this application will implement many capabilities; however, we consider here a single task that does require extracting a fine-grain feature of the videos, namely the identification of the license plates using the open-source Automated License Plate Reader (ALPR) software [3].[1] Using the accuracy of this task as evaluation metric, we show that the combination of YLLO and BATS can maintain an object identification accuracy of over 95%, by only sending ~5-10% of all the recorded frames. In the process, we show that our mechanism saves at least 60% energy at the ED. The savings at the EC are expected to be even higher because of the fewer frames sent to ECs that are marked with object IDs, thereby further reducing the work for EC.

The remainder of this paper is organized as below. Section II discusses the related work. Section III discusses the YOLOv4 algorithm that forms the basis for our YLLO algorithm. Section IV presents the YLLO algorithm and compares it against YOLOv4. Section V formulates the object identification optimization problem in a multi-camera environment and presents the BATS algorithm. Experimental evaluation and results are summarized in section VI. Section VII then concludes the paper.

[1]This is only an example task for EC; we are not focused on optimizing licence plate detection per se.

II. Related Work

Applying CNNs to object classification has been showing excellent performance improvements recently. In [4], the authors have used CNNs to detect the objects that demonstrate exceptional performance gains in real-time. Reference [5] compares the speed and accuracy of various CNN models. In [6] the authors have developed a cognitive assistance framework using Google Glass along with edge computing to provide satisfactory performance. A framework by Glimpse [7] analyzes data from multiple sources and presents a case study of filtering unnecessary frames for face detection. The authors in [8] suggested an application using smartphone cameras to increase pedestrian safety. This application detects vehicles approaching the pedestrian using the rear camera and alerts them. CarSafe [9] presents an Android application that detects driver fatigue using the front-facing camera along with tracking road conditions using a rear-face camera. Live video analytics using edge computing has been increasingly popular in the research community. Reference [10] has discussed the feasibility of distributed edge computing architecture for meeting the real-time requirement of large-scale video analytics. Videostorm [11] is a video analytics platform that performs query processing on live video streams, optimizing both the query knobs and resource allocation. Similar to our work, Chameleon [12] presents a video analytics system that optimizes the inference accuracy and computational resources by exploiting the temporal and spatial correlation among video frames. VideoEdge [13] uses a hierarchical architecture for edge-based video analytics to maintain higher accuracy in real-world video queries by placing and merging the queries for processing. However, it does not discuss the reduction in bandwidth required when uploading the frames to the cloud for further analysis. Vigil [14] and LAVEA [15] also explore edge-based real-time video analytics system by partitioning the analytics between the edge and the cloud. In [16], [17] the authors have proposed a roadmap for scaling video analytics using cross-camera correlations. Similar proposals of edge computing for video surveillance are studied in [18], [19], [20]. The proposal that is closest to ours is reported in [21] where the authors have used video analytics to extract some relevant information from the frames and send the metadata to the controller. In contrast, our scheme exploits the correlation between successive frames and transmits frames that have substantial change with respect to the previous frames. This is done via a scene change detection algorithm. The BATS algorithm then exploits overlaps between camera views to further avoid redundant frame transmission. Metadata transmission may be adequate in some cases, but this would allow correlation of only the metadata across multiple views from same or different cameras. A collaborative detection of fine-grain features at the edge controller would invariably require raw frames from the cameras.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

III. Device Based Object Recognition

Avoiding the transmission of all video frames from edge devices to the EC requires that the device be able to analyze the frames for the presence of "interesting" objects so that the other frames can be either discarded or stored locally and sent later for archival purposes (if necessary). Thus, the critical challenge for edge devices is the real-time recognition of evidence of relevant objects in the video stream.

A. Feature vs. Object Detection

In general, there are two ways to identify frames with interesting objects: (a) look for features that relate to the presence of relevant objects, or (b) directly detect and classify the objects in the stream. From the perspective of real-time operation on low-end devices, option (a) appears quite attractive. We initially explored option (a) using state of the art feature detection methods such as FAST, BRIEF, SIFT, and ORB [22]. Of these, the combined algorithm "Oriented FAST and rotated BRIEF" (ORB) is the fastest and designed for real-time use [23].

Key points are low-level features which do not use a hierarchical layer-wise representation. Fig. 2 shows the key-point characterization of a frame using ORB. The accuracy of key points is affected by the image noise, the similarities between the foreground and the background, and high texture repetition. Under such conditions, it is challenging to achieve discriminative features across the frames using ORB or any other traditional feature detectors. Our experimentation showed that ORB could indeed work reasonably fast (frame rate of 26/sec on Lenovo machine, with Intel core i7 generation 10 processor, using 12 GB memory and 1 TB of HDD). However, the object identification accuracy of ORB failed to classify objects like cars of different types accurately. Therefore, after considerable experimentation, we abandoned option (a) and focused on (b). Fig. 3 shows the detection of the object with a bounding box, which is of high importance in our research.

Fig. 2: Keypoint based characterization of a frame (ORB).

An accurate object recognition requires deep learning, such as a convolutional neural network (CNN) as a backbone network [24]. The complexity of CNN depends on the variety of objects to be recognized (e.g., vehicles, people, animals, bikes, etc. in the traffic context) and the discrimination desired such as vehicle type (e.g., car, minivan, small truck, etc.), color, and other features. Running such a CNN model on an ED using HDVS at 30 frames/sec requires high computing power and not practical given the constraints (explained above).

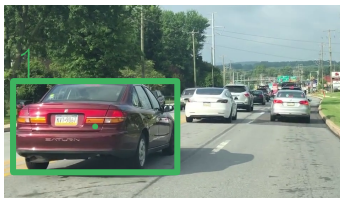


Fig. 3: Object based characterization of a frame (YLLO).

We are interested in finding frames if the scene has changed sufficiently for a new object added, to limit the number of frames sent to the EC. Thus a CNN with a smaller number of layers may be useful provided that it does not miss interesting objects or changes to the scene. In particular, we should have a very low false-negative rate for recognizing the objects in the scene, but a higher false-positive rate is acceptable. In addition, the new approach should reliably predict the new objects and/or the new position of the objects in successive frames, to skip other frames intelligently. We shall discuss in detail these issues after an introduction to the primary approach.

B. Object Detection Algorithms

A variety of algorithms have been proposed in the literature for object detection [25], [26], but most of them are not appropriate for the real-time video-stream use that we are targeting. The existing object detection algorithms can be classified into two categories: (A) a two-step process that generates region proposals at first and then classifies each proposal into different object categories, and (B) a single-step procedure that regards object detection as a regression or classification problem for the entire frame. The first category includes R-CNN, SPP-net, Fast R-CNN, Faster R-CNN, R-FCN, FPN, and Mask R-CNN. The second category comprises MultiBox, AttentionNet, G-CNN, YOLO, SSD, YOLOv2, DSSD, YOLOv3, DSOD, RetinaNet, YOLOv4 etc. For brevity, we do not discuss these; a comprehensive review of most of these is presented by the authors in [2]. Generally, the type (A) methods are inappropriate for real-time use because of their computational costs and hence will not be considered.

