Machine Learning at the Network Edge for Automated Home Intrusion Monitoring

Aditya Dhakal, K.K. Ramakrishnan
Department of Computer Science and Engineering
University of California, Riverside
Email: adhak001@ucr.edu, kk@cs.ucr.edu

***Abstract*— Monitoring of residences and businesses can be effectively performed using machine learning algorithms. As sensors and devices used for monitoring become more complex, having humans process the information to detect intrusions would be expensive and difficult to scale. We propose an automated home/business monitoring system which resides on edge servers performing online learning on streaming data coming from homes and businesses in the neighborhood. The edge servers run OpenNetVM, a Network Function Virtualization (NFV) platform, and host multiple machine learning applications instantiated on demand. This enables us to serve a set of customers in the neighborhood on a timely basis, permitting customization and learning of the behavior of each home. We combine the results of the multiple classifiers, with each classifier examining a distinct feature related to a distinct sensor, to finally infer whether the entry is a normal one or an intrusion. Our results show that our system is able to classify intrusions better than basing the decision on a single classifier, thus reducing false alarms. We have also shown that our system can effectively scale and monitor thousands of homes.**

I. Introduction

Monitoring of residences and businesses is a significant industry, and continues to grow. Multiple home monitoring security systems and services that are used to monitor homes and businesses to detect intruders are available in the market. Monitoring can range from being quite simple, with simple sensors for doors or windows opening to quite complex ones with multiple cameras, motion sensors inside and outside the building, and biometric scanners in addition to the door and window sensor. Together with these sensors, a typical security system also includes one or more control panel that is used to arm and disarm the alarm. Most of these monitoring systems have teams of humans working in a central monitoring station to analyze and respond to each and every alarm. The response usually is to call local law enforcement and the home/business owner. However, the alarms can be triggered by an actual intrusion or by mistake (a false positive). For example, it may be because the homeowner, or possibly a guest, being late to disarm the alarm. Each alarm that has to be handled by human intervention will cost the monitoring service time as well as money. A false positive alarm is expensive. Similarly, missing a true alarm and letting an intrusion go undetected (a false negative) can be equally, if not more serious.

Automating such monitoring is highly desired. Especially, as the sensors and devices used for monitoring become more complex, having a human to detect intrusions would be increasingly expensive, error-prone and more difficult to scale. With the growth of machine learning algorithms, and their effectiveness increased by the availability of more powerful computation resources, automating home monitoring is becoming increasingly feasible[1]. While it could be performed by a server complex in data centers at the central monitoring station, we believe it may be more cost-effective and more importantly, bandwidth efficient, to have network resident edge-computing resources perform the detection and inference of intrusions in homes and businesses.

We propose an automated home/business monitoring system which resides on edge servers performing online learning on streaming data coming from homes and businesses in the neighborhood. This learned model will be then used to detect each human entry into home and classify/infer whether the entry is a normal, safe entry by the occupants or an entry by an intruder. We wish to ensure that the inference is performed with a low false positive rate and with a sufficiently high accuracy (very low false negative rate). We also want the system to have high performance in learning and inference, so as to be scalable in supporting a relatively large number of homes in a neighborhood.

The monitoring framework is hosted on OpenNetVM[2], a Network Function Virtualization (NFV) platform.We use the capability of OpenNetVM to host multiple machine learning applications and use them on demand, depending on the sensors being used in each home/business. For example, a home may have a camera, and could benefit from face recognition, and a network function (NF) for face detection is instantiated on demand. Packet data from sensors in the home is delivered to the edge network OpenNetVM platform where an NF processes and updates a per-home data structure. Upon receiving the complete data related to an entry event (e.g., when the alarm is disarmed by entering the appropriate identification code by the homeowner) or when a timer expires, the entry information is presented to multiple classifiers for inference. Each classifier examines a different feature related to the entry event. We combine the results of the multiple classifiers to finally infer whether the entry is a normal one or is an intrusion. Having the detection and inference performed at the edge enables the platform to serve a more manageable set of customers of the alarm service, permitting customization and learning of the behavior of each home.

