Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 6, December 2021, pp. 3127~3136
ISSN: 2302-9285, DOI: 10.11591/eei.v10i6.2698

On detecting and identifying faulty internet of things devices and outages

Feng Wang[1], Eduard Babulak[2], Yongning Tang[3]
[1]School of Engineering, Liberty University, Lynchburg, Virginia, USA
[2]School of Business, Liberty University, Lynchburg, Virginia, USA
[3]School of Information Technology, Illinois State University, USA

Article Info

Article history:

Received Jul 24, 2020
Revised Mar 13, 2021
Accepted Oct 24, 2021

Keywords:

Entropy
Fault detection
IoT
IoT outages
Traffic deviation

ABSTRACT

As internet of things (IoT) devices play an integral role in our everyday life, it is critical to monitor the health of the IoT devices. However, fault detection in IoT is much more challenging compared with that in traditional wired networks. Traditional observing and polling are not appropriate for detecting faults in resource-constrained IoT devices. Because of the dynamic feature of IoT devices, these detection methods are inadequate for IoT fault detection. In this paper, we propose two methods that can monitor the health status of IoT devices through monitoring the network traffic of these devices. Based on the collected traffic or flow entropy, these methods can determine the health status of IoT devices by comparing captured traffic behavior with normal traffic patterns. Our measurements show that the two methods can effectively detect and identify malfunctioned or defective IoT devices.

Corresponding Author:

Eduard Babulak
School of Business
Liberty University
Lynchburg, Virginia 24515, USA
Email: ebabulak@liberty.edu

1. INTRODUCTION

As internet of things (IoT) devices play an integral role in our everyday life, it is critical to monitor the health of the IoT devices because IoT sensors can suffer from a wide range of faults or security attacks [1]-[3]. For example, a temperature sensor may stop reporting measurement results since its battery is out of power or it loses network connectivity. The main goal of this paper is to develop efficient methods that can monitor the health status of IoT devices.

However, fault detection in IoT is much more challenging compared with that in traditional wired networks [4]-[6]. First, traditional observing and polling are not appropriate for detecting faults in resource-constrained IoT devices. For example, periodic polling IoT devices can quickly deplete the power supplies of IoT devices because of high energy consuming wireless communications. It is also hardly possible or practical for human beings to take care of a large number of IoT devices across a large area. When deploying IoT devices in a large geographical area, it is not possible to know when the battery dies or if a sensor device is malfunctioned. In addition, dynamic network conditions, such as temporary link breakdown and frequent changing topologies, increase the complexity of fault detection. Second, many fault detection techniques for wireless sensor networks (WSNs) have been proposed in recent years. For example, previous work [7] proposes a method that uses a Bayesian network to describe the spatial and temporal correlations of data to detect sensor failures. A Hidden Markov random field based method is used to detect faults [8]. Work [9] uses the I-V curve of a solar panel to monitor solar panels online at different time intervals. However, those detection methods for WSNs or solar panels are inadequate for general IoT fault detection because of the dynamic feature of IoT devices. For example, in an IoT network, new services and IoT devices may be added, removed, and changed over time.

Therefore, it is imperative to continually monitor the pulse of the IoT devices, to detect anomalies and faults, and drive recovery in a timely manner. Consequently, network-wide monitoring has emerged as a promising approach. Since large-scale IoT devices could generate large traffic volume, traffic-based detection methods must use very low memory space and processing time. Thus, designing a scalable and efficient detection method is critical for IoT fault detection.

In this paper, we propose two methods that can monitor the health status of IoT devices through monitoring IP packet traffic of these devices. The two methods utilize traffic deviation and entropy as indicators to detect IoT outages. The first method, which uses traffic deviation as outage indicators, can detect malfunctioned or defective IoT devices that contribute a large volume of IP traffic. To address the limitation of the first method, we propose an entropy-based method to detect the outages of IoT devices that do not generate large normal traffic volume. To efficiently monitoring traffic, the two methods uitilize a compact data structure proposed in our previous work [10]. More specificially, the compact data structure, called *Bitcount*, is used to capture IP traffic and identify traffic deviations. Note that our previous work [10] focuses on using Bitcount to detect heavy hitters. In this paper, we leverage Bitcount to capture IoT traffic and recover the identities of outages. More specifically, we extend Bitcount to design a new data structure to detect and identify a different traffic anomaly problem, traffic deviation, which is more challenging than detecting heavy hitters [11]. The contributions of this work are listed as show in:

- The proposed two key indicators, traffic deviation and flow entropy, are compact and can scale to monitor a large scale of IoT devices.
- The proposed detection methods can effectively detect and identify malfunctioned or defective IoT devices.

The rest of paper is organized as shown in: in section 2, we present our research methods. We evaluate the effectiveness of the two proposed methods in section 3. We conclude the paper in section 4 with a summary.

2. RESEARCH METHODS

In this section, we present our research methods. More specifically, we first present the data structure, Bitcount, the key enabler of our methods. Then, we present the traffic deviation and entropy indicators. Finally, we present the details of two detection methods.

