Bandwidth-Efficient Cross-Camera Video Analytics

Hyunwoo Jung
20183573
School of Computing
KAIST

Dongmin Kim
20184220
School of Computing
KAIST

Nick Heppert
20186505
School of Computing
KAIST

Taeksoo Kim
20193162
School of Computing
KAIST

I. Revised problem statement

Camera network system has become common these days. The advance in image recognition technology like deep learning and the decrease in the cost of IoT devices have lead the pervasiveness of the camera network system. One example of such camera network system is Amazon Go where there is no cashier but customers can do shopping without checkout. Customers just grab what they want to buy and leave the store then payment is automatically done because the video system tracks down who takes out what. Another example is video surveillance system in the metropolitan city like London and Beijing where a million of cameras have been deployed. However, it is very challenging to apply the technology used in Amazon Go to such large scale video surveillance system because it is expensive to have cameras equipped with compute devices or to wire all cameras to cloud systems that have powerful compute devices such as GPUs.

Recently, Jian et al. [6] have proposed an algorithm for cross-camera video analytics called RexCam. The authors evaluate RexCam on the DukeMTMC dataset [5], which contains footage from 8 cameras located on the Duke University campus. The authors show the high degree of spatial and temporal locality of the footage. That is, the traffic between camera A and camera B is more likely than the one between camera A and camera C if camera A is located close to camera B but far away from camera C. Thus, they propose algorithms utilizing spatial and temporal locality to reduce compute cost and delay.

However, RexCam assumes a somewhat unrealistic environment. Some of the assumptions are that cameras are high-definition as well as that they are connected to each other and to a central powerful computation unit (most likely a cloud) over a high-speed local area network. RexCam also assumes that bandwidth between cameras and to the cloud is sufficient so that it can transmit video data in real-time with 60 frames per second. According to Zhang [10], however, existing network infrastructure is insufficient to accommodate many simultaneous high-definition video feeds such as Dropcam [1]. That is, the traffic between cameras and the centralized cloud system is bounded by the bandwidth of Wi-Fi, which results in the lack of scalability. In section II we will do an depth-in analysis of bottlenecks in the network. For this end, we propose resource-efficient distributed system for large scale camera network in Section III.

Source	Bandwidth [Mbps]
www.cctvcalculator.net	4.5
www.arxys.com	14.25
www.security.us.panasonic.com	5.7
www.aventurasecurity.com	13.82
www.supercircuits.com	9.4
www.camscan.ca	6
www.stardot.com	12
www.ivcco.com	9.977
Average for 30 FPS	9.4559

TABLE I
All estimations were made for streaming with highest h.264 codec settings for Full HD frames (1920 x 1080 pixels) at 30 FPS as the data in Duke Dataset [5] is recorded at 60 FPS one would need to double the average in order to get the requiuerements for the Duke Dataset which results in 18.91 Mbps.

	Coverage	Theoretical Throughput (Mbps)	Practical Throughput (Mbps)	RexCam's Requirement (Mbps)
802.11n	97.7%	288.8	20.473	18.91

TABLE II
Coverage and theoretical/practical throughput of 802.11n. Data are adopted from [2], [3], [9]. Required bandwidth calculation for one camera for the Duke MTC dataset [5] can be seen in Table I.

II. Design Considerations

A. Bandwidth of wireless network

Wired cameras are inefficient in terms of cost and effort due to their characteristics. Thus, many studies on video surveillance system have been done by assuming wireless network as in [10]. In this paper, we also assume that all cameras in our system communicate with a Wi-Fi network. If cameras send their video streams to the cloud through a Wi-Fi network, RexCam's approach is not realistic because of the bandwidth issue of Wi-Fi.

For example, DukeMTMC dataset [5], which is used by RexCam for experiments, consists of 1080p video recorded at 60 frames per second (FPS) from 8 cameras. This means that each camera sends approximately 18.91 Mbps of video data, which is fairly large, to the cloud. A calculation of this can be seen in Table I. Next, we will see if Wi-Fi Access Points (AP) receiving this amount of data can handle it. There are various versions of the Wi-Fi protocol and their bandwidths are different. Recent studies