978-1-5090-6501-1/17/$31.00

II. Motivation and Related Work

Going forward, monitoring systems will likely become richer in functionality with more complex sensing, including the use of face recognition and behavior monitoring, with cameras to capture images and videos, complementing the binary triggers from the current set of devices such as motion detectors or detectors for door or window opening. It would be difficult to have humans in the loop for observing, detecting and inferring intrusions. Moreover, it doesn't scale to large numbers of homes being monitored. Automation of these tasks is needed.

There are several options for performing this automated monitoring. One could be at the home itself, with the inclusion of computation and communication capabilities, in addition to sensors. Communication could be to a central monitoring station as is done currently, or to law enforcement. However, for a large percentage of the time, with only normal activity, these resources will remain unutilized. For this reason, sharing the resources (compute and communication) across homes is desirable, to get the (potentially quite significant) benefit of statistical multiplexing. Replacing the central human resource pool at the monitoring station with computational resources for machine learning is another option. While this is feasible, it does require all information to be shipped from the home to the central monitoring station/data center, which can become bandwidth intensive with video being shipped to the central data center. Network and server I/O bandwidth, and processing resources at the central data center will be a concern. A third alternative is to have distributed edge computing resources located at neighborhoods to be able to perform the monitoring for the homes and office buildings in a neighborhood. Having learning and inference in the edge would be closer to the homes being monitored, saving network bandwidth. This enables the inclusion of more complex surveillance with video and other bandwidth intensive monitoring capabilities. Having the ability to learn and infer at the edge allows us to efficiently manage and customize these tasks as needed at the edge resources are focused on providing service to a smaller number of homes than compared to the centralized systems. For example, every home could have a different setup of sensors and have different types of sensors. E.g. some homes and offices could be equipped with cameras and sound sensors, others may not. In those cases, each home could possibly require a different set of machine learning algorithms. We have to keep some amount of state once a person initiates a sensor event upon entering a home, till either the alarm is disabled or the monitoring system times-out and sounds a local alarm and requires the central station to be notified. Keeping the state for a large number of homes presents scale challenges at the central data center. Having an edge computing platform would imply that it has to keep less state, for fewer homes, in one location. This is again more manageable and potentially more scalable.

Using machine learning (ML) algorithms to learn human activities in monitoring and security context has been researched by other groups [1]. Similarly, community wide alarm system using SDN controller has been discussed [3]. Our primary goal is to demonstrate the use of ML functionality across multiple network functions (NFs) running on an edge computing platform with OpenNetVM. We use existing ML algorithms in NFs running on OpenNetVM, with the added functionality of running the Dempster-Shafer algorithm in another NF. We find OpenNetVM, with its high-speed processing capability, to be a suitable platform to host the ML applications as middle-box features in the network. Most of the research in home security utilizes a server in data-center to store the incoming data so it could be processed subsequently. However, in our framework, with network resident ML algorithms, we perform the task on streaming data. Stream processing algorithms like KNN with data replacement [4] lets us classify incoming packets without storing all the incoming data points. Since we receive data streams from multiple sensors in a home, having those processed in the network edge to extract desired features from the packet stream for learning and inference can save bandwidth and load on the central servers having to process the information from a very large number of homes continuously. Furthermore, use of algorithms working on streaming data would require us to store very little information in the edge server.

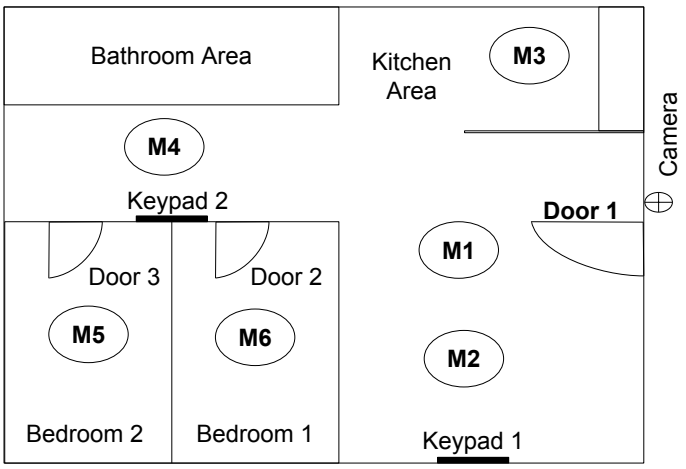

Fig. 1: Apartment Layout

III. System Design

We focus our discussion on home monitoring by examining what it would take to monitor a typical small apartment. Homes and businesses would have a much more complex architecture and potentially many more sensors (motion, door/window opening, cameras and alarm enabling/disabling keypads).