2.1. Bitcount

As shown in Figure 1, Bitcount has a hash function h that evenly maps source/destination IP addresses onto m buckets. The hash table is used to resolve the collisions due to mapping multiple addresses into the same bucket. Each bucket maintains a SUM counter and a set of bit counters to record the occurrence of source/destination IPs that are mapped onto this bucket. The SUM counter counts the number of packets in the bucket. The SUM counter and the bit counters are denoted as $S[i]$, where $i \in [0,m\text{-}1]$, and $count[i][0],\ldots,count[i][31]$, respectively. Formally, whenever a packet arrives, it is hashed to a bucket. The value of the SUM counter is updated, $S[i]=S[i]+1$. The corresponding bit counters will be updated, for $j \in [0, 31]$, $count[i][j]=\Sigma bit_i(j)$, where $bit_i(j)$ represent the bit value of the j-th bit of the binary representation of the packet's IP address.

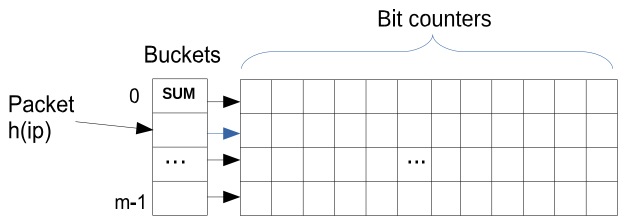


Figure 1. Bitcount data structure consists of a set of buckets. A hash function is used to map every packet into a corresponding bucket that maintains a SUM counter and a set of bit counters to record the occurrence of source/destination IPs that are mapped onto this bucket

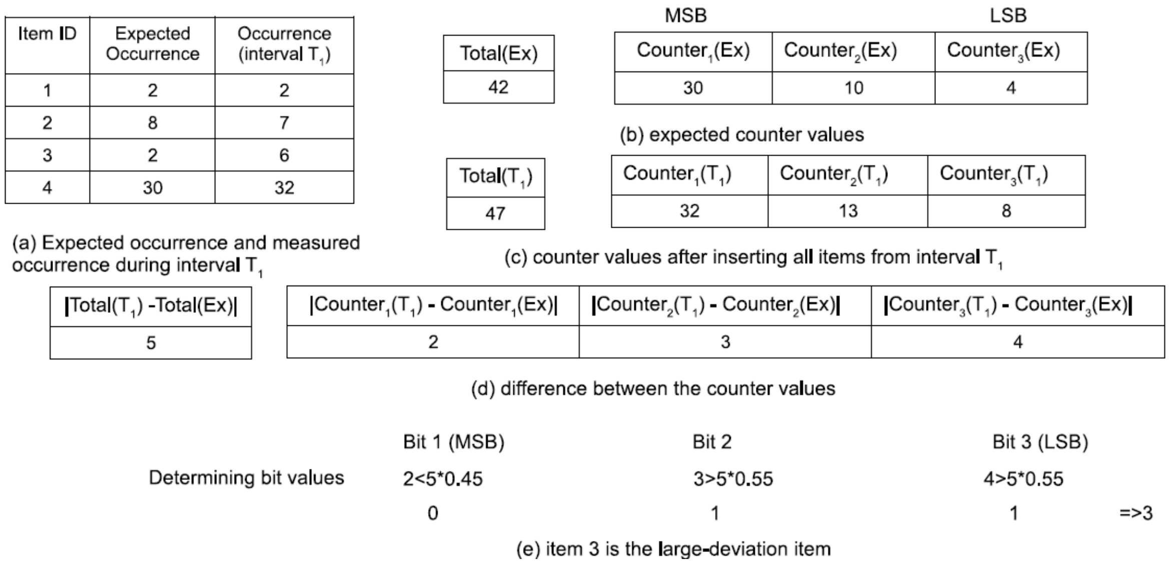
Bitcount can quickly detect and recover the identifiers of heavy hitters, where a heavy hitter is an item whose frequency is higher than a given frequency threshold. Various approaches, such as counter-based or sketch-based methods, have been proposed to address the scalability issue. Sketch-based approaches are the most commonly used method. For example, the most recent works, sketchvisor [12], flowradar [13], univmon [14], and elastic sketch [15] are sketch-based approaches. Although previous works have shown that counter-based approaches, such as lossy counting and spacesaving [16], are preferable to sketch-based ones, there are some problems, such as outage detection, which can only be solved by sketches [16], [17]. Comparing with those methods, Bitcount is more efficient in terms of memory space and the number of memory accesses [10].

2.2. Traffic deviation indicator

The goal of traffic volume deviation detection is to efficiently identify the IoT devices that have large deviations of their traffic volume from their expected volume. Any IoT device whose traffic volume deviation exceeds a pre-specified threshold is a suspicious heavy deviation device. Note that our traffic deviation problem is different with the heavy change problem defined in existing works [18]-[20].

