manuscript No.
(will be inserted by the editor)

Implementation of a partial order data security model for the Internet of Things (IoT) using Software defined networking (SDN)

Abdelouadoud Stambouli[a,1], **Luigi Logrippo**[b,1,2]

[1]Université du Québec en Outaouais
[2]University of Ottawa

Received: date / Accepted: date

Abstract In previous work, the authors have shown that a generally applicable method for data security (involving secrecy, integrity and conflicts) can be built by generalizing to partial orders the well-known lattice security model and by associating simple set labels to network entities. They have have also shown how, in principle, this method can be used for data security in the Internet of things (IoT). We show in this paper how our method can be implemented by using the architecture of Software defined networking (SDN). Essentially, the labels of the entities can be used to compose SDN forwarding tables, thus ensuring that each entity can send or receive only data that is authorized to according to security constraints. We propose a centralized IoT architecture with a cloud structure using SDN as networking infrastructure, where storage entities (i.e. cloud servers) are associated with application entities. We introduce also methods for network transformations, to allow for adding or removing entities, or for changing their levels of secrecy and integrity. Finally, we show how our architecture can be used in the normal case where several data flows must be allowed in a network. A small 'hospital' example is developed for illustration. Considerations of scalability complete the paper.

Keywords:: Internet of things (IoT), Software defined networking (SDN), data and information security, data flow control, access control, secrecy- confidentiality-integrity.

1 Introduction

The Internet of things (IoT) is seen here as an evolving set of entities among which data flow. Data security, information security and data privacy are major research issues in this context, see Alaba et al.[5] Suo et al.[46], Qiang et al. [36], Hou et al. [22]. This paper presents an implementable method for directing data flows in the IoT in such a way that common data security requirements are satisfied. We decompose data security requirements into the two aspects of data *secrecy* (often also called *confidentiality*) and data *integrity* (Bishop [8]). Further, our method is capable of dealing with *conflict* requirements, which are also security requirements. As a corollary, data *privacy* requirements can also be addressed, insofar as they can be addressed by constraining data flows (Landwehr [29]).

Common approaches for data security in the IoT are based on data encryption, where the responsibility is for the entities to encrypt and decrypt data so that only certain entities can read or write them; however these operations can be burdensome or unfeasible for some devices. Our solution is based on routing: entities are labeled according to the data that can flow to them (or equivalently, that they can contain). Thus labels can be used to construct routing mechanisms allowing only secure data flows. This solution is based on the results of [31] where it was shown that the well-known lattice security model can be generalized to a partial order security model, corresponding to an entity labeling method. These principles were shown to be applicable to any network that can be represented as a directed graph (note that partial order security models are also called multi-level models in the literature [31]).

We propose an implementation of this method on a highly recognized telecommunications network architecture, the Software defined networking (SDN). We formulate an SDN architecture where SDN routing tables are compiled by the SDN controllers using the entities' labels. We use a centralized IoT architecture where all data are transferred and stored in cloud platforms and accessed by user applications.We also formulate methods for implementing transformations in the partial order of entities, following administrative or policy decisions or events determining security changes. Fi-

[a]e-mail: staa16@uqo.ca
[b]e-mail: luigi@uqo.ca

nally, we show how several coexisting data flows can be defined and implemented in a system.

We have drawn inspiration from the work by Etalle et al. [15], where a function *Tag* is defined, that maps subjects or objects to the set of tags assigned to them, and where a security administrator can formulate logic-based authorization policies that define access rights in terms of these tags. In Singh et al.[42], entities and data are labeled with two labels, one for secrecy and another one for integrity, and security policies are defined in such a way that data from entities can only flow into other entities labelled to receive them (only one label is required in our method). This research traces back to well-known foundational work by Denning, Sandhu and others on labeled lattice security models [11][41], which we have generalized to partial orders.

In Sect. 2, we briefly introduce Software defined networking. In Sect. 3, we provide a literature review, with a brief comparison to our contributions. Sect. 4 presents some background about our method. Sect. 5 presents our method in principle. Sect. 6 presents a concrete 'hospital' example. Sect. 7 present methods for network transformation. Sect. 8 shows how our method can be generalized to deal with several data flows in a single network. Sect. 9 presents how our method was tested. Sect. 10 deals with efficiency and scalability. Sect. 11 concludes the paper.

