IoTMap: A protocol-agnostic multi-layer system to detect application patterns in IoT networks

Jonathan Tournier, François Lesueur, Frédéric Le Mouël, Laurent Guyon, Hicham Ben-Hassine

IoTMap: A protocol-agnostic multi-layer system to detect application patterns in IoT networks

Jonathan Tournier
jonathan.tournier@insa-lyon.fr
Univ Lyon, INSA Lyon, CITI
Villeurbanne, France
AlgoSecure
Lyon, France

François Lesueur
Frédéric Le Mouël
francois.lesueur@insa-lyon.fr
frederic.le-mouel@insa-lyon.fr
Univ Lyon, INSA Lyon, CITI
Villeurbanne, France

Laurent Guyon
Hicham Ben-Hassine
laurent.guyon@algosecure.fr
hicham.ben-hassine@algosecure.fr
AlgoSecure
Lyon, France

ABSTRACT

The growth of the Internet of Things (IoT) results in a proliferation of different protocols (ZigBee, Bluetooth, 6LowPAN, Z-Wave, Wi-Fi, etc.). Organizations tend to quickly deploy several IoT applications over time and thus face heterogeneous IoT systems, combining different IoT protocols in different places of the overall system. This heterogeneity of protocols makes these networks hard to monitor or control, and some misconfigurations or unexpected device behaviours may even expose users to security issues. In this work, we propose the IoTMap system. IoTMap models interconnected and heterogeneous IoT networks, combining different protocols, by providing a generic stack and a unified packet format. IoTMap builds an iterative graph model where high-level semantics can progressively be deduced, ranging from packet transmission to application-type analysis. As such, IoTMap detects application behaviours amongst devices implementing different protocols, interconnected through a multi-protocol hub. In its current implementation (available at https://github.com/AlgoSecure/iotmap), IoTMap can inspect ZigBee, BLE and 6LowPAN networks.

CCS CONCEPTS

• **Security and privacy** → **Mobile and wireless security**; • **Networks** → *Network management*.

KEYWORDS

IoT, IoT networks, IoT modelling, heterogeneous network, IoT Security, pattern detection

ACM Reference Format:
Jonathan Tournier, François Lesueur, Frédéric Le Mouël, Laurent Guyon, and Hicham Ben-Hassine. 2020. IoTMap: A protocol-agnostic multi-layer system to detect application patterns in IoT networks. In *IoT '20: 10th International Conference on the Internet of Things, October 06–09, 2020, Malmö, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

IoT '20, October 06–09, 2020, Malmö, Sweden

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Internet of Things (IoT) refers to a network of physical objects (*things*) able to communicate (between them and with external entities) as well as to sense and interact with the real world. With different computing and sensorial capabilities, these connected devices provide complex interactions with their environment and users. IoT is rapidly growing as the number of devices strongly increases and should reach several billions in 2020[1].

The proliferation of connected devices, as well as the development of multitudinous applications, show the interest expressed by various sectors such as industry, healthcare or energy management. However, the constraints and needs of these applications lead to an overall heterogeneity in the IoT. We find a plethora of protocols settling the same problem. Hence, this heterogeneity can end up in networks with devices which communicate using multiple protocols [11]. For instance, two devices with similar sensor capabilities could use different protocols to send back their data to the same controller. However, existing monitoring tools are still protocol-specific, which means they cannot model IoT systems composed of different IoT protocols. This heterogeneity of protocols and associated monitoring tools increases the difficulty to control and monitor these networks. For instance, some misconfigurations, unexpected device behaviours or even security vulnerabilities may expose users or companies to security issues or critical data leaks. Many examples show that devices deployed in the wild with insufficient security can be turned into zombies exploited by botnets [7]. Besides, attackers can use flaws in protocol design and implementation [1, 5, 13, 19] to tamper data or take control of the network. This attack surface is difficult to monitor because of the deployment of various protocols and highlights the need to find a holistic solution to monitor and understand these networks.

In this work, we introduce IoTMap, a system capable of modelling interconnected and heterogeneous short- and mid-range IoT networks. Our contribution is two-fold: first, we describe a set of generic graphs to model heterogeneous IoT protocols (Zigbee, BLE or 6LoWPAN); second, we propose algorithms to calculate this set of graphs. In Section 2, we present our motivating scenario on which the whole paper is illustrated. In Section 3, we describe our graph-based methodology to model IoT networks and applications. In Section 4, we propose algorithms to construct these graphs from raw network captures. In Section 5, we evaluate our proposition. Finally, we describe related work in Section 6 and conclude in Section 7.

[1] Gartner - 5.8 Billion IoT Endpoints in 2020, https://www.gartner.com/[...]-io

2 MOTIVATING SCENARIO

This section presents a typical setup we target to map and which we use as an example throughout this paper. To assess the genericity of our model, this scenario is composed of twelve devices communicating over three different IoT protocols (some devices acting as gateways between different protocols). To show different cross-protocol interactions, it runs four actuator-sensor applications and four data-aggregation applications. It is illustrated in Figure 1.

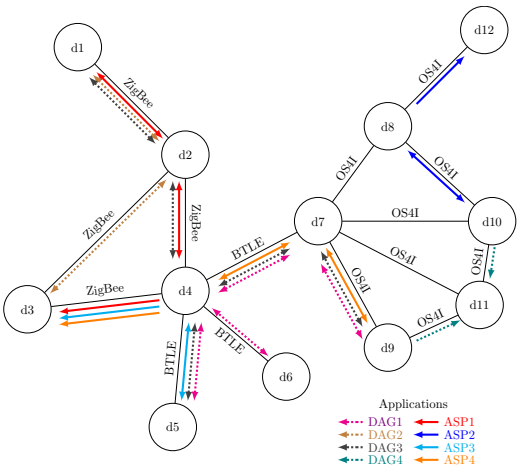

Figure 1: Experimental lab.