The works most closely related to our approach are the reversible sketch [21] and group testing-based method [22]. The reversible sketch uses modular hashing and IP mangling to provide a reversible procedure to reconstruct original IP address. However, because the reverse calculation is very expensive, many pre-computing tables are used. More importantly, the reversible sketch cannot provide any accuracy guarantees so that it has to use another sketch to verify IP addresses. Thus, the space and time costs of this method are much higher than ours. In [22], a set of hash functions are used to divide items into groups, and perform testing on each group to detect heavy-hitters. On the contrary, our method can quickly identify large-deviation devices without performing group testing.

We use an example to illustrate the basic idea of the detection method. Suppose that there are 4 unique items. For each item, its expected occurrence and measured occurrence during measurement interval T_1 are shown in Figure 2 (a). In this example, item 3 is the heavy deviation item because its change rate (4/7=0.57) is higher than the other items.



Item ID	Expected Occurrence	Occurrence (interval T_1)
1	2	2
2	8	7
3	2	6
4	30	32

(a) Expected occurrence and measured occurrence during interval T_1

Total(Ex)
42

	MSB		LSB
	$Counter_1(Ex)$	$Counter_2(Ex)$	$Counter_3(Ex)$
	30	10	4

(b) expected counter values

$Total(T_1)$
47

$Counter_1(T_1)$	$Counter_2(T_1)$	$Counter_3(T_1)$
32	13	8

(c) counter values after inserting all items from interval T_1

$\lvert Total(T_1) - Total(Ex)\rvert$
5

$\lvert Counter_1(T_1) - Counter_1(Ex)\rvert$	$\lvert Counter_2(T_1) - Counter_2(Ex)\rvert$	$\lvert Counter_3(T_1) - Counter_3(Ex)\rvert$
2	3	4

(d) difference between the counter values

	Bit 1 (MSB)	Bit 2	Bit 3 (LSB)	
Determining bit values	2<5*0.45	3>5*0.55	4>5*0.55	
	0	1	1	=>3

(e) item 3 is the large-deviation item

Figure 2. An example of detecting a heavy deviation item; (a) the expected occurrence and measured occurrence of 4 unique items, (b) the expected values of bit counters, (c) the values of bit counters for the measured occurrence, (d) the difference values of bit counters, (e) the heavy deviation item 3 is identified according to the bit-set rate

We use 3 bit counters to count the total number of 1-bits in each bit position of binary representation of items' IDs. A bit counter is denoted as counti. For each item, the ith bit counter is incremented by one if the ith bit of the binary representation of the item's ID 1. The most significant bit of an item is recorded in the first bit counter, and the second most significant bit is kept in the second counter, and so on. There is an extra counter to count the total occurrence. In this example, we use 2 sets of bit counters to record the expected occurrence and the measured occurrence, respectively, as shown in Figure 2 (b) and (c). To detect the heavy deviation item, we first calculate the difference between the two total occurrences Sumd, and the difference between each bit counter, which is shown in Figure 2 (d). Then, for each bit counter, we define its bit-set rate as $\frac{|count_i(T_i)-count_i(E_x)}{Sum_d}$.

Finally, according to the bit-set rate, we use two thresholds (0.55, 0.45) to recover the heavy deviation item. That is, if a bit's bit-set rate is higher than 0.55, the bit is 1. Otherwise, if the bit-set rate is lower than 0.45, the bit is 0. As shown in Figure 2 (e), we can finally detect and identify the heavy deviation item (item 3). For more details about the identification algorithm, readers are referred to our previous work [9].

2.3. Entropy indicator

Although we can use the heavy deviation indicator to detect IoT outages, this indicator can only detect malfunctioned or defective IoT devices that contribute a large volume of IP traffic. Here, we focus on detecting the outages of IoT devices that do not generate large normal traffic volume. We propose to use a generalized information entropy as the outage indicator:

$$\mathrm{H}_\alpha(x) = \frac{1}{1-\alpha}\log_2\left(\sum_{i=1}^{n} p_i^{\alpha}\right) \tag{1}$$

In our measurement, we set parameter α to 2. The entropy indicates the entropy of the probability when the number of packets is smaller than the mean of the number of packets.

2.4. Heavy deviation detection

The heavy deviation detection algorithm is composed of two stages: a learning stage and a detecting stage. The goal of the learning state is to establish a profile of normal traffic. The normal/expected traffic profile contains the the expected number of packets for each block, and the expected bitwise summation of source IP addresses. During a detecting stage, deviations from the established profile are considered as anomaly. The two stages can operate in parallel or in series. We present the two stages in the following two subsections.

2.4.1. Learning expected traffic profile

Previous works [23], [24] have shown that IoT devices have active/sleep periods over which IoT devices are generating traffic or remain sleep. IoT devices are more likely to generate synchronized traffic resulting in bursty aggregate traffic volumes. We exploit the characteristics to build the expected traffic profiles. In particular, we separate aggregate traffic into active traffic and inactive traffic by using a threshold θ_a. For each bucket in the hash table, if its aggregate traffic volume is lower than θ_a, the traffic is called inactive. Otherwise, the traffic is called active traffic. The threshold θ_a is assigned to the first-level hash table, and each bucket may have the same or different thresholds. Note that the expected traffic profile is built from active traffic, but the detection can monitor both active and inactive traffic.