TABLE I: Comparison of Various Algorithms

Algorithm	**AP_{50}**	**Speed**
SSD300	43.1	43
SSD512	48.5	22
EfficientDet 512 × 512	52.2	62.5
YOLOv2 608 × 608	48.1	40
YOLOv3 416 × 416	55.3	45
YOLOv3 608 × 608	57.9	20
YOLOv4 416 × 416	62.8	38
YOLOv4 608 × 608	65.7	23

Table I list the performance of some of the algorithms known for speed measured in Frames Per Second (FPS) and accuracy measured as Average Precision (AP_{50}). For comparison, we also include Faster R-CNN, which is among the fastest algorithms of type (A). Note that the numbers here are for a certain combination of CPU and GPU and on a specific dataset (MS-COCO); therefore, *the FPS numbers especially have no meaning beyond the comparisons shown in the Table.* A modification of YOLOv2 [27], called YOLOv3 [28] improves accuracy but at the cost of much higher number of layers (106 vs 32). It is less appropriate for edge device applications because of its computational cost and memory requirements. A lighter version of YOLOv3 and YOLOv4 called, YOLOv3-tiny and YOLOv4-tiny respectively, still has a higher FPS but its accuracy is compromised [29]. The Single Shot Multi-Box Detector (SSD) [30] is rather fast and was specifically designed for mobile devices. However, unlike YOLO, SSD

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

outputs the low resolution version of the classified image which may be unsuitable for sending to EC since the analytics done by EC will usually focus on fine-grain features of the video recognition. In fact, we verified that SSD performs poorly with the license plate detection task that we assigned to the EC.

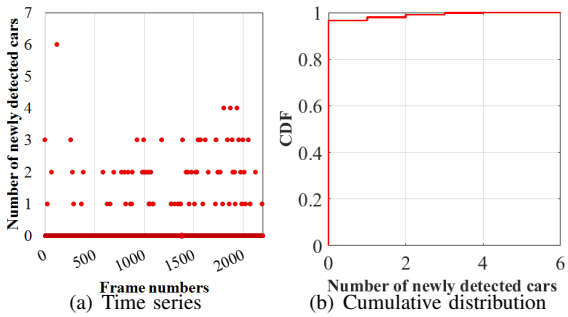
C. An Overview of YOLOv4

In this section, we provide a brief overview of the two most promising lightweight algorithms, namely YOLOv4 and SSD, mainly from the perspective of their appropriateness for our application. As stated earlier, we use the bounding box, and object detection features of such algorithms, and thus could, in principle, use any single-step algorithm.

YOLO [31] was the first effort to create a fast-real object detector with subsequent update to YOLOv2 in 2017 to increase model performance. This was followed by YOLOv3 and then YOLOv4, generally with more features and hence better accuracy but with increased processing requirements. In the following we briefly describe YOLOv4 and its simpler version YOLOv4tiny.

YOLOv4 is a more complex, deeper network architecture based on the same principles as YOLOv3 and TinyYOLO, but with improvements incorporated. The authors examined the possibilities of combining a large number of state-of-the-art improvements in object-based neural network detection with YOLO-based backbone architecture. Among many, they decided to use: (a) cutting-edge model training techniques such as mosaic-based data increase and self-adversarial training; (b) continuously differentiated activation function (Swish and Mish) and self-normalizing networks (SELU) function; (c) additional bounding box regression loss term (parametric intersection over union (IOU) based losses); (d) advanced regularization methods; (e) state-of-the-art normalization methods (such as Cross mini-batch normalization). After additional modification and tuning, the authors optimized YOLOv4's performance for both CPU- and GPU-based processing and showed superiority in both detection accuracy and time requirements on the general object detection dataset. YOLOv4 is twice as fast as EfficientDet (competitive recognition model) with comparable performance. In addition, AP (Average Precision) and FPS (Frames Per Second) increased by 10% and 12% compared to YOLOv3 [2].

Any YOLO-based detector will try to find an object in each grid cell at each central point. An image's upright rectangular division makes the grid. Each grid cell has a 32x32 pixel neighborhood, so the number of grid cells depends on the image size. The vertical and horizontal number of grid cells is equal to the image height and width divided by 32. Therefore, the number of objects to detect increases with the size of the image input. Furthermore, as the entire image context is used for training, objects from nearby grid cells can overlap, and each grid cell can be used to detect an object at different scales. The input image is downsampled with a stride size of 32 during the propagation process, so the first detection layer is used to detect large objects. Successive layers include up-sampling convolution, skip-connections, and



(a) Time series (b) Cumulative distribution

Fig. 4: New car detection per frame during a rush hour traffic.

feature concatenations. The second layer receives a feature map, which is downsampled with a stride size of 16 that can detect medium-sized objects. Finally, the third detection layer receives a downsampled feature map with a stride size 8 to detect small objects.

Table I shows the comparison of various object detection algorithms. It is seen that YOLOv4 achieves better performance with comparable or faster speed compared to predecessor architecture YOLOv3 and Tiny-YOLO.

YOLOv4 also has a simpler version, called YOLOv4-Tiny, which uses only two YOLO layers (instead of three) and has fewer anchor boxes for prediction. YOLOv4-Tiny is roughly 8X as fast at inference time and 2/3 as performance as YOLOv4. We performed experiments on some videos and observed that both detection and tracking accuracy dropped by around 30 percent. Because of this, we have decided not to use YOLOv4-tiny.

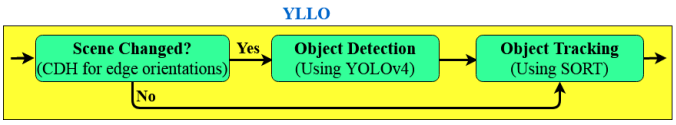


Fig. 5: Structure of YLLO Algorithm.

IV. YLLO: Efficient Object Identification in Video streams

A. Motivation for YLLO Algorithm

As stated earlier, our proposed YOLO algorithm which uses YOLOv4's basic building blocks and is designed specifically for continuous video-streams, rather than individual snapshots. Thus our focus is only exploiting redundancies in the video-stream to quickly recognize the essential frames for transmission to the edge controller.

Fig. 4 shows the distribution of the number of new cars detected in a video that was captured on the road during a typical rush hour. As seen from this figure, no new objects (vehicles) are detected in most of the frames (~97%). This phenomenon is true for many surveillance applications, including occupancy detection in an office building, or face detection within a cafeteria, etc.

B. Description of YLLO

In the traffic monitoring application, many fast-moving vehicles may enter and exit the scene frequently. YLLO uses the underlying object recognizer to identify the objects and tracks them over the successive frame. By doing object recognition,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

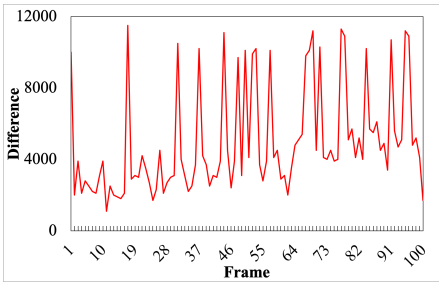


Fig. 6: Histogram difference of video frames

we not only reduce the computational load on the edge device but also reduce the bandwidth used for frame transmission to the EC. Depending on the application requirements, the untransmitted frames may either be discarded or stored locally by the edge device for some time (e.g., a few hours) and retrieved by the EC as-needed basis.

The overall structure of YLLO is shown in Fig. 5. YLLO uses the coordinates of the detected object and initializes the object tracker when it receives the first frame. The object tracker associates each detected object between successive frames using the bounding box coordinates. To perform this multiple object tracking task, the *Simple Online and Real-time Tracking (SORT)* [32] algorithm is used. YLLO also performs scene change detection by analyzing inter-frame differences. Below we explain the components of Fig. 5 separately.