The three protocols are ZigBee, BLE and OS4I. ZigBee and BLE are well known and widely used IoT protocols. OS4I, standing for "Open Stack For IoT", is our local naming for an open-source stack for IoT using IEEE 802.15.4, 6LoWPAN, UDP/TCP and CoAP/MQTT.

Four applications are sensors-actuators. ASP1 is a ZigBee-only application involving $d1$, $d3$ and $d4$. The controller $d4$ requests the sensor $d1$ every sixty seconds to obtain its sensed value, and, then updates the actuator $d3$ according to the obtained the value. ASP2 is an OS4I-only application involving $d8$, $d10$, $d12$. Device $d12$ acts as the actuator and the device $d10$ as the sensor. The controller $d8$ then requests the sensor every forty seconds and sends a command regarding the value returned by device $d12$. ASP3 is a BLE-ZigBee application involving $d5$, $d3$ and $d4$. The controller $d4$ queries the sensor $d5$ every sixty seconds using BLE communications and updates accordingly the actuator $d4$ using ZigBee communications. ASP4 is a BLE-ZigBee-OS4I application involving $d3$, $d4$, $d7$ and $d9$. The controller $d4$ initiates a request to the controller $d7$ to retrieve data from the sensor $d9$. According to the returned value, the controller $d3$ sends a command to the actuator $d3$. Communications between the controllers $d4$ and $d7$ are carried over BLE. Messages between the controller $d7$ and the sensor $d9$ are sent over OS4I communications. Finally, the ZigBee protocol is used to carry communication between the controller $d4$ and the actuator $d3$.

Four other applications are data-aggregation procedures. In DAG1, $d6$ displays the temperatures sensed by $d5$ and $d9$ using BLE and OS4I. In DAG2, $d2$ monitors the status of $d1$ (temperature sensor) and $d3$ (actuator status) using Zigbee. In DAG3, $d4$ gets temperatures from $d1$ and $d5$ using BLE and Zigbee. In DAG4, $d11$ monitors the sensors $d10$ and $d9$ over OS4I.

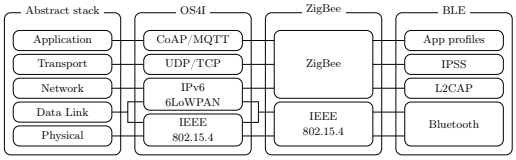

Figure 2: IoT stacks comparison.

We assume this scenario to be relevant enough to evaluate the efficiency of IoTMap as fitting with the scenarios provided in the literature [3, 16].

3 GRAPH-BASED MODELLING

This section presents the model used to represent multiple layers of an IoT network. Each layer, as illustrated in Figure 2, corresponds to a specific graph and highlights a specific point in the modelling. The modelling relies on an iterative approach where the creation of each graph (except for the first one) requires information obtained from the underlying one. For all graphs, we define N the set of nodes of the network and E the set of edges according to the graph, the whole defined as follows:

- **Data-Link graph** $G_{DL}(N, E_{DL})$ represents information obtained at the data link layer and highlights single-hop (radio) communications;
- **Network graph** $G_{NWK}(N, E_{NWK})$ represents information obtained at the network layer and highlights end-to-end communications;
- **Transport graph** $G_{TRANS}(N, E_{TRANS})$ represents information obtained at the transport layer and highlights the types of application deployed in the network with the role of each device;
- **Application graph** $G_{APP}(N, E_{APP})$ represents information obtained at the application layer and highlights the logical flows between nodes.

Some nodes, for instance gateways, may provide several network interfaces for different protocols, in which case they have multiple data-link and network addresses (one for each network interface). A node is thus defined by these characteristics: ID, a unique identifier for the node; a set of data-link (DL) and network (NWK) addresses; a set of roles, initially empty, which is completed during the analysis.

Throughout this section, each graph is illustrated with the scenario described in Section 2. The algorithmic generation of these graphs is presented in Section 4.

3.1 Data-link graph

This graph represents all connections between nodes at the data-link layer, it thus corresponds to single-hop radio communications. To construct this graph, we draw an edge for each communication between two nodes from the source node to the recipient. We then label every edges with the timestamp of the communication, the data-link and networks addresses (source and destination) and the protocol used.

Figure 3 represents the data-link graph of the scenario described in Section 2. We limit to one the number of edges between two nodes to improve the readability of the figure, but in reality, as many edges as communications are present is the graph. For instance, the

edge from $d1$ to $d2$ is labelled with the timestamp 1.5, the data-link address source $0x7b65$, the data-link address destination $0x0$ and the remaining information such as network addresses and protocol are summarized with tag P (*Payload*).

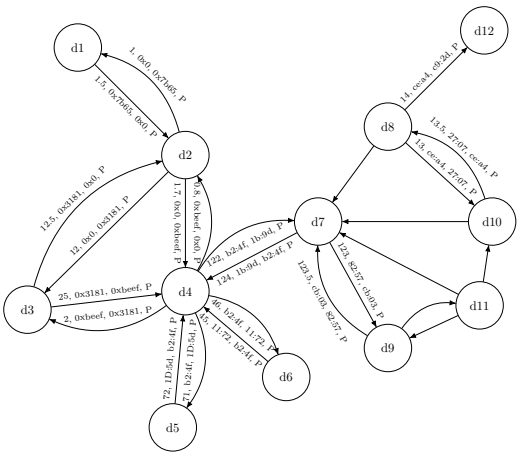

Figure 3: Data-Link graph.