The learning stage spans several consecutive intervals in order to collect a sequence of measurements. Since the expected number packet for each block can be simply derived from SUM counters, we focus on how to learn the expected bitwise summation. Instead of directly using the bit counter values, the algorithm first attempts to separate the source IP addresses that generate a locally significant amount of packets in each block. The reason is that the bit counters can only provide aggregate information. The accuracy of the traffic profile of those significant devices can affect the accuracy of the expected numbers. For each specific source, the algorithm calculates its average number of packets. Then, the expected traffic profile is derived based on expected number of those specific sources and the left bit counter values.

2.4.2. Detecting and identifying heavy deviation

Before starting a detecting stage, the algorithm initializes the SUM counter to the negative expected number of packets. *k*th bit counter is initialized to be negative expected bitwise summation. When a packet arrives, its source IP is hashed to a bucket, and the value of the bucket counter is incremented by 1. Then, for each source IP, the SUM counter and each bit counter is incremented by the corresponding bit value. At the end of the detection stage, the algorithm applies the following process to recover the source IPs and estimate their changes. Finally, the heavy change IP addresses are identified according to the given threshold. The stage works is being as follow. The first step is to identify source IPs according to two redefined thresholds θ^+ and θ^-:

$$IP = \begin{cases} \sum_{j=0}^{31} 2^{31-j}, if\ count[n][j] > \theta^+ \times SUM[n] \\ Cannot\ be\ identified, if\ \theta^- \times SUM[n] \leq count[n][j] \leq \theta^+ \times SUM[n] \end{cases} \tag{2}$$

The thresholds are used to improve the accuracy of the identified IPs. In our measurements, we use $\theta^{+} = 0.55$ and $\theta^{-}=0.45$. The second step is to estimate the number of packets of the identified IP from the first step. Suppose that the IP is in block i, for each bit counter $count[i][j], j \in \{0,...,31\}$, we define its bit occurrence as:

$$C(j) = \begin{cases} count[i][j], if\ \theta^{+} \times SUM[i] \\ SUM[i] - count[i][j],\ otherwise \end{cases} \tag{3}$$

The third step is to update SUM counter and bit counters or the next recovery process:

$$SUM[i] = SUM[i] - Num(IP) \tag{4}$$

$$count[i][j] = \begin{cases} count[i][j] - Num(IP), if\ bit_j(IP) = 1 \\ count[i][j],\ otherwise \end{cases} \tag{5}$$

where $bit_j(IP)$ represent the value of the jth bit of the binary representation of IP. After that, we can recover the next IP address from the same bit counters. The number of packets $Num(IP)$ is estimated by the minimum bit occurrence: $Num(IP) = \min\{C(j)\}.0 \leq j \leq 31$. In the previous example, we can estimate the estimated frequency of item 4 is 30.

2.5. Entropy detection

In order to identify the IoT devices with small traffic volume, we combine a counting bloom filter (CBF) [25], [26] with a Bitcount. The CBF is to check the membership of IoT devices. As shown in Figure 3, the left hash table is used to collect traffic volume to calculate the entropy, and the bit counters are used to record the faulty devices' IP addresses.

The detection process consists of two stages: a detection stage and an identification stage. During the outage detection stage, the algorithm compares each bucket's entropy against its normal profile. When a bucket exhibits a behavior that is outside its normal range or profile, the algorithm moves to the identification stage to identify the faulty device.

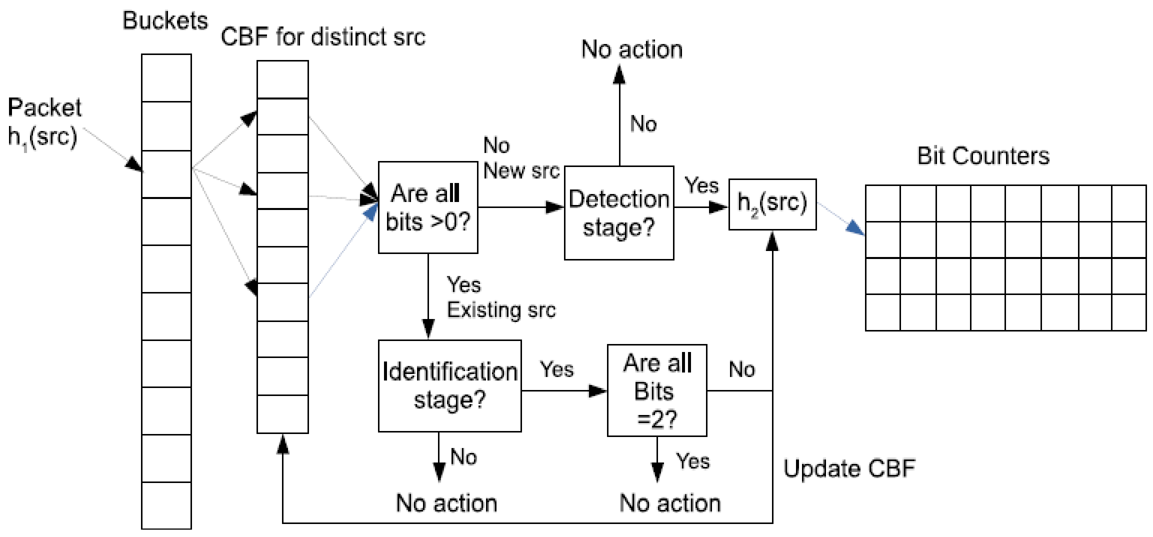


Figure 3. Implementaiton of the entropy based detection method. The counting bloom filter (CBF) is used to detect the first packet from a source IP at two dierent stages. The Bitcount is used to record the source IP addresses