Scene change detection: In most videos, a scene changes typically slowly, but occasionally abrupt changes do occur. The scene change detection considers both local (a portion of the region in a frame) and global (the whole frame) scene changes. In our scheme, we combine the edge orientation and histogram-based difference for scene change detection [33].

Edge orientation can represent the object boundaries and texture structures, thereby providing more semantic information about the object shape than the flat regions. This is based on the Laplacian of the image, defined as the second order derivative of pixel intensity values. A high Laplacian value indicates area of rapid change in density. Unfortunately, the second order derivative makes this approach susceptible to noise, which we address by first smoothing the spatial images. In particular, we combine *Gaussian Smoothing* with *Laplacian Filter* for edge detection as in [34]. These filters are utilized for identifying sudden intensity changes by locating irregular sharp edges. The Laplacian-Gaussian edge detection framework convolves the images with a mask to detect zero crossings of the calculated second derivatives. The pixels that define local maximum gradient are considered as edges by the edge detector. Next, for the scene change detection, we generate *Color Difference Histograms (CDH)* H_n for edge orientations. Such differences are primarily insensitive to moving cameras, object motion, and changes in illumination.

In our experiments, given a video frame (color image), we quantize the luminance (brightness) channel into 10 bins and the two chrominance channels (U and V components of YUV system) into 3 bins each. We also quantize the edge orientations into 18 bins, which corresponds to angle intervals of 20 degree (similar to [33]). The result is a 10×3×3+18 or 108-dimensional color feature vector. The distance metric for measuring the similarity between histograms is the L1 norm defined as:

$$d_n = \sum_{i=1}^{108} |H_n(i) - H_{n-1}(i)| \quad (1)$$

where $H_0(i) = 0$ for $0 \le i \le 108$. Finally, the scene changes S is calculated using a threshold T of the distance of histograms:

$$S = \{n \mid d_n \ge T, n = 1,2,\ldots N\} \quad (2)$$

where N is the number of frames.

Fig. 6 shows the histogram difference between successive frames. For scene change detection, we determined the appropriate threshold T at a color distance score greater than 8500.

Object detection: For object detection we have used YOLOv4, which is summarized in section III-C.

Object tracking: The SORT algorithm used for object tracking implements (a) Kalman Filter [35] that predicts the location of objects for the next frame by using successful detection from the current frame, and (b) The Hungarian algorithm [36] for optimally solving the association and matching between predictions and the bounding boxes. The detected objects between the successive frames that are successfully associated with a tracked object will inherit the existing object ID, otherwise a new unique ID is assigned to the object and the tracker is updated. If none of the detected bounding boxes can be associated with the tracked bounding box, the tracker component is deleted. Fig. 7 shows some of the frames that are sent to the EC upon detecting a new object.

C. Comparison of YLLO and YOLOv4

Dataset: We investigated the performance of our YLLO mechanism in terms of speed and accuracy. We evaluated the accuracy and performance of our model to detect vehicles on two data sets. A data set by Longyin Wen [37], which is a real-world multi-object detection and multi-object tracking benchmark, consisting of 10 hours of videos captured at 25 frames per seconds (fps), with resolution of 960×540 pixels. This data set has more than 140,000 frames with 8250 vehicles that are manually annotated. Another dataset by Fisher Yu [38] comprising of over 100,000 images with rich annotations such as image level tagging, object bounding boxes, drive-able areas, lane markings, and full-frame instance segmentation.

Ground Truth: In both the datasets, the objects like cars, trucks, bikes, persons, traffic lights etc. were annotated in the video by the authors. Therefore, we assess the performance of our model by comparing results to the ground truth.

Metrics: The primary performance metric for YLLO is detection accuracy. We use detection accuracy as the percentage of correctly predicted examples out of all predictions, formally known as $\frac{TP+TN}{TP+FP+TN+FN}$. The metric parameters are defined as:

- YLLO is said to generate a True Positive (TP) when there is an object and the model detects it with an IOU against ground truth box above the threshold. When multiple boxes detect the same object, the box with the highest IOU is considered as TP. All other boxes are considered as False Positives (FP). We set the IOU threshold to 0.5.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

(a) Frame 1 - YLLO (b) Frame 17 - YLLO (c) Frame 31 - YLLO

Fig. 7: Frames sent for EC when a new object is detected.

- YLLO is said to generate a False Negative (FN) if the object is present and the predicted box has an IOU < threshold with ground truth box.
- If the object is not in the image, yet the model detects one then the prediction is considered as False Positive (FP).

We considered an additional metric called Multiple Object Tracking Accuracy (MOTA) as defined by Bernadin. [39] to measure accuracy of YLLO. This metric denotes the characteristics of a tracking system in terms of its accuracy in recognizing object configurations like localizing the object and their ability to track objects across the frames consistently. Table II shows the comparison of the tracking performance of YOLOv4 along with our proposed YLLO scheme.

TABLE II: Comparison of Original YOLOv4 with YLLO.

	Detection Algorithm	Speed (FPS)	Detection Accuracy	MOTA Accuracy
Dataset1 [37]	YOLOv4	8-9 FPS	80.15 %	79.65 %
	YLLO	23-24 FPS	86.35 %	88.14 %
Dataset2 [38]	YOLOv4	8-9 FPS	79.62 %	79.9 %
	YLLO	23-24FPS	86.63 %	88.15 %

The original YOLOv4 model with tracker has no scene change detection introduced between frames, so all frames are considered for object detection. The YLLO model with the tracker is more accurate than the YOLOv4 model with tracker by introducing a scene change detection between successive frames before performing classification that reduces the total number of false positives. The experiments for calculating the accuracy tests were run on a Lenovo 90HV0005US machine, with Intel core i7 (generation 10) processor, using 32 GB memory and 1 TB of HDD. These experiments on a somewhat dated desktop processor explain how much faster YLLO can run compared with YOLOv4; the numbers show the performance when using Intel Neural compute stick2 (NCS2) [40]. The Intel® NCS2 is built on the Intel® Movidius™ Myriad™ X VPU featuring 16 programmable shave cores and a dedicated neural compute engine for accelerating deep neural network inferences. The actual numbers will be reduced by 50 percent when run on the same machine configuration without NCS2. On both datasets, YLLO achieves ~3-4× speed as compared to YOLOv4. In particular, YLLO achieves a frame rate of ~23-24 FPS which is near-real time, without compromising the accuracy.

V. BATS: Bandwidth Adaptive Transmissions of Streams

Next, we discuss the second component of our scheme, namely, an intelligent transmission of the video stream frames that accounts for the available transmission bandwidth. When multiple cameras are covering the same scene, it takes advantage of the redundancy in deciding which frames to transmit. YLLO selectively transmits frames whenever the cameras detect a new object; however, the available bandwidth may not allow transmission of all such frames. For this, we exploit the overlapping coverage areas of multiple cameras to transmit selective frames, while maximizing the level of coverage.

In a multi-camera system, there may be a significant amount of coverage area overlaps in between the cameras. For example, a pair of cameras at both sides of a highway will capture approximately the same number of cars passing the highway. Similarly, two cameras on the same pole on different angles may have overlapping coverage. Thus, if a user query asks to capture all the cars passing the highway within a time interval, then sending frames from one of these cameras may be sufficient. A set of such cameras monitoring a single geographic area with a significant amount of overlapping views form a cluster. BATS provide an *intra-cluster* algorithm, which will only transmit the selected frames after eliminating the redundant/overlapping ones when the wireless channel has limited channel capacity.

