Computer Networks and Communications
https://ojs.wiserpub.com/index.php/CNC

UNIVERSAL WISER PUBLISHER

Article

Detecting Application-Level Associations Between IoT Devices Using a Modified Apriori Algorithm

Juan Benedicto L. Aceron, Marc Elizette R. Teves and Wilson M. Tan*

Department of Computer Science, University of the Philippines Diliman, Quezon City, Philippines
E-mail: wmtan@dcs.upd.edu.ph

Received: 22 June 2023; **Revised:** 2 September 2023; **Accepted:** 15 September 2023

Abstract: Internet of Things (IoT) for home systems enables new functionalities and results in significant conveniences. However, their reliance on a stable, continuous Internet connectivity reduces their overall reliability. Losing connectivity to the Internet, for many of these devices, translates to the cessation of even the most basic of functionalities (e.g., being able to turn on the light, even from within the house). A possible solution to this problem is to shift some functionalities done by cloud-based servers to the edge (e.g. the home router), but doing so conveniently would necessitate the ability to dynamically identify pairs of IoT devices (usually sensor-actuator pairs) that communicate (their associations), and the rewriting/rerouting of packets or messages between those devices. The first problem is also known as the association detection problem, and is where this paper makes a contribution. We describe a solution to the association detection problem using a modified Apriori algorithm and a method to create its input from network traffic, and then revise the solution to respond to fluctuating network conditions. The final design accurately discovers sensor-actuator pairs using a simple approach with low computational complexity, and with only the hardware addresses of monitored IoT devices as its starting knowledge.

Keywords: IoT, resiliency, edge computing, cloud computing, association rule learning, SDN, Apriori algorithm

1. Introduction

As of 2018, the number of Internet of Things (IoT) devices was estimated to be around 7 billion, and this number is expected to reach 22 billion devices by 2022 [1]. Most of these devices have simple computing hardware, with most of them being simple sensors or actuators. Edge computing devices, such as an IoT hub, or more commonly, cloud servers managed by the device manufacturer, handle services such as automation, data collection, and system integration. This simplicity leads to affordability, albeit at the cost of losing some functionality when the Internet connection goes down or is unstable. This is sometimes true even for the simplest of operations which only require local information, such as turning on a smart oven from a mobile device [2]. Even for such a simple operation, the command actually travels from the mobile device, to a cloud server, to another cloud server, and finally back to the smart oven.

A lack of one open standard governing how devices made by different manufacturers should remain inter-operable is the primary culprit for this arguably poor communication model. Without a de-facto open communication standard, manufacturers instead opt to make proprietary and even ad hoc approaches to compatibility [3]. At times, the message format used by a control device may not be understood by the actuator, so inter-operability between them is facilitated by cloud servers.

Even major IoT manufacturers and software providers such as Amazon, Apple, and Samsung have incomplete solutions to the problem. When faced with the loss of the Internet connection, none of them can even

DOI: https://doi.org/10.37256/cnc.1220233263

provide simple functionalities such as changing the state of a device from a mobile phone in the local network [4–7].

The Open Connectivity Group with their IoTivity [8] framework project is an attempt at a solution to this problem. The IoTivity framework is an open protocol which can potentially be followed by all IoT manufacturers. Once adopted by IoT manufacturers, the services provided by their cloud servers can, in theory, be implemented in the local network. However, IoTivity adoption is still not widespread enough [9]. As such, an approach which would not need a large-scale collaborative effort may be more desirable.

One alternative approach is to predict and replicate the cloud server's response to an IoT device's request, locally. To use this approach, however, there must first be a method to detect the *associations* between IoT devices, or sensor-actuator pairs, that exist in the network. This is the association detection problem, and a solution to it will be the main concern of our paper.

The association detection problem has similarities to the problem of finding hidden connections in a Tor network. Flow correlation techniques figure prominently as proposed solutions to this problem [10], and notable examples include [11] and [12]. These examples utilize computationally expensive deep learning techniques, which cannot be run efficiently or practically on the hardware that most IoT networks are expected to have. However, we borrowed and adopted their use of inter-packet delays and flow directions in our feature set.

The Apriori algorithm was used by Huang et al [13] in discovering spatio-temporal relationships between IoT services, a problem which has similarities to association detection. Unfortunately, the Apriori algorithm has a time and space complexity of $O(2^n)$ given n transactions. For our work, we also rely the Apriori algorithm, but since we are only concerned with detecting pair associations, we were able to modify it and reduce the resulting complexity down to $O(n)$.

This modified algorithm will be applied to a list of *buckets* generated using two parts: a stream of captured packets coming to and from monitored IoT devices, and a dynamic *threshold* that will pair those packets into buckets. The correctness of the buckets generated will depend on the suitability of the threshold, so arriving at an appropriate value is key to getting accurate results. Several candidate associations will be generated from this analysis, and the subset of candidates with the highest *lift* scores will be selected as the most likely associations.

To emphasize, the point of finding associations between IoT device pairs (usually sensor-actuator pairs; e.g., smart button and smart light) is to eventually be able to rewrite (and reroute) packets sent by the IoT devices so that they communicate directly or through an edge device (at least during period of no Internet connectivity) instead of communicating through a cloud-based server.

The main contributions of the work are as follows:

- The general idea of providing resilience in an IoT system by rewriting (and rerouting) packets between sensor-actuator pairs so that total dependence on a cloud-based server (and the Internet) is broken
- The identification of the association detection problem as crucial initial step in the realization of the idea mentioned above
- An application of the Apriori algorithm to the association detection problem mentioned previously
- A method to dynamically adjust the optimal threshold of the Apriori algorithm so that it keeps up with changing network conditions

The rest of the paper is structured as follows. Section 2 discusses work related to the paper. Section 3 provides an overview of the methodology used in this work. Section 4 describes our solution to the association detection problem. Section 5 discusses the performance of our solution, as well as its current shortcomings. Section 6 describes solutions to the shortcomings discussed in Section 5. Finally, Section 7 contains a summary of our findings, as well as recommendations for future work.

2. Related Work

We will briefly describe the network environment of our system by borrowing terms as used in Dorsemaine et al [14]. The environment we describe in this paper is composed of two classes of *objects* - *sensors* and *actuators*. They are both directly connected to a *transport* (the switch), with no *pickup point* (a hub) in between the transport and the objects. Messages travel from sensors to actuators exclusively. Both object classes must connect to cloud servers before messages can be sent or received. Cloud servers sit in a critical position between sensor to actuator communication; that is, the objects cannot communicate with each other in the absence of these cloud servers.

In section 2.1 we will discuss the association detection problem and prior research related to it. Section 2.2 will discuss research related to providing IoT network resiliency. Section 2.3 will introduce the Apriori algorithm, the main algorithm used in our solution to the association detection problem.

2.1 *Association Detection*

We define an *association* as a pair containing a sensor that communicates with an actuator. Information about all associations present in an IoT network are not given to observers within the local network. The *association detection problem* is what we call the task of finding hidden associations. In this subsection, we will explore prior research closely related to the association detection problem.