3.2 Network graph

This graph represents all connections between nodes at the network layer, it thus corresponds to end-to-end communications. Hence, this graph highlights nodes that act only as routers (only forward packets). We use the same specification presented in Section 3.1 to label every edges of the network graph, i.e the timestamp, data-link addresses (source and destination), network addresses (source and destination) and the protocol.

Figure 4 represents the network graph of our motivating scenario. We observe many edges are added in comparison to the data-link graph such as from $d1$ to $d4$ or between $d11$ and $d8$. From this addition and with the data-link graph, we can deduce that $d2$ and $d10$ act as routers. Indeed, from the data-link graph, we know that $d1$ communicates with $d2$ and $d2$ communicates with $d4$ and with the knowledge provided by the network graph, we validate that $d2$ is a router that forwards messages between $d1$ and $d4$. Some edges may also be similar to those of the data-link graph. We can see that communications between the device $d7$ and $d9$, for instance, are identical in both data-link and network graphs. This is mainly due to the proximity of each node deployed in the network.

3.3 Transport graph

This graph exposes the orientation of the data flow and assigns a role for every nodes. We (1) replace every edges between two nodes by a single oriented one and (2) assign a *role* label to nodes. We merge the timestamps of every communications together and store it as a value in the label along with data-link addresses, network addresses and the protocol. We define four roles for nodes: *source (Src)*, *sink (Sk)*, *source-sink (Src-Sk)* and *controller (Ctrl)*. A node with the role *source-sink* is involved in some communications where it sends data and other ones where it receives data. The *controller* role represents a nodes aggregating and analysing data from one or several *sources* to control one or several *sinks*.

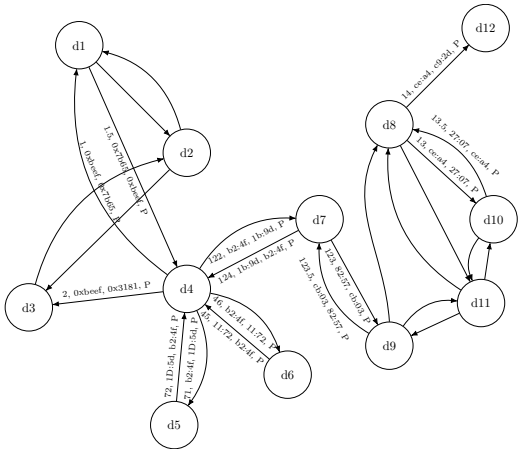

Figure 4: Network graph.

Figure 5 depicts the transport graph of our scenario. We observe that every bidirectional communications from the network graph are replaced by a single oriented edge. Roles are also assigned to nodes corresponding to the orientation of the data flow. For instance, in Figure 4, $d1$ and $d2$ use a bidirectional scheme to communicate. However, the data flow between those two devices is oriented from $d1$ to $d2$. Since the data flow starts from $d1$ to reach $d2$, $d1$ should be defined as the *source* of the data flow and $d2$ as the *sink* of the data flow. A *controller*, such as $d4$ or $d7$, is a node which is involved in at least two communications with the role of *source* in one and the role of *sink* in another. However, contrary to a *source-sink* node, these two communications are correlated. This means that $d4$ and $d7$ aggregates and analyses the data from the *source* and sends a message in consequence to the *sink*. In Figure 5, $d4$ is identified as a *controller* where $d5$ is the *source* and $d3$ is the *sink*.

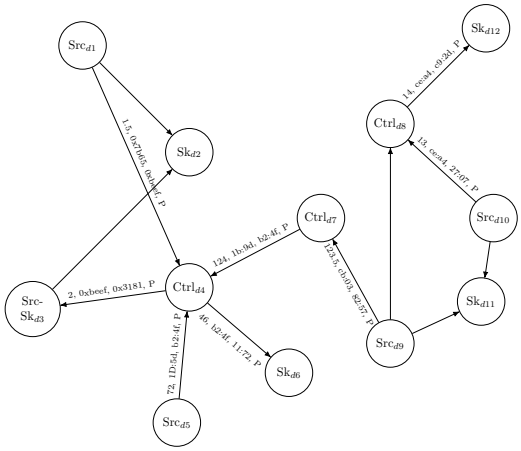

Figure 5: Transport graph with roles and oriented edges.

3.4 Application graph

This last graph highlights applications detected in the network. Applications are defined by patterns described in Subsection 4.4. Figure 6 represents Actuator-Sensor applications in the network illustrated in Section 2.

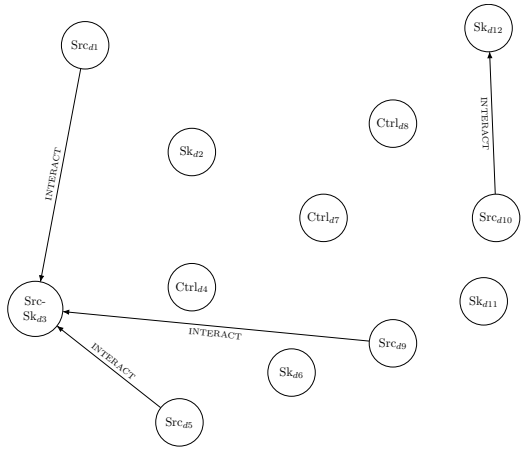

Figure 6: Application graph.

We observe that every edges from the transport graph are replaced by new edges labelled as *INTERACT*. We can observe transprotocolar interactions, such as $d9$, which is a OS4I device, interacting with $d3$, which is a ZigBee device.