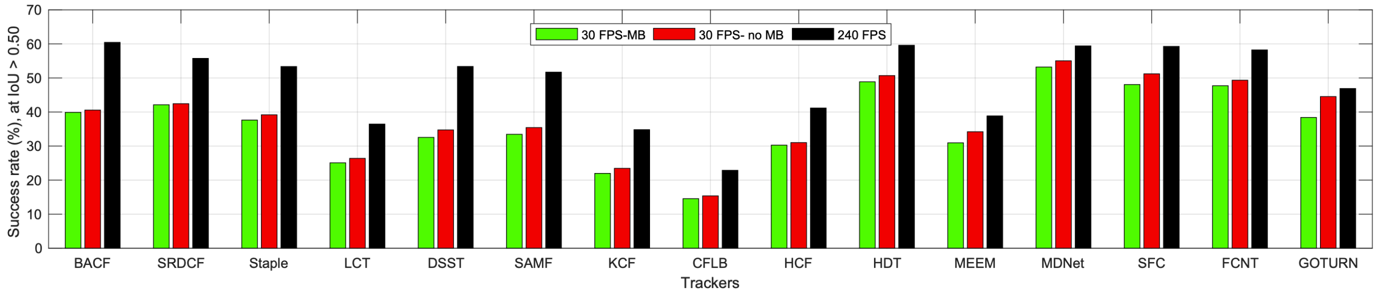

Fig. 1. Success rate vs. FPS on a variety of trackers. MB means motion blur. This figure is adopted from [4].

including [3] have shown that most Wi-Fi models support up to 802.11g and 802.11n among many Wi-Fi standards.

However, 802.11n that covers 97.7% of Wi-Fi AP in the world can support merely 20.473 Mbps while the theoretical bandwidth of 802.11n is 288.8 Mbps. To calculate the real world bandwidth of 802.11n, we took three measured points in [9] and fit an exponential curve to the points. We chose an exponential curve rather than a line since we found bandwidth decreases exponentially with respect to distance in [9]. Hence we got $y = 2.24 + 104.44 \cdot e^{-0.299 \cdot x}$ where y is bandwidth and x is distance to the Wi-fi AP. We calculated the expected value of the bandwidth within the distance range from 1 m to 15 m as follows:

$$\frac{1}{(15-1)} \int_{1}^{15} 2.24 + 104.44 \cdot e^{-0.299 \cdot x} dx \approx 20.473\text{Mbps}$$

We assumed a maximum distance of 15 m as the authors in [9] couldn't measure any more signals after exceeding this distance to the Wi-Fi AP.

Wi-Fi bandwidth plays an important role in the assumed video surveillance system. In general, trackers' performance in video systems degrade if FPS is low (see Figure 1) [4]. Since bandwidth directly affects FPS, it is crucial to guarantee enough bandwidth for video surveillance system. However, as described above, current prevalent Wi-Fi access points are not capable to provide adequate bandwidth for all cameras to be streamed simultaneously.

B. Filtering cameras to stream

Our solution is to filter the camera to send video data to the cloud using the filtering techniques presented in RexCam. RexCam builds spatial-temporal correlation matrix between cameras in the preparation phase to reduce computational overhead of identity tracking algorithms by picking up only relevant images to current object of interest among video streams. This is a reasonable choice because, from a locality point of view, there is a high probability that an object will be found again in an image close to the image where the object is already observed. We propose to dynamically filter which "cameras" would send their frames to the cloud, rather than filter "images" already sent to the cloud. That is, we suggest a system which make only the cameras adjacent to the camera that is currently sending the video containing objects stream all frames of video, while other cameras stream partial frames of video. By doing so, we can naturally solve the bandwidth problem as well as reduce the computational overhead which is already achieved in RexCam.

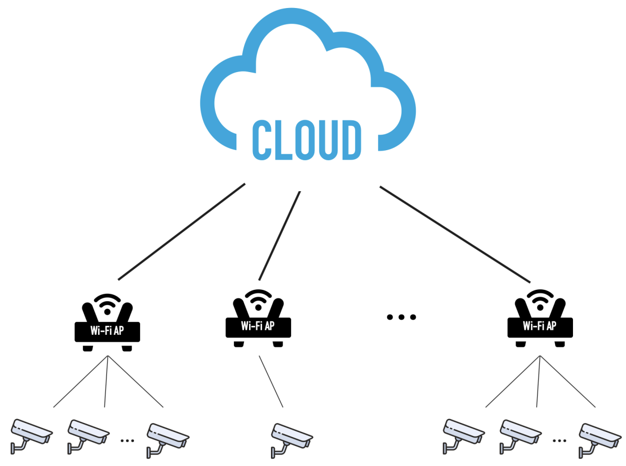

Fig. 2. Architectural overview of our new system.

