Combining Device Behavioral Models and Building Schema for Cyber-Security of Large-Scale IoT Infrastructure

Ayyoob Hamza, Hassan Habibi Gharakheili, Trevor Pering, and Vijay Sivaraman

***Abstract*—Modern buildings are increasingly getting connected by adopting a range of IoT devices and applications from video surveillance and lighting to people counting and access control. It has been shown that rich connectivity can make building networks more exposed to cyber-attacks, and hence difficult to manage. Currently, there is no systematic approach for evaluating or enforcing cyber-security of building systems with a large number of heterogeneous IoT devices. In this paper, we aim to enhance cyber-security of large-scale IoT infrastructure by formally capturing the expected behavior of the system using static profile of devices' intended usage, buildings information, and network configurations (pre-deployment) along with dynamic diagnosis (post-deployment) of network activity using machine-learning models.**

Our contributions are three-fold: (1) We develop a tool that automatically generates a formal ontology of network communications for a connected infrastructure by taking a description of buildings (in the form of Brick schema), devices network behavior (in the form of Manufacturer Usage Description specifications, MUD profile), and network configurations (address, port, VLAN) as inputs. We contribute our tool as open-source, apply it to a subset of our university smart campus testbed, covering 20 IoT devices of three types deployed in seven different buildings. We translate the formal model into network flow rules and enforce them to the network at run-time using programmable networking techniques; (2) We, then, measure the network activity of device-specific flow rules and diagnose their health using a set of trained anomaly detection models (one-class classifiers) each corresponding to a particular type of device and specific building location, and demonstrate how our method detects attacks with reasonable accuracy of 92.5%; and (3) Lastly, we demonstrate three types of location-defined network policies (deployment, administrative, and organizational) that can be verified by this formal model.

***Index Terms*—IoT System Ontology, Behavioral Modeling, Anomaly Detection, MUD Profile, Building BRICK**

I. Introduction

OPERATORS of modern buildings and infrastructure [1], [2], [3] are increasingly adopting a range of IoT devices to better manage utilization of physical spaces, improve the safety of occupants, save energy, and reduce maintenance costs [4]. From a traditionally static and proprietary environment of standalone systems, smart buildings are moving towards a dynamic environment driven by connected systems and standard protocols. In these systems, automatic decisions are often made by IoT controllers based on data collected from a large number of devices such as security cameras, smart-lights, smoke-alarms, or occupancy sensors sourced from a diversity of vendors. Integrating cloud-based servers with many different types of devices, each with their own security flaws (*e.g.,* weak/no encryption, open ports, default username/passwords) [5], [6], [7], [8], [9], [10], [11], [12], exponentially increases the potential attack surfaces on smart environments [13], [14], [15].

Though many research papers (including ours [7], [16]) have identified myriad security flaws in IoT devices, few have suggested solutions beyond the patching of these flaws by the respective manufacturers, which is doomed to failure given the large number of vendors and their limited motivation to support a device beyond its sale. A promising direction is to monitor and lock-down the network activity of IoT devices to detect and block misbehavior [17], [18], giving the network operator a second line of defense against compromised or misbehaving devices without relying solely on appropriate security protections by the IoT supplier. The success of this approach, however, relies on knowing the expected network behavior of each IoT device, and the interactions of these devices in a specific deployment environment such as a commercial building or an enterprise. We assume that individual traffic flows can be uniquely mapped to a corresponding device (by way of device's unique IP and/or MAC address). In other words, analyzing network traffic whereby the identity of devices is obfuscated by NAT [19] is beyond the scope of this paper.

The corporate world experienced a significant uptick in physical and cybersecurity threats due to the pandemic sending employees home to work [20]. With office buildings vacated, the properties were ripe for exploitation by malicious actors. Today, a large-scale digital infrastructure is typically managed by two entities Estate Management (assets) and IT department (network). Such disjoint management of information makes it challenging to verify the operation of the entire system (building, campus) and secure it against cyber threats. However, given the changing nature of security risks, organizations need to start combining the previously disjointed approaches to combat threats more systematically and automatically in a pandemic and post-pandemic world. In this paper, we systematically combine information from three sources namely intended behavior of individual IoT devices, physical assets and building data, and network configurations. We first draw upon emerging frameworks such as the IETF

A. Hamza, H. Habibi Gharakheili, and V. Sivaraman are with the School of Electrical Engineering and Telecommunications, University of New South Wales, Sydney, NSW 2052, Australia (e-mails: m.ahamedhamza@unsw.edu.au, h.habibi@unsw.edu.au, vijay@unsw.edu.au).

T. Pering is with Google (e-mail: peringknife@google.com).

MUD (Manufacturer Usage Description) standard [21] that provides an ACL[1]-like language for describing the expected network communications (*e.g.,* protocols over which and/or controllers with which they talk) of an IoT device, and Brick [22] that provides a metadata schema for the deployment environment (such as a building) including locations of sensors and their sub-system relationships.

For our **first** contribution, we develop a tool that combines behavioral profile of devices and buildings schema information with network configurations to automatically derive a model of all potential network communications across the IoT system. We contribute our tool as open-source, apply it to a subset of our university smart campus network, covering 20 IoT devices of three types deployed in 7 different buildings. Following generation, we automatically translate the formal model into network flow rules, and enforce them to a programmable switch. Our **second** contribution periodically measures the activity of IoT flows (via network telemetry) and diagnoses their health by checking against anomaly detection models, identifying and interpreting unexpected behaviors. Our **third** contribution applies three representative location-defined policies (*e.g.,* restricting access of certain resources to specific operational zones) to this formal model, verifying the intended system behavior.

II. Motivation and Related Work

A large IoT infrastructure for smart buildings may consist of many subsystems such as HVAC, lighting, access controllers, occupancy sensors, or physical security systems. These subsystems are often managed by a variety of stakeholders from network architect, network engineers, facility management engineers, and cyberlsecurity analysts to device manufactures, system integrators, and building managers throughout the lifecycle of a smart building [4]. These stakeholders produce different data schema to maintain information about the physical location, network configuration, or security policies of IoT devices. The lack of a common data model is a major challenge in limiting the interoperability and holistic analysis of heterogeneous IoT systems. This has led to many cyber-attacks – for example, the Shodan search engine [23] has listed publicly exposed building management systems that allows attackers to penetrate those networks.

Current methods for evaluating the security posture of such environments is at best ad-hoc, and enforcement and monitoring of appropriate access controls from outside and within the organization are lacking. However, securing large IoT systems demands a formal model that enables, at design stage, an evaluation of the attack surface exposed by the smart environment, including assessments of firmware updates, breached elements, and organization policy changes on overall security. Also, the model needs to be enforced at runtime, including monitoring the communication flows to detect anomalous patterns indicative of volumetric attacks. This paper presents the first systematic effort to both model (statically) and enforce (dynamically) cyber-security for large-scale IoT systems.

[1]Access Control List

Building Data: Haystack [24], Brick [22] and IFC (Industry Foundation Classes) [25] provide constructs to formally define a metadata model to specify sensors, controllers, their location in buildings, and their inter-relationships. We use Brick in this paper because: (a) it describes building entities (sensors, equipment, room, floor and many more) and their relationships by abstracting classes and tags; (b) its hierarchical constructs allows to extend the Brick model to express new entities (*e.g.,* Camera can be derived from Sensor); (c) its expressiveness and ease of adaption allow us to build a better query processor; and (d) it uses the Resource Description Framework (RDF) syntax to maintain the system ontology, this enables application developers to interact with the ontology using query-based language (*e.g.,* SPARQL [26]). Work in [27] discusses various applications that can benefit from building data such as energy optimization, fault detection, and risk analysis. To the best of our knowledge, we are the first to use building metadata to enhance the network security of IoT systems.

IoT Behavioral Profiles: As opposed to general-purpose computers, IoT devices have a limited and recognizable pattern of communications [28], [29], [30], [31]. This allows IoT device behavior to be captured succinctly and verified formally. IETF MUD (Manufacturer Usage Description) is a standard [21] that provides a set of machine-readable constructs to capture the flow information of a device, MUD allows a device manufacturer to define the behavior of their device in the form of access control lists. A valid MUD profile comprises several access control entries (ACE), serialized in JSON format. Access-lists are explicit in describing the direction of communication, *i.e.,* from-device, and to-device. Each ACE would match on source/destination port numbers for TCP/UDP, and type and code for ICMP. The MUD specifications also distinguish local network traffic from Internet communications and also provide the support of ambiguities through controller tag which during the deployment system integrators can configure the server location.