Motivated to allow users to generate insight about the behavior of IoT devices in the home, [13] proposes a novel algorithm that identifies groups of related IoT services in a network using spatio-temporal data from these devices. These composite IoT services are inferred from a database of usage data recorded from multiple IoT devices across a network. Each datapoint describes a period of activity for a specific device, as well as a stationary location described as a point in 3-dimensional space. The first part of their algorithm is based on the Apriori algorithm, modified to reduce the creation of item sets that fall under a minimum support value. The result of this first part is a set of candidate composite IoT services. The second part of the algorithm calculates the proximity value of each candidate and filters out any candidates with a proximity value under a specified target. The aim specifically being to filter out candidates under a minimum support and proximity, and not trying to identify groups of devices which are known beforehand to be related. The approach to association detection still relies on using the Apriori algorithm on a large dataset grouping multiple IoT services together. Because of this approach, even near real-time analysis using their methods is not possible.

The Tor network is an anonymizing network that uses onion routing to hide the identity of communicating pairs. Flow correlation is an active field of study on the problem of attempting to find communicating pairs despite the obfuscation of anonymizing networks [10]. The problem of flow correlation closely resembles our association detection problem - both are trying to identify communicating pairs by analyzing network traffic at the gateways that the communicating devices use. Due to this similarity, it is worthwhile to look at the prior work in this field.

The AttCorr [11] and DeepCorr [12] models are used to correlate flows hidden through the Tor network. Raw traffic features, such as packet sizes, flow directions, and inter-packet delays are extracted from ingress and egress flows. The feature set collected by AttCorr and DeepCorr matches features collected by our association detection component, except that we do not use packet sizes. Associated and unassociated flows are then used to train the models. AttCorr uses an attention-based deep learning model, DeepCorr uses a convolution neural network, while our approach does not use expensive to train deep learning models of any sort. We opted for a simpler approach because while Tor actively attempts to obfuscate the association, our environment only hides the association simply because it is not explicitly known. Due to a lack of the adversarial nature found in the Tor environment, and because there is no need to analyze and collect large amounts of data, we do not use a deep learning model for association detection.

The above flow correlation attacks are passive in nature. Active correlation is also possible via watermarking, that is, tampering with flows in order to create noticeable effects on either side of an associated flow. RAINBOW [15] watermarks flow by introducing artificial delays on the order of milliseconds to packets crossing their controlled routers. These delays are so small that they are virtually invisible to defenders against this correlation attack, since they are on the same order of duration as regular jitters on the Tor network. If another flow is observed to have the same delay pattern as a manipulated flow, then those two flows are correlated. Like our approach, RAINBOW analyzes the timing of when packets are sent and when they are received. Unlike RAINBOW, our approach is not active, and does not tamper with the flows during association detection in order to avoid possible side effects. This also avoids another layer of complexity, which is detrimental to the reliability of a resiliency system.

2.2 *Resilient IoT Networks*

This section will discuss attempts to create resilient IoT networks that will take over cloud service functions in the event of an unreliable network connection. The common theme with these papers is that they still require the cooperation of the manufacturers of the devices as well as their respective cloud server providers. Our solution to the association detection problem may pave the way for a system that can provide reliability regardless of manufacturer of an IoT device.

The RES-Hub is a device proposed by [16] designed to fit in a framework in which it will act as a fallback for IoT-connected devices in the case that the home is disconnected from the internet. Listed in the paper are the

minimum functionalities that their implementation requires, such as a method for detection if the connection to the internet is down, a method of notifying the home owner that the system has lost connection to the internet, as well as specifications for the security of the system. The RES-Hub is able to do this because of its persistent mobile connection to the internet for authentication needs, as well as a local database to store the states of the various IoT devices. Although this is a mechanism for achieving resiliency, it ultimately depends on the availability of a mobile connection. Furthermore, it relies on the adoption of an open connectivity framework, putting the burden of compliance on manufacturers of IoT devices.

Ride is a middleware edge-computing architecture proposed in [17] that will allow for an IoT system to continue functioning in the event of an unreliable internet connection. The system continuously collects network information when in normal operation, and will construct updated optimized routes. With an unreliable internet connection, the system aims to divert device traffic along these routes in to allow IoT devices to continue functioning. Although there is no association detection component, this architecture requires the cooperation of the maintainers of the IoT devices to achieve resiliency. New IoT devices need to be configured with an edge service that will act as a fail-over in case their normal servers cannot be reached. This need for configuration is an overhead that can be made obsolete by an automatic association detection system.

Another example of a resilient framework for IoT systems is proposed by [18]. This approach leverages the strengths of fog computing to create a system where the states of various configurable IoT devices can be saved in the event of a failure scenario, and can be restored once the problem has been addressed. This is possible by using low-powered devices distributed in the network called fog nodes, which act as administrative entities, which act as mini-hubs to which, IoT devices will connect. In addition to this, various APIs are needed to allow developers to interact with this system. Although this approach introduces resiliency, it still requires manufacturers to build this support into their devices, highlighting the need for a manufacturer-agnostic resiliency approach.

On the other hand, in a paper by [19] its authors propose an architecture that for efficient fault-management at different levels of edge-computing ranging from the cloud, to the mist (extreme edge) where edge devices are very close to the end system. The architecture emphasizes the importance of replicating data so as not to lose any information which may be critical to IoT systems that rely on sensors. The described system can switch to different levels of edge computing (from mist to fog) depending on the delay tolerance of the IoT application.

To address the unstable internet connectivity in some areas, [20] propose a method where a specific IoT application can still remain available despite an unstable internet connection. The application that they demonstrated this on was an e-voting system, and the methods they propose might not be applicable to all cases. The main mechanism to achieve resiliency is the fact that the system buffers messages that are intermittently synchronized with the online database. The most important parts of their proposed solution are: a local database for temporary data storage; the Mosquitto (MQTT) broker for forwarding messages; a low-powered devices (Intel Galileo) to host some web-pages for the application. However, when an internet connection is lost for an extended period, the system will lose all functionality.

Our vision or long-term goal for IoT resiliency has similarities to self-organizing networks. Instead of automatically deciding which end nodes or devices are connected however, the system (in the long run) aims to "shortcut" the connection or communication between pairs, especially during situations when the connection to the cloud or Internet is down or unavailable. There have been several recent notable works in the field of self-organizing networks. Dou et al [21] presents a routing algorithm for self-organizing networks, which is based on link reliability, and uses a Markov decision process as well as Q-learning in its computation. Romanov et al [22] proposes the use of a self-routing algorithm, originally designed for use in WSNs (wireless sensor networks), in NoCs (networks-on-chip). Daas et al [23] discusses a multi-sink routing protocol for self-organizing wireless networks. Hamrioui [24] proposes a cross-layer approach for IoT devices to self-organize and self-configure their communications.

Our approach to IoT resiliency features ideas from edge computing, which has seen several interesting works in recent years. Anees et al [25] discusses edge computing and how it can benefit IoT (to be more specific, WoT, or Web of Things). The paper mentions the benefits outside of resiliency (which we are aiming for, long term, in our work) - improved bandwidth, minimized latency, lower energy, and lower cost. Rawashdeh et al [26] presents an algorithm which predicts a service's optimal host (cloud, fog, or edge), to aid in the migration of the said service. The algorithm also takes into account host power levels, and has the long-term goal of minimizing service latencies for IoT devices. This concept or approach has tangential similarities with our long-term vision for resilient IoT, though the location of the service is not of our concern. We always assume that the service(s) is/are hosted in a fog-like system, and the challenge we are tackling long-term is automatically adapting the service(s) to the IoT devices installed in the system.