A. Sensor Placements, Data Generation

The apartment floor plan used for data generation is shown in Figure 1. The sensors in this design include a camera outside the apartment, 3 door sensors for the front door (Door 1) and two bedroom doors and multiple motion sensors labeled M1-M6. We define a "home-entry" as a sequence of the events which starts when the front door is opened. If a camera is available in that home, the facial image taken by the camera is processed by OpenFace [5] face recognition after the front door is opened. Actions like unlocking the front door and walking through the

home will activate the door sensors and motion sensors in those rooms. Once activated, all the sensors (door, motion) will send a packet to the edge server with a time-stamp of when the sensor was activated. The "home-entry" event is considered complete once the user reaches a control keypad and enters the PIN code to disarm the alarm. Normal behavior of a legitimate person entering the home through Door 1 could be to enter and walk directly to Keypad 1 to disable the alarm; or walk to Keypad 2; or through the kitchen (drop stuff off) and then to Keypad 2; or from the bedroom to Keypad 2 to disable the alarm (e.g., sleeping at night, steps out of the room). We set time limits of 2 minutes after opening a door to disable the alarm.

We model our data after the data of real life home entries in the Kyoto dataset [1]. As the Kyoto dataset did not have enough data points for home entry scenario, we created synthetic data with average time and the standard deviation of time between the sensors the same as in the Kyoto dataset.

IV. NFV & Testbed Implementation

We have used OpenNetVM[2] as the platform to run our ML classifiers. OpenNetVM provides a high-performance packet processing framework to host multiple Virtual Network Functions (VNFs) on lightweight docker containers. We also use the NF Manager in OpenNetVM to provide efficient load balancing, flow management and service name abstractions for those NFs. OpenNetVM uses Intel's DPDK libraries to provide high performance I/O with zero copy delivery of packet data to NFs. OpenNetVM's simplifies the process of developing and hosting the NFs and efficiently route the packets through the NF service chains[6].

OpenNetVM uses Docker containers to host applications and its NF Manager directs network flows to those applications with simple flow rules. NFs in OpenNetVM can be dynamically initiated, and with the Flurries implementation [7], a service chain of NFs can be created on demand for each flow, without significant performance penalties. These features would be useful in the home monitoring scenario as different vendors competing in the market may offer customized and proprietary ML solutions. Some may offer facial recognition while others might offer inference engines that find intrusion based on human behavior, like walking speed, time of day etc. Thus, the ability of OpenNetVM to host and run multiple applications dynamically allows us to host ML applications in the network (on relatively small 'edge cloud' platforms). The use of containers to run those applications allows for isolation between application, for performance and security.

A. Testbed Implementation

Our testbed implementation includes one system running OpenNetVM that intercepts packets generated by sensors. These packets are delivered to the appropriate NF running the machine learning algorithm for processing the sensor's traffic.

For traffic generation, another system running OpenNetVM is used, emulating the sensor traffic generated from a large number of homes. The traffic generator reads the data from a trace file related to each home and sends packets related each sensor in those homes with the same time difference that a person entering the home would have generated the packets also contain the information about keypad (when the alarm is disarmed) and images generated by the camera.

V. Algorithms

Our system uses four classifiers to determine if an entry event is normal or an anomaly generated by an intruder. Each classifier learns and infers from different features observed from the entry events. The features classifiers currently use are time between activation of each sensor, the time taken to complete the entry event, the image from camera sensor and the number of times the PIN is entered on the keypad. Furthermore, all the classifiers also perform a *basic belief assignment*, i.e., they provide a numerical belief value [0,1] on how certain they are that the event is an intrusion or not. These belief values are then used to arrive at an overall determination using the Dempster-Shafer combination rule.

A. Learning

We have an initial training phase where the packets generated by sensors of a home for a timespan of say 2 weeks are collected. When the packets from each sensor are received, the above-mentioned features are extracted and stored in the relevant home's data structure. Once enough data is collected, it is used for creating ML models for that particular home.