2.5.1. Outage detection

The process consists of two stages: a detection stage and an identification stage. During the outage detection stage, each bucket's entropy is compared against a pre-defined threshold. When the value exceeds the threshold, the process moves to the identification stage.

During a detection stage, when a packet arrives, the CBF is used to determine if the packet has a new source IP address. If this is true, the source address is inserted into CBF and the bit counters. Thus, the bit counters remember all source IPs during the detection stage.

A sliding window containing the most recent elements is used to calculate the proposed entropy value. More specifically, the number of packets from each bucket is used to calculate its generalized information entropy. When the entropy value is lower than a given threshold, this bucket is reported as a suspicious bucket.

2.5.2. Outage identification

Once detecting a change in a bucket's entropy, the identification stage starts. Upon receiving a packet, the source IP address is checked in the CBF to see if it contains the address. If the source IP is not in the filter, the source is a new one and it will be discarded because we are interested in detecting the outages occurring in the existing sources. If at least one of the counter values from the CBF is one, the packet is identified as the first packet during the identification stage. The counters in the CBF are incremented by one. Next, the algorithm deletes the source IP from the bit counters. If all the counter values are 2, the incoming packet is a duplicated packet. At the end of the identification stage, the source IP address left in the bit counters is the faulty device.

3. MEASUREMENT RESULTS AND DISCUSSION

In this section, we evaluate the effectiveness of the proposed two methods. We first present our dataset. Then, we present the measurement results, and discuss the future work.

3.1. Dataset

We use two types of dataset to evaluate our methods. First, we evaluate the scalability and effectness of the enabler of our methods, Bitcount, by using sythetic datasets. We study the impact of various choices of parameters on the accuracy of data strucutre. We also use the synthetic datasets to compare our method with other implementations. We generate the synthetic datasets according to a Zipf distribution with different skewness (α=[0.8,2.0]). Each dataset is comprised of 70 M packets.

Second, we evaluate the effectiveness of our detection methods on real IoT traffic data from [24]. The IoT traffic are collected from a wide range of IoT devices. There are 96 IoT devices, including Smart Things, Amazon Echo, Samsung SmartCam, Dropcam, Sleep sensor, Smart plug, Smart Bulb. The IoT traffic trace contains 2.6 million IPv4 packets.

3.2. Performance of bitcount

First, we detect 10 different numbers of heavy keys (from 50 to 500) based on the synthetic datasets. For each case, we measure the lower bound of memory usage without false positives and negatives. The memory usage is determined by the array size of the number of buckets in Bincount. Each bucket in Bincount has 33 4-byte counters. From Figure 4 (a), we find that when the distribution is less skewed (α=0.8 case), we need more memory. On the other hand, when the distribution is more skewed, such as α=2, we need less memory. For example, we need less than 46 buckets to detect 100 heavy keys when α>1. However, we need 150 buckets when α=0.8.

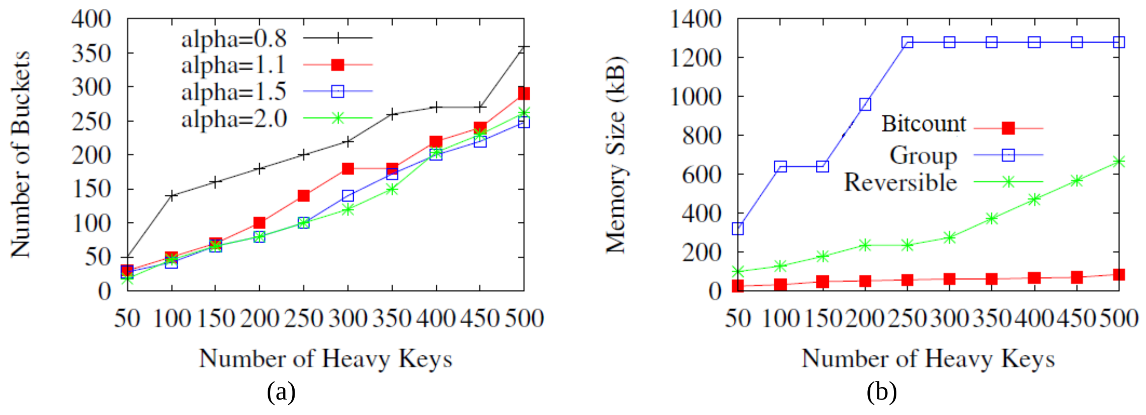


Figure 4. Measurement results based on the synthetic datasets; (a) the lower bound of number of buckets used in Bitcount to correctly detect and recover all heavy keys without false positives and negatives, (b) memory usage for three different approaches