Dynamic Network Telemetry: Collection of network data using software-defined and/or programmable networking paradigms has been increasingly used for network management and intrusion detection [32], [33], [34], [35]. Authors of [32] develop an SDN-based monitoring system for identifying and classifying network flows in real-time at scale. Work in [33] proposes an adaptive network telemetry system using SDN for an accurate attack detection. Given a topology and traffic distribution, authors develop an algorithm to determine a set nodes and sampling resolution for collecting network data during volumetric and distributed attacks. Authors of [34] employ programmable data-plane to develop a scalable telemetry for collecting and analyzing network traffic in real-time. In a similar work [35] P4 programmable switches were used for a better defense mechanism against DDoS attacks. We, in this paper, use SDN control plane to: (a) automatically insert network rules obtained from the formal ontology, and (b) periodically collect the activity of network rules for real-time diagnosis by trained inferencing models.

Static Security Verification: In our earlier works [36], [37] we showed MUD profiles can be used for verifying the compatibility of an IoT device with an organizational network

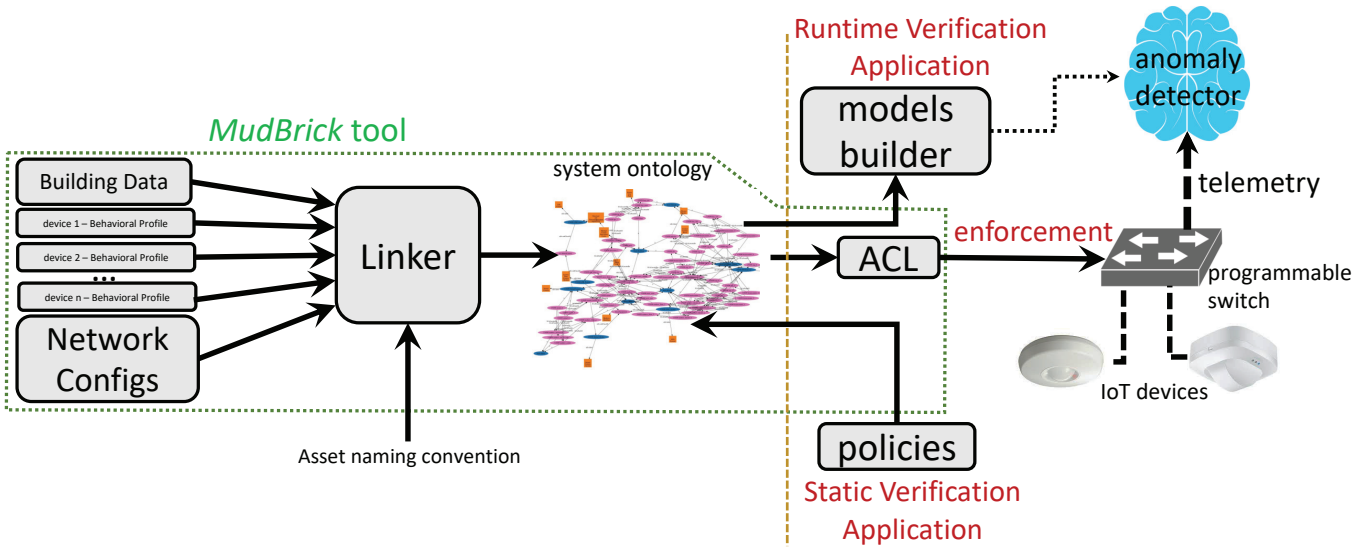


Fig. 1. Our system architecture: formal model generation, policy verification, and rules enforcement and monitoring.

policy for acceptance. Authors of [38], [39], [40] aim to detect/resolve conflicts among trigger-and-action-based policies set by network administrators in IoT environments. Work in [40] extends trigger-and-action-based policies to support MUD access-control rules and building/floor constructs. In this paper, we use a combination of MUD profiles and building Brick schema to verify location-defined network policies for large-scale IoT systems.

Rutime Security Verification: MUD specifications can be fed to an IDS (Intrusion Detection System) to detect observed behavior that is not as specified, thereby indicating an anomaly or threat [41], [42]. MUD enables enforcement of a baseline security control for IoT devices by isolating exception traffic that does not match the device intended ACEs. However, studies [43], [44], [45], [46] have shown that the attacks are still possible. Authors of [44], [47], [48] have used anomaly detection techniques to secure devices by modeling the traffic characteristics of individual devices. In this paper, we detect anomalies by looking at the traffic characteristics of both individual devices and a group (based on location) of devices in a building.

Our Novelty and Key Differentiation: This paper builds upon our previous works [36], [37]. It is the first to fuse three disjoint data sources of digital and physical assets, including (i) a description of a physical environment, (ii) a description of network behaviors for connected IoT device types, and (iii) network configurations, generating a formal and holistic model of the entire system (physical assets, digital assets, and underlying networks) in §III. We demonstrate how the resulted model can serve two important purposes: (a) continuously checking the conformity of the system behavior to its intended model (cyber health) and automatically detecting deviations from expected behaviors in real-time in §IV, and (b) systematically assessing cyber risks and verifying organizational policies. Our real-time network telemetry (flow-level activity features without inspecting packet payload) is lightweight, and the formal model is responsive and automatically enforceable, making it conducive for scale deployment in §V.

III. Generating Formal Model of Communications for IoT Infrastructure

In this section, we discuss the formal model of communications for an IoT system and how we automatically generate this model using a tool, *MudBrick* which we developed and is released as open-source [49]. We apply this tool to an IoT infrastructure testbed [50], [51] consisting of 20 IoT devices of three types (*i.e.,* 6 units of people counting camera from Steinel, 12 units of beam counter from EvolvePlus, 2 units of license plate recognition camera from Nedap) that are spread across 7 buildings on our university campus.

Let us begin with our system architecture shown in Fig. 1. *MudBrick*, shown by the dotted green region, takes three sources of information namely building data in the form of Brick schema, usage behavior of individual devices in the form of MUD profiles and their corresponding network configurations. The linker module then combines these data sources and generates the formal model of the IoT system (*i.e.,* system ontology) which is machine-readable and captures data of assets and their relationships. We note that data from Brick and MUD contains formal semantics while the format of network configurations data varies across different organizations. Therefore we have designed the linker module to be extensible accepting various formats of configurations.

There exist two applications that consume the knowledge representation (formal model) to enhance the security of the entire infrastructure: **(1) dynamic verification app**: once the ontology is created, *MudBrick* generates flow rules (in the form of access control lists or ACLs) that will get enforced in operational networks using programmable networking techniques [41]. The run-time activity of network flow rules is periodically collected from the network switch and fed to a set of pre-trained anomaly detection models, each specific to the controller of devices from a particular type and their building location. In what follows, we will describe the structure of data for various sources specific to our university campus use-case used in our *MudBrick* tool; and **(2) static verification app:** the system ontology can also be checked whether it is

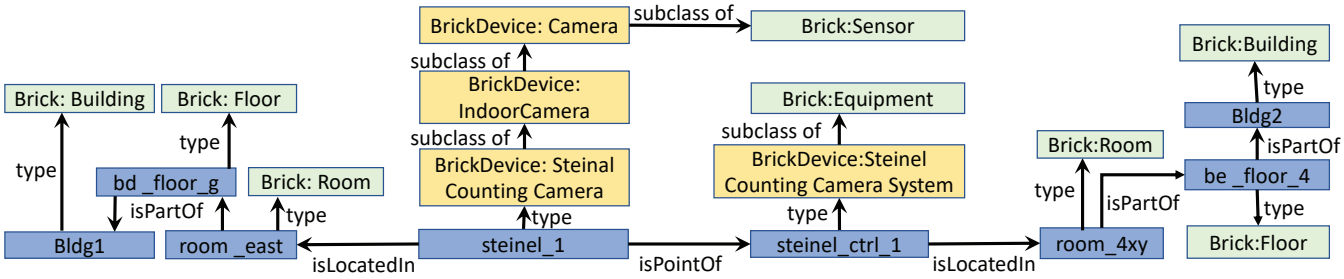


Fig. 2. An example of building data (Brick representation) for a subset of our IoT infrastructure.