4 PATTERNS DETECTION

IoTMap uses a graph-based iterative approach to provide a modelling of the network presented in Figure 7. This approach uses four functions where each one takes the output of the previous function and the desired pattern associated as inputs. The modelling starts with the function F_{DL} converting intercepted communications stored in *PCAPs* to the first graph G_{DL} using the pattern P_{DL}. The second function F_{NWK} inputs G_{DL} and the pattern P_{NWK} and generates the graph G_{NWK}. Then G_{NWK} and P_{TRANS} feed the function F_{TRANS} to model the graph G_{TRANS}. Finally, the last function F_{APP} inputs the G_{TRANS} graph and a pattern P_{APP} and model the graph G_{APP}. As represented in Figure 7, IoTMap may iterate on the F_{APP} with different patterns to produce different application graph. Only the F_{DL} requires human intervention to map the potential multiple interfaces of a device using several protocols, and then the graph generation is entirely automatic. All proposed patterns for each function are detailed in the following sections.

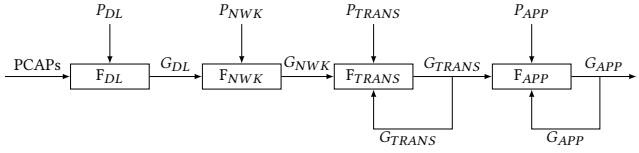

Figure 7: Graph-based iterative approach.

4.1 Data-Link pattern P_{DL}

The first step is to create G_{DL} from the captured packets. Since F_{DL} inputs are protocol-specific PCAPs, we provide a pattern for each supported protocol such as PDL_{OS4I} for OS4I protocol, PDL_{ZB} for ZigBee protocol and PDL_{BTLE} for BTLE protocol. Each pattern analyses protocol-specific information from related PCAP and computes a partial P_{DL} graph as described in Section 3.1. After analysing every PCAP, F_{DL} merges all the graphs and outputs G_{DL}.

Every pattern use a similar layer-per-layer methodology to extract information from the PCAP. For instance, the PDL_{OS4I} pattern extracts the data-link addresses from the 802.15.4 layer and the network addresses from the 6LoWPAN layer. The CoAP layer is then used to differentiate data packets from control packets (such as ACK or SYN, which we do not consider). Finally, the PDL_{OS4I} pattern creates an edge between two nodes for each packet in the PCAP and uses the information extracted to fill the label of the edge.

4.2 Network pattern P_{NWK}

The Network pattern P_{NWK} identifies end-to-end communications from the graph G_{DL}. In other words, this pattern detects communications between two nodes excluding intermediate nodes such as routers or gateways. We use information stored in each edge $e \in E_{DL}$ of the graph G_{DL} and correlate it with each node $n \in N$, where N is the set of node of each graph. Then, if the network source address stored in the edge e and represented by e_{src} is equal to the network address of the node n_{src}, we draw an edge from the node n to the recipient identified by the network destination address stored in the edge e_{dst}.

4.3 Transport pattern P_{TRANS}

In this section, we describe three different patterns to construct G_{TRANS}. We first use patterns OWT and PBC(1) to identify the direction of the data flow and accordingly assigning a role to each node amongst: *source*, *sink*, *source-sink*. This produces a temporary graph that is used by the last pattern CTRL(2). Each pattern generates a sub-graph of the final graph G_{TRANS}. Those sub-graphs are then merged by the function F_{TRANS} to output the graph G_{TRANS}.

Algorithm 1 Periodic bidirectional communications pattern (PBC)

Input: G_{NWK}
Output: Part of G_{TRANS}
  N_{NWK} the set of nodes of G_{NWK}
  E_{NWK} the set of edges of G_{NWK}
  $E_{TRANS} \leftarrow \emptyset$
  δ # Threshold
  for all $(n, m | n \in E_{NWK}, m \in E_{NWK}$ and $n \neq m)$ **do**
    $e_{(n,m)} \leftarrow \{e \in E_{NWK} | e.src = n$ and $e.dst = m\}$
    $e_{(m,n)} \leftarrow \{e \in E_{NWK} | e.src = m$ and $e.dst = n\}$
    for all $e_1 \in e_{(n,m)}$ and $e_2 \in e_{(m,n)}$ **do**
      $ts_1 \leftarrow e_1.ts$ # timestamp of e_1
      $ts_2 \leftarrow e_2.ts$ # timestamp of e_2
      if $ts_1 > ts_2$ and $ts_1 - ts_2 < \delta$ **then**
        $addEdge(e_{n,m}, E_{TRANS})$
        $addRole(n, source)$
        $addRole(m, sink)$
      end if
      if $ts_2 > ts_1$ and $ts_2 - ts_1 < \delta$ **then**
        $addEdge(e_{m,n}, E_{TRANS})$
        $addRole(m, source)$
        $addRole(n, sink)$
      end if
    end for
  end for

4.3.1 Source and Sink patterns (OWT, PBC). We propose two different patterns to detect if a node is a *source*, a *sink* or a *source-sink*. The first pattern OWT, considers one way transmissions between two nodes and identifies the sender of the communication as *source* and the recipient as *sink*. The second pattern PBC, illustrated by Algorithm 1, considers periodic bidirectional communications between two nodes where one node sends a request and the other one responds. We suppose in this pattern that no acknowledgements are sent back after a response. We identify the device that responds to a query as *source* and the one that initiates the communication as *sink*. To ensure the right role of each device, we have to determine the orientation of the exchange, it means which device initiates the communication and which one responds. We use a threshold δ as the time required to a device that received a request to respond.