A. Determining the primary cameras

We consider a set of cameras as *primary cameras* whose frames are essential for sufficient coverage purposes. For simplicity, we divide the region into several small grids. Next, we mark the grids that are covered by these cameras, and the associated weights based on the quality of coverage of the grids by the corresponding cameras. Assume that γ_g is a binary variable if grid-g is covered by at least one camera and zero otherwise. Assume that C_{cg} is a binary variable which is one if camera-c covers grid-g, and the coverage quality of camera-c corresponding to grid point g is q_{cg}. x_c is a binary decision variable if camera-c is chosen as a primary camera and zero otherwise, whereas w_c is its weight. Thus to ensure that none of the viewpoints are missed, we need to at least accumulate camera-frames such that the amount of coverage (i.e the number of grids covered) is maximized, whereas the number of cameras in the primary set is minimized, while improving the overall quality of coverage. This results in the following optimization problem:

$$\text{Maximize} \qquad \sum_g \gamma_g - \alpha \sum_c x_c / w_c + \beta \sum_g Q_g \tag{3}$$

$$\text{subject to} \qquad \mathbf{C_1}: \gamma_g \geq \zeta \sum_c C_{cg} x_c \tag{4}$$

$$\mathbf{C_2}: Q_g = \max_c \; q_{cg} x_c \tag{5}$$

where α, β, ζ are small numbers. A small α ensures that for two solutions with same coverage, the solution with fewer cameras and higher weights is chosen. When all these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

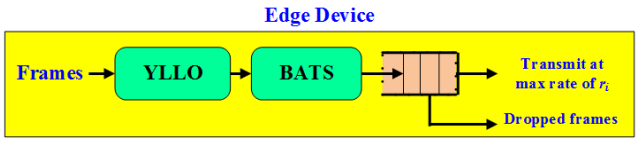


Fig. 8: Illustration of an edge device operations flow.

are identical, then the solution with better overall quality is chosen, which is ensured by the variable $\beta << \alpha$.

In constraint $\mathbf{C_1}$, the parameter ζ ensures that γ_g is one if grid-g is covered by at least one camera. Because γ_g is binary, it can be at most one irrespective of the number of cameras covering it. The constraints $\mathbf{C_2}$ records the variable Q_g, which is the quality of coverage of a grid-g by at least one of the primary cameras.

Theorem 1: The primary camera selection problem is NP-complete.

Proof 1: We prove the NP-completeness of the primary camera selection problem by considering its special case, where all the grids are covered by at least one camera, and the quality of coverage of all the cameras are identical. This special case can be seen as an instance of the set cover problem [41] which is known to be NP-complete.

As the primary camera selection problem is NP-complete, we propose a simple heuristic that flows as follows. We first mark all the grids as *uncovered*. At first, the camera with maximum coverage (calculated in terms of the total number of uncovered grids) is chosen to be a primary camera. If multiple cameras have equal coverage, then ties are broken by selecting the camera with (a) higher weight, and (b) better quality of coverage. After choosing the first primary camera, the grids that are covered by the camera is marked as *covered*. Then the same process is repeated for choosing the next set of cameras that maximizes the number of uncovered grids. The iteration stops when adding more cameras does not result in exploring any more uncovered grids. In the special case where all cameras are of identical weights, and all the grids are covered by at least one camera, the proposed heuristic becomes the approximation algorithm proposed in [41] that achieves an approximation ratio of $H(n)$, where n is the number of grids, and $H(n)$ is the n-th harmonic mean.

B. Collaborative rate adaptation in multi-camera system

We now formulate an optimization problem to maximize the amount of information availabile from a cluster at the controller end, with the constraints that (a) the channel capacity is not overshot, and (b) at the same time the controller can receive frames from the cameras at a certain minimum rate. Let us assume that there are C cameras, and the channel capacity is assumed to be $\mathbb{C}$. The controller can dynamically measure the channel capacity by periodically measuring the packet transmission rate and their corresponding losses. Thus if the time averaged packet loss rate is L and the physical-layer bit rate is R, then $\mathbb{C} = R(1-L)$ [14]

Considering these factors, the **weighted proportional fairness** within a cluster can be achieved by modeling the utility function of node i as $U_i(r_i) = \alpha_i.\log(r_i)$, where α_i is the normalized weight of the window i. The weights may be assigned by the controller based on their locations and area of coverage, orientations and can be time-dependent. Our objective is to maximize the overall utility of the cluster, i.e. $\sum_{i=1}^{N} U_i(r_i)$, after satisfying the required constraints. We also assume that each frame consumes F bps, thus to avoid the channel to be overloaded, i.e. $\sum_{i=1}^{N} r_i.F \leq \mathbb{C}$, or $\sum_{i=1}^{N} r_i \leq \frac{\mathbb{C}}{F} = \mathbb{M}$. Intuitively we can think that the cameras in a cluster work as a *single virtual camera* that reports at a maximum rate of $\mathbb{M}$ frames/interval. $\mathbb{M}$ is a controlling parameter that controls the overall sampling rate of the coalition, i.e. if the controller wants to receive the frames more frequently, it increases $\mathbb{M}$ and vice versa.

Let us denote the set of primary cameras after solving problem (5) is S. Thus each of these cameras need to transmit at a rate of $\mathcal{R}/|S| = \mathbb{S}$ to ensure that any point is tracked at least at a rate of $\mathcal{R}$, i.e. $r_i \geq \mathbb{S}, \forall i \in S$. For all the other cameras can transmit at least at a rate of some minimum rate of R_m. We assume that all the cameras need to transmit at least at a rate of R_m to ensure that the controller can do some analytic in regards to the surveillance. Thus the overall optimization problem becomes:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{N} U_i(r_i) \\ \text{subject to} \quad & \mathbf{C_1}: \sum_{i=1}^{N} r_i \leq \mathbb{M}, \\ & \mathbf{C_2}: r_i \geq \mathbb{S}, \forall i \in S, \quad \mathbf{C_3}: r_i \geq R_m, \forall i \in \bar{S} \end{aligned} \tag{6}$$

Algorithm 1 Collaborative Multi-Camera Rate Allocation scheme

1: INPUT : Maximum sampling rate $\mathbb{R}_i$, utility weights α_i and $\mathbb{M}$.
2: OUTPUT : Sampling rates r_i $\forall i \in \{1,2,...,N\}$.
3: $r_i = \mathbb{S}$, $\forall i \in S$;
4: $r_i = R_m$, $\forall i \in \bar{S}$;
5: $Tot = \sum_i r_i$;
6: $\mathbb{M} = \mathbb{M} - Tot$; ‘
7: **for** each node $i = \{1,2,...,N\}$ **do**
8: $\quad r_i = r_i + \frac{\alpha_i}{\sum_{i \in U} \alpha_i} \mathbb{M}$;
9: **end for**
10: return r_i $\forall i$;

We propose an algorithm to solve this problem, which is presented in the Algorithm 1, and is addressed centrally at the edge controller. Algorithm 1 first assigns the minimum required frame transmission rate for each camera (line 3-4), and then the total rate is calculated (line 5). The remaining $\mathbb{M}$ is calculated in line 6. After this step the constraints $\mathbf{C_2} - \mathbf{C_3}$ are satisfied.

We then construct the modified optimization problem after eliminating constraints $\mathbf{C_2} - \mathbf{C_3}$ as follows:

$$\text{Maximize} \sum_{i=1}^{N} U_i(r_i) \text{ subject to } \mathbf{C_1}: \sum_{i=1}^{N} r_i \leq \mathbb{M} \tag{7}$$

As log is a concave function, the above problem becomes a convex optimization problem. Thus by solving the KKT conditions of problem (7), we obtain $r_i = \frac{\alpha_i}{\sum_{i \in U} \alpha_i} \mathbb{M}$.