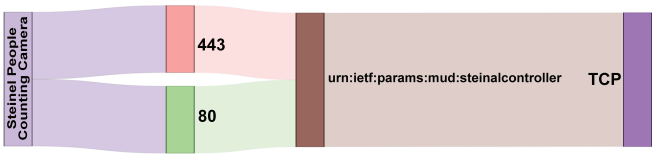


Fig. 3. Sankey diagram of MUD profile for Steinel people counting camera.

compatible with a given set of policies in an organization – such verification helps to identify links which violate intended policies, and hence need to be pruned. This enables enterprise network operators to request the installation team or respective manufacturers to make necessary changes for acceptance.

Building data is a machine-readable data structure for specifying entities and subsystems in a physical building and their relationships. To better visualize this data structure, we show in Fig. 2 the Brick[2] representation [22], corresponding to a subset of our IoT infrastructure. In this figure, each node represents either a class or an entity instance, and each edge describes the relationship between nodes. Green nodes are classes, which are defined by the original Brick schema [22] and yellow nodes are classes that we have extended by inheriting from the original schema. For example, the current version of Brick does not have a definition for cameras, but it provides an extendable hierarchy. This allows us to create a new device class called "Camera" which is a *subclass* of existing "Sensor". Blue nodes represent various entities in buildings.

In Fig. 2, a people counting camera (`steinel_1`), manufactured by Steinel [52], is an instance of class *"Steinel Counting Camera"*. This device is installed in the eastern room (`room_east`) on the ground floor (`bd_floor_g`) of building `Bldg1` – actual names of buildings and rooms are obfuscated for privacy reasons. It communicates with its controller (`steinel_crtl_1`) which is derived from Brick's class *Equipment*. The controller is located in room `room_4xy` at the fourth floor of building `Bldg2`.

Device Behavioral profile: We collected the network traffic trace of the campus IoT infrastructure, and then generated the MUD profile for the three types of devices using our MUDgee tool [36]. Fig. 3 visualizes the MUD profile of the Steinel people counting camera. It is seen that the device exposes TCP port 443 and 80 to its controller which periodically communicates with the camera to collect measurements. Note that the controller (*i.e.,* `urn:ietf:params:mud:steinelbroker`) can provision either locally or in a remote network.

Network Configs: We obtained (from our campus IT department) a spreadsheet of network configurations for all connected IoT devices and their corresponding controllers in the campus IoT system. It contains MAC address, reserved IP address of every device, physical port number they are connected to, their VLAN configurations, and host-names.

Linker: This module is responsible for fusing three "independent" sources of data structures. In order to merge these data sources, we need to "extend" their current schema. One may choose to extend all three schemas or a subset of them, making them fusible. We note that the description of a physical environment is more comprehensive than the other two input sources. We, therefore, extend only the BRICK schema with two new semantics, providing "hooks" (adapters) to incorporate MUD and Network Configs data.

We augmented the Brick schema with two semantics (one for MUD and one for network configs) each consisting of classes and properties. This enables us to combine the building metadata with data of IoT MUD profiles and network configurations. Fig. 4 depicts the MUD and Network semantics. We introduced two properties called *FromDevice* and *ToDevice* to capture the direction of communication. These properties are applied to the class Sensor and point to an ACE class. The ACE class would have properties from the MUD data – it contains an endpoint to a fixed IP address, domain name or a controller tag. In the case of the controller tag, it needs to be an object derived from the class Sensor or Equipment in Brick. Similarly, for network configurations, we added semantics that captures all configurations as properties of a Sensor or Equipment. Note that the automatic correlation of these three data sources (*i.e.,* Brick, MUD, network configs) could be challenging since we need a unique identifier across data sources. Therefore, our design offers extensions, allowing the use of various strategies for asset naming conventions that could be specific to organizations and deployments. In the context of this paper, with the available dataset, we use the MAC address of IoT devices as a unique key for combining the building data with network configurations, and the device type is used to combine MUD profiles with the building data.

IoT Ontology: Using the extended semantics introduced above, we generate a machine-readable knowledge representation of the entire system. Fig. 5 visualizes a part of

[2]Brick is an open-source effort to standardize semantic descriptions of the physical, logical and virtual assets in buildings and the relationships between them.

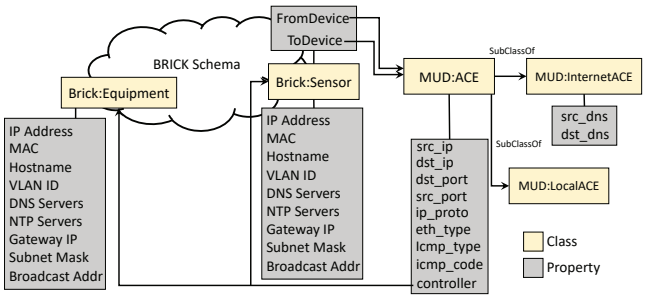


Fig. 4. Partial illustration of intermediate semantics after combining MUD profile, Network Configs, and Brick schema.

intermediate form (not the semantics). We can see how `steinel_1` communicates with `steinel_ctrl_1` – protocols, port numbers, VLAN, IP/MAC addresses of the Steinel counting camera and its controller are captured in this form. Now, we are able to make various queries over this ontology. For example, the IP address of all cameras located in building *Bldg1* can be obtained by:

```
<sensor.ip> := (sensor.type = Camera   ∧
                sensor.isLocatedIn.isPartOf = Bldg1   ∧
                sensor.isLocatedIn.isPartOf.type = Building)
```

In the above statement, `<sensor.ip>` indicates the returned value and `"."` operator (*e.g.,* `.type`) indicates a property of a given node and a node, in this case, is a class in ontology. In listing 1, we show the actual SPARQL query corresponding to the statement above.