4.3.2 Controller pattern (CTRL). This pattern, illustrated by Algorithm 2, considers communications with nodes tagged both *source* and *sink* and the nodes involved in corresponding communications. We start from the nodes tagged *source*-only and we follow the communication flow to the nodes tagged *sink*. We define the threshold δ as the time required by the node being both *source* and *sink* to get information from the *source*, analyse it and send a command to the *sink*. If the timestamp between the node with both roles and the *sink* and the timestamp between the node and the *source* is lower than the threshold, then we assume that this node acts as a *controller*. This pattern is used after the assignment of the *source* and *sink* roles for each device.

Finally, we build the transport graph according to the role of the nodes. We draw an oriented edge from a node tagged as *source* or *controller* to a node tagged *sink* or *controller* according to the communications observed in the network graph.

Algorithm 2 Controller pattern (CTRL)

Input: G_{TRANS}
Output: Part of G_{TRANS}
  N_{TRANS} the set of nodes of G_{TRANS}
  E_{TRANS} the set of edges of G_{TRANS}
  δ # Threshold
  for all ($n \in N_{TRANS} | n.role$ is source-sink) **do**
    $e_{sk} \leftarrow \{e | e \in E_{TRANS}, e.src = n\}$
    $e_{src} \leftarrow \{e | e \in E_{TRANS}, e.dst = n\}$
    for all $e_1 \in e_{sk}$ **do**
      $ts_1 \leftarrow e_1.ts$
      $s \leftarrow \{s | s \in N_{TRANS}, s = e_1.src\}$
      for all $e_2 \in e_{src}$ **do**
        $ts_2 \leftarrow e_2.ts$
        if $ts_2 > ts_1$ and $ts_2 - ts_1 < \delta$ **then**
          $setRole(n, "controller")$
        end if
      end for
    end for
  end for

4.4 Application pattern P_{APP}

Application pattern defines a high-level application analysis. It aims to detect specific application that may be deployed intentionally or not in the network based on the transport graph G_{TRANS}. We propose in this section two application patterns: an actuator-sensor application scheme that emphasizes interactions between two nodes and a data-aggregation scheme that identifies devices gathering data in the network.

Algorithm 3 Actuator-Sensor pattern (ASP)

Input: G_{TRANS}
Output: G_{APP}
  N_{TRANS} the set of nodes of G_{TRANS}
  E_{TRANS} the set of edges of G_{TRANS}
  δ # Threshold
  $E_{APP} \leftarrow \emptyset$
  for all ($n \in N_{TRANS} | n.role$ is $controller$) **do**
    $e_{sk} \leftarrow \{e | e \in E_{TRANS}, e.src = n\}$
    $e_{src} \leftarrow \{e | e \in E_{TRANS}, e.dst = n\}$
    for all ($e_1 \in e_{sk}, (e_2 \in e_{src})$ **do**
      $ts_1 \leftarrow e_1.ts$
      $ts_2 \leftarrow e_2.ts$
      if $ts_2 > ts_1$ and $ts_2 - ts_1 < \delta$ **then**
        $s \leftarrow \{s | s \in N_{TRANS}, s = e_1.src\}$
        $d \leftarrow \{d | d \in N_{TRANS}, d = e_2.dst\}$
        $e_{s,d} \leftarrow createEdge(s, d)$
        $addEdge(e_{s,d}, E_{APP})$
      end if
    end for
  end for

4.4.1 Actuator-Sensor pattern (ASP). This application scheme, represented in Algorithm 3 considers only multihop sensor-actuators schemes. Indeed, in the case of a direct transmission between the sensor and the actuator, we can identify the interaction in the transport graph, where the sensor is the *source* and the actuator is the *sink*. So we consider paths with at least one *controller* between a *source* and a *sink*. Then to ensure that an interaction exists between the *source* and the *sink*, we analyse the timestamp of packets sent by the *source* and those sent by the *controller*. If the difference between them is inferior than the threshold δ then we assume that the *controller* sends data or command to the *sink* due to the information received from the *source*, so this latter interacts with the *sink* through the *controller*. If multiple devices tagged *controller* exist on the path, we apply this pattern recursively.

4.4.2 Data aggregation pattern (DAG). This pattern considers node that receives packets from at least t different nodes tagged *source*. We define t as the number of sources required for a node to be detected as a data aggregator. We assume that if a node gathers data from t sources, then it analyses it or stores it for further use. This pattern is represented in Algorithm 4.

5 EVALUATION

In this section, we evaluate our proposition. First, we detail the experimentation settings. Then, we show (1) that our protocol-agnostic approach allows to detect interactions which would have remain unnoticeable if each protocol was studied independently

Algorithm 4 Data aggregation pattern (DAG)

Input: G_{TRANS}
Output: G_{APP}
  N_{TRANS} the set of nodes of G_{TRANS}
  E_{TRANS} the set of nodes of G_{TRANS}
  $N_{DAG} \leftarrow \{n | n \in N_{TRANS}$, n.role is ctrl or sk or src-sk$\}$
  t # Threshold to determine if the node is a DAG
  $E_{APP} \leftarrow \emptyset$
  for all $(n \in N_{DAG})$ **do**
    $E_{subAPP} \leftarrow \emptyset$
    $e_{sk} \leftarrow \{e | e \in E_{TRANS}, e.dst = n\}$
    $counter \leftarrow 0$
    for all $e \in e_{sk}$ **do**
      $s \leftarrow \{s | s \in N_{TRANS}, s = e.src\}$
      if $s.role$ is *source* or *controller* **then**
        $counter \leftarrow counter + 1$
        $addEdge(e, E_{subAP})$
      end if
    end for
    if $counter < t$ **then**
      $E_{subAPP} \leftarrow \emptyset$
    end if
    merge(E_{subAPP}, E_{APP})
  end for

and (2) that the controller detection pattern, quite central in our approach, is precise.