2 SDN: Software defined networking

Just as the IoT, SDN is a networking technology introduced at the beginning of this century. The literature on SDN is abundant, we mention some points in this section for completeness. Reviews of SDN and its use for security can be found in several papers, a recent one that cites many others is Huang et al.[24]. Kalkan and Zeadally [26] is a review paper that focuses on the use of SDN specifically for security in the IoT.

SDN is an evolution of the classic network model into a network defined by applications. SDN architecture separates the network control (control plane) and forwarding functions (data plane) enabling the network control to become directly programmable and centrally managed. This programming is done via SDN controllers instead of classical Internet protocols. The centralization allows the controller to maintain a global view of the network and control it through standards such as Open Flow, which is a protocol defined by the Open Networking Foundation to transfer forwarding rules from the controllers into the routers using APIs. We use in our work the most common way of programming SDN networks, where applications give abstract rules to controllers, which translate them into commands to the network equipment concerned, the SDN routers.

To justify our choice of the SDN architecture, we start from the observation that global security solutions are more efficient and focused when they are centralized, as SDN is. Further, SDN is a system designed for efficient networking and so its use for data security will be efficient. Finally, we will see that SDN allows a straightforward translation of our labels into rules for controllers and then routers. Many types of controllers and routers exist in practice, but our approach appears to be feasible on any of them.

There is research in the literature that proposes SDN-based security frameworks for the IoT. This literature will be reviewed. However, the main concerns of this literature are the management and deployment of security policies, identity management, and detection or prevention of intrusions and attacks, these subjects are outside of the scope of this paper, and some solutions proposed in these fields could be combined with our solution. Little has been done on subjects related to data secrecy and integrity and data flow control with SDN as we do in this paper.

3 Related work

We have mentioned in the Introduction some papers that have influenced our work. Other notable contributions will be discussed in this section, starting with some that propose the use of SDN within the IoT for data flow and security management. Many papers propose IoTs architectures based on SDN for the evolution of such networks.

Mamdouh et al. [33] present a new architecture for IoT infrastructure based on network virtualisation including SDN. Their SDN paradigm for the IoT consists of three different planes. The data plane regroups all the IoT network elements as simply forwarding devices. The control plane residing in the SDN controller and the management plane are complementary planes and they are jointly responsible for the management and control of network operations. This architecture provides efficient network sharing and can handle large data input from IoT devices, as well it simplifies management tasks.

In a recent paper, Quinn et al. [38] propose MLS-Enforcer, a Software-defined networking (SDN) controller that enforces MLS policies while retaining the ability to securely relabel network nodes under changing topology state and network traffic demands; this is done by using a polynomial-time heuristic relabeling algorithm. The method is restricted to lattice-structured networks, and the labels used are more complex than ours. Future research can deal with combining the ideas of this paper with the ones of ours, possibly leading to more general results.

Yassein et al. [50] propose some solutions that combine SDN and IoT networks in order to respond to the latter's challenges. Hakiri et al. [19], Wu et al. [48], and Qin et al. [37] propose solutions based on SDN to handle and to manage large numbers of devices and to schedule the flow of the data generated by those devices.

Other papers propose specific solutions that use SDNs to secure IoT networks. Flauzac, Gonzalez et al. [16][17][18] first introduced the notion of multi domain SDN. The network is divided into multiple SDNs where for each SDN we have an SDN controller as a cluster head. Then the securisation of domains will be the task of the controller that authenticates the network devices and then pushes the appropriate flow to the switch software. To ensure security of the whole network, security policies are shared among other domain controllers using the concept of security grid. In their work, an IoT device is seen as a combination of legacy interfaces and an SDN controller. They do not use a centralized SDN controller, but some devices will have SDN capabilities and will act as SDN controllers. In addition, those controllers alongside border controllers distribute routing and security rules. They also suggest DISFIRE, a Smart Firewall to provide a safe structure for SDN networks. The network is divided into clusters with an SDN controller in each cluster. These clusters execute safety strategies. For this objective, they use a protocol named OpFlex as an alternative to OpenFlow. The SDN controller can then execute a firewall that can exclude any unauthorized devices. A critique for such solutions is the use of IoT devices to play the role of controllers, given the limited resources that some of these devices have.