Our first classifier is K-NN classifier that uses the time spent between the activation of each sensor to classify each entry as either a normal entry or an anomaly. After analyzing the data of home entry events from the Kyoto dataset [1], we saw that the average time spent between entering the home and reaching the kitchen was 9.88 seconds and the standard deviation was 3 seconds. This shows that humans would be somewhat consistent in the time they take to reach the keypad on entry into their home. Therefore, we decided to use the time difference between the activation of each sensor as a feature to create a model with K-NN. Once the training data is collected, our system runs 5-fold CV on the training dataset to get the average number of neighbors (KNN_{avg}) and standard deviation (KNN_{SD}) of the number of neighbors from the training examples. Then it calculates the threshold parameter K as $K = \text{KNN}_{avg} - \text{KNN}_{SD}$.

Our second classifier is the Openface face recognition software used for homes with a camera. While we could use human subjects for training and test data, we instead use existing images. Openface provides us with a pre-trained model that was trained using a database of 6000 images of 10 different celebrities. (we selected this dataset for expediency). Openface provides a score in the range [0,1] for every image it examines, with respect to the image data base. For images that it has not seen, Openface does not provide an inference with probability 1 that the image is not among the 6000 images it has been trained with. Instead, it continues to assign a score, albeit lower, for that image. We took a subset of the images from the database as residents of each home for training our model. We used a different set of images for the intruders.

Our third classifier looks at the number of times the keypad is pressed before the person enters a valid access code. A resident of a home would know the PIN code and would likely not make mistakes while entering the code. However, an intruder is likely to make multiple attempts to get the correct code. Thus, we could infer from the number of times someone is trying to disarm the system as to whether the person is an intruder or not. However, the homeowner could still make a simple mistake while entering the code, if it is dark or for other reasons. Thus, our setup considers the person entering the code is a resident if the person is able to put the correct code within the first three tries.

Our fourth classifier, Disarm-time uses the time needed by the home's users to complete an entry event as a feature. A person not familiar with the home layout might take longer to find out where the alarm keypad is. Thus, this classifier would be useful to detect an anomaly in the time taken for disarming. After gathering the initial training data, the mean and standard deviation of time taken to disarm the alarm is calculated and used to compute a threshold for inference.

As we seek to perform the classification and inference in the network, rather than a server that might examine all the stored monitoring data, we seek algorithms that perform well with streaming data. While working with streaming data, we should note that as the time passes and we receive more data, statistical properties of the data can change, e.g. a resident of the home could get slower due to illness or injuries etc. Thus, the model created some time in the past becomes less accurate on classifying events in the present. This phenomenon is known as concept drift [8]. To avoid concept drift, our system has to update the classifiers frequently as new data arrives. In our system, once we have a certain number of new entry events classified as normal, we replace the existing data with these newly received data points. Then we run the cross validation once more to update the models for the classifiers. To conserve computational resources, our system only updates the original models after 30 new normal entries. If the data point is inferred to be from an intruder, an alarm is raised (which in an actual system may involve forwarding the packets and the inference to the central station for further handling).

B. Inference

Once the initial training is complete, the system is ready to infer and classify new entry events. When the "alarm disarmed" packet is received, i.e when the user disarms the alarm successfully by entering the right PIN or the time to enter the PIN (120 secs) expires, then the data collected from the packets are inferred and classified. Packets generated by a camera (e.g., an image file), are passed to and processed by the face-recognition software OpenFace[5]. Together with classifying the events as normal and anomalous, the classifiers also assign beliefs to each inference.

For KNN, we use the K computed during cross-validation. Any inferred event having less than K neighbors is classified as an anomaly. The *basic belief assignment* for event x, where x has n neighbors would be:

$$Bel(x) = \begin{cases} 1, & \text{if } n < K \\ 0, & \text{if } n > \text{KNN}_{avg} \\ 1 - \frac{n-K}{\text{KNN}_{avg}-K} & \text{otherwise} \end{cases}$$

For the OpenFace classifier, the system will take the image coming in from the camera and feed it to the OpenFace application and receive a score based on if the image belongs to a resident of the house or not. We have assigned a threshold score of 0.95 for classifying an image as a resident of the home. Any image getting score lower than 0.95 would be classified as an intruder. We computed the belief as $Bel(image) = 1 - \text{OpenFace}(image)$.