Thus in line 7-9 of Algorithm 1, the remaining $\mathbb{M}$ is then shared proportionately among all the cameras, i.e $r_i =$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

Fig. 9: Example of a frame where ALPR cannot identify one of the license plate **(i.e. silver car)** due to lack of clarity.

$r_i + \frac{\alpha_i}{\sum_{i \in U} \alpha_i} \mathbb{M}$. The calculated sampling rates then sent to the corresponding edge devices.

C. Queue Management

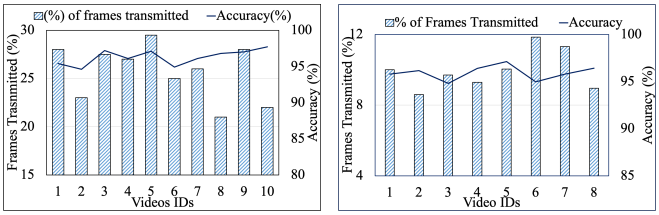
Fig. 8 shows the overall block diagram of the set of steps executed in an edge device. The selected frames sent by the YLLO are stored in an outgoing queue, whose maximum transmission rate is determined by Algorithm 1. Thus, after getting the maximum allowable transmission rate from the controller, the individual edge devices set this as the maximum service rate of their queues. Typically any queue management policy can be adopted; however, for our experiments we have adopted a simple packet dropping policy. When the queue is filled up by a certain threshold Γ, then we drop ξ percentage of equally spaced packets from the queue. We believe that dropping the equally spaced packets is suitable for the streaming applications, rather than dropping some packets in succession, which may hurt the object identification accuracy. For our experiments, Γ and ξ are assumed to be 65% and 20% respectively.

VI. Experimental Evaluation

In this section, we evaluate the performance of our scheme-with the real video traces. The effectiveness of our schemeis measured using these metrics: (a) object identification accuracy, (b) the fraction of total packets transmitted and, (c) energy consumed. We pass the frames received by EC to the license plate detection software (ALPR) and find out the percentage of vehicle nameplates identified correctly for all the objects that are classified by YLLO. Fig. 9 shows an example of a transmitted frame to the EC. Here, one of the license plates is not identified due to clarity. Unless otherwise mentioned, the frame resolution is kept as 720p (1280 × 720) which is a recommended resolution level specified by the ALPR software.

A. Experimental Setup

We have an experimental setup with three machines, which consists of an EC and two EDs with machine configurations supporting Intel Neural Compute Stick2. The ED connected to the camera transmits the relevant frames to the EC whenever it detects a new object as described in section IV. The EC then passes these frames to the ALPR software to record the vehicle number-plates. The EC machine is (6th generation) 2.40GHZ Intel i5-6300U CPU with 32 GB memory and 1TB SSD with Linux Ubuntu 18.04, and among the two EDs, ED1



(a) Video from car dashboard (b) Video from static cameras

Fig. 10: Vehicle identification accuracy and % of frame transmissions for independent transmissions.

(Machine1) sports a recent (8th generation) 1.80GHz Intel i7 CPU, 16GB memory, 512GB SSD with Windows-10 OS. ED2 (Machine2) is (8th generation) 3.20GHz Intel i7 CPU, 32GB memory, 1TB HDD with Linux Ubuntu 18.04. All the algorithms are written in Python3.

B. Performance of transmitting frames independently

We first evaluate the performance when each camera transmits the frames independently to the EC. We collected video traces from four different environments around the Philadelphia metropolitan area. We have recorded a total of 10 video traces with ~45,000 image frames/video for moving camera and 8 video traces for static camera.

For videos captured using a moving camera (see Fig. 10(a)), the object identification accuracy is more than 95% for all video, and the frame transmitted to EC is around ~20-30%. The inaccuracy of 5% arises mainly because new frames are sent to the EC whenever the cameras detect a new object. This selective transmission can cause some frames to be incomplete or obscure views, thereby hindering the ALPR software incorrectly identifying the nameplate. For video from a static source (see Fig. 10(b)), the scheme achieves the same high accuracy level of around 95% by transmitting only ~5-11% of its captured frames. In both cases, the lower frame rate *directly equates to energy saving* in both the ED and EC.

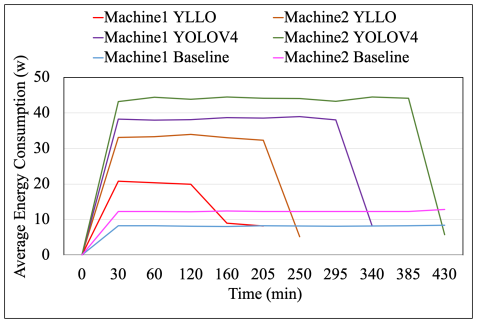
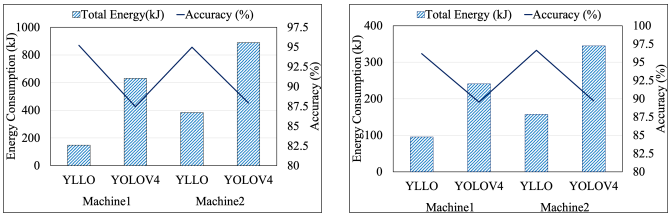


Fig. 11: Power consumption for moving camera videos.

C. Energy Consumption Analysis

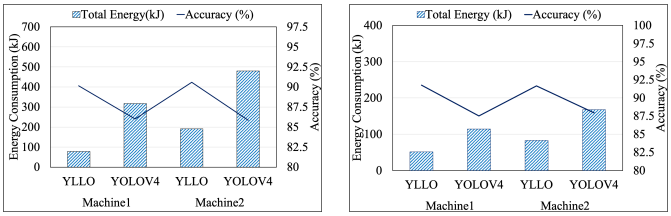
Energy consumption is a primary concern in video surveillance since video cameras collect a vast amount of data that must be transmitted to the edge controller over the wireless link. A significant part of the energy consumption in a smart camera is due to data transmission and the computation energy of local video frame processing. As part of **additional energy evaluation** of the proposed YLLO algorithm, tests were

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing



(a) Dashboard video (b) Static Camera Capture

Fig. 12: Energy & accuracy of YLLO vs YOLOv4 at 30 FPS.



(a) Dashboard video (b) Static Camera Capture

Fig. 13: Energy & Accuracy of YLLO vs YOLOv4 at 15FPS.

conducted with multiple HD video streams on the EDs with specifications listed above. We compared the test results with energy consumed by YOLOv4 at two frame rates of 30 and 15 FPS. In [12], authors have found that the frame sampling rate independently impacts accuracy. We used a single-phase wattmeter to record the average AC power consumption for both machines (with batteries removed). To minimize influence of other processes, all unnecessary background jobs and services were terminated. The power measurements reported exclude the idle power consumption when the machines are not running our scheme. It also excludes the power consumption of the power adapter itself.

1) Energy Consumption Analysis of Video Traces from Moving Camera: Fig. 11 shows the comparison of average power vs. time-taken by our scheme vs YOLOv4. It can be noted that our scheme in both the machines takes less than ~50% the time taken by original YOLOv4. This reduction is mainly due to intelligently avoiding unnecessary frame processing. Yet, the identification accuracy remains over 95% for all video traces, much better than the YOLOv4 accuracy. The reason for the inaccuracy in YOLOv4 is that sending redundant frames to the EC increase the chance of identifying false positives in license plate recognition. As a result, there is a reduction incorrectly identified license plates compared to YLLO. On the other hand, YLLO keeps track of objects with specific object IDs and does not send a frame if the object IDs stay the same for two consecutive frames.