Second, we compare the memory usage of Bitcount with that of reversible sketch [21] and group testing [22]. The memory usage includes the total memory space. We refer to the Reversible sketch as ``Reversible", and the group testing as ``Group" for brevity. Figure 4 (b) shows the comparison result. We find that our method always uses much less memory than the others. In order to detect all heavy keys, group testing has to use more than 1.28 MB memory, but our method only needs less than 85 KB. The memory benefit of our method becomes more impressive as the number of heavy keys grows. For example, when the number of heavy keys is increased from 50 to 500, the space requirement of our method grows by a factor of about 3.4, yielding a total size of about 85 KB. On the contrary, the space used by Reversible sketch are exploded by a factor 7.

From the measurement results from the synthetic data, we find that the enabler of our methods, Bitcount, maintains a small number of counters for faster and more accurate detection of heavy keys while incurring a small increase in memory usage.

3.3. Real IoT traffic measurements

In this subsection, we use the real IoT traffic to evaluate the effectiveness of our methods on detecting faulty IoT devices and IoT device outages in term of detecting latency. We use two malfunctioning scenarios. In the first scenario, we simulate a faulty IoT device by generating more packets than its typical traffic volume. This type of fault could be due to a malicious attacker who controls this device. We use the traffic deviation method to detect the scenario. The second malfunctioning scenario simulates an IoT outage by disconnecting the devices after running it for a certain time. The type of fault could be due to low battery. We use the entropy method to detect it.

3.3.1. Scenario 1: faulty devices

In the first malfunctioning scenario, we select a smart things, which typically generates less than 5% of packets during fault free operation. We increase the traffic volume from this device from 5% to 30% after 100th time bins (indicating by a dashed vertical line) as shown in Figure 5 (a). Since the traffic volume of the smart things is higher than its expected volume, the traffic deviation method can quickly detects this fault as a heavy change device. After the detecting stage, the method can identify the IP address of the device by using Bitcount, as shown in Figure 5 (b).

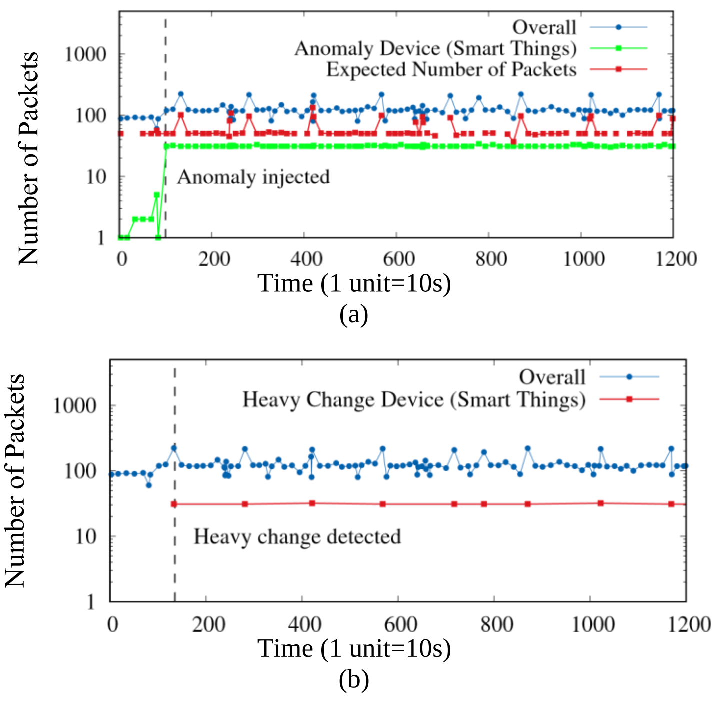
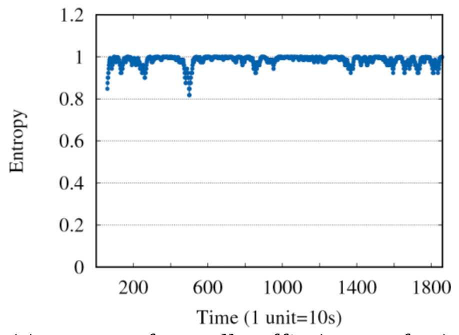


Figure 5. Malfunctioning scenario 1: detecting and identifying a faulty IoT device (smart things). Both figures only show the number of packets during active periods. The dashed vertical lines indicate when the fault starts and identified; (a) a faulty device (b) the faulty device is identified as a heavy change device

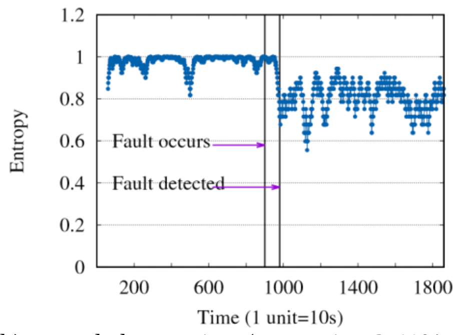
3.3.2. Scenario 2: IoT outages

In the second malfunctioning scenario, we simulate two outages occurring at two IoT devices. The IoT devices are 1) a smart baby monitor, which generates 3.41% of packets, and 2) a smart sleep sensor, which generates 1.85% of traffic volume. The sliding window is set to 60 inactive periods, and the threshold is set to 0.8. Figure 6 (a) shows the overall traffic (fault free). Figure 6 (b) and (c) show that our proposed detection method successfully detects the malfunctioning baby monitor after about 17 minutes, and the sleep sensor after 30 minutes.