III. Proposed scheme

In Section II we showed that the approach in [6], streaming all cameras to the cloud, is unrealistic in the real world. In Figure 2 we propose a new architecture which considers the previously outlined real-world requirements. To summarize our idea in one sentence from the clouds perspective our proposed system improves the following: instead of filtering out unwanted camera streams in the cloud we will control which cameras stream to the cloud.

A. Assumption

Similar to [6] we place multiple cameras in the environment but rather than letting stream every camera to the cloud and then do the filtering there we propose a another layer as shown in [7]. As analyzed in Section II we would assume that the amount of Wi-Fi cameras increases we therefore need to connect each camera to a Wi-Fi access point rather than using wired connections. Also, the bandwidth of the Wi-Fi access point is insufficient to meet the fps of all cameras as described in II.

B. Algorithmic Changes

With this updated architecture we need to modify the flow of streams in our system. We propose to shift the "filtering" step based on the spatial-temporal model in [6] on cameras. This will reduce the traffic which is sent to the cloud heavily as no longer every camera stream is uploaded to the cloud. There are no restrictions which could hinder us in doing so. To achieve this the cloud can change the available bandwidth for each camera dynamically. We will still use the same model as described in [6] but we will move the camera filtering part on cameras themselves by sending signal for a bandwidth change from the cloud to them. As a consequence the cloud has the ability to balance loads from cameras.

Please keep in mind that we only focus on the inference part as we assume that the spatial-temporal model is constructed in an offline pre-phase. For all analysis regarding amount of data needed for offline profiling etc. we point towards [6] where they extensively studied the offline model creation phase. As we don't do any improvements here we assume the model as "given" in the following. Such a spatial-temporal model gives us information whether a camera is correlated spatially or temporally based on a defined threshold for both cases respectively.

IV. Proposed Algorithm

In order to understand how we split bandwidth for one Wi-Fi Access Point among its connected cameras we will provide an algorithmic outline of three possible methods. The baseline (see Algorithm 1), binary Algorithm 2 and relative sharing of bandwidth Algorithm 3.

A. Baseline Bandwidth Splitting

The baseline for bandwidth splitting can be seen in Algorithm 1. Based on the maximum amount of bandwidth $b_{\max}$ available all cameras n_{cameras} get an equal portion of it. The algorithm returns a list of bandwidths b_{cameras} indexed by camera indices for one particular Wi-Fi access point.

Algorithm 1 Baseline

```
1: procedure BaselineSplit($b_{\max}$, $cs_{\text{all}}$)
2:     for all $c \in cs_{\text{all}}$ do
3:         $b_{\text{cameras}}[c] \leftarrow b_{\max}/|cs_{\text{all}}|$
4:     end for
5:     return $b_{\text{cameras}}$
6: end procedure
```

B. Baseline Binary Splitting

In this model we leverage the proposed filter from [6] which is defined as

$$M(c_s, c_d, t_{\text{curr}}) = \begin{cases} 1, & d_{sc}(c_s, c_d) \geq s_{\text{thresh}} \\ & \quad \text{and} \\ & d_{tc}(c_s, c_d, [t_0, t_{\text{curr}}] \leq 1 - t_{\text{thresh}} \\ 0, & \text{otherwise} \end{cases}$$

Where d_{sc} is the spatial correlation between the source camera c_s and the destination camera c_d. The second filtering step d_{tc} captures the temporal correlation between by calculating the volume of historical traffic that arrived in the destination camera c_d from the source camera c_s in the timeframe between $[t_0, t_{\text{curr}}]$. Here f_0 is the frame at which the first historical arrival of the subject c_d from c_s was recorded. For further information we kindly refer to [6].

Given that model we can derive Algorithm 2 for a binary split case. Again as before the maximum available amount of bandwidth $b_{\max}$ for one access point needs to be split across all connected cameras n_{cameras}. Also, the algorithm returns a list of bandwidths b_{cameras} indexed by camera indices for this particular Wi-Fi access point. Additionally we receive the filter model M. We assume that both threshold s_{thresh} and t_{thresh} as well as the current time t_{curr} are already incorporated in the model and leave them out for clarity. Additionally we are provided with the optimal bandwidth per camera b_{optimal}.