2.3 *Apriori Algorithm*

The *Apriori algorithm* [27] is used for association rule learning. Frequently occurring itemsets are extracted from a database of transactions to obtain association rules. The Apriori algorithm normally has an $O(2^n)$ time and space complexity for a database with n transactions, but this is only true for when the size of the desired itemset is unknown. If the size of the desired itemset is fixed, like in our system (itemsets are fixed to two), the complexity is drastically reduced to $O(n)$. A version of the Apriori algorithm with a fixed itemset size is the main driver of our solution to the association detection problem.

An improvement to the Apriori algorithm proposed by [28] optimizes the algorithm for datasets with a temporal dimension. The main contribution is the concept of time-interval expansion and mergence. Meaning, discrete itemsets with overlapping time intervals can be coerced to combine with each other, reducing the total number of calculations to be done for the algorithm.

The Apriori algorithm has featured in several IoT systems before.

Yan et al [29] describes a smart shopping system based on a cart with a Raspberry Pi board, an RFID reader, and a monitor display, as well as RFID tags attached to the products. The system features a product recommender system (which takes its inputs from the RFID reader) based on fuzzy logic and Apriori system. While our work also features the Apriori algorithm, we use it for detecting associations between IoT devices, not associations between products or items.

Du et al [30] notes that data mining algorithms are important in generating value out of the Big Data produced by IoT systems. One well-used algorithm which they identified is the Apriori algorithm. Aside from running an energy analysis of the Apriori algorithm as it is used in a cloud platform, they also presented a modification of the
algorithm based on Boolean matrix and sorting index rules. While we also modified the Apriori algorithm in this work, we did so for it to fit the nuances of our application, and the said modification may have limited application out of our work, as opposed to the more general nature of their proposed modified algorithm.

We note that there are alternatives to the Apriori algorithm when it comes to association rule learning. The most notable of these alternatives are the FP-growth algorithm and the Eclat algorithm.

The *FP-growth* or *Frequent Pattern Growth algorithm* [31] uses a tree-like structure called Frequent Pattern Tree to find frequent itemsets. The algorithm repeatedly scans the dataset to add items in the tree in the order of their support. A conditional FP-Tree is generated by traversing the tree from the bottom. The conditional FP-Tree leads to the generation of the frequent itemsets. The FP-Growth algorithm makes fewer passes on the dataset and does not require candidate generation; as such, it is more scalable than the Apriori algorithm. However, it is fairly complex and may not fit in the memory of memory-constrained machines.

The *Eclat (Equivalence Class Clustering and bottom-up Lattice Traversal)* [32] algorithm differs from the Apriori algorithm and the FP-growth primarily in the sense that it works on “horizontal” datasets, as opposed to “vertical” datasets. The dataset is only scanned by the Eclat algorithm once, and it finds frequent itemsets by taking the intersection of the transaction sets. A possible disadvantage of the Eclat dataset mani-fests when the dataset is large, since transaction IDs associated with each itemset may become very long. As such (intermediary) results can become larger than the original dataset and may no longer fit into the memory.

The FP-growth algorithm and the Eclat algorithm are more complex algorithms than the Apriori algorithm, and both basically trade off (memory) space for speed. The Apriori algorithm was chosen as the association rule mining technique for this work for its relative simplicity, although both the Eclat algorithm and the FP-growth algorithm can be modified for use as an alternative to the Apriori algorithm, if so desired.

3. Methodology

The experimental network environment is described first before the actual association detection system. Un-monitored and monitored hosts connected to a switch with Software-Defined Networking (SDN) capabilities primarily comprise our local area network. SDN [33] is used in this project to program necessary additional behaviors into the network, such as filtering packets matching specific characteristics, cloning packets, and redirecting packets to destinations they would not normally be routed to. Instead of adding these behaviors at the hardware level, SDN allows for them to be defined and installed by writing software.

All hosts are internet-connected, although the monitored hosts are the only ones that are relevant, as they are stand-ins for IoT devices whose associations we are interested in finding. The monitored hosts are divided equally into sensors and actuators.

An application-level association exists between a sensor and a specific actuator. The sensor sends application-level messages to the actuator, but the communication does not occur over shortest possible route

(i.e., directly, with the switch as the only hop). Each monitored host has an associated *remote server* (which we also call *remote*), which maintains a Websocket [34] connection. When a sensor communicates to an actuator, the sensor does so by sending a message to its remote server, which then forwards the message to the actuator's remote server, which forwards the message down to the actuator back in the sensor's local network. This flow or sequence of events can be seen in Figure 1.

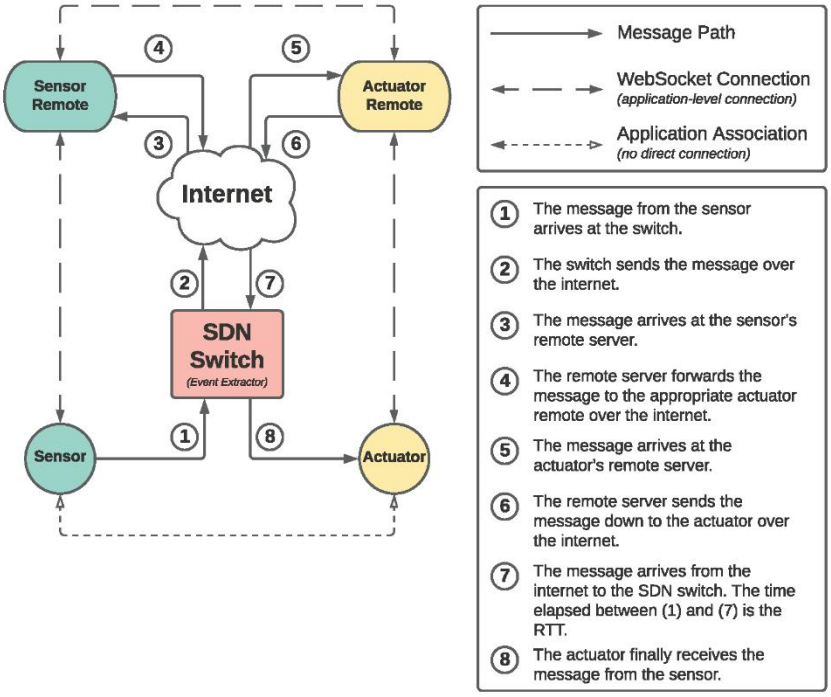

Figure 1. Message Paths

Sensors periodically send messages to an actuator, after waiting for a period between a minimum wait time and a maximum wait time. All sensors share the same minimums and maximums. However, the actual wait time differs from message to message and from sensor to sensor. Here we define the round-trip time, or RTT as the amount of time between a sensor's message arriving on the SDN switch, and the triggered response from an actuator's remote arriving on the SDN switch.

We note that while we did not use any actual IoT devices in our experiments, we used hosts (laptops and desktop computers) programmed to behave as how actual IoT devices likely would (i.e., we used the same network protocols that IoT systems would likely use), thus resulting in message exchanges or network traffic that is similar to what would be observed in an actual IoT system. We decided against using a simulator such as NS-3 [35] to ensure the realism of the network traffic. The presence of anomalous events (to be discussed later), for instance, is something that may have been missed by an NS-3-based experimental setup. Likewise, we decided against the use of emulation systems such as mininet [36] to avoid performance fidelity issues that arise when the emulation platform becomes severely oversubscribed.

4. Design

Three subsystems comprise the proposed solution, but this section will only describe the first two. Figure 2 illustrates the system. The *event extractor*, utilizes programmed SDN behavior in the controller and the switch to dissect and analyze traffic from the *monitored hosts*, and turns certain, specific packets into *network events*, or simply, *events*. The events generated by the subsystem flow into the *association detector*. Continuously taking in events from the event extractor, the association detector pairs them into buckets using some *threshold* as a basis. It leverages a real-time modified Apriori algorithm on the collected buckets to identify *associations*.