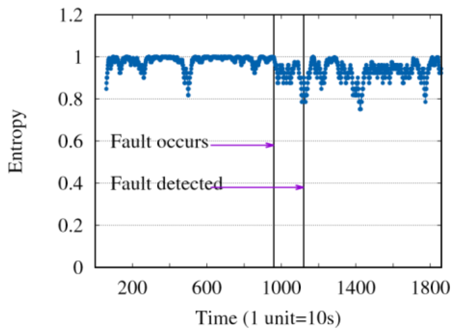
One of the factors that can affect the scalability of this method is the size of the Bloom Filter. The size of the Bloom Filter is determined by the number of unique source IP addresses and the false positive. In order to have a small false positive for a large number of IoT devices, we need a very large Bloom Filter. Therefore, we must consider how to design the Bloom Filter. One possible solution is to use multiple Bloom Filters. We will investigate how to effciently divide IoT devices into multiple filters.



(a) entropy of overall traffic (outage free)



(b) smart baby monitor (generating 3.41% of packets)



(c) smart sleep sensor (generating 1.85% of packets)

Figure 6. Measurement results of detecting two IoT outages; (a) the overall traffic without any outages. In (b) and (c), around 920th time bins (indicating by the left vertical line), the two devices are disconnected. The entropy detection method successfully detects the malfunctioning baby monitor after about 17 minutes, and the sleep sensor after 30 minutes (indicating by the right vertical line)

4. CONCLUSION

In this paper, we propose two fault and outage detection methods for IoT devices. Both methods are more space-efficient than the existing sketch-based method. The experimental evaluations show that the two methods can be used to detect and identify faulty IoT devices or IoT outages with less memory and computational overhead. Based on our analysis and measurements, our methods can be efficiently provisioned as a utility in IoT devices as a virtual service to detect IoT faults or outages.

REFERENCES

[1] P. R. J. Pêgo and L. Nunes, "Automatic discovery and classifications of IoT devices," *2017 12th Iberian Conference on Information Systems and Technologies (CISTI)*, 2017, pp. 1-10, doi: 10.23919/CISTI.2017.7975691.
[2] H. Ning, H. Liu, and L. T. Yang, "Cyberentity Security in the Internet of Things," *Computer*, vol. 46, no. 4, pp. 46-53, 2013, doi: 10.1109/MC.2013.74.

[3] J. Granjal, E. Monteiro and J. Sá Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1294-1312, thirdquarter 2015, doi: 10.1109/COMST.2015.2388550.

[4] I. Hafeez, A. Y. Ding, and S. Tarkoma, "IOTURVA: Securing device-to-device (D2D) Communication in IoT Networks," *Proceedings of 12th ACM Workshop on Challenged Networks, MobiCom CHANTS '17. ACM*, pp. 1-6, 2017, doi: 10.1145/3124087.3124093.

[5] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2177-2184, doi: 10.1109/ICDCS.2017.283.

[6] Younghee Park, S. Daftari, P. Inamdar, S. Salavi, A. Savanand and Youngsoo Kim, "IoTGuard: Scalable and agile safeguards for Internet of Things," *MILCOM 2016 - 2016 IEEE Military Communications Conference*, 2016, pp. 61-66, doi: 10.1109/MILCOM.2016.7795302.

[7] E. W. Dereszynski, Thomas G. Dietterich, "Spatiotemporal Models for Data-Anomaly Detection in Dynamic Environmental Monitoring Campaigns," *TOSN*, vol. 8, no. 1, pp. 3-36, 2011, doi: 10.1145/1993042.1993045.

[8] Jianliang Gao, J. Wang and X. Zhang, "HMRF-based distributed fault detection for wireless sensor networks," *2012 IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 640-644, doi: 10.1109/GLOCOM.2012.6503185.

[9] P. K. Malik, R. Singh, A. Gehlot, "Online Monitoring of Solar Panel Using I–V Curve and Internet of Things," *Proceedings of First International Conference on Computing, Communications, and Cyber-Security (IC4S 2019)*, 2020, vol 121, pp. 225-234, doi: 10.1007/978-981-15-3369-3_17.

[10] F. Wang and L. Gao, "Simple and Efficient Identification of Heavy Hitters Based on Bitcount," *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*, 2019, pp. 1-6, doi: 10.1109/HPSR.2019.8808101.

[11] G. Cormode and S. Muthukrishnan, "What's new: finding significant differences in network data streams," in *IEEE/ACM Transactions on Networking*, vol. 13, no. 6, pp. 1219-1232, Dec. 2005, doi: 10.1109/TNET.2005.860096.