5.1 Experimentation settings

To conduct our experimentation, we setup the lab corresponding to our motivating scenario (Section 2). It features twelve IoT devices deployed over three different protocols to ensure communications between devices (Figure 1). For each protocol we use different hardware. For the subnet using the ZigBee protocol, we use a SmartThings hub, a RaspberryPi with the ConBee microchip and the RaspBee distribution, a smart plug and a temperature sensor. For the OS4I subnet, we deploy a RaspberryPi with a CC2530 dongle configured as a 6LoWPAN Border Router (6lbr) and five CC2650 sensortags programs with Contiki OS. For the subnet using the BLE protocol, we deploy two MICRO:BIT devices and we configure the two RaspberryPi devices, used in the ZigBee and 6LoWPAN subnets, to enable BLE communications.

To capture the traffic, we equipped a computer with five dongles: an ATMEL RZUSBStick with the KillerBee framework to intercept ZigBee communications, a CC2530 sensortag with the sensniff application to intercept OS4I traffic and three MICRO:BIT devices with the btlejack firmware to intercept BLE communications.

We provide some applications which are internal to a subnet (mono-protocolar) and some others that are cross-subnet (multi-protocolar). We use a BLE-ZigBee application, where the RaspberryPi acts as an interface between the BLE protocol and the ZigBee protocol. For this application, we setup a MICRO:BIT device as a temperature sensor that is requested by the Raspberry every ten seconds. For each request, the Raspberry analyses the returned value, compares it to some threshold and then sends the corresponding command to the smart plug. Another application is designed to allow devices in the BLE, ZigBee and OS4I subnets to interact with each other. It uses five devices, one MICRO:BIT that initiates the requests, one sensortag CC2650 that is the target of the requests, the ZigBee and OS4I subnets Raspberry acting as router and the smart plug. All these application are illustrated in Figure 1.

5.2 Results

We perform two evaluations of IoTMap. The first one focuses on the pattern detection according to the numbers of protocols observed. The second one measures the relevance of a specific algorithm of IoTMap using the precision and recall functions.

5.2.1 Protocol-based comparison. We perform an analysis on patterns successfully detected according to the number of protocols simultaneously observed. We focus the evaluation on patterns used to build the transport graph G_{TRANS} (OWT, PBC(1) and CTRL(2)) and the application graph G_{APP} (ASP(3)), presented in Section 4. For patterns OWT, PBC(1) and CTRL(2), we compare the number of nodes correctly tagged (CT) and those wrongly tagged (WT) amongst the four different possible values *source*, *sink*, *source-sink* and *controller*. We then evaluate the pattern ASP(3) using three parameters:

- **TP:** True Positives are edges created between two nodes and existing in our ground truth
- **FP:** False Positives are edges created between two nodes and not existing in our ground truth
- **FN:** False Negatives are edges not created between two nodes and existing in our ground truth

Three protocols are deployed in our IoT platform, we hence perform seven experimentations to cover the whole set of combination of protocols. We use the '-' separator to identify protocols observed simultaneously while a '∪' is used to symbolize that protocols are put together but not observed simultaneously. Moreover, each experimentation is based on the same dataset of intercepted communications and we use the same value of each parameter used to generate graphs. We also define a graph as ground truth (row labelled *Expected Values* in Table 1) corresponding to the applications deployed in the network and illustrated by both Figure 5 for G_{TRANS} and Figure 6 for G_{APP}. We then compare this graph to each one produced per experimentation. Results of these experimentations are presented in Table 1.

We observe that IoTMap wrongly detects the role of two nodes in almost each experimentation where only one or two protocols are observed simultaneously. This result shows that nodes involved

Table 1: Patterns detection according to the number of protocols simultaneously observed.

Patterns / Protocol(s)	*OWT/PBC/CTRL*		*ASP*		
	CT	WT	TP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	10	2	2	1	2
ZigBee-BTLE ∪ OS4I	10	2	3	3	1
ZigBee-OS4I ∪ BTLE	10	2	2	1	2
OS4I-BTLE ∪ ZigBee	12	0	2	1	2
ZigBee-OS4I-BTLE	12	0	4	2	0
Expected values	12	0	4	0	0

in several subnets using different protocols are hard to classify when information, or a part of it, is missing. As expected, when all protocols are simultaneously observed, IoTMap correctly identifies the role of every nodes, even those involved in many subnets. We observe the same result when we focus on G_{APP} pattern detection. When protocols are observed independently or in pairs, IoTMap identifies only protocol-specific interaction. We expect false positive results because IoTMap requires three parameters to generate the complete modelling. Depending on the value for each parameter, the given results can be more or less relevant with few or many false positives. Moreover, as IoTMap relies on a time-based analysis to identify patterns, two communications with timestamps very close (for instance with a difference of 0.001s) may match a pattern even if it is not expected. We evaluate the relevance of IoTMap in a second evaluation.

5.2.2 Precision and recall. We evaluate the relevance of the pattern CTRL(2). This pattern detects if a node with the role *source-sink* is considered as a *controller*. We use the precision and recall methodologies to measure the relevance of the pattern detection we consider. We set three parameters such as TP for true positive, FP for false positive and FN for false negative, defined as follows:

TP: nodes detected as controller where the two other nodes involved in the communication are correlated.

FP: nodes detected as controller where the two other nodes involved in the communication are not correlated.