Listing 1. SPARQL query to retrieve IP address of all cameras in building Bldg1.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX brick: <https://brickschema.org/schema/1.0.3/Brick#>
PREFIX bricknetwork: <https://iotanalytics.unsw.edu.au/schema/1.0.0/
      BrickNetwork#>
PREFIX bf: <https://brickschema.org/schema/1.0.3/BrickFrame#>
PREFIX brickdevice: <https://iotanalytics.unsw.edu.au/schema/1.0.0/BrickDevice
    #>
PREFIX : <https://iotanalytics.unsw.edu.au/ontology/>
SELECT ?ip WHERE {
?sensorType rdfs:subClassOf* brickdevice:Camera.
?device rdf:type ?sensorType.
?device bf:isLocatedIn ?location.
?location bf:isPartOf* ?partOf.
?partOf rdf:type brick:Building.
?device bricknetwork:reservedIP ?ip.
Filter ( ?partOf in (:Bldg1))}
```

Scalability of our Approach: It is important to note that the scalability of our approach (shown in Fig. 1) stems primarily from scalable network telemetry and scalable system ontology. Our previous works [36], [37] highlighted the scalability of flow-level network telemetry through extensive evaluations performed on a prototype consisting of real IoT devices. The scalability of system ontology is determined by its completeness in representing digital and physical information as well as the memory/time complexity of its underlying data structure. The completeness aspect is inherited from the Brick schema that was extensively evaluated by prior work in [22]. In order to quantify the time/memory complexity of our system ontology for a very large network, we conduct experiments in an emulator environment where parameters can be configured to represent a network with up to 10000 connected devices, each with up to 150 rules.

We developed the MudBrick tool using Apache Jena [53], and ran it on a machine with Intel 8 Core CPU 2.7 GHz and 8

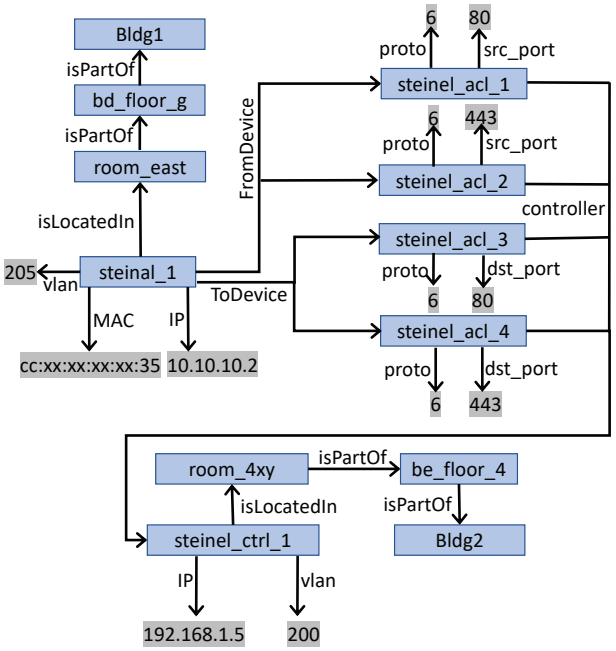


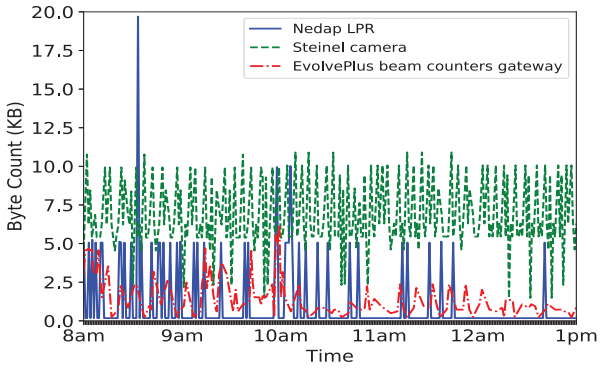
Fig. 5. Partial illustration of intermediate form of our IoT infrastructure.

GB of RAM on Mac OS X. For our IoT infrastructure, the size of input data sources was 32 KB while MudBrick generated the formal model of size 59 KB. Also, a search query (*e.g.,* Listing 1) on average is responded in less than 200 ms. We emulate larger and more complex networks by increasing the number of devices up to 10000, and the count of their MUD flows up to 150 per device type. Table I summarizes the impact of these two parameters on the size of our formal model and its search response time. It can be seen that scaling the network size and complexity by factors of 500 (10000/20=500) and 10 (150/14≈10), respectively, leads to only a 60 times larger model (ratio of 3200 KB to 52 KB). This trend highlights almost a linear growth at the beginning that tends to saturate for larger and more complex networks. Note that the largest model (with the size of 3200 KB) is fairly lightweight for such a scale. In terms of the search response time, it consistently remains sub 200 ms, which is very attractive.

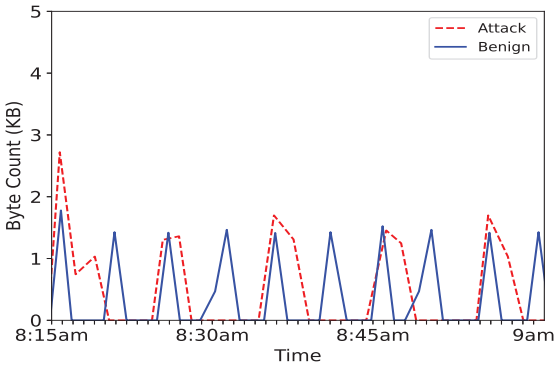
Once the system ontology (formal model) is generated, MudBrick translates the ontology into a set of flow rules (per device) that can be enforced to the network. We note that in the formal model it is allowed to have Internet endpoints specified by their domain-name. But, ACEs pertinent to Internet communications (with domain-name) can not be directly translated to flow rules, and hence need further inspection to infer DNS bindings (mapping DNS names to IP addresses for the various servers/controllers) at run-time [41]. Obviously, ACEs with endpoints specified by IP address are proactively inserted into the switch – others are reactively inserted after bindings are determined. Moreover, ACEs obtained from our formal model can be directly translated to flow rules, but they may require a notion of rules priority to tightly enforce the activity of IoT devices on the network – details on how to enforce such rules can be found in our previous work [41].

TABLE I
IMPACT OF DEVICES COUNT AND THE COMPLEXITY OF THEIR BEHAVIOR ON THE SIZE OF FORMAL MODEL AND SEARCH RESPONSE TIME (FROM SIMULATED DATASET).

# IoTs	# MUD ACL per IoT	Model size (KB)	Search response time (ms)
20	14	52	~ 200
20	150	210	~ 200
100	14	175	~ 200
100	150	588	~ 200
10000	150	3200	~ 200



(a) Temporal patterns of network activity for three device types.



(b) Benign versus attack traffic pattern of beam counters gateway.

Fig. 6. Time-trace of aggregate traffic exchanged between IoT device types and their respective controller: (a) benign traffic of three IoT types, and (b) benign vs. attack traffic of beam-counters gateway.

IV. ANOMALY DETECTION

We now monitor the network activity of the IoT system (flow rules and associated location of devices obtained from the ontology) at run-time. In this section, we begin by looking at various patterns of IoT traffic. We, next, describe a method we developed for detecting attacks. Lastly, we discuss our attack dataset and evaluation results.

A. IoT Traffic Pattern

Fig. 6(a) shows the temporal pattern of network traffic (aggregate of all flows during 8am-1pm on a typical weekday) for the three device types in our infrastructure. Note that devices communicate only with their respective controller. Nedap license plate recognition (LPR) cameras, shown by solid blue lines, generate network traffic whenever they detect a moving object (vehicle). It can be seen that the LPR camera is fairly active early in the morning, during 8am-10am, transmitting data up to 20 KB per minute to its controller. Its activity slowly becomes infrequent after that – such a network behavior matches normal usage of the car park on campus [51]. Steinel cameras (shown by green dashed lines in Fig. 6(a)), instead, periodically send counts of people to their controller – a fairly consistent traffic rate of 7.5 KB per minute. For the third category of network behavior, we see EvolvePlus beam counters gateway, which publishes data (count in/out) every minute if there are movements of people; otherwise, it communicates "Hello" messages every 10-minute (shown by red dashed-dotted lines in Fig. 6(a)). Note that beam counter sensors talk wirelessly to a gateway (in their proximity) which then relays (over Ethernet) aggregate data from connected sensors to a remote controller [50].

In Fig. 6(b) we show the change of traffic pattern during an attack on beam counters. In one of the lecture theaters, we launched a frequency jamming attack (emulated by manually removing the battery from sensors), hence not transmitting data to their local gateway. As shown by solid blue lines, early morning (between 8.15am and 9am) on a typical weekday, this gateway displays a periodic traffic pattern (normal Hello messages every 10-min) since there is no class scheduled during this period and hence no movements. In Fig. 6(b), an evident deviation from the norm is observed. Large spikes of about 2.7 KB per minute are caused due to error messages generated by the gateway missing sensors. It is important to note that normal traffic patterns can vary highly over time, particularly on a per-flow basis. Therefore, a simple thresholding method would not suffice for distinguishing normal from abnormal patterns. Instead, it is needed to model and learn the dynamics of traffic profiles across individual flows.

B. Anomaly Detection Model

We develop a machine learning technique to determine if our IoT infrastructure is affected by a cyber-attack (the "attack detection"), and if so, to determine the contributing building or device (the "attack identification"). Our objective is to train our machine with benign traffic profile (one-class classifier) of each IoT controller, and detect attacks by flagging deviation (from expected pattern) in traffic flows of a controller (obtained from the system ontology). Fig. 7 illustrates a high-level structure of our anomaly detection engine. For each IoT controller, we train models at two levels of granularity, namely building level (**Stage1:** M_{B_i}) and device level (**Stage2:** M_{d_k}).

At Stage1, building models (coarse-grained) monitor the behavior of aggregate of IoT devices in a building that

TABLE II
NETWORK FLOWS MONITORED FOR EACH UNIT OF STEINEL PEOPLE COUNTING CAMERA.

source	destination	proto	srcPort	dstPort
<deviceIP>	<controllerIP>	6	80	*
<deviceIP>	<controllerIP>	6	443	*
<controllerIP>	<deviceIP>	6	*	80
<controllerIP>	<deviceIP>	6	*	443