The keypad classifier would classify an event where a user would press the keys more than 3 tries as an intrusion. If the user disables the keypad in x attempts, the belief would be:

$$Bel(x) = \begin{cases} 1, & \text{if } x > 5 \\ 0, & \text{if } x < 3 \\ \frac{x-2}{4} & \text{otherwise} \end{cases}$$

The disarm-timer would flag an event as an intrusion if the time taken to disarm the timer would exceed 1 standard deviation more than the mean disarm time learned from the training example. Belief assignment for an entry event which takes x-units of time is computed by the following expression:

$$Bel(x) = \Phi(z), \text{where, } z = \frac{x - \mu}{\sigma}$$

Where, μ and σ are the mean time of entry, and its standard deviation in training samples, z is the z-score of an inferred entry and $\Phi(z)$ is the probability obtained from the probability density function of a normal distribution.

Whether it is with Openface or other face recognition software, they will inevitably have to process new faces, beyond the ones that have been presented as training data for the legitimate residents and visitors of the homes and buildings being monitored, and we have to deal with uncertainty when recognizing an image and conclude that it is an intruder. This is where combining multiple sensors and fusing the information across all these sensors is key for an effective monitoring system.

C. Dempster-Shafer Theory

The Dempster-Shafer (D-S) theory of evidence [9] [10] has been widely used as a method to fuse multiple decisions from different ML classifiers to make an overall inference.

1) Terminology: Frame of Discernment (Θ) : The frame of discernment is the exhaustive set of all the hypotheses in the same problem domain. For example, for the case of intrusion detection in the home monitoring scenario the frame of discernment, Θ = $\{\phi, \{Normal\}, \{Intrusion\}\}$ where proposition ϕ is an empty set ,the proposition $\{Normal\}$ and $\{Intrusion\}$ sets indicate that the inferred entry is normal or an intrusion.

Belief: Belief in a proposition is a measure of certainty that is committed to that particular proposition.

Basic Belief Assignment (BBA) is a function m: $2^{\Theta} \rightarrow [0, 1]$, that assigns a belief to each subset of the frames of discernment Θ such that,

$$m(\phi) = 0 \qquad \text{and} \qquad \sum_{A \subset \Theta} m(A) = 1$$

Here, $m(A)$ is also known as mass value (measure of certainty) of A. Where A a member of the Θ.

2) Dempster's Rule of Combination: We can use Dempster's rule of cumulative combination (DRC) [11] that is used to combine evidence from different independent sources. If we have 2 classifiers A and B, and classifier A provides belief for intrusion, $B_A(Intrusion)$ = 0.6 then rest of the belief is assigned to the frame of discernment such that $B_A(\Theta) = 0.4$. Then, equations 1 and 2 can be used to combine the beliefs for intrusion I from the two models A and B.

$$CUM(I) = \frac{B_A(I)B_B(\Theta) + B_B(I)B_A(\Theta)}{B_A(\Theta) + B_B(\Theta) - B_A(\Theta)B_B(\Theta)} \quad (1)$$

$$CUM(\Theta) = \frac{2B_A(\Theta)B_B(\Theta)}{B_A(\Theta) + B_B(\Theta) - B_A(\Theta)B_B(\Theta)} \quad (2)$$

D-S theory requires no *a priori* knowledge, thus is suitable for anomaly detection as it can potentially detect scenarios not seen before [12]. In contrast, other algorithms that combine results of multiple models like Bayesian Inference requires some *a priori* knowledge. Furthermore, the D-S theory allows us to provide the uncertainty on each model for a sensor, based on domain knowledge. Combining multiple models optimally using the Dempster's Rule of Combination would help reduce the amount of false positives and false negatives.