Aggarwal and Srivastava [3] propose a solution to secure IoT devices against external attacks instead of the data flow security that we implement. The method uses the implementation SDN & Edge Computing and the security of the devices depends on the way they are connected to the Internet.

Karmakar et al. [27] propose a security architecture for IoT networks using SDN features. This solution is divided in two phases. First, the devices are authenticated to the SDN controller using a lightweight protocol based on Elliptic Curve Cryptosystem (ECC), and then using a Policy based Security Application (PbSA). Security policies are enforced by the SDN controller. To enforce such security policies, each device of the network is assigned a number of attributes. Then, predefined security expressions will use those attributes to determine the behavior of the switches of the network.

Prabhakar et al. [35] present an SDN framework for securing IoT networks against external attacks and principally against distributed denial-of-service attacks (DDoS). This paper presents a design that incorporates the Cloud and Fog to demonstrate the capabilities of SDN in monitoring dynamic policy enforcement and access control at run time. Finally, they simulate DDoS attacks to show the capability of their solution to detect and mitigate such attacks. Chakrabarty et al. [9] propose a solution with SDN involving encrypting headers and payload to mitigate a range of attacks, but they don't consider data flow control.

Papers [13][20] propose secured solutions for the Cloud based IoT. Djouani et al. [13] use the same domains architecture presented in the mentioned work by Flauzac et al., with the addition of encryption by the devices before they send data through the network. In Han et al. [20] the authors develop a three- layer framework (perception layer, software defined network layer, and cloud-based application layer) that integrate SDN and Cloud-IoT. The developed framework consists of 23 indicators for security features, those indicators are scattered in each layer meaning that each layer has its own indicators. Each one of those indicators was given a weight based on online interviews with researchers alongside with three weighting methods. Finally, those indicators are mapped into Cloud IoT platforms such as Google Brillo and Microsoft Azure IoT to get an overall end-to-end security framework.

Liu et al.[30] address issues of SDN network latency and load balance as well as protection against spoofing and flooding attacks.

Hou et al. [22] is a recent survey paper that deals with the data perspective of security in the IoT. The authors mention the problem of data flow observation and control but they present no specific solution for it.

As we have seen, the vast majority of these papers tackle security factors such as exchange and deployment of security policies within the network in the case of SDN domains, intrusion detection, security against external attacks, etc. Some of the proposed security solutions use cryptographic algorithms that normally require sizeable computational resources. Considering that IoT devices have often limited computing resources, such solutions may be impossible to implement. Surely, some of the techniques reviewed may be compatible with our approach and, in combination with it, may lead to efficiency improvements; this will be the subject of further research. It should be clear however that none of these papers presents a data flow control method comparable to the one presented here.

Coming to work more related to ours, some papers propose the use of different types of access control and data flow control policy models in the IoT, for example Smriti and Sandhu[43] propose the use of Attribute-based Access Control (ABAC), and Xie et al. [49] propose the use of provenance based data flow control (PDFC), defined by fairly complex authorization rules. Our policy model is simpler, covers both access and flow control, and has well-defined concepts of secrecy and integrity. It also considers prove-nance to the extent that our labels express provenance.

One of the closest approaches to ours, since it deals with data flow control and privacy concerns, is Al-Haj and Aziz [6]. This paper presents a solution to enforce security policies to control the routing configuration in database-defined networks. To achieve this, the authors use row-level security checks and the lattice-based model [11][41] alongside with

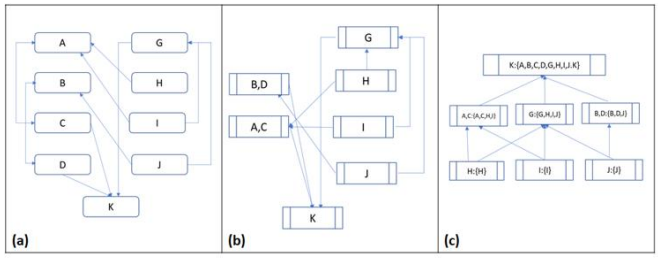


Fig. 1 a) arbitrary directed graph. b) Graph condensation. c)Partial order of component

the RAVEL architecture (Wang et al. [47]). Their solution consists in constructing routing tables by using the lattice model, encoding the tables in the data base-defined network architecture of RAVEL and enforcing multi-level security policies using row-level security as an enforcement mechanism. The authors deal separately with secrecy and integrity. To enforce upward flow of data, the authors propose to define the flow path in the Can Flow table. This path consists of sequences of nodes that data can flow into. Once a path is defined, each node in this path starting from the first one will be given a security label. Finally, a security policy is defined in respect to a multi-level model, which states that data can only flow upward from a security level into a higher one. The enforcement of downward data flow for integrity is dual. Our work considerably generalizes the work done in this paper, and in several directions. One idea that we retain for further research is the use of a data base approach to represent data flow policies.