communicate with a given controller, while at Stage2, device models (fine-grained) capture the activity of individual devices per controller. An anomaly is detected only when both Stage1 and Stage2 models raise alarms. For our models, we compute features from time-series signal of flow activities. Note that network flows are obtained by applying policies to the system ontology and enforced to the network. For example, Table II displays flow rules for each unit of Steinel camera. We retrieve flow counters every minute to construct the required time-series at run-time (Fig. 1).

Feature Extractor: It is important to note that our system is privacy-preserving because we develop the profile from observing network traffic pertaining to the device; we emphasize that we do not look into the data to/from the IoT device, but instead extract the "meta-data" associated with the network activity of the device at flow levels. Using this meta-data, we build behavioral models of devices. We maintain a sliding window of 60 data-points (byte count of individual flow rule). There exist a large number of features that can be extracted by time-series analysis [54]. For our infrastructure, we use Wrapper method [55] to select the most important features per each flow. The flow-level features are: (1) mean, (2) variance, (3) sum, (4) count above mean, (5) count below mean, (6) the longest strike above mean, (7) the longest strike below mean, (8) zero count, (9) number of peaks, (10) number of time crossing, (11) absolute sum of changes, (12) mean absolute change, (13) hour-of-day, and (14) day-of-week. Note that the total number of features for each model (M_{B_i} and M_{d_k}) varies depending on the count of flows per device and count of devices per building.

Dispatcher: This module batches a collection of features (computed on network telemetry) and disseminates them across their corresponding models. Note that Stage1 models are trained by the network behavior of a group of devices that reside in a building. The dispatcher obtains situational information (mapping of devices to buildings) from the ontology. As illustrated in Fig. 7, the dispatcher feeds the flow-level features of two devices d_1 and d_2 to their building's model M_{B1}. Detection of an anomaly at Stag1 would activate corresponding Stage2 models where the dispatcher simply presents the features of individual devices to their device models (M_{di}) – no situational information is needed for Stage2 inferencing.

Anomaly Worker Models: Inspired by our previous work [44], we choose one-class classifier workers for detecting anomalies in Stages 1 & 2. These workers are trained by the features of benign traffic (normal) from their respective IoT device type, offering the ability to determine whether a traffic observation belongs to the normal (benign) class or not. To

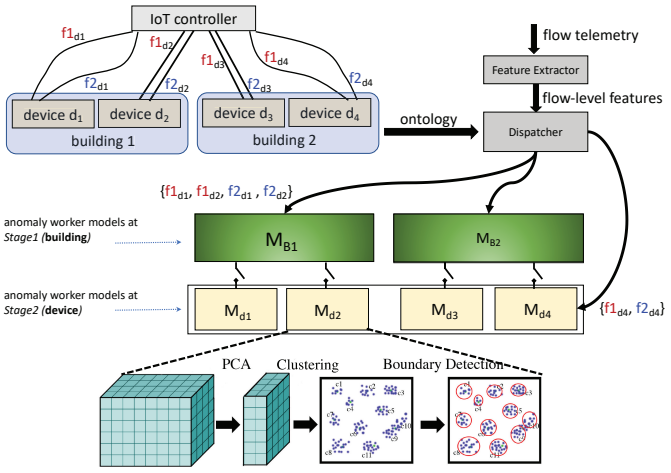


Fig. 7. Structure of our anomaly detection – illustrating a sample IoT controller that communicates with four devices located in two buildings (two devices in each building) where each device generates two flows ($f1$ and $f2$) for its network activity.

train the model for anomaly workers, we employed three main techniques, including probabilistic (*i.e.,* Gaussian mixture), domain-based (*i.e.,* one-class SVM), and cluster-based (*i.e.,* DBSCAN, Kmeans). We found that the clustering approach best models the benign behavior of the IoT types we tested. Therefore, we employ a clustering-based outlier detection algorithm comprising three steps, namely PCA, Clustering, and Boundary Detection, as shown at the bottom of Fig. 7.

Principle Component Analysis (PCA): We discussed earlier in this section that each flow rule contributes to 14 features. This means that the profile of a device type containing four flow rules would require a total of 56 (4×14=56) features. This number of features (per connected IoT device) can be computationally expensive to analyze, particularly at scale (say, thousands of connected devices). We note that certain features are highly correlated and hence can be transformed, reducing the space dimension. We, therefore, employ PCA [56] to extract the principal components of our features that are orthogonal to each other. We use the Kaiser rule [57] (eigenvalues >1) to deduce and select the most suitable set of principal components that capture as much variation across benign instances in our dataset. As per PCA requirement, we normalize all features using the z-scores method. Table. III summarizes the count of features identified for three Steinel people counting cameras in a building. We see how PCA reduces the feature dimension significantly while a considerable level of variations in the training dataset is covered.

Clustering: As discussed above, the network traffic of each connected device will be analyzed by a collection of anomaly detection models specific to the device. Therefore, management of such complex inference machinery demands an efficient and inexpensive clustering algorithm that can: (a) set the parameters automatically (*i.e.,* self-tuned), and (b) deal with our benign dataset containing a mix of sparse and dense regions. Among many possible clustering algorithms, we use X-means [58] (*i.e.,* a flavor of K-means algorithm), which is a fairly lightweight yet efficient clustering method. The accuracy of high dimensional data clustering depends

TABLE III
PCA ANALYSIS FOR 3 STEINAL CAMERAS LOCATED IN A BUILDING.

Worker	Building	Device $d1$	Device $d2$	Device $d3$
# Features	168	56	56	56
# PCA	16	9	6	6
Coverage (%)	98.7	99.3	85.8	82.7

on the distance function used [59]. After conducting several experiments, the Manhattan distance function provided the most optimal result. Lastly, we train the clustering algorithm by the principal components of our training dataset (obtained from PCA). As a result, the outputs of the X-means algorithm are the coordinates of the cluster heads, as shown at the bottom of Fig. 7 by green dots labeled as c_i.

Boundary Detection: We employ boundary detection to identify outliers (*i.e.,* determining whether an instance is anomalous or not). An anomaly flag is raised when an observation falls outside benign clusters. Given cluster heads and our training dataset, we compute the 97.5th percentile as a boundary for each cluster. Therefore, anomalies observed outside these boundaries are alarmed – this may lead to a minor detection of benign traffic as an anomaly (false positive alarm).

C. Dataset

We collected traffic traces (benign and attack) of 20 devices installed in 7 different buildings during six weeks of operation. To obtain traffic traces, we mirrored the entire network traffic (incoming and outgoing) to our collector server. This allows us to capture the traffic before any NAT is applied. We then used the "`tcpdump`" tool to record PCAPs on the collector machine. We replayed PCAPs on our SDN emulator [60] to generate telemetry data (continuously measured) of flow-level activity needed for model training and testing.

Table IV summarizes our dataset. For example, for each LPR camera unit (the first row), we generated more than 50K benign instances (22K for training and 33K for testing) and more than 2500 attack instances. Each instance corresponds to 1-min traffic counters (*i.e.,* byte) of all flows corresponding to ACLs contained in the MUD profile of a device. We obtained ethics clearance (UNSW Human Research Ethics Advisory Panel approval number HC190171) from our university for this trial.

To collect malicious traffic (attack) instances, we launched three types of attacks: (i) for LPR cameras (located at the entrance/exit of our campus multi-story car park), we covered their lens to stop normal operation (detecting cars entering/exiting), emulating physical jamming attacks; (ii) for beam counters, we launched another emulated jamming attack (Fig. 6(b)) on beam counter sensors located at the doorways of three large lecture theaters on campus – each of these theaters has two, three, and four doorways; and (iii) for Steinel cameras, we launched a malware attack (emulated by doubling the rate of pulling data from the camera) with an intent to exhaust victim's battery.

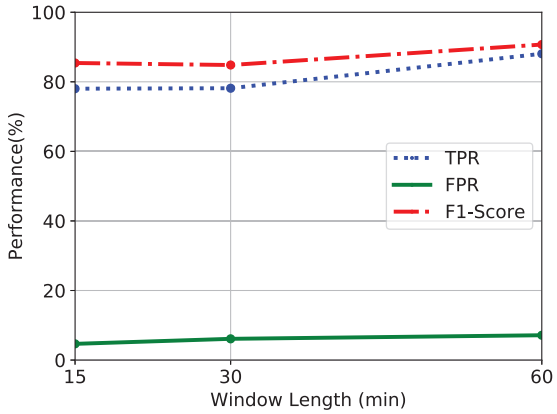


Fig. 8. Impact of window size on the performance of anomaly detection.

D. Feature Analysis