VI. Evaluation

We evaluate our system in terms of accuracy in predicting intrusion and in terms of scalability. The synthetic data we generated for the experiment purpose is on the timescale of seconds as it mimics the real human behavior. To complete the experiment in a reasonable time, we scaled the event inter-arrival times from secs to millisecs. However, in our implementation, the OpenFace face recognition still takes on average 2.9 secs on a single CPU core, per image. As it is small compared to the time taken for the entire entry process (60 seconds) in real time, we safely assume that the image would be processed and the belief on that image would be available before the entry event ends. To avoid this significant compute overhead and simplify the evaluation, we inferred all the images of the people living in the home as well as the intruders separately and the traffic generator sends a packet with the result of OpenFace immediately after the packet for the door open event.

A. Classifier Performance

The four classifiers we use are KNN, OpenFace, Keypad and D-Timer (disarm timer), shown in tables I and II. For testing, we created synthetic data for the following two intrusion scenarios, to test each classifier.

1) In scenario 1, an intruder enters the apartment via door 1, moves around the home (looking for the keypad or to steal articles). In this scenario, there is no camera.

Classifier	Accuracy	False +ve	False -ve
KNN	98.5%	1	2
D-Time	95%	9	1
DS	99.5%	1	0

TABLE I: Classifiers' performance in Scenario 1

Classifier	Accuracy	False +ve	False -ve
KNN	26%	54	94
D-Time	41%	65	53
Keypad	75%	0	50
OpenFace	92%	16	0
DS	96.5%	7	0

TABLE II: Classifiers' performance in Scenario 2

2) In the second intrusion scenario, the intruder knows where the keypad is located and moves towards it very similar to a resident, but takes multiple attempts to disable the alarm.

We compute the accuracy (ratio of number of correct classifications to the total number of test samples), the count of false positive and false negative classifications for each classifier.

For testing scenario 1, 200 entry events were generated for each house for learning; 100 normal and 100 anomalous entries were used for testing. Similarly, for scenario 2, we tested with 100 normal and 50 anomalous entries, with the intruder trying to disable the alarm 4 times (1 lower than the threshold we set for the alarm classifier to classify the entry as anomalous). We also generated 50 additional anomalous entries with the intruder trying to disable the alarm 5-6 times.

Table I shows that using Dempster's rule of combination to combine the output of different models reduces the false positive and false negative classifications. For scenario 2, shown in Table II, we see that the KNN and disarm-time classifier does not work well to catch an anomaly as the intruder walks inside in a manner similar to a resident. Thus, our training is incapable of determining it as an intrusion with high belief. However, the keypad classifier infers the intrusion with strong belief, which is reflected on the D-S's combination. To see how the combination of different classifiers is effective, we ran another experiment with fewer entry points and plotted the individual beliefs of each classifier as well as combined belief in figure 2. The threshold for D-S to classify an entry as an anomaly is 0.75. In case 1 all the classifiers have a low belief of intrusion. Thus, D-S combination also shows a low belief. In scenario 2, some classifiers show higher belief. However, Dempster's rule of combination brings the belief to a higher value, enough to raise an alarm. The third entry is an example of false positive alarm. Here, two classifiers show a very low belief, but one classifier is showing a high belief. Thus, the combination shows a high belief and classifies it as an anomaly. This is because we have chosen a lower threshold for D-S for classification, to reduce the number of false negative classifications.

B. Scalability

To test the scalability and latency of inference, we run an experiment where the system gets packets from a number of houses. We measure time taken to process an entry event and number of inferences our system can perform at a time. As