Fig. 12 shows the energy consumption (in Joules on the left axis) and accuracy (as a percentage on the right axis) of YLLO for 30 FPS over different video traces. The comparison is shown for both Machine1 and Machine2. Also, while Fig. 12(a) is for videos taken from car dashboard on busy streets, Fig. 12(b) is for videos taken from static camera in a residential street. It is seen that *YLLO consumes only about 40% of the energy of YOLOv4, and yet can provide substantially higher accuracy.* The accuracies are 95% (YLLO) vs. 87.5% (YOLOv4) in Fig. 12(a) and increase somewhat (96% YLLO vs. 89.5% YOLOv4) for both algorithms with statically placed cameras in Fig. 12(b). This is attributed to objects (e.g., buildings, parked cars, trees, etc.) remain static from frame to frame and are easily discounted in the processing.

A similar situation is seen in Fig. 13 where we eliminate alternate frames and thereby run the videos at 15 FPS. Both algorithms suffer accuracy loss in these cases, as expected. However, YLLO still consumes only about 40% of the energy of YOLOv4 and still manages to beat YOLOv4 in accuracy by 5 percentage points (90% vs. about 85%). At both frame rates, machine 1 has lower power consumption due to energy efficient CPU and DRAM, and faster SSD (newer model).

The number of network calls to EC (for license plate reading) is another contributor to reduced energy consumption in YLLO. Fig. 14 shows the total network calls by YLLO (to EC) is reduced by 75% because of selective frame transmission (as explained earlier). In contrast, the number of network calls made in YOLOv4 is proportional to the number of frames recorded in the video frames. This reduction in network calls *directly translates* into lower EC computing power requirements and a decrease in network BW usage for communication between ED and EC. *That is, using our schemewill cut down the bandwidth needs to about 1/4th of the original.* This, in turn, can reduce the energy consumption of the device's radio, although the details very much depend on the communications technology used and energy saving aspects such as batching of transmissions, use of low power modes for the transmitter and receiver, etc.

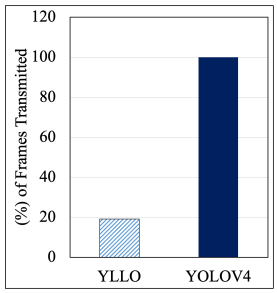


Fig. 14: Percentage of frames transmitted to EC.

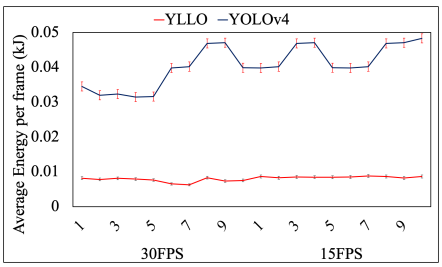


Fig. 15: Mean and variance of per frame energy.

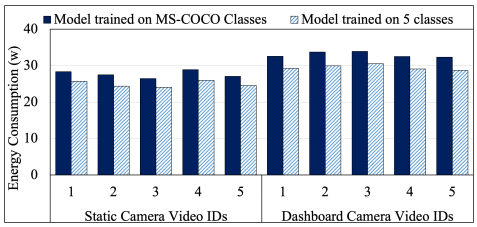
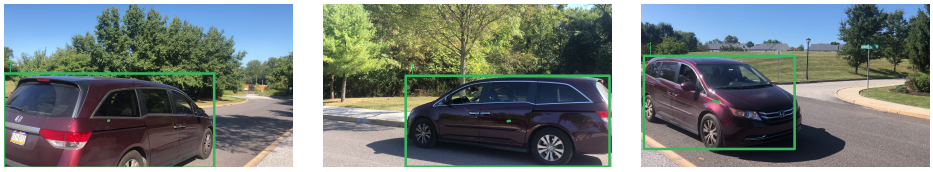


Fig. 16: Variation of energy consumption on different trained models.

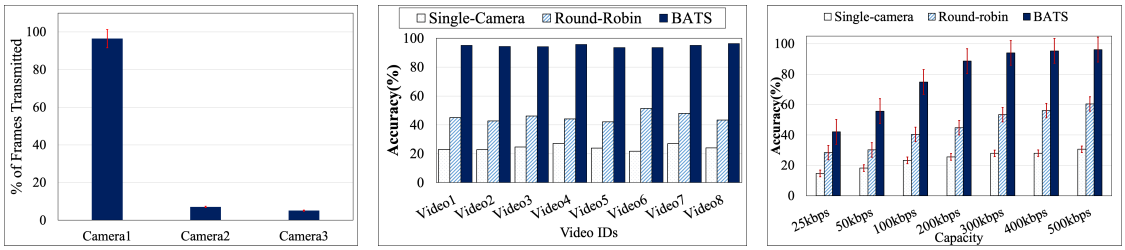
2) Per-Frame Energy Consumption: To better understand the energy results, we further analyzed the energy consumption across various video streams and compared the per-frame energy consumption between YOLOv4 and YLLO. Fig. 15 shows this behavior across the 10 dashboard videos, both with

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing



(a) Camera1 (b) Camera2 (c) Camera3

Fig. 17: (a)-(c) Videos captured by 3 cameras with significant overlap in coverage area.



(a) Frac of frames (b) Accuracy vs. Video ID (c) Accuracy vs. Channel capacity

Fig. 18: (a) Fraction of frames transmitted by different cameras, (b) Accuracy and (c) channel capacities of BATS vs. simpler mechanisms.

30 FPS and 15 FPS (with the line in the middle separating the 30 FPS case from 15 FPS). The graph shows the mean energy in KJ and the vertical bars show the variation across frames of each video. It is seen that the variation in the mean energy across videos is rather limited, and the variation across frames in a video is also quite small. Thus, it is meaningful to speak of per-frame energy consumption in that the total energy consumption for a video of this type can be obtained by simply considering the total number of frames. We believe that this also applies to the frame processing in the EC, although we have not conducted detailed experiments directly.

3) Energy Consumption Analysis of Trained Models: To understand the variation in energy consumption of YOLOv4 as a function of number of objects that it can recognize, we retrained it to detect five types of objects, namely cars, trucks, bicycles, busses and people. (The original YOLOv4 was trained on 80 categories of object classes [42].) We used the pretrained weights of YOLOv4, built a new training dataset along with required modifications in the configuration file. The training was conducted on 2080 Ti GPU. We evaluated the energy consumption with a few videos of both the static camera and the dashboard cameras in both models. Fig. 16 shows the results. It can be seen that the average difference in energy consumption is around 10-11%. While this is a significant energy saving, it may not be worthwhile. Note that many types of objects can occasionally appear on the road (e.g., discarded boxes, animals, rocks, etc.). We do not want to miss such unusual objects by training the algorithm on only the everyday objects (e.g., cars, pedestrians, etc.)

D. Performance of Collaborative Frame Transmissions

We evaluate the performance in a multi-camera network where there is a significant overlap of coverage areas between the cameras. The purpose of this experiment is to examine to what extent BATS can maintain the accuracy as wireless capacity becomes scare, by exploiting the overlaps in between the camera coverage.