We now look into the importance of features in our anomaly detector models. Table V summarizes the impact of three different feature sets on the performance of anomaly detection: (a) "***feature-set-1***" corresponds to the 14 features of individual ACL flows, *e.g.,* HTTP, HTTPS, we identified in §IV-B; (b) "***feature-set-2***" corresponds to the same features, but computed on aggregate of ACL flows (one incoming and on outgoing); and (c) "***feature-set-3***" corresponds to features used in [44] for detecting volumetric attacks – this set computes sum, mean, and variance of packet size and count at multiple time-granularities including 2-min, 3-min, 4-min, 8-min, 16-min, and 64-min. It can be seen that even though the three feature sets give almost the same overall accuracy (about 92%) the feature-set-2 completely misses attack instances, highlighting the fact that coarse-grained flow telemetry would not be able to tightly model the network behaviors, and hence results in poor visibility. Also, we observe that the feature-set-3 yields a lower TPR (59.5%) compared to the feature-set-1 (88.0%). We observed feature-set-1 provides the best result in terms of attack detection and FPR. This is because features-set-1 captures more information of the timeseries waveform and hence is able to detect subtle changes in traffic rates, but features-set-2 fails to capture fine-grained behaviors, hence giving poor performance. Looking into individual attacks, we found that feature-set-1 was able to detect all attack streams with some delays (particularly, early attack instances, closer to the start, went undetected), causing a reduction in TPR. Note that, the average delay in detecting attacks on license-late recognition (LPR) camera, beam counters (EvolvePlus), and people counting cameras (Steinel) was 55 min, 19 min, and 1 min, respectively.

We further analyzed the impact of the length of sliding windows on the performance of models. Fig. 8 shows our results. Unsurprisingly, the overall accuracy (F1-Score) and TPR improve with larger windows, but at the cost of slightly higher false positives (mis-detecting benign instances), specially when attacks are low-profile and long in duration, displaying behaviors closer to benign traffic at least in short-term. Lastly, it is important to note that a larger window would result in a higher cost for computing and maintaining features.

TABLE IV
OUR DATASET: TRAINING, TESTING, AND ATTACK INSTANCES.

IoT device type	# devices	# training instances per unit	# testing instances per unit	average # attack instances per unit
LPR camera	2	22000	33575	2674
EvolvePlus beam counters GW	4	22000	21940	1036
Steinel counting camera	6	22000	21158	2752

TABLE V
IMPACT OF VARIOUS FEATURES ON THE PERFORMANCE OF ANOMALY DETECTION.

	Anomaly detectors	TPR (detected attack instances)	FPR (mis-detected benign instances)	Accuracy: (TP+TN)/(TP+FP+TN+FN)
feature-set-1	our 14 features of individual ACL flows	88.0	7.2	92.5
feature-set-2	our 14 features of aggregate ACL flows (incoming & outgoing)	0.0	0.0	92.4
feature-set-3	Features [44] of individual ACL flows	59.5	5.0	92.7

TABLE VI
PERFORMANCE RESULTS OF ANOMALY DETECTION MODELS.

Anomaly detectors	All devices			Nedap LPR		EvolvePlus GW		Steinel camera	
	Accuracy	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
Stage1 only	84.6	97.5	16.5	92.0	13.2	98.4	16.8	100.0	6.7
Stage2 only	88.3	88.1	11.7	78.9	11.1	52.6	17.3	100.0	6.0
Stage1 & Stage2 combined	**92.5**	88.0	7.2	78.8	8.4	52.4	5.11	100.0	5.1

E. Evaluation Results of Attack Detection

Our primary objective here is to quantify the effectiveness of our hierarchical inference models in detecting sophisticated attacks on IoT devices spread across different locations. We now evaluate the accuracy of our trained models (Stage1 and Stage2) in detecting attacks. A summary of our results is shown in Table VI. We quantify the performance of models in three scenarios namely "Stage1 only", "Stage2 only", and "Stage1&Stage2 combined". Note that Stage2-only inferencing is a device–specific anomaly detection scheme (our previous work [44]) which we use as baseline in our analysis. It can be seen that the best accuracy (92.5%) is obtained when a combination of both Stages is used. We also note that combining Stage1 and Stage2 reduces the false-positive-rate (FPR) to a minimum of 7.2%. This FPR may still sound high for a real infrastructure. We note that the majority (more than 90%) of false-positive alarms do not persist for successive one-minute epochs (intermittent alarms), and hence the risk of mis-classification is low. One can reduce the FPR by raising alarms only when attack persists over successive epochs. For example, expecting persistence over two epochs would reduce the FPR to 5% – this filtering can also lead to a lower TPR, and hence incurring a delay in detecting attacks. In terms of detection rate, we see that Stage1 only gives a better result (97.5% TPR). However, it does not provide any information on which device is involved in the attack.

Note that for Steinel controller (the last column in Table VI), given its periodic traffic pattern, having two stages of inferencing does not enhance the performance of models – each of Stage1 or Stage2 gives the same accuracy as the combined models (100% TPR and 5-6% FPR). Moving to LPR and EvolvePlus controllers, we observe that Stage1 only gives an acceptable detection rate (92.0% in LPR and 98.4% in EvolvePlus), but the TPR for combined models is relatively lower. For example, models for two of the beam counters detected 75% of attacks while the other two models detected only 27% of attack instances. This is mainly because the model for Stage2 has a very broad view of normal behavior,

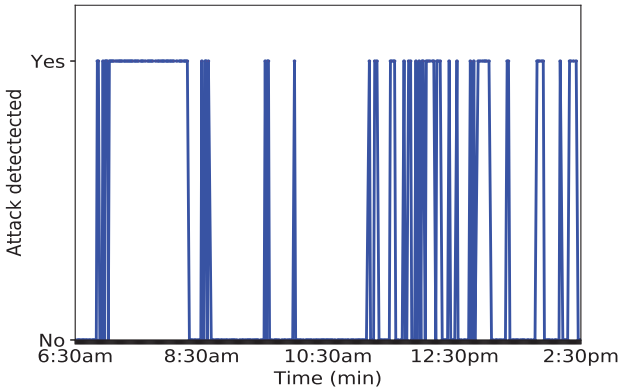


Fig. 9. Attack instances on EvolvePlus sensors gateway detected with 27% TPR.

and hence misses some attack instances. Another reason is due to delay in detecting attacks as the change in traffic patterns (by our attacks) is not very significant to be detected immediately, or expected traffic rate during certain hours (*e.g.,* between 9am-11:30am in Fig. 9) is fairly low. Note that one can further enhance the performance of attack detection by using an ensemble method [61] fed by our Stage1 and Stage2 models.

Summary: We have demonstrated how devices' location (derived from the formal knowledge representation) can be incorporated to model the normal behavior of IoT systems, and how location-aware models augment device-specific models in detecting distributed anomalies in network behaviors. Note that some of the illustrative sensors (*e.g.,* LPR cameras) we considered in this paper may generate traffic in response to activities in their physical environment. Also, our sensors in this specific setting communicate only with their corresponding controllers on a hub-and-spoke basis. That said, this does not necessarily limit the generality of our method, which is extensible to accommodate other types of communications between sensors themselves with/without sensors-controller interactions. In that

case, the MUD profile of those IoT device types is expected to include sensor-to-sensor communications. Lastly, one may consider incorporating temporal dependencies across sensors (*e.g.,* motion sensors triggering temperature control sensors), which is beyond the scope of this paper.

V. Verifying Compatibility of Formal Model with Location-Defined Network Policies

In traditional IT infrastructure, policies are typically enforced at run-time and often implemented in the form of "Match" and "Action" pairs – if packet headers match the criteria specified by the policy, then the policy action is applied to the packet. Such implementations are reactive and often do not consider the situational context of policies. In this section, we develop a method to (proactively) verify the compatibility of our formal model with location-defined network policies (pre-deployment), using semantics we defined in §III.

For a policy verification system, it is essential to have three main components: (1) semantics of policy intent, (2) methods to detect/resolve conflicting policies, and (3) verification of policies against the system ontology. Note that conflict detection/resolution is beyond the scope of this paper. In what follows next, we consider semantics and verification of three representative policies.

1. Deployment Policies: These policies are considered during the installation phase. For example, enterprise IoT devices typically support multiple communication protocols (*e.g.,* BACnet, Modbus, and HTTP/HTTPS). They may only use a specific protocol when integrated with the building automation system (with variable protocol capabilities). However, selectively disabling unsupported communication protocols is not accommodated by the MUD standard. We, instead, prune the system ontology to meeting desired deployment policies before it is enforced to the network.

We consider the following scenario to emulate a representative deployment policy in our infrastructure: "Steinel people counting camera supports both HTTP and HTTPS protocols. However, the controller in building Bldg2 only supports HTTPS". This policy is stated by:

```
acl = (sensor.isFromDevice ∪ sensor.isToDevice)
P1: <sensors, acls> := [sensor.type = "Steinel Counting Camera"
      ∧  acl.controller.isLocatedIn.isPartOf = Bldg2
      ∧  acl.controller.isLocatedIn.isPartOf.type = Building
      ∧  acl.protocol = 6
      ∧  (acl.src_port = 80  ∨  acl.dst_port = 80)]
```

We implemented the above policy SPARQL, checked against the ontology of our IoT infrastructure, and extracted devices and ACEs that violate this policy (top row in Table VII). Unsurprisingly, we found that every Steinel camera (6 cameras) in building Bldg2 has two violating ACEs in their MUD profile. Note that 6 device nodes are pointed to the same couple of ACE nodes in the ontology. For such a policy the default action would be to prune violating rules. Note that pruning ontology is a nontrivial task since it may affect other devices that are connected to the same ACE node. In the case of overlapping device nodes, the violating device nodes are separated-out along with a new pruned ACE node in the ontology.

TABLE VII
Representative policies and violations.

Policy ID	# violating devices	# violating MUD ACEs
P1	6	2
P2	4	2
P3	8	4

2. Administrative Policies: These policies are set for administrative purposes. An example of such a policy is that devices in a "highly-restricted zone" of a building are not allowed to communicate outside the building. A synthesized policy intent of such scenario is stated by:

P2: IoT devices in the MAT Theater (located in MAT building) are not allowed to have any network communication outside the building.