The quality of the results from the *association detection system* are affected by fluctuating network conditions, because network conditions effectively change the optimal threshold. This phenomenon motivates the existence of the third subsystem, called the *threshold optimizer*. Using a set of the most recently recorded

events to infer the proper parameters for the association detection system, the threshold optimizer ensures that the system's outputs and results stay accurate.

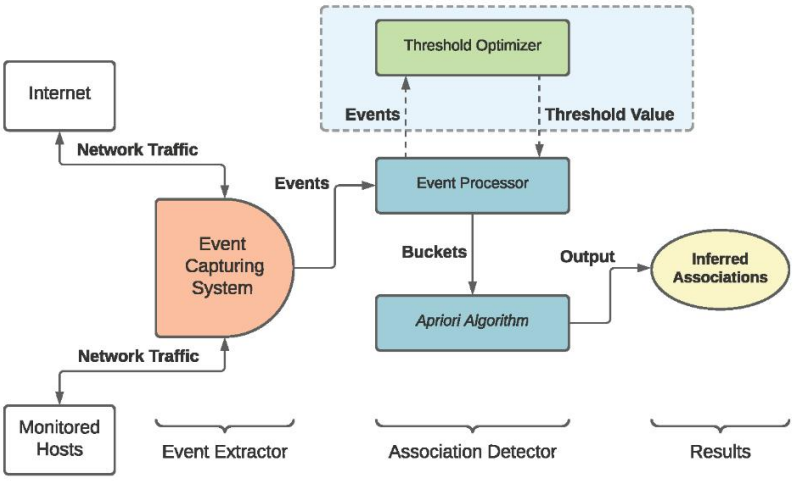

Figure 2. System Diagram

4.1 *Event Extractor*

The event extractor monitors packets that pass through the gateway switch, filters packets that match a certain criteria, extracts relevant features, and converts those features into network events (or simply, events).

4.1.1 Location of the Event Extractor on the Network

Tables 1 and 2 detail the monitoring flows which are installed on the SDN controller. The flows are installed for each monitored host.

Table 1. Egress Monitor Flow

Match	Action
in_port = HOST_PORT eth_dst = GATEWAY_ETH eth_src = HOST_ETH	send_to_port (DEST_PORT) send_to_port (CONTROLLER)

Table 2. Ingress Monitor Flow

Match	Action
in_port = WAN_PORT eth_dst = HOST_ETH eth_src = GATEWAY_ETH	send_to_port (DEST_PORT) send_to_port (CONTROLLER)

The egress monitor flow captures packets that flow from a sensor to the internet; the ingress monitor flow, on the other hand, captures packets that flow from the internet to an actuator. Packets that match these flows are sent to their intended destination, although duplicates of the packets are also created and forwarded to the controller for analysis.

4.1.2 Converting Packets to Events

At the controller, the packet proceeds to the next phase of processing if and only if satisfies two conditions. The first condition is that it has a TCP header. The second condition is for its payload to be non-empty TCP. Admittedly, this method of matching the application level messages sent to and from monitored hosts is not without flaws. The method will also match irrelevant messages, such as those sent to unrelated hosts over the internet. However, in our setup, the hosts were setup to only communicate with their corresponding remote servers. As such, our setup avoids the problems brought about by stray matches.

Packets that fail the criteria are filtered out. For each packet that survives the filtering stage, we extract three relevant features:

1. Address - address of the monitored host which the packet is either sent from, or sent to.

2. Direction - the direction is up if it matched on the egress flow. It is down if it matched on the ingress flow.

3. Time - Unix timestamp of when the controller received the packet.

Every 3-tuple of (*address*, *direction*, *time*) is called a network event, or event for short.

4.2 *Association Detector*

The association detector works off a constant event stream generated by the previous subsystem. Using a threshold, the association detector pairs events into buckets; in addition, it also continuously updates a modified Apriori model in order to infer the correct associations within the network. The central intuition behind the approach is that whenever up and down events happen close enough in time, it increases our confidence that the hosts involved are truly associated with one another.

4.2.1 Bucketing

In our modified version of the Apriori algorithm, we use special transactions which we call *buckets*. Single items feed into the association detector; as such, buckets must be formed from these items before the Apriori algorithm can use them.

A 2-tuple of *sensor, actuator* comprises a bucket. The sensor device's ethernet address forms the first element. The ethernet address of the actuator device comprises the second element. A bucket, therefore, is effectively a representation of two events that happened close enough in time.

Every time an up event occurs, we allow a span of time where any down event that occurs is bucketed with that up event. The span of time's length is set by the threshold. At any time, there can be multiple spans that are active. We note that an up event being bucketed does *not* expire the span; it will only expire once the threshold has been reached or has elapsed. As such, a single up event can be part of several buckets; this possible multiple pairing also holds true for down events. Figure 3 illustrates the process of bucketing.

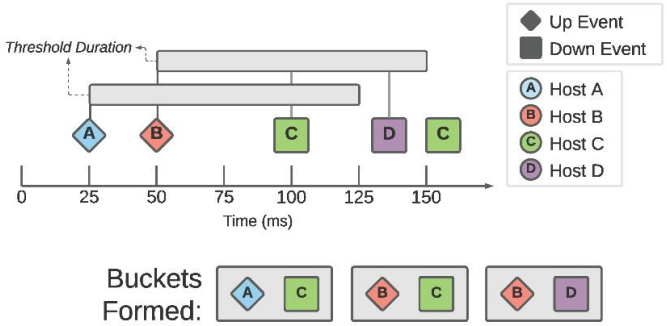

Figure 3. A timeline of 5 events recorded by the system. The threshold is 100ms. In this example, the list of buckets recorded would be (A,C), (B,C), and (B,D).

We do not form buckets if the up event has the same address as the down event. Ideally, a device is either a sensor, in which case it never sends a down event, or an actuator, in which case it never sends an up event. However, in the course of running our experiments, we observed situations or sequences that defied this ideal situation. We would later discuss remedies or solutions for these non-ideal situations.

The association detector discards up events after the relevant thresholds have passed; likewise, down events are discarded by the association detector after they are bucketed as per the spans they are part of. Nevertheless, events are still kept in memory, and would later form a critical part in the operation of the threshold optimizer subsystem.

4.2.2 Modified Apriori Algorithm

The Apriori algorithms works off a database of pairs, and our only concern is finding pairs. As such the algorithm can be modified to rin in real-time, as opposed to operating in batches. This has the tremendous advantage of improving its complexity from $O(2^n)$ to $O(n)$.

Three values can be derived from each possible sensor-actuator association, or S, A (note that $Count(X, Y)$ is the total number of buckets that contain both X and Y):

1. $Confidence(S, A) = \dfrac{Count(S, A)}{Count(S)}$

2. $Lift(S, A) = \frac{Confidence(S, A)}{Support(A)}$
3. $Support(X) = \frac{Count(X)}{Totalbuckets}$

To handily derive these values, when a bucket (S, A) comes into existence, we increment by 1 four values:

1. $Count(S, A)$
2. $Count(S)$
3. $Count(A)$
4. $Totalbuckets$

Post-incrementation, the association detection subsystem can actually purge the bucket from memory as it is no longer necessary. With the values now available, deriving the *Lift*(S, A) for any (S, A) should not be troublesome computationally.