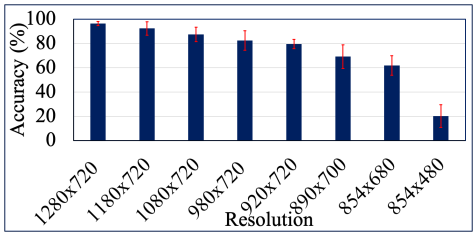


Fig. 19: Object identification accuracy with different frame resolutions.

We recorded video traces from 3 cameras deployed in our university, with a significant amount of coverage area overlap. Fig. 17 shows the views of these 3 cameras showing significant overlaps in their coverage areas. We consider camera-1 as primary camera whose frames are essential for identifying the vehicle nameplates. These cameras log video traces synchronously for around an hour with ~36,000 image frames.

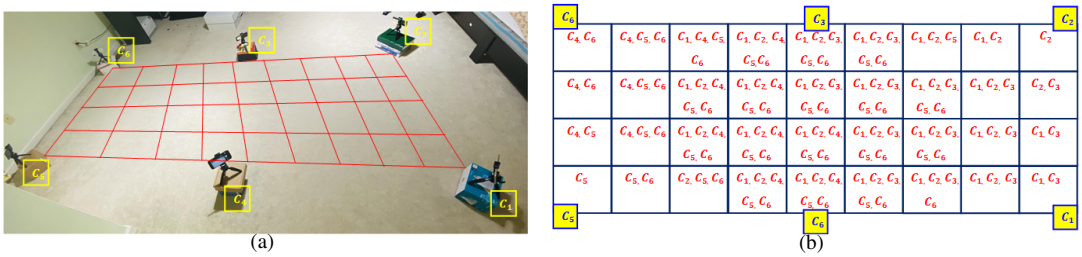
We compare the accuracy of BATS with two approaches: (a) a round-robin approach that cycles through all the cameras within a cluster in a round-robin manner to upload the frames and (b) a single-camera approach that arbitrarily selects a single camera to upload the frames. The average frame size from the cameras is around ~2.5 MB. We are using the H.264 codec standard for compression which provides approximately 20× compression.

E. Impact of Frame Resolution on Detection Accuracy

Fig. 18(a) shows the fraction of frames transmitted by different cameras, which confirms that the primary camera (i.e. camera 1) transmits more than 95% of the frames as compared to the secondary cameras. In Fig. 18(b),we assume that the channel capacity to be equal to 300 kbps. Across all videos (i.e video 1 to video 8) in Fig. 18(b) clearly shows the object detection accuracy of BATS increases by ~4× compared to the single-camera and ~2× compared to round-robin schemes.

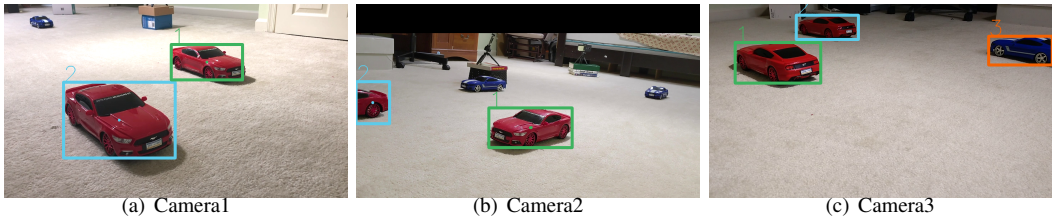
Next, using different channel capacities (x-axis), we show in Fig. 18(c) the average number-plate identification accuracy (y-axis) of all the videos. These figures show the effectiveness of the collaborative frame transmission policy of BATS as

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2021.3064245, IEEE Transactions on Sustainable Computing

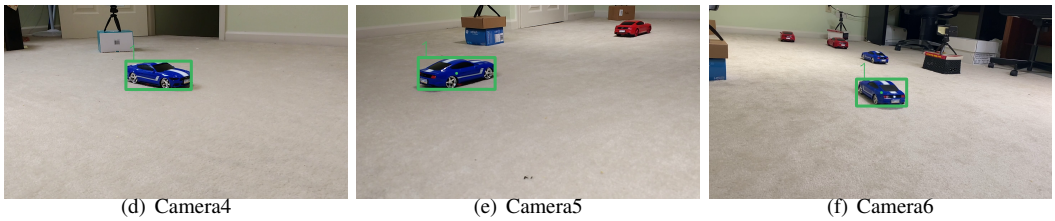


(a) (b)

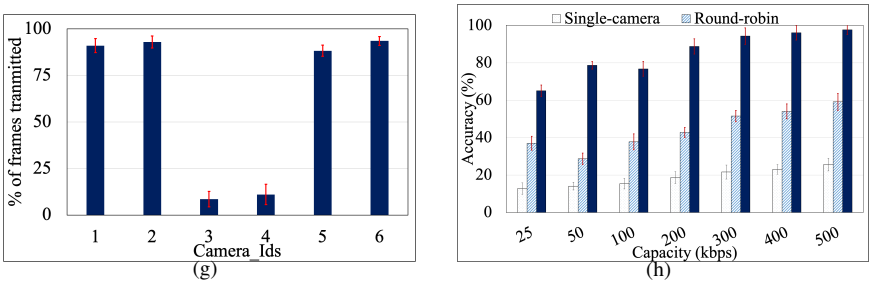
Fig. 20: (a) The experimental setup in an indoor environment with six cameras, and their (b) coverage areas.



(a) Camera1 (b) Camera2 (c) Camera3



(d) Camera4 (e) Camera5 (f) Camera6



(g) (h)

Fig. 21: (a)-(f) Videos captured from 6 cameras showing effect of coverage area overlaps. (g) Percentage of frames transmitted to EC. (h) Accuracy of our schemeagainst simpler mechanisms.

opposed to the independent transmissions. In particular, the accuracy improves significantly as the channel capacity creases from 25 kbps to 300 kbps, but saturates after that.

We evaluate the object detection accuracy of our schemefor different frame resolutions. In Fig. 19, we vary the frame resolution from 1280 × 720 to 854 × 480, which results in an accuracy degradation of 97% to 20%. This is because the performance of the ALPR software degrades significantly with lower resolution. In particular, it goes below 80% with a resolution of less than 920 × 720. However, frame resolution can be a tuning parameter that the EC can exploit in applications where it only needs to monitor the number of objects passing, rather than doing some deep analytics on the frames like the number-plate identification.

F. Performance in Multi-Camera Environment

As suitable datasets in a multi-camera environment with partial area overlaps are relatively sparse, we experimented with toy cars setup in an indoor environment. To imitate the scenario of partial area coverage, we place six cameras as shown in Fig. 20(a) as if they are placed by the two sides of a road. We printed the license plates and stick them on both sides of these cars (putting license plates on front and rear end is a requirement in 31 states in USA). This ensured that their license plates could be read by cameras that can see the cars from possible sides.

We divide the coverage areas into 9×4 grids, and record the area of coverage by these cameras in Fig. 20(b). From Fig. 20(b) we can determine (C_1,C_2,C_5,C_6) as primary cameras and (C_3,C_4) as secondary cameras. Fig. 21(a)-(f) show the snapshot of the videos captured by these six cameras at any time instance, which clearly shows the effects of coverage area overlaps that can be exploited for suppressing redundancy.

Fig. 21(g) shows the fraction of frames transmitted by different cameras, where the channel capacity is assumed to be 200 kbps. It is evident that the primary cameras transmit ~22% of all the frames whereas the secondary ones on transmit a tiny fraction of ~5-6%. This confirms the ability of BATS in redundancy suppression for better channel utilization.