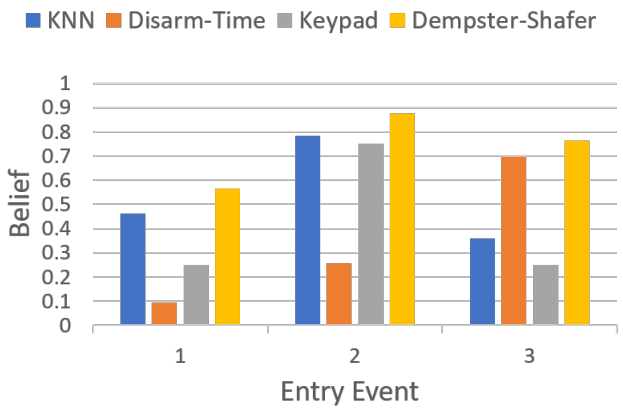

Fig. 2: Beliefs of Classifiers and D-S Combination

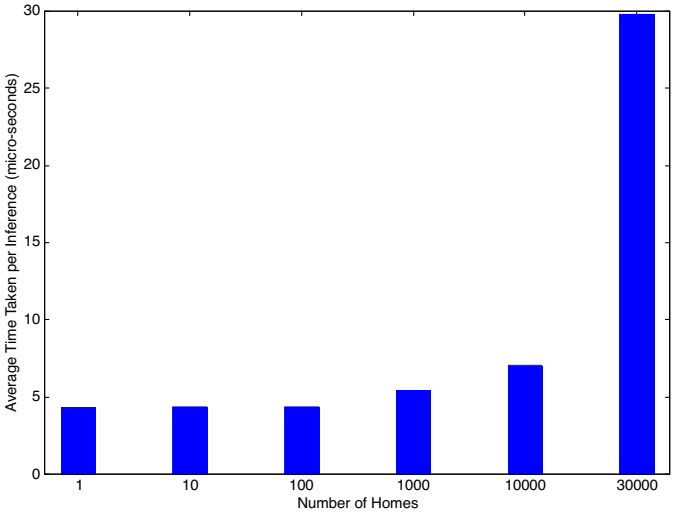

Fig. 3: Inference Time vs Number of Homes

shown in Figure 3, our system performs 1 inference in about 4.5 μ seconds. Increasing the number of inference to 10,000 (i.e., 10,000 different homes having simultaneous entry events) does not increase the average time for inference much. Furthermore, our system can perform 30,000 simultaneous inferences with only slight increase in the processing time. Supporting 30,000 simultaneous inferences would imply that the service can have a much larger number of subscribers to the service in the neighborhood being managed.

VII. Summary

In this paper we design and implement a system, that is resident at the network edge, for automated monitoring of homes and businesses, to reliably detect intrusions. Our system is customized to perform on-line learning of the behavior of residents in each home and use multiple classifiers, each examining a distinct sensor input as a feature, to detect intrusions. We showed that by combining the results from multiple classifiers, the detection of intrusion is much more reliable and reduces false alarms and can scale to support a large number of homes in a neighborhood (thousands of homes) close to where an edge cloud platform is situated.

Acknowledgment

This work was supported in part by NSF grant CNS-1619441 and by the Department of the Army, US Army Research, Development and Engineering Command grant W911NF-15-1-0508.

References

[1] J. Dahmen, B. L. Thomas, D. J. Cook, and X. Wang, "Activity learning as a foundation for security monitoring in smart homes," *Sensors*, vol. 17, 2017.

[2] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. Ramakrishnan, and T. Wood, "Opennetvm: a platform for high performance network service chains," in *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*. ACM, 2016, pp. 26–31.

[3] R. Abhishek, S. Zhao, D. Tipper, and D. Medhi, "Sesame: Software defined smart home alert management system for smart communities," in *2017 IEEE International Symposium on LANMAN*. IEEE, 2017.

[4] M. Tennant, F. Stahl, G. Di Fatta, and J. B. Gomes, "Towards a parallel computationally efficient approach to scaling up data stream classification," in *Research and Development in Intelligent Systems XXXI*. Springer, 2014, pp. 51–65.

[5] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," *CMU School of Computer Science*, 2016.

[6] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *2015 IEEE International Workshop on LANMAN*. IEEE, 2015, pp. 1–6.

[7] W. Zhang, J. Hwang, S. Rajagopalan, K. Ramakrishnan, and T. Wood, "Flurries: Countless fine-grained nfs for flexible per-flow customization," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. ACM, 2016, pp. 3–17.

[8] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, no. 2, 2004.

[9] A. P. Dempster, "Upper and lower probabilities induced by a multivalued mapping," *The annals of mathematical statistics*, pp. 325–339, 1967.

[10] G. Shafer *et al.*, *A mathematical theory of evidence*. Princeton university press Princeton, 1976, vol. 1.

[11] A. Jøsang, J. Diaz, and M. Rifqi, "Cumulative and averaging fusion of beliefs," *Information Fusion*, vol. 11, no. 2, pp. 192–200, 2010.

[12] Q. Chen, A. Whitbrook, U. Aickelin, and C. Roadknight, "Data classification using the dempster–shafer method," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 26, no. 4, pp. 493–517, 2014.