Algorithm 2 Binary Split

```
1: procedure BinarySplit($b_{\max}$, $cs_{\text{all}}, b_{\text{optimal}}$)
2:     $cs_{\text{rel}} \leftarrow \{c_d \mid c_d \in n_{\text{cameras}} \wedge M(c_s, i)\}$
3:     $cs_{\text{irel}} = cs_{\text{all}} \setminus cs_{\text{rel}}$
4:     $b_{\text{rel}} = \min(b_{\max}, b_{\text{optimal}} \cdot |cs_{\text{rel}}|)$
5:     $b_{\text{irel}} = b_{\max} - b_{\text{rel}}$
6:     for all $c \in cs_{\text{rel}}$ do
7:         $b_{\text{cameras}}[c] \leftarrow b_{\text{rel}}/|cs_{\text{rel}}|$
8:     end for
9:     for all $c \in cs_{\text{irel}}$ do
10:        $b_{\text{cameras}}[c] \leftarrow b_{\text{irel}}/|cs_{\text{irel}}|$
11:    end for
12:    return $b_{\text{cameras}}$
13: end procedure
```

The most important difference to the baseline algorithm is that we assign the bandwidth in two steps. First, we calculate all relevant cameras denoted by $cs_textrel$ (Line 2) and the remaining ones (Line 3). Second we calculate the maximum bandwidth which gets assigned to these cameras (Line 4) and as before the remaining bandwidth (Line 5). It is important to notice that in the potential case that more cameras are relevant than the actual bandwidth supports the irrelevant cameras will get assigned no bandwidth at all. This issue could be overcome by adding a dampening factor in Line 4. Then from Line

6 and onwards the bandwidths get assigned just as before in Algorithm 1.

C. Relative Split

For this splitting algorithm (Algorithm 3 we propose to even further leverage the model and rather than plain thresholding we will use the correlation directly to calculate the portions. The splitting algorithm has the same input as before, when assuming one can access $d_s(c_s, c_d)$ and $d_t(c_s, c_d, t_{\text{curr}}$ directly. Also, as we calculate a relative split we don't need to know the optimal bandwidth anymore.

Algorithm 3 Relative Split

```
 1: procedure RelativeSplit($b_{\max}$, $cs_{\text{all}}$)
 2:     for all $c \in cs_{\text{all}}$ do
 3:         $d_{\text{cameras}}[c] \leftarrow d_s(c_s, c) \cdot d_t(c_s, c, t_{\text{curr}})$
 4:     end for
 5:     $d_{\text{cameras}} \leftarrow \text{normalize}(d_{\text{cameras}})$
 6:     for all $c \in cs_{\text{all}}$ do
 7:         $b_{\text{cameras}}[c] \leftarrow b_{\max} \cdot d_{\text{cameras}}[c]$
 8:     end for
 9:     return $b_{\text{cameras}}$
10: end procedure
```

The algorithm itself is straight fowarbd. First we calculate the relevance for each camera (Line 3) but compared to Algorithm 2 we will not have a binary decision of 0 or 1 but rather capture the relevance relative to the others by normalizing it (Line 5).

D. Discussion

As previously discussed in II-A obviously the real-time accuracy of the network increases when we provide more camera to the relevant cameras. When following this idea, one obviously will choose Algorithm 2 as it provides the highest bandwidth when possible to the relevant cameras.

But one thing that also needs to be considered is the impact of fast-replay. Fast-replay is also proposed by [6]. Fast-replay is triggered when the current target is lost. The cloud will then go through all cameras and analyze them. This assumes though that all the data is saved on the cloud and hence all data is streamed to the cloud. We will now analyze how our proposed algorithms improve or hinder fast-replay.

When we use Algorithm 2 and reduce the remaining bandwidth for the irrelevant cameras to 0 the cameras will not stream any videos to the cloud. Therefore, we can not save their streams in order to do fast-replay on this data. As already mentioned IV-B one way of overcoming this issue might be to use decay factor. As this is one more parameter to tune for each specific setup we came up with our idea for relative split depending on the relevance of the camera in IV-C. This variant also ensures that irrelevant cameras are streamed to the cloud but with reduced bandwidth depending on their relevance. Of course the baseline provides the best base for fast-replay but one might argue that on the other side a lot of unnecessary data is streamed to the cloud.