It must be noted that in the Apriori algorithm, the lift of a rule corresponds to the correlation of the precedent to the antecedent. In other words, it is how much more likely it is that an up event from one host would appear to cause a down event with another host. A positive correlation can be deduced from a lift value (significantly) greater than 1; on the other hand, a negative correlation can be deduced from a lift value that is less than 1. A lift value which is close to 1 implies the lack of any causal relationship.

At most ${}_nP_2$ candidate associations will exist in a network with n pairs of communicating IoT devices. We infer that the n candidates with the highest computed *Lift* scores have genuine associations. The greater the difference in *Lift* between the inferred real associations and the rest of the candidate associations, the better the confidence that these are indeed the real associations.

4.2.3 Filtering Out the Effects of Anomalous Events

We define *anomalous events* as up events associated with an actuator address, or down events associated with a sensor address. Ideally these should not be produced by the event extractor; however, the naturally bidirectional nature of communication between hosts and servers (e.g., keep-alive packets) means that they *will* be observed from time to time.

Ideally, anomalous events must be removed since they do not help within the functioning our model. However, this elimination is neither straightforward nor easy, because we lack Apriori knowledge regarding which hosts are sensors and which hosts are actuators. Fortunately they have obvious effects on the results of the system's predictions, so a more effective solution is to eliminate their effects on the predictions, as opposed to eliminating the events early.

Anomalous associations stem from events which are anomalous. They are backed by relatively unlikely events, which can be characterized or understood as events with low support values. It is helpful to be reminded that *Lift*(S, A) is negatively correlated with *Support*(A). Anomalous events usually have support scores that are extremely low. As such, to eliminate the effects of anomalous events, we can simply filter out any candidate (S, A) if *Support*(S) or *Support*(A) is lower than a support limit. In this work, we used the minimum support of 0.1 to filter out the effects of anomalous events.

4.3 *Effectiveness of the System*

The system was run on a setup with three communicating host pairs. The hosts are in a local network, and they have associated cloud-hosted remote servers. Figure 4 shows the result of the test or experiment.

It can be seen that over time, as data is collected by the system, the average of the lift values from real candidate associations stabilizes under 2.5, indicating a strong positive correlation. As for false candidate associations, the average of their lift values falls to 0.50, signalling strong negative correlation. If configured to only show candidate associations with lift values above 2.00, the system will stably return the correct candidate associations after 50 seconds.

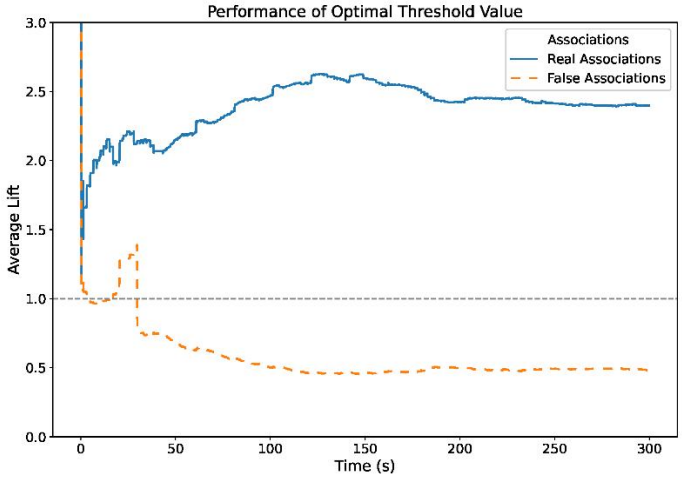

Figure 4. Threshold = 0.25 seconds

5. Flaws of the Original Design

While the design described so far works in certain conditions, it is, in general, still flawed. Its weakness stems from the threshold period being fixed across time. To demonstrate how this is a flaw, we show the effect of the threshold being appropriate, being too long, and being too short.

In Figures 4, 5 and 6 we show the results of processing the same list of events, under three different values of the threshold. The average *Lift* of all the correct candidate pairs is indicated by a solid line; the average *Lift* of all the false candidate pairs is indicated by the dashed line. We measure the average round trip time (RTT) to be around 250 milliseconds.

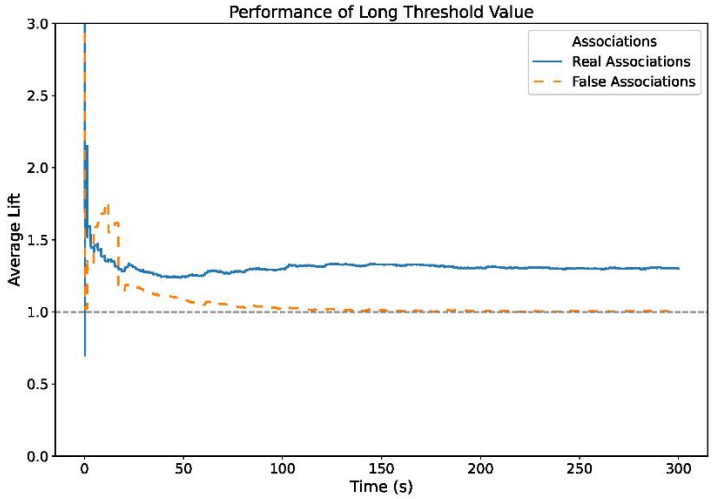

Figure 5. Threshold = 2.0 seconds

The threshold is set to 250 milliseconds for Figure 4. From 0 to around 50 seconds, the difference between the false connections and the real connections is insufficient for the generation of a confident prediction. Given that there is not enough data gathered yet, this is understandable. Eventually, we will see stability in the predictions - the difference in lift between the false and real associations would be significant enough that prediction can now be made confidently.

We next set the threshold to 2000 milliseconds. Figure 5 shows the effect of the change. Note that the gap between false and authentic connections still exists, but at a much smaller magnitude compared to Figure 4.

Finally, we test the threshold of 100 milliseconds, which is less than half of the average RTT. Figure 6 shows that disappointingly, the system assessed the genuine associations as some of the worst candidates.

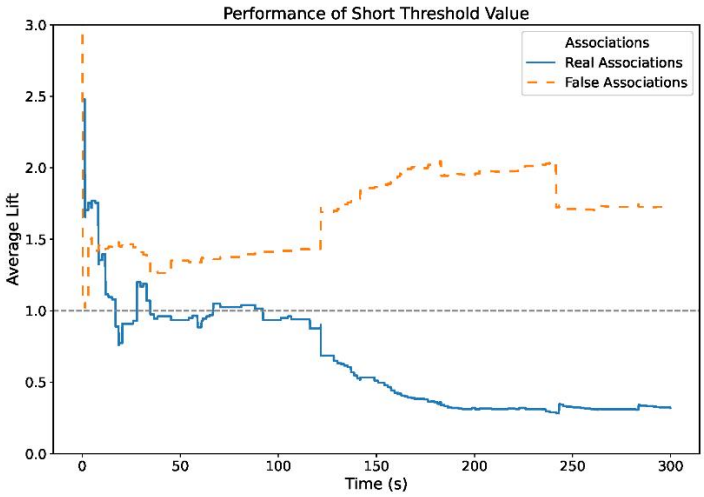

Figure 6. Threshold = 0.1 seconds

Figure 7 diagrams a comparison between long, short, and optimal threshold values, as well as their consequences on the process of generating buckets. Light-colored buckets indicate real associations, while false associations are represented by dark-colored buckets. Correct buckets are not formed under a short threshold; on the other hand, with a long threshold, correct buckets are (eventually) guaranteed to be formed, but this comes as the cost of also forming a significant number of erroneous buckets.