FN: nodes not detected as controller although the two other nodes involved in the communication are correlated.

Based on these parameters, we provide two measures, the precision(1) and the recall(2), defined as follows:

$$Precision = \frac{TP}{TP + FP} \quad (1) \qquad Recall = \frac{TP}{TP + FN} \quad (2)$$

The ground truth is represented by a graph defined with the knowledge of the applications deployed in the platform. This graph is illustrated in Figure 5. Each node and each edge of this graph correspond to the correct and expected value.

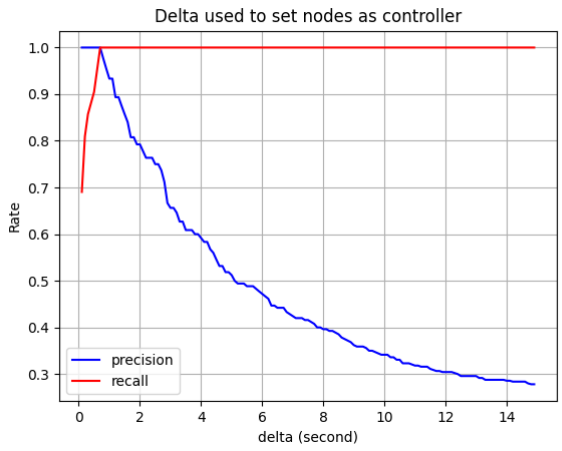

Figure 8: Rate of nodes detected as controller amongst nodes with *source-sink* role.

Figure 8 demonstrates that IoTMap is very relevant when the delta is small, even very small (< 1s), with a precision about almost 100%. We then observe a strong decrease of the precision as soon as the delta exceeds 1s, to reach a rate lower than 30% with a delta equal to 15s. As expected the recall curve grows very fast to reach 100% with a delta value close to 1s, then the curve stands to 100% along with the increase of delta. This result is explained by the short frame where false negative can be found. Indeed, false negative element are nodes that should be defined as controller but not detected by IoTMap. However, with a delta close to 1s all false negative elements are replaced by true positive elements.

6 RELATED WORK

Multiple studies focus on the capture and analysis of IoT networks. We divide those works intro four different groups: works focusing on the packet capture and manipulation, those focusing on monitoring, those focusing on traffic analysis and those focusing on monitoring and traffic analysis.

The first group of works presents frameworks that provide an interface between radio signals and digital information. Indeed, those frameworks are sets of tools for users to interact with the protocol and understand it. They provision multiple complex functionalities such as sniffing, network reconnaissance, devices discovering and simpler programs to receive and send packets over the radio for instance. In [20], Joshua Wright presents his ZigBee attack framework named KillerBee. This set of tools can intercept, inject and manipulate 802.15.4 and ZigBee traffic. Hall Joseph *et al.* introduce their Zwave attack framework in [6]. As KillerBee, EZ-Wave is able to intercept and manipulate traffic in ZWave networks. In our approach, those works are related to the physical and data link layers of our IoT stack.

The second group of studies presents works focused on traffic monitoring. We denote two different approaches to monitor networks: passive and active. An active approach relies on modules installed on network nodes to capture the sent and received traffic. An active method offers better interception capacities but uses a significative part of resources of the nodes. Several works [12, 14, 18] provide solutions to monitor traffic in networks. They can detect node failures, list neighbours, identify topology and offer the capacity to visualise the state of the network. However, those tools involve adding code to the node's applications. This requirement is not always possible, even more if sensors are already deployed and can not be reprogrammed. A passive approach consists of using sniffers to intercept radio signals and to convert them to digital information. A passive method does not impact the node resources, but it is more affected by interference and packets loss. In [2, 4, 8, 21], multiple problems are tackled to improve traffic monitoring using passive methods. They provide solutions to process traces merging using multiple sniffers such as time normalisation and duplicate transmissions. They also propose algorithms to analyse traffic to reconstruct node interactions and interfere routing path. Finally, all those works provide an interface to visualise the network as it is captured. In our approach, we only consider traffic monitoring based on the passive method. Furthermore, compared to our approach, those works are related to the data-link and network layers of our IoT stack.

The third group of studies focuses on traffic analysis. Some works are focus on devices identification or recognition to identify network threats or abnormal traffic. Authors use device identification by fingerprinting techniques to distinguish trusted and untrusted devices. Miettinen *et al.* in [10] describe a device-type classification relying on the analysis of the joining sequence of a new node in the networks. This approach raises multiple issues if all devices have already paired or if they change behaviour afterwards. In [3, 15] authors propose to classify devices from its behaviour after observing its communications. They analyse devices traffic collected during a finite time and observe specific information to deduct a different behaviour. Those approaches rely on the knowledge of devices and the regular behaviour traffic to be able to detect a malicious one. Others studies [9, 17] are focused on traffic classification to identify specific applications in the network. Sivanathan *et al.* in [17] provide a methodology and solutions to instrument and classify traffic generated by IoT devices deployed throughout campus environments. Works of [9] are an improvement of network traffic classifier using convolutional neural networks. Like [17], Lopez *et al.* capture the traffic and analyse it using multiple characteristics. Those two studies rely on a proxy installed beforehand on the network to intercepted all the traffic incoming and outgoing. All studies presented in this paragraph are related to the application layer of our approach. However, all those works are specific to a unique layer of the IoT stack. Furthermore, only few IoT protocols are supported, and devices are mainly IP-based or use no-specific IoT protocols such as HTTP, DNS, *etc.* Our approach is protocol-agnostic and we treat all layers except the physical one.