Fig. 21(h) shows the accuracy of identifying the license-

plates of BATS as opposed to a single camera and round-robin schemes, with varying channel capacities. BATS improves the license-plate detection accuracy by ∼4 times as compared to the single camera scenario. As compared to the round-robin scenario, BATS enhance the accuracy by ∼65-67%. The accuracy improves till the channel capacity is 300 kbps and saturates beyond that.

This reduction in the number of frames can also provide corresponding energy savings in the radio transmitters if suitable power saving mechanisms are available and engaged. For example, IEEE 802.11ah includes power saving mode using the Target Wake Time (TWT) mechanism [43], [44] that permits the transmitter to enter into a sleep state occasionally. During their sleep state, the incoming packets can be buffered and transmitted when the radios wake up. There is an energy-latency tradeoff here which needs to be adjusted depending on the application requirements.

VII. Conclusions

In this paper, we have developed a lightweight object detection technique called YLLO for video streams that significantly reduces the number of frame transmissions to the edge controllers (ECs) while still maintaining the object identification accuracy of more than 95%. In addition, our BATS scheme can further reduce the frame rate by exploiting coverage overlaps among multiple cameras under the control of the same EC. Our notion of overlap between cameras is rather simplistic and static; it is possible to have more semantically significant notion by exploiting techniques like homography transformation of overlapping camera views, inter-camera feature correspondences, etc.

Acknowledgements

This research was support by NSF grant CNS-1527346. The authors would like to acknowledge the help of Dr. Sanjeev Sondur in doing energy consumption measurements.

References

[1] A. Jolfaei *et al.*, "Data security in multiparty edge computing environments," *GOMACTECH*, 2019.
[2] A. Bochkovskiy *et al.*, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.
[3] "Opensource Automatic License Plate Recognition," http://www.openalpr.com/cloud-api.html.
[4] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[5] J. Huang *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *IEEE CVPR*, 2017, pp. 7310–7311.
[6] K. Ha *et al.*, "Towards wearable cognitive assistance," in *ACM Mobisys*, 2014, pp. 68–81.
[7] S. Han *et al.*, "Glimpsedata: Towards continuous vision-based personal analytics," in *ACM WPA*, 2014, pp. 31–36.
[8] T. Wang *et al.*, "Walksafe: a pedestrian safety app for mobile phone users who walk and talk while crossing roads," in *ACM HotMobile*, 2012.
[9] C.-W. You *et al.*, "Carsafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones," in *ACM Mobisys*, 2013, pp. 13–26.
[10] G. Ananthanarayanan *et al.*, "Real-time video analytics: The killer app for edge computing," *IEEE Computer*, vol. 50, no. 10, pp. 58–67, 2017.
[11] H. Zhang *et al.*, "Live video analytics at scale with approximation and delay-tolerance," in *USENIX NSDI*, 2017, pp. 377–392.
[12] J. Jiang *et al.*, "Chameleon: scalable adaptation of video analytics," in *ACM SIGCOMM*, 2018, pp. 253–266.
[13] C.-C. Hung *et al.*, "Videoedge: Processing camera streams using hierarchical clusters," in *IEEE/ACM SEC*, 2018, pp. 115–131.
[14] T. Zhang *et al.*, "The design and implementation of a wireless video surveillance system," in *ACM MobiCom*, 2015, pp. 426–438.
[15] S. Yi *et al.*, "LAVEA: latency-aware video analytics on edge computing platform," in *ACM/IEEE SEC*, 2017, pp. 15:1–15:13.
[16] S. Jain *et al.*, "Scaling video analytics systems to large camera deployments," in *HotMobile*, 2019, pp. 9–14.
[17] A. Misra *et al.*, "Dependable machine intelligence at the tactical edge," vol. 11006, pp. 64 – 77, 2019.
[18] S. Y. Nikouei *et al.*, "Smart surveillance as an edge network service: From harr-cascade, SVM to a lightweight CNN," in *IEEE CIC*, 2018, pp. 256–265.
[19] J. Wang *et al.*, "Elastic urban video surveillance system using edge computing," in *SmartIoT*, 2017, pp. 7:1–7:6.
[20] H. Sun *et al.*, "*VU*: video usefulness and its application in large-scale video surveillance systems: an early experience," in *SmartIoT*, 2017, pp. 6:1–6:6.
[21] J. Barthelemy *et al.*, "Edge-computing video analytics for real-time traffic monitoring in a smart city," *Sensors*, vol. 19, no. 9, p. 2048, 2019.
[22] E. Salahat *et al.*, "Recent advances in features extraction and description algorithms: A comprehensive survey," in *IEEE ICIT*, 2017, pp. 1059–1063.
[23] E. Karami *et al.*, "Image matching using sift, surf, brief and orb: performance comparison for distorted images," *arXiv preprint arXiv:1710.02726*, 2017.
[24] A. B. Amjoud *et al.*, "Convolutional neural networks backbones for object detection," in *ICISP*, 2020, pp. 282–289.
[25] J. Han *et al.*, "Advanced deep-learning techniques for salient and category-specific object detection: A survey," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 84–100, Jan 2018.
[26] L. Weng, "Object detection part 4: Fast detection models," https://lilianweng.github.io/lil-log/2018/12/27/object-detection-part-4.html, Dec 2018.
[27] J. Redmon *et al.*, "Yolo9000: better, faster, stronger," in *IEEE CVPR*, 2017, pp. 7263–7271.
[28] J. Redmon *et al.*, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[29] Y. Cai *et al.*, "Yolobile: Real-time object detection on mobile devices via compression-compilation co-design," *arXiv preprint arXiv:2009.05697*, 2020.
[30] W. Liu *et al.*, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.
[31] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.
[32] A. Bewley *et al.*, "Simple online and realtime tracking," in *IEEE ICIP*, 2016, pp. 3464–3468.
[33] G.-H. Liu *et al.*, "Content-based image retrieval using color difference histogram," *Pattern recognition*, vol. 46, no. 1, pp. 188–198, 2013.
[34] S. Paris *et al.*, "Local laplacian filters: Edge-aware image processing with a laplacian pyramid." *ACM Trans. Graph.*, vol. 30, no. 4, p. 68, 2011.
[35] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.
[36] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.
[37] L. Wen *et al.*, "Ua-detrac: A new benchmark and protocol for multi-object detection and tracking," *arXiv CoRR*, vol. abs/1511.04136, 2015.
[38] F. Yu *et al.*, "Bdd100k: A diverse driving video database with scalable annotation tooling," *arXiv preprint arXiv:1805.04687*, 2018.
[39] K. Bernardin *et al.*, "Multiple object tracking performance metrics and evaluation in a smart room environment," in *IEEE International Workshop on Visual Surveillance*, 2006.
[40] "Intel Neural Compute Stick2," https://ark.intel.com/content/www/us/en/ark/products/140109/intel-neural-compute-stick-2.html.
[41] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, pp. 233–235, 1979.
[42] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *ECCV*, 2014, pp. 740–755.
[43] E. M. Khorov *et al.*, "A tutorial on IEEE 802.11ax high efficiency wlans," *IEEE Commun. Surv. Tutorials*, vol. 21, no. 1, pp. 197–216, 2019.
[44] E. M. Khorov *et al.*, "A survey on IEEE 802.11ah: An enabling networking technology for smart cities," *Comput. Commun.*, vol. 58, pp. 53–69, 2015.