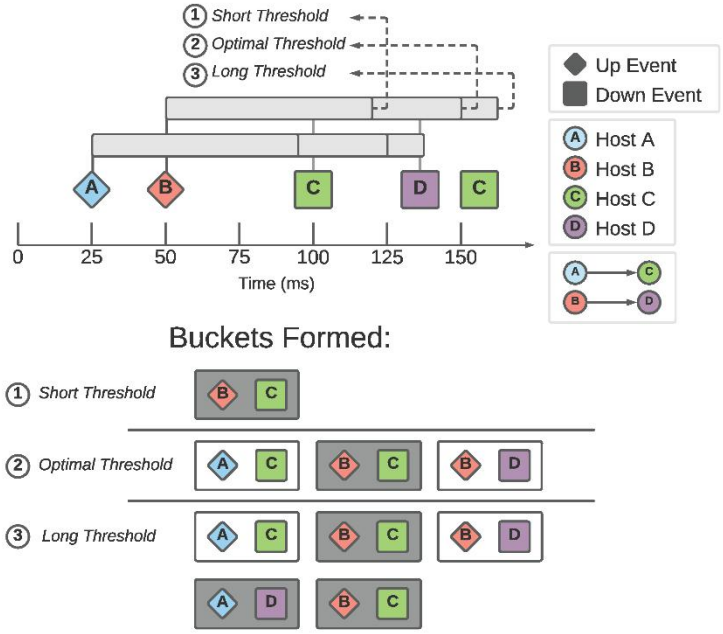

Figure 7. Comparison of Threshold Values

For buckets corresponding to real connections to be formed, it is absolutely necessary that the threshold be set to a value longer than the average RTT of the real connections. Setting it shorter will result in the situation shown in Figure 6; on the other hand, make it too long, and it will result in Figure 5. Since network conditions naturally vary over time, it is to be expected that the average RTT will fluctuate. Thus a system which relies on a static threshold value will perform problematically. In Section 6, we will present a method to dynamically change the threshold value, so that the system can properly react to a changing environmental average RTT.

6. Improved Design

We utilize an *event log* to demonstrate the consequences of varying the RTT. An event log is simply the list of event used in the earlier experiments. We used a script to modify the event log. The script has another input which is a list of (*duration, offset*) tuples. The *offset* is how much the average RTT should be modified, while the *duration* is the amount of time that the offset is in effect. What the script does is that it effectively adds an offset to the time value for down events in the event log, based on what is active at the original time. For the earlier experiment, the average RTT was found to be 237 ms, with 22 ms standard deviation.

We can see the performance of the system as configured so far as it deals with RTT fluctiations in Figure 8. Using the measured average RTT, we have a fixed threshold of 250 ms. 150 seconds after the beginning of the experiment, the RTT is increased by 200 ms. As such the fixed threshold would be now be too short for the correct creation of buckets. As discussed and as expected, the system does not stabilize towards an accurate prediction over time. The average lift values of the real connections actually steadily decline to 1. The average lift values of the false connections, on the other hand, approach 1 as well, but from the other side.

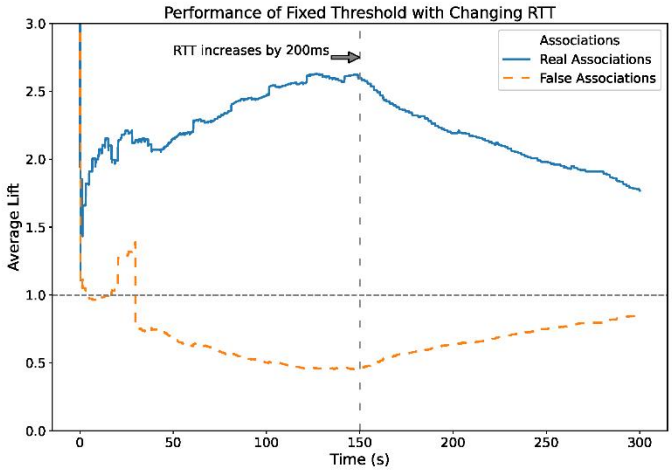

Figure 8. Performance of the original system when the average RTT changes

As expounded in Section 5 the predictions' accuracies degrade once the RTT veers from the fixed threshold, and this can be clearly seen in Figure 8. For the system to stabilize and produce accurate predictions, it is imperative that it detect and "chase" the ever-changing RTT.

The solution to threshold adjustment can not rely on prior knowledge of the correct associations, since that is unknown to the system. We can also not base the answer on current prediction, since those can be erroneous; obviously, using erroneous predictions as the basis for the dynamic or new threshold value can only result to more predictions which are wrong. The solution, as such, should rely on a property of the monitored environment that can be confidently assumed to be true - and that is that each down event will eventually be bucketed.

6.1 *Threshold Optimizer*

A *threshold optimizer* subsystem is added to the design to make threshold value dyna-muc. This subsystem runs in unison with the design described in Section 4. It is shown in the boxed area in Figure 2.

Figure 9 illustrates the operation of the threshold optimizer. Its operation will be discussed in detail in the following paragraphs.

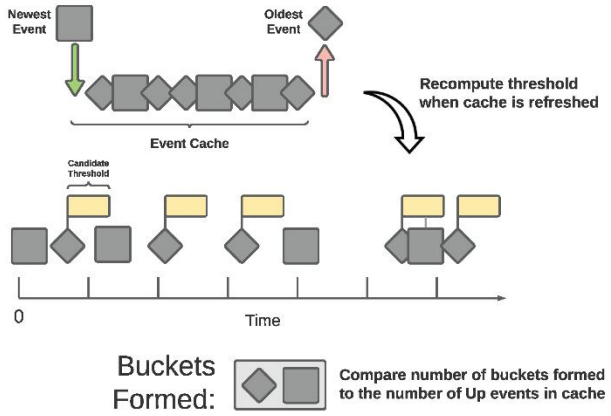

Figure 9. Threshold Optimizer

6.1.1 Event Cache

To solve the problem, we introduce a new component called the *event cache*. This is a component which records events created by the event extractor. The cache effectively holds information about the network conditions from the oldest recorded event to the newest recorded event, and using this information will allow the system to select an appropriate threshold for the future using the events in the event cache as a basis.

Since the event cache cannot have an infinite size, we set its size to *max queue size*. All events generated the event extractor are pushed into the event queue. Should it happend that the queue is full when a push is made, we evict the oldest event in the queue. A larger max queue size translates to more extensive information about the past, and would likely lead to more accurate threshold values. Nevertheless it comes at the cost of increasing the amount of time it takes to fully refresh the event cache, resulting in less frequent threshold recomputes, and thus, a less dynamic system.

6.1.2 Threshold Recompute

Every time an oldest event is evicted from the cache (or when the event cache is fully refreshed), the contents of the cache will be analyzed by the threshold optimizer. The optimizer will then produce an optimized threshold. The optimized threshold is defined as the lowest threshold where the amount of buckets formed is at least 90% of the number of down events. This stems from the aforementioned assumption that every down event is eventually bucketed. Nevertheless, some leeway is allowed for anomalous down events or down events whose up events are not in the cache anymore. Algorithm 1 describes how the threshold is recomputed.