The last group includes all-in-one solutions presented in the previous groups. Siby *et al.* in [16] describe an approach covering all steps from the capture of the traffic to the analysis of this one with devices using IoT protocol such as BLE or ZigBee. Their framework uses passive monitoring methods to intercept, capture and visualise traffic. The framework is also able to analyse the trace to identify and classify devices to determine potential threats. However, their main experimentation relates to WIFI enabled devices. Furthermore, their classification relies only on detecting the type of devices, and the classification per type requires to define each type of device before. In our approach, we classify devices according to their role in the network, and we describe a holistic analysis of IoT systems using different IoT protocols and interconnected, where an event in a system can produce actions in another one.

7 CONCLUSION AND FUTURE WORK

In this work, we present IoTMap, a system capable of modelling interconnected IoT networks using multiple protocols simultaneously. First, we introduce an iterative graph-based representation to model the different layers of an IoT system, ranging from packet transmission to application-type analysis. Then, we propose different patterns to fuel the construction of each graph from the underlying one. Finally, we demonstrate that we can identify specific application behaviour amongst devices belonging to different protocols on a physical testbed.

In its current state, IoTMap supports three different protocols (ZigBee, BLE and 6LowPAN) and relies on unencrypted communications. In our future work, we will introduce new interaction patterns to identify more application types, e.g. at the network level with multicast protocols, and at the application level with streaming or data leak behaviours. We will model encrypted IoT networks using network analysis and flow tracking. Long-range IoT networks are also a perspective of improvement for IoTMap. However, even if the mesh topology exists for these IoT protocols, it is not widely deployed, making the modelling of these networks less of a priority. IoTMap is licensed under GPL 3.0 and available on Github (https://github.com/AlgoSecure/iotmap).

REFERENCES

[1] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, and Y-F Hu. 2007. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards. *Computer Communications* 30, 7, 1655–1695.

[2] B. Benmoshe, E. Berliner, A. Dvir, and A. Gorodischer. 2011. A joint framework of passive monitoring system for complex wireless networks. In *Consumer Communications and Networking Conference (CCNC)*. IEEE, 55–59.

[3] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. 2018. Iotsense: Behavioral fingerprinting of iot devices. *CoRR*.

[4] B. Chen, G. Peterson, G. Mainland, and M. Welsh. 2008. LiveNet: Using Passive Monitoring to Reconstruct Sensor Network Dynamics. In *Distributed Computing in Sensor Systems, 4th IEEE International Conference (DCOSS)*. Springer, 79–98.

[5] B. Fouladi and S. Ghanoun. 2013. Security evaluation of the Z-Wave wireless protocol. *Black hat USA* 24.

[6] J. Hall, B. Ramsey, M. Rice, and T. Lacey. 2016. Z-Wave network reconnaissance and transceiver fingerprinting using software-defined radios. In *International Conference on Cyber Warfare and Security (ICCWS)*. Academic Conferences International Limited, 163.

[7] R. Hallman, J. Bryan, G. Palavicini, J. DiVita, and J. Romero-Mariona. 2017. IoDDoS - The Internet of Distributed Denial of Sevice Attacks - A Case Study of the Mirai Malware and IoT-Based Botnets. In *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security (IoTBDS)*. SciTePress, 47–58.

[8] G. Kunzel, J-M. Winter, I. Muller, C-E. Pereira, and J-C. Netto. 2013. A passive monitoring tool for evaluation of routing in WirelessHART networks. In *4th International Embedded Systems Symposium (IESS)*. Springer, 159–170.

[9] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE Access* 5, 18042–18050.

[10] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A-R. Sadeghi, and S. Tarkoma. 2017. IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT. In *37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE Computer Society, 2177–2184.

[11] M-V Moreno, J. Santa, M-A Zamora, and A-F Skarmeta. 2014. A holistic IoT-based management platform for smart environments. In *International Conference on Communications (ICC)*. IEEE, 3823–3828.

[12] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. 2005. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*. ACM, 255–267.

[13] A. Reziouk, A. Lebrun, and J-C. Demay. 2016. Auditing 6LoWPAN Networks UsingStandard Penetration Testing Tools. *DEF CON*.

[14] S. Rost and H. Balakrishnan. 2006. Memento: A health monitoring system for wireless sensor networks. In *3rd Annual Communications Society on Sensor and Ad Hoc Communications and Networks (SECON)*, Vol. 2. IEEE, 575–584.

[15] M. Shahid, G. Blanc, Z. Zhang, and H. Debar. 2018. IoT Devices Recognition Through Network Traffic Analysis. In *International Conference on Big Data (Big Data)*. IEEE, 5187–5192.

[16] S. Siby, R. Ranjan Maiti, and N. Ole Tippenhauer. 2017. IoTScanner: Detecting Privacy Threats in IoT Neighborhoods. In *Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security (IoTPTS@AsiaCCS)*. ACM, 23–30.

[17] A. Sivanathan, D. Sherratt, H-H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. 2017. Characterizing and classifying IoT traffic in smart cities and campuses. In *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 559–564.

[18] M. Turon. 2005. Mote-view: A sensor network monitoring and management tool. In *2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*. IEEE, 11–17.

[19] L. N. Whitehurst, T. R. Andel, and J. Todd McDonald. 2014. Exploring security in ZigBee networks. In *Cyber and Information Security Research Conference (CISR)*. 25–28.

[20] J. Wright. 2009. Killerbee: practical zigbee exploitation framework. In *11th ToorCon conference*.

[21] X. Xu, J. Wan, W. Zhang, C. Tong, and C. Wu. 2011. PMSW: a passive monitoring system in wireless sensor networks. *Int. Journal of Network Management* 21, 4, 300–325.