E. Experimental Setup

In our experimental setup, cameras, Wi-Fi APs and the cloud will be deployed in a simulated environment. Each camera is able to adjust the frame rate of their video stream being sent to the cloud. Cameras are connected to one Wi-Fi AP which has a limited bandwidth capacity. We assume that the cloud tries to filter out less important cameras by sending them signal for themselves to lower their frame rates while it keeps running object tracking algorithm which is affected by frame rate. Right now, we will utilize ns-3 simulator [8] to emulate the network of our system and test bandwidth across the system. Each entity, camera, Wi-Fi AP and cloud will have be a separate instance on the simulated system. Our goal is to observe how frame rates are changing across cameras and how object tracking accuracy is going up and down according to those frame rates. Currently, we are setting up our simulated network environment.

F. Further Extensions

With this new architecture there are multiple extensions we could apply in order to make our system more autonomous or robust. Introducing them in an extended manner would be out-of-scope for the course, therefore we only give a brief overview of them.

1) Initial Query Discovery: Currently the system proposed in [6] does not provide any solution for doing an initial query. They say queries have to be issued by hand which is not scalable to a lot of cameras. To solve this problem there could be multiple solutions.

First, one could use a system as described in [7] where the authors use the cameras resources efficiently in order to fulfill simple tasks (such as detection or tracking). Of course, in order to save hardware costs for more powerful cameras, the advanced re-identification algorithm would be run on the cloud as described previously.

Second, if the previous solution is not accurate enough, one could leverage the cloud in order to get more stable results. Instead of selecting specific cameras to be streamed to the cloud, the bandwidth between cameras and the Wi-Fi access point is shared equally. Of course, this will result in a lower frame rate, but as we are currently not tracking anything in real time we could sacrifice this frame rate drop. On the other hand when constantly streaming to the cloud at maximum capacity the costs for running the cloud server will increase.

To make a clear decision one has to study these two solutions in addition different methods. Doing so would be out of scope for this project we therefore kindly assume this as future work or an open issue in our new, proposed system and in the original RexCam [6] implementation.

2) Fast Replay Search: One problem we ignore in our project is how to maintain accuracy for fast-replay search as described in [6]. As already said, fast-replay search cannot be carried out as proposed in the original paper, because at the point where a fast replay is needed not all data is available at the cloud. Therefore, one has to provide the historical data one after another to the cloud for processing while caching all incoming data simultaneously. If we use the reasonable assumptions that the cloud can process streams faster than real-time (if in question just deploy more GPUs) as well as our upload bandwidth allows us streaming faster than real-time we can show that fast replay search is still possible even though not all data resides on the cloud but rather on the cameras itself.

References

[1] Dropcam website. https://www.dropcam.com/. Accessed: 2019-04-07.

[2] Ieee 802.11. https://en.wikipedia.org/wiki/IEEE_802.11. Accessed: 2019-05-06.

[3] S. Biswas, J. Bicket, E. Wong, R. Musaloiu-e, A. Bhartia, and D. Aguayo. Large-scale measurements of wireless network behavior. In ACM SIGCOMM Computer Communication Review, volume 45, pages 153–165. ACM, 2015.

[4] H. K. Galoogahi, A. Fagg, C. Huang, D. Ramanan, and S. Lucey. Need for speed: A benchmark for higher frame rate object tracking. CoRR, abs/1703.05884, 2017.

[5] M. Gou, S. Karanam, W. Liu, O. Camps, and R. J. Radke. Dukemtmc4reid: A large-scale multi-camera person re-identification dataset. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 10–19, 2017.

[6] S. Jain, J. Jiang, Y. Shu, G. Ananthanarayanan, and J. Gonzalez. Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale. arXiv preprint arXiv:1811.01268, 2018.

[7] S. Y. Jang, Y. Lee, B. Shin, and D. Lee. Application-aware iot camera virtualization for video analytics edge computing. In 2018 IEEE/ACM Symposium on Edge Computing (SEC), pages 132–144. IEEE, 2018.

[8] G. F. Riley and T. R. Henderson. The ns-3 Network Simulator. In K. Wehrle, M. Güneş, and J. Gross, editors, Modeling and Tools for Network Simulation, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] Z. Shah, S. Rau, and A. Baig. Throughput comparison of ieee 802.11ac and ieee 802.11n in an indoor environment with interference. In Proceedings of the 2015 International Telecommunication Networks and Applications Conference (ITNAC), ITNAC '15, pages 196–201, Washington, DC, USA, 2015. IEEE Computer Society.

[10] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee. The design and implementation of a wireless video surveillance system. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom '15, pages 426–438, New York, NY, USA, 2015. ACM.