[12] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.C. Chen, and G. Zhang, "SketchVisor: Robust Network Measurement for Software Packet Processing," *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 113-126, 2017, doi: 10.1145/3098822.3098831.

[13] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI)*, pp. 311-324, 2016, doi: 10.5555/2930611.2930632.

[14] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 101-114, 2016, doi: 10.1145/2934872.2934906.

[15] T. Yang, *et al.*, "Elastic Sketch: Adaptive And Fast Network-Wide Measurements," *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 561-575, 2018, doi: 10.1145/3230543.3230544.

[16] G. Cormode and M. Hadjieleftheriou, "Finding Frequent Items In Data Streams," *PVLDB*, vol. 1, no. 2, pp. 1530-1541, 2008, doi: 10.14778/1454159.1454225.

[17] R. Berinde, G. Cormode, P. Indyk, and M. J. Strauss, "Space-Optimal Heavy Hitters With Strong Error Bounds," *Proceedings of the Twentyeighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 157-166, 2009, doi: 10.1145/1559795.1559819.

[18] W. Feng, Z. Zhang, Z. Jia, Z. Fu, "Reversible Sketch Based on the XOR-Based Hashing," *Services Computing 2006. APSCC '06. IEEE Asia-Pacific Conference on*, pp. 93-98, 2006, doi: 10.1109/APSCC.2006.91.

[19] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC'03)*, vol. 10, pp. 234-247, 2003, doi: 10.1145/948205.948236.

[20] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. "Reversible Sketches for Efficient and Accurate Change Detection over Network Data Streams," *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC'04)*, pp. 207-212, 2004, doi: 10.1145/1028788.1028814.

[21] R. Schweller *et al.*, "Reversible Sketches: Enabling Monitoring and Analysis Over High-Speed Data Streams," in *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1059-1072, Oct. 2007, doi: 10.1109/TNET.2007.896150.

[22] G. Cormode and S. Muthukrishnan, "What's Hot and What's Not: Tracking Most Frequent Items Dynamically," *ACM Trans. Database Systems*, vol. 30, no. 1, pp. 249-278, 2005, doi: 10.1145/1061318.1061325.

[23] N. Apthorpe, D. Reisman and N. Feamster, "Closing the Blinds: Four Strategies for Protecting Smart Home Privacy from Network Observers," *CoRR*, vol. 1705, 2017.

[24] A. Sivanathan, *et al.*, "Characterizing and classifying IoT traffic in smart cities and campuses," *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 559-564, doi: 10.1109/INFCOMW.2017.8116438.

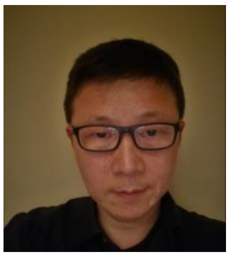
[25] B. Bloom, "Space/Time Tradeoffs in Hash Coding With Allowable Errors," *Communications of the ACM*, vol. 13, no.7, pp. 422-426, 1970, doi: 10.1145/362686.362692.

[26] Li Fan, Pei Cao, J. Almeida and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," in *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281-293, June 2000, doi: 10.1109/90.851975.

BIOGRAPHIES OF AUTHORS

Feng Wang is a professor with the School of Engineering and Computational Sciences at Liberty University. He received his Ph.D degree in Electrical and Computer Engineering at the University of Massachusetts, Amherst. He received his B.E degree from Zhejiang University in China, and M.S. degree from Yanshan University in China. His research interests include Internet of Things, Internet routing, and networks security.

Eduard Babulak is a professor and professional engineer (P.Eng.) with School of Business and Computet Science at Liberty University. He received his Ph.D., degree in Computer Science at the Staffordshire University in UK, Doctor-Habilitated (Docent D.Sc., degree) in Information Technology at University of Pardubice in Czech Republic, M.Sc. in Information Systems degree at University of East London in UK, High National Certificate in Computer Aided Engineering at Brighton College of Technology in UK, and degree in Electrical Engineering at Technical University of Kosice in Slovakia. His research interests are in the field of Computing, Data Networks, Cybersecurity, the IoT, ICT eServices, Future Internet and Smart Cyberspace. He is Life Chartered Fellow, Mentor and Elite Group Member of the British Computer Society and Fellow of Royal Society RSA in London, UK. He is Mentor and Senior Member of the IEEE and ACM, Chartered Engineer and Professional Member of the IET in London, UK, Professional Member of the American Society for Engineering Education (ASEE) & American Mathematical Society (AMS). He serves as Editor & Reviewer for number of prestigious IEEE, Elsevier, Springer, Hindawi & MDPI, plus Journals.



Yongning Tang is a professor with the School of Information Technology at Illinois State University. He received his Ph.D degree in Computer Science from DePaul University. He received his B.E. and M.S. degrees, both from Hefei University of Technology in China. His research interests include fault diagnosis in large-scale distributed networking systems, cloud computing, network security, wireless networks, and intelligent defined networking.