Algorithm 1. Threshold Recompute Algorithm

$threshold \leftarrow 1ms$
$step\_size \leftarrow 50ms$
$term \leftarrow 0.9$
$num\_buckets \leftarrow form\_buckets(evt\_cache, threshold)$
while $num\_buckets < term * (num\_down(evt\_cache))$ do
  if $threshold > evt\_cache[last].time - evt\_cache[first].time$ then
    return $threshold$
  esle
    $threshold \leftarrow threshold + step\_size$
    $num\_buckets \leftarrow form\_buckets(evt\_cache, threshold)$
  end if
end while
return $threshold$

6.2 *Results*

The updated system's performance is evaluated by having it process the same set of inputs used in generating Figure 8. We overlay its results with the original, and this can be seen in Figure 10. Similar to what happened in the previous design (the one described in Section 4), its accuracy degrades immediately after the RTT changes. However, there is the crucial difference that the accuracy stops its degradation after a certain point.

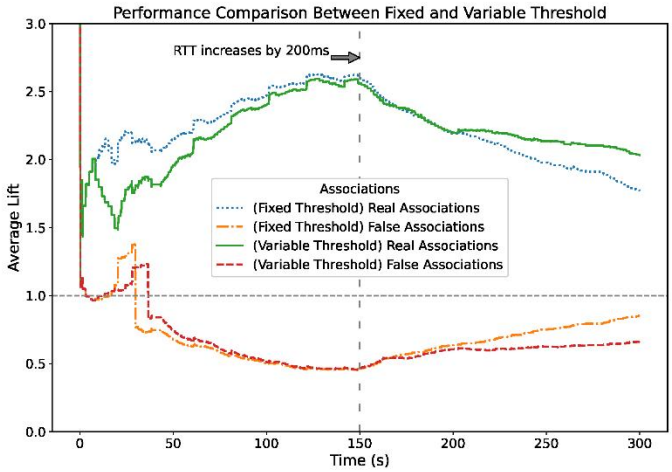

Figure 10. Comparison of the old system (Fixed Threshold) vs the improved system (Variable Threshold)

Figure 11 helps us visualize how the RTT is pursued by the improved and extended system. The figure plots the chosen candidate threshold values, calculated by the threshold optimizer system, against a known, but fluctuating, RTT. It can be appreciated from the figure that the RTT is closely estimated by the system. As such, the accuracy of the system is maintained to some degree, as opposed to the free-fall degradation exhibited by the original design.

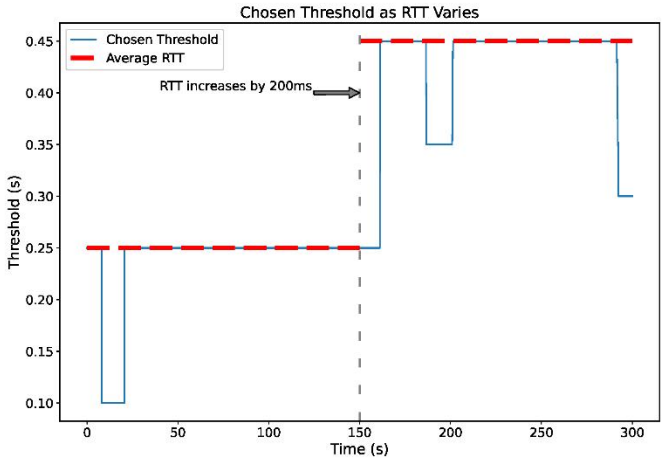

Figure 11. Calculated threshold value over time of the improved system

Note that we see dips in the chosen threshold value. This is caused by two things: optimizer's "short-sighted" nature (i.e., only the most recent events are used to compute the RTT), and the suitability of the termination case (Step 3) of our Threshold Recompute algorithm. The improvement of the second factor may warrant further investigation.

We should also note that there is some delay before the system is able to adjust. Given that the event cache has to be fully refreshed before threshold optimization starts, this is reasonable and to be expected. In the mean time, this opens up the possibility of the RTT changing while the event cache is only partially filled with new events. This has the consequence that the threshold calculated for that refresh period would not be perfectly

adjusted to the most recent RTT. For a refresh to be result in a threshold value that is truly accurate, all cache events must be recorded after the most recent RTT change.

7. Conclusion

In this work, a solution to the association detection problem is proposed. The solution centers on the analysis of inter-packet delays and flow directions which are used to generate sets of transactions, which are then used as inputs to a modified Apriori algorithm which then identifies associations between actuators and sensors with high accuracy. Unfortunately when network conditions fluctuate, the RTT also fluctuates, reducing the accuracy of the original design. We added a threshold recompute step which has the task of adjusting the threshold on a periodic basis to conform the system better to prevailing network conditions. This modification enables the system to deliver accurate results despite a fluctuating RTT.

The approach can be implemented with relative simplicity in existing IoT networks - it is not expensive computationally, primarily due to specific modifications to the Apriori algorithm. The association detection system's accuracy is critical to systems that provide offline-reliability to IoT networks, especially those whose operations center on identifying sensor-actuator pairs, analyzing their traffic, and replicating the correct remote server responses. Our association detection system was created with such an approach in mind, an approach that may be tackled in the future.

Possible improvements to our approach include the exploration of the use of features other than inter-packet delays and flow direction. It may be possible to efficiently improve accuracy through the active correlation of packets via watermarking. Investigations may also be done on the effects of adjusting various fixed parameters, such as *step_size* and *term*. Another possible improvement in the future is to explore the use of association detection techniques other than the Apriori algorithm, such as the FP-growth algorithm and the Eclat algorithm. These algorithms are potentially faster than the simpler Apriori algorithm, but their operations come at the cost of greater memory consumption. Their application may be a necessity for households or systems with a significant number of IoT devices, though they would also need more powerful machines or edge computers to run.

Conflict of Interest

There is no conflict of interest for this study.

References

[1] Lueth, K.L. State of the iot 2018: Number of iot devices now at 7b – market accelerating. Available online: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/ (accessed on 17 October 2023).

[2] Yadav, P.; Li, Q.; Mortier, R.; Brown, A. Network service dependencies in commodity internet-of-things devices. In Proceedings of the International Conference on Internet of Things Design and Implementation, IoTDI'19, New York, NY, USA, 2019, https://doi.org/10.1145/3302505.3310082.

[3] Noura, M.; Atiquzzaman, M.; Gaedke, M. Interoperability in Internet of Things: Taxonomies and Open Challenges. *Mob. Networks Appl.* **2018**, *24*, 796–809, https://doi.org/10.1007/s11036-018-1089-9.

[4] Samsung. SmartThings API. Available online: https://developer.smartthings.com/docs/api/public/ (accessed on 17 October 2023).

[5] Samsung. What automations can run locally on the connect home hub. Available online: https://www.samsung.com/us/support/troubleshooting/TSG01109364/ (accessed on 17 October 2023).

[6] Apple. Home app – apple. Available online: https://www.apple.com/home-app/ (accessed on 17 October 2023).

[7] Amazon. How aws iot works. Available online: https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html (accessed on 17 October 2023).

[8] Iotivity. Iotivity about page, 2019. Available online: https://iotivity.org/about (accessed on 17 October 2023).

[9] Elfstrom, K. Evaluation of iotivity: A middleware architecture for the internet of things. Master Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2017.