Although many approaches have been proposed for security in SDN-enabled scenarios, several of the reviewed papers are short and present only ideas of solutions. Many do not concentrate on data security. We note the following contributions of our work: instead of using the lattice model, we use the partial order model, applicable to any network; we represent secrecy and integrity policies with a single mechanism, based on the use of a simple labeling method; we develop a generic SDN framework; we show how different data flows can be defined in a single network; we have methods for network transformations; finally, we propose an implementation and a simulation of our SDN-enabled networks.

4 Preliminaries

This work is based on results published in papers [45][32][31]. To make our paper self-contained, the essence of these results is presented in this section. We start by illustrating our use of basic results of graph theory, so far little known in the theory of data security. We refer to Figure 1.

In Figure 1a), we see a directed graph, which we call a *data network*, or simply *network*. It represents a set of *entities* in the IoT, with directional communication *channels* between them. E.g. one path in the graph is $I \rightarrow A \rightarrow C \rightarrow K$ and represents a data path such that data in entity I can go to entity A, C and K. We say that data *can flow* from I to A and the other entities, written *CanFlow(I,A* etc. Channels can be defined by capability lists or access control lists. A *component* (or *equivalence class*) in the graph is defined as a set of entities such that data can flow between any pair of them. Such entities are thought as being data flow equivalent, in the sense that whatever data might be in any of them might also be in the other. In Figure 1 b), we have identified two non-trivial components in the graph (a), they are *A,C* and *B,D*, with several trivial components containing only one entity. Double-sided rectangles will be used to represent components. A basic graph-theoretical result says that a directed graph where all components have been reduced to a single node (such is graph (b)) is a partial order of components. For our example the partial order is represented in (c), transitively reduced and not showing self-loops. The graph has been reoriented in order to show clearly the partial order. In Figure 1 c) we have also added *labels* to entities, by using the rule that an entity named X contains the name of Y in its label iff *CanFlow(Y,X)*. Data flow equivalent entities have the same label.

Arrows represent permissions for data *receiving* (or *reading*) and data *sending* (or *writing*), which in the IoT are often expressed as permissions to *pull* and *push* data (an example related to the one that we will present in Sect. 6 is in Abawajy and Hassan [2]). Henceforth we shall use the terms *send* as a synonym of *write* or *push*, and *receive* as a synonym of *read* or *pull*.

When *CanFlow(X,Y)* we also say that *Y can contain X* or that *Y can know X*.

In papers [45][32][31] it was shown that the previous graph theoretical result can be used to conclude that:

1. Any network can be seen as a partial order of equivalence classes of entities, where:
   - The entities in the top equivalence classes, the data sinks, cannot send to entities outside their class and thus can be considered to be the most *secret*: in the example of Figure 1, entity$K$ is the most secret.
   - The entities in the bottom equivalence classes, the data sources, cannot receive from entities outside their class and thus can be considered to be of maximum *integrity*: in the example of Figure 1, entities *H,I,J* are of maximum integrity.
   - Other equivalence classes are at intermediate levels of secrecy and integrity according to their receiving or sending permissions.
2. The position of equivalence classes of entities in partial orders (top, bottom or in between) can be defined by assigning to them labels; conflicts can be addressed by ex-

cluding certain combinations of entity names in labels; entities get the labels of their equivalence classes.

3. As in established theory [8], data in a network can flow from an entity *A* to an entity *B* iff *B dominates A* in the partial order of equivalence classes. This is the case iff the label of *B* includes or equals the label of *A*. This implies the existence of flows between each entity and itself, such reflexive flows are postulated for order-theoretical reasons, but are not shown and do not have to be implemented. There are two important consequences from these points, two basic facts that justify our method:
   - Complex labels showing both security levels and categories, commonly used in multi-level data security theory, can be reduced to simple sets.
   - Set labels can be used for routing data in networks, from the entities where data originate to all the entities where they can flow.
4. Given a network, efficient algorithms exist to calculate the partial order of equivalence classes and the labels of entities in the network [45]; conversely, a partial order of equivalence classes or an assignment of labels to entities define a network, which can be implemented with different channel configurations where the partial order relation is respected.
5. These results apply to any network that can be specified by means of access control matrices or permission lists, including the widely implemented Role-based access control (RBAC).