```
acl = sensor.isFromDevice ∪ sensor.isToDevice
P2: <sensors, acls> := sensor.isLocatedIn = "MAT Theatre"
  ∧  sensor.isLocatedIn.type = Room
  ∧  .isLocatedIn.isPartOf = "MAT"
  ∧  sensor.isLocatedIn.isPartOf.type = Building
  ∧  acl.controller.isLocatedIn.isPartOf != "MAT"
```

As shown in the second row in Table VII, four sensors (EvolvePlus beam counters) have violated this policy since they communicate with a controller located in another building (publishing their measurements to TCP 55555 on the controller). To address this issue, a possible solution would be to install a separate controller for those beam counters in the MAT building itself. This shows that the ontology not only is used to identify the violation but also provides the context of violating devices. It can potentially help during the design phase (pre-deployment) to cater to such administrative policies.

3. Organizational Policies: This category of policies is typically applied to the entire network. An example of this policy is given:

P3: IoT devices are not allowed to communicate over "unsecured" protocols (HTTP, FTP) across buildings, but may use these protocols within a building.

```
acl = sensor.isFromDevice ∪ sensor.isToDevice
  P3: <sensors, acls> := [(sensor.isLocatedIn.isPartOf !=
acl.controller.isLocatedIn.isPartOf)
 ∧  sensor.isLocatedIn.isPartOf.type = Building
 ∧  acl.protocol = 6
 ∧  (acl.src_port = 80 ∨ acl.dst_port = 80 ∨
     acl.src_port = 21 ∨ acl.dst_port = 21)]
```

Applying this policy, we found 8 devices have violated P3 (last row in Table VII). Violating devices include six units of Steinel counting camera and the two Nedap license plate recognition (LPR) cameras since Steinel cameras use HTTP protocol while Nedap cameras use FTP protocol. Such pre-

deployment compatibility checks help device manufacturers automatically identify violating behaviors of their device that may lead to a change and firmware upgrade to pass the acceptance testing.

Our MudBrick code, the three semantic files, dataset of our IoT infrastructure, and queries used for policy verification are available at [49] – source code and data will be publicly accessible once this paper is accepted.

VI. Conclusion

Today, IoT-connected building systems with a large number of heterogeneous devices are exposed to cyber-attacks, yet lack a systematic approach for evaluating or enforcing cyber-security measures. In this paper, we have developed a tool that automatically generates a formal model for the intended behavior of the entire IoT system by combining MUD profile of devices, Brick schema of buildings, and network configurations – our tool is publicly released as open-source. We showed how our model allows for static or dynamic security evaluations by applying our tool to a real IoT infrastructure on a university campus, and derived the system ontology. Following generation of the ontology, we then automatically translated it to flow rules and enforced them into the network using programmable control plane. We developed anomaly detection models to continuously monitor the activity of IoT traffic flows at building-level and device-level. Our trained models yield an acceptable accuracy of 92.5%. Lastly, we showed how the compatibility of the IoT system behaviors can be systematically checked against three representative organizational policies, prior to deployment.

References

[1] Australia Government. (2016) Smart Cities Plan. [Online]. Available: https://www.infrastructure.gov.au/cities/smart-cities/plan/index.aspx

[2] University of Wollongong. (2018) Liverpool Smart Pedestrian. [Online]. Available: http://digitallivinglab.uow.edu.au/portfolio/liverpool-smart-pedestrian/

[3] Sidewalk Labs . (2018) Sidewalk Labs Neighborhood of the Future in Toronto Is Getting Closer. [Online]. Available: https://www.citylab.com/design/2018/11/sidewalk-labs-quayside-toronto-smart-city-google-alphabet/577078/

[4] IoT Security Foundation, "Whitepaper: Can You Trust Your Smart Building?" Tech. Rep., Jun 2019.

[5] Lucian Constantin. (2016) Backdoor accounts found in 80 Sony IP security camera models. https://bit.ly/2pPNgDi.

[6] Pen Test Partners. (2017) Too cold to work? School closed? Sure your BMS hasn't been hacked? https://bit.ly/33BRLQh.

[7] F. Loi *et al.*, "Systematically evaluating security and privacy for consumer iot devices," in *Proc. ACM IoT S&P*, Dallas, Texas, USA, November 2017.

[8] M. Lyu *et al.*, "Quantifying the Reflective DDoS Attack Capability of Household IoT Devices," in *Proc. ACM WiSec*, Boston, Massachusetts, July 2017.

[9] Cisco Systems, "Midyear Cybersecurity Report," Tech. Rep., 2017.

[10] T. Brewster. (2018) Google's Doors Hacked Wide Open By Own Employee. https://bit.ly/2JrXv7A.

[11] A. Charlton. (2018) How a thermostat in the lobby fish tank let hackers steal a casino's high-roller database. https://bit.ly/2J2hy9N.

[12] S. Moss. (2017) University suffers DDoS attack from IoT vending machines. https://bit.ly/2S79dY4.

[13] H. Habibi Gharakheili, A. Hamza, and V. Sivaraman, "Cyber-Securing IoT Infrastructure by Modeling Network Traffic," in *Security and Privacy in the Internet of Things: Architectures, Techniques, and Applications*, A. Ismail Awad and J. Abawajy, Eds. John Wiley & Sons, 2021, ch. 6, pp. 151–176.

[14] J. Anand *et al.*, "PARVP: Passively Assessing Risk of Vulnerable Passwords for HTTP Authentication in Networked Cameras," in *Proc. ACM Workshop on Descriptive Approaches to IoT Security, Network, and Application Configuration (DAI-SNAC)*, Virtual Event, Germany, 2021.

[15] F. Li, Y. Shi, A. Shinde, J. Ye, and W. Song, "Enhanced Cyber-Physical Security in Internet of Things Through Energy Auditing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5224–5231, 2019.

[16] V. Sivaraman, H. Habibi Gharakheili, C. Fernandes, N. Clark, and T. Karliychuk, "Smart IoT Devices in the Home: Security and Privacy Implications," *IEEE Technology and Society Magazine*, vol. 37, no. 2, pp. 71–79, June 2018.

[17] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a Trillion (Unfixable) Flaws on a Billion Devices: Rethinking Network Security for the Internet-of-Things," in *Proc. ACM HotNets*, Philadelphia, PA, USA, Nov 2015.

[18] V. Sivaraman, H. Habibi Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home iot devices," in *Proc IEEE WiMob*, Abu Dhabi, UAE, Oct 2015.

[19] A. Pashamokhtari *et al.*, "Inferring Connected IoT Devices from IPFIX Records in Residential ISP Networks," in *IEEE LCN*, Edmonton, AB, Canada, Oct 2021.

[20] S. V. Till. (Accessed on 20.12.2021) The Convergence of the Physical and Digital Security Worlds. [Online]. Available: https://www.securityinfowatch.com/access-identity/article/21227403/the-convergence-of-the-physical-and-digital-security-worlds

[21] E. Lear, D. Romascanu, and R. Droms, "Manufacturer usage description specification," IETF Secretariat, RFC 8520, March 2019. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8520.txt

[22] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal *et al.*, "Brick: Metadata schema for portable smart building applications," *Applied energy*, vol. 226, pp. 1273–1292, 2018.

[23] J. Matherly. (2018) Shodan. https://www.shodan.io/.

[24] Project Haystack. (Accessed on 05.7.2019) Project Haystack. project-haystack.org.

[25] V. Bazjanac and D. Crawley, "Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings," in *Proc. of Building Simulation*, 1999.

[26] W3. (2007) SPARQL Query Language for RDF. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/

[27] Memoori, "Whitepaper: Brick Schema Building Blocks for Smart Buildings," https://bit.ly/3EII1FN, Tech. Rep., March 2019.

[28] A. Sivanathan *et al.*, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," *IEEE Transactions on Mobile Computing*, vol. 18, no. 8, pp. 1745–1759, Aug 2019.

[29] S. Marchal *et al.*, "AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1402–1412, 2019.

[30] F. Li, A. Shinde, Y. Shi, J. Ye, X.-Y. Li, and W. Song, "System Statistics Learning-Based IoT Security: Feasibility and Suitability," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6396–6403, 2019.

[31] R. SharmaHabibi *et al.*, "Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment," in *Proc. USENIX Security Symposium*, Boston, MA, USA, Aug 2022.

[32] H. Habibi Gharakheili, M. Lyu, Y. Wang, H. Kumar, and V. Sivaraman, "iTeleScope: Softwarized Network Middle-Box for Real-Time Video Telemetry and Classification," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1071–1085, 2019.

[33] D. Zhou, Z. Yan, G. Liu, and M. Atiquzzaman, "An Adaptive Network Data Collection System in SDN," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 562–574, 2020.

[34] A. Gupta *et al.*, "Sonata: Query-driven streaming network telemetry," in *Proc. ACM Special Interest Group on Data Communication*, Budapest, Hungary, Aug 2018, pp. 357–371.

[35] M. Zhang *et al.*, "Poseidon: Mitigating Volumetric DDoS Attacks with Programmable Switches," in *Proc. USENIX NDSS*, San Diego, California, USA, Feb. 2020, pp. 1–18.

[36] A. Hamza, D. Ranathunga, H. Habibi Gharakheili, M. Roughan, and V. Sivaraman, "Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles," in *Proc. ACM IoT S&P*, Budapest, Hungary, August 2018.

[37] A. Hamza *et al.*, "Verifying and Monitoring IoTs Network Behavior using MUD Profiles," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 1–18, 2020.

[38] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *Proc Usenix ATC*, Boston, MA USA, 2018.

[39] Z. B. Celik, G. Tan, and P. D. McDaniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT," in *Proc. NDSS*, San Diego, CA USA, 2019.

[40] V. Nagendra, A. Bhattacharya, V. Yegneswaran, A. Rahmati, and S. R. Das, "Viscr: Intuitive & conflict-free automation for securing the dynamic consumer iot infrastructures," *arXiv preprint arXiv:1907.13288*, 2019.

[41] A. Hamza, H. Habibi Gharakheili, and V. Sivaraman, "Combining MUD Policies with for IoT Intrusion Detection," in *Proc. ACM IoT S&P*, Budapest, Hungary, August 2018.

[42] M. Ranganathan, "Soft mud: Implementing manufacturer usage descriptions on openflow sdn switches," in *International Conference on Networks (ICN)*, Valencia, Spain, 2019.

[43] A. Hamza, H. Habibi Gharakheili, and V. Sivaraman, "Combining MUD Policies with SDN for IoT Intrusion Detection," in *Proc. ACM IoT S&P*, Budapest, Hungary, Aug 2018.

[44] A. Hamza, H. Habibi Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting Volumetric Attacks on loT Devices via SDN-Based Monitoring of MUD Activity," in *Proc. ACM SOSR*, San Jose, USA, April 2019.

[45] S. Singh, A. Atrey, M. L. Sichitiu, and Y. Viniotis, "Clearer than Mud: Extending Manufacturer Usage Description (MUD) for Securing IoT Systems," in *Proc. ICIoT*, San Diego, USA, Jun 2019.

[46] Y. Afek, A. Bremler-Barr, and A. Noy, "Eradicating attacks on the internal network with internal network policy," *arXiv preprint arXiv:1910.00975*, 2019.

[47] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *Proc SPW*. San Francisco, CA, USA: IEEE, 2018.

[48] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, and M. Guizani, "A survey of machine and deep learning methods for internet of things (iot) security," *arXiv preprint arXiv:1807.11023*, 2018.

[49] A. Hamza. (2019) MUD Brick Generator. [Online]. Available: https://github.com/ayyoob/mud-brick-gen

[50] T. Sutjarittham *et al.*, "Experiences with IoT and AI in a Smart Campus for Optimizing Classroom Usage," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7595 – 7607, October 2019.

[51] ——, "Measuring and Modeling Car Park Usage: Lessons Learned from a Campus Field-Trial," in *Proc IEEE WoWMoM*, Washington DC, USA, Jun 2019.

[52] Steinel. (Accessed on 15.11.2019) Steinel Presence Detectors. https://www.steinel.com.au/product-category/presence-detectors/.

[53] Apache. (2000) Apache Jena. [Online]. Available: https://jena.apache.org/

[54] Blue Yonder. (Accessed on 16.10.2019) Tsfresh. https://tsfresh.readthedocs.io.

[55] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *Proc IEEE SAI*, London, UK, Oct 2014.

[56] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[57] H. F. Kaiser, "The application of electronic computers to factor analysis," *Educational and psychological measurement*, vol. 20, no. 1, pp. 141–151, 1960.

[58] D. Pelleg, "X-means: Extending k-means with efficient estimation of the number of clusters." in *Proc ICML*, San Francisco, CA, USA, June 2000.

[59] C. C. Aggarwal *et al.*, "On the surprising behavior of distance metrics in high dimensional spaces," in *Proc ICDT*. Berlin, Heidelberg: Springer, October 2001.

[60] Ayyoob Hamza. (Accessed on 5.5.2019) SDN pcap simulator. https://github.com/ayyoob/.

[61] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[62] A. Hamza, H. Habibi Gharakheili, and V. Sivaraman, "Securing iot networks using formal behavioral modeling and dynamic flow management," Ph.D. dissertation, UNSW Sydney, 2020.

Ayyoob Hamza received his Bachelor's degree in Computer Science from the University of Colombo, Sri Lanka in 2014 and his Ph.D. in Electrical Engineering and Telecommunications from University of New South Wales (UNSW) in Sydney, Australia in 2020. He is currently the Chief Architect at CyAmast and Adjunct Associate Lecturer at UNSW. His research interests are in IoT technologies, network security, distributed systems, and software-defined networking.

Hassan Habibi Gharakheili received his B.Sc. and M.Sc. degrees of Electrical Engineering from the Sharif University of Technology in Tehran, Iran in 2001 and 2004 respectively, and his Ph.D. in Electrical Engineering and Telecommunications from UNSW in Sydney, Australia in 2015. He is currently a Senior Lecturer at UNSW Sydney. His current research interests include programmable networks, learning-based networked systems, and data analytics in computer systems.

Trevor Pering is a senior systems software engineer at Google. His research interests include building-scale Internet of Things systems, mobile devices, and interactive experience design. He received a Ph.D. in Electrical Engineering and Computer Science from the University of California, Berkeley. He is a member of ACM.

Vijay Sivaraman received his B. Tech. from the Indian Institute of Technology in Delhi, India, in 1994, his M.S. from North Carolina State University in 1996, and his Ph.D. from the University of California at Los Angeles in 2000. He has worked at Bell-Labs as a student Fellow, in a silicon valley start-up manufacturing optical switch-routers, and as a Senior Research Engineer at the CSIRO in Australia. He is now a Professor at the University of New South Wales in Sydney, Australia. His research interests include Software Defined Networking, network architectures, and cyber-security particularly for IoT networks.