[10] Murdoch, S.; Danezis, G. Low-Cost Traffic Analysis of Tor. In Proceedings of 2005 IEEE Symposium on Security and Privacy (S&P'05), Oakland, CA, USA, 8–11 May 2005, https://doi.org/10.1109/sp.2005.12.
[11] Li, J.; Gu, C.; Zhang, X.; Chen, X.; Liu, W. AttCorr: A Novel Deep Learning Model for Flow Correlation Attacks on Tor. In Proceedings of 2021 IEEE International Conference on Consumer Electronics and Co mputer Engineering (ICCECE), Guangzhou, China, 15–17 January 2021, https://doi.org/10.1109/iccece51 280.2021.9342534.
[12] Nasr, M.; Bahramali, A.; Houmansadr, A. Deepcorr: Strong flow correlation attacks on tor using deep learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October, 2018, https://doi.org/10.1145/3243734.3243824.
[13] Huang, B.; Bouguettaya, A.; Neiat, A.G. Discovering Spatio-Temporal Relationships among IoT Services. In Proceedings of 2018 IEEE International Conference on Web Services (ICWS), San Francisco, CA, USA, 2–7 July 2018, https://doi.org/10.1109/icws.2018.00058.
[14] Dorsemaine, B.; Gaulier, J.-P.; Wary, J.-P.; Kheir, N.; Urien, P. Internet of Things: A Definition & Taxon omy. In Proceedings of the 2015 9th International Conference on Next Generation Mobile Applications, S ervices and Technologies, Cambridge, UK, 9–11 September 2015, https://doi.org/10.1109/NGMAST.201 5.71.
[15] Houmansadr, A.; Kiyavash, N.; Borisov, N. Rainbow: A robust and invisible non-blind watermark for network flows. *NDSS*, **2009**, *47*, 406–422.
[16] Doan, T.T.; Safavi-Naini, R.; Li, S.; Avizheh, S.; K., M.V.; Fong, P.W.L. Towards a Resilient Smart Home. In Proceedings of the 2018 Workshop on IoT Security and Privacy, San Francisco, Budapest, Hungary, 20 August 2018, https://doi.org/10.1145/3229565.3229570.
[17] Benson, K.E.; Wang, G.; Venkatasubramanian, N.; Kim, Y.-J. Ride: A Resilient IoT Data Exchange Middleware Leveraging SDN and Edge Cloud Resources. In Proceedings of 2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI), Orlando, FL, USA, 17–20 April 2018, https://doi.org/10.1109/iotdi.2018.00017.
[18] Ozeer, U.; Letondeur, L.; Ottogalli, F.-G.; Salaun, G.; Vincent, J.-M. Designing and Implementing Resilient IoT Applications in the Fog: A Smart Home Use Case. In Proceedings of 2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 19–21 February 2019, https://doi.org/10.1109/icin.2019.8685909.
[19] Grover, J.; Garimella, R.M. Reliable and Fault-Tolerant IoT-Edge Architecture. In Proceedings of 2018 IEEE SENSORS, New Delhi, India, 28–31 October 2018, https://doi.org/10.1109/ICSENS.2018.8589624.
[20] Sahadevan, A.; Mathew, D.; Mookathana, J.; Jose, B.A. An Offline Online Strategy for IoT Using MQTT. In Proceedings of 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA, 26–28 June 2017, https://doi.org/10.1109/cscloud.2017.34.
[21] Dou, Y.; Liu, H.; Wei, L.; Chen, S. Design and simulation of self-organizing network routing algorithm b ased on Q-learning. In Proceedings of 2020 21st Asia-Pacific Network Operations and Management Sym posium (APNOMS), Daegu, Korea, 22–25 September 2020, https://doi.org/10.23919/apnoms50412.2020. 9237020.
[22] Romanov, A.; Myachin, N.; Sukhov, A. Fault-Tolerant Routing in Networks-on-Chip Using Self-Organizi ng Routing Algorithms. In Proceedings of IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society, Toronto, ON, Canada, 13-16 October 2021, https://doi.org/10.1109/iecon48115.2021. 9589829.
[23] Daas, M.S.; Chikhi, S.; Bourennane, E.-B. A dynamic multi-sink routing protocol for static and mobile self-organizing wireless networks: A routing protocol for Internet of Things. *Ad Hoc Networks* **2021**, *117*, 102495, https://doi.org/10.1016/j.adhoc.2021.102495.
[24] Hamrioui, S.; Lloret, J.; Lorenz, P.; Rodrigues, J.J.P.C. Cross-Layer Approach for Self-Organizing and Se lf-Configuring Communications Within IoT. *IEEE Internet Things J.* **2022**, *9*, 19489–19500, https://doi.or g/10.1109/jiot.2022.3168614.
[25] Anees, T.; Habib, Q.; Al-Shamayleh, A.S.; Khalil, W.; Obaidat, M.A.; Akhunzada, A. The Integration of WoT and Edge Computing: Issues and Challenges. *Sustainability* **2023**, *15*, 5983, https://doi.org/10.3390/ su15075983.
[26] Rawashdeh, M.; Al Zamil, M.G.; Samarah, S.M.; Obaidat, M.; Masud, M. IOT-based service migration fo r connected communities. *Comput. Electr. Eng.* **2021**, *96*, 107530, https://doi.org/10.1016/j.compeleceng. 2021.107530.
[27] Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. *J. SIGMOD Rec.* **1993**, *22*, 207–216, doi:10.1145/170035.170072.

[28] Ning, H.; Yuan, H.; Lu, Z.; Guo, J. Research on Association Rules Mining with Temporal Restraint. In Proceedings of 2007 2nd IEEE Conference on Industrial Electronics and Applications, Harbin, China, 23–25 May 2007, https://doi.org/10.1109/iciea.2007.4318678.
[29] Yan, S.-R.; Pirooznia, S.; Heidari, A.; Navimipour, N.J.; Unal, M. Implementation of a Product-Recommender System in an IoT-Based Smart Shopping Using Fuzzy Logic and Apriori Algorithm. *IEEE Trans. Eng. Manag.* **2022**, *PP*, 1–15, https://doi.org/10.1109/tem.2022.3207326.
[30] Du, Z. Energy analysis of Internet of things data mining algorithm for smart green communication networks. *Comput. Commun.* **2020**, *152*, 223–231, https://doi.org/10.1016/j.comcom.2020.01.046.
[31] Han, J.; Pei, J.; Yin, Y. Mining frequent patterns without candidate generation. *ACM SIGMOD Rec.* **2000**, *29*, 1–12, https://doi.org/10.1145/335191.335372.
[32] Zaki, M. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.* **2000**, *12*, 372–390, https://doi.org/10.1109/69.846291.
[33] Benzekki, K.; El Fergougui, A.; Elalaoui, A.E. Software-defined networking (SDN): a survey. *Secur. Commun. Networks* **2016**, *9*, 5803–5833, https://doi.org/10.1002/sec.1737.
[34] Fette, I.; Melnikov, A. The WebSocket Protocol. **2011**, https://doi.org/10.17487/rfc6455.
[35] Riley, G.F.; Henderson, T.R. The ns-3 Network Simulator. In *Modeling and tools for network simulation*, Springer: Heidelberg, Germany, **2010**, 15–34, https://doi.org/10.1007/978-3-642-12331-3_2.
[36] Xiang, Z.; Seeling, P. Mininet: an instant virtual network on your computer. In *Computing in Communica tion Networks*, Academic Press: Cambridge, MA, USA, **2020**, 219–230, https://doi.org/10.1016/b978-0-12-820488-7.00025-6.