But how do we find networks such as the one of Figure 1a), representing data flows useful for specific applications? For this, we use labels that identify the possible contents of entities instead of simply entity names. The term *data category* or simply *category* is used in security theory (Bishop [8]) to identify such contents. This will becomes clear in the example of Sect. 6.

5 Our implementation method

5.1 Network configurations and graphic representation

On the basis of the mentioned previous results, we propose an SDN configuration where SDN forwarding tables are compiled by the SDN controllers using the entities' labels.

As mentioned, we choose to work on a centralized IoT architecture with a cloud structure using SDN as communication infrastructure. Several papers in the literature propose centralized configurations for IoT security such as Christos et al. [10], Hany et al. [21], and Roy et al. [39]. Centralization might seem to be inconsistent with the decentralized nature of the IoT, however our efficient centralized algorithms can reconfigure networks dynamically as necessary, see Sect. 7. SDN will work very well in closed systems such as hospitals, industrial plants, smart homes, and the like, since its architecture is well conceived for scalability and speed.

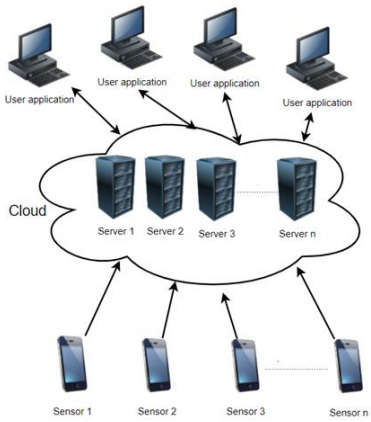


Fig. 2 Generic centralised cloud-based IoT implementation configuration

We will not discuss the characteristics and advantages of cloud architecture since this subject is part of general knowledge. In the Cloud, a data container and a server can be two distinct entities interconnected via the network. For simplicity, we choose to represent them as a single entity. In centralized IoT systems, all devices are connected through centralized cloud servers and communication between different devices must be achieved through these servers. This IoT configuration, shown in Figure 2, consists of three main layers: Sensing layer, Networking layer and Application layer. The Sensing layer consists of different types of sensors, RFIDs and other data collecting devices. This layer collects data from the environment and sends them to the cloud servers via centralized gateways. Entities requiring high integrity are found in this layer. The Application layer involves various IoT applications that use the data collected by sensors in context such as healthcare system, smart cities, etc. High secrecy entities are found in this layer. The Cloud constitutes the IoT Networking layer and all communication passes through it. Figure 3 represents a view of this architecture with the graphical notation to be used in this paper.

The Networking layer is used to connect IoT objects to the Internet, it also contains all the servers used to store the data collected from the sensing layer. Several communications technologies and protocols are used in this layer such as 3G/4G/5G, Zigbee, Bluetooth, Wi-Fi to transport data from the sensing layer to the application layer on one hand, and inside the networking layer between the servers on the other hand. Our solutions are oriented towards Wi-Fi since with this technology every entity or object in the system will have an *IPAD* (or IP address) that identifies it. This simplifies our presentation, but our approach can be extended. Some of the concepts used or implied in our architecture are:

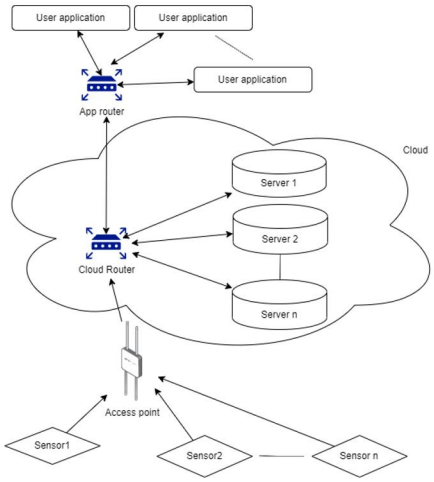


Fig. 3 Our generic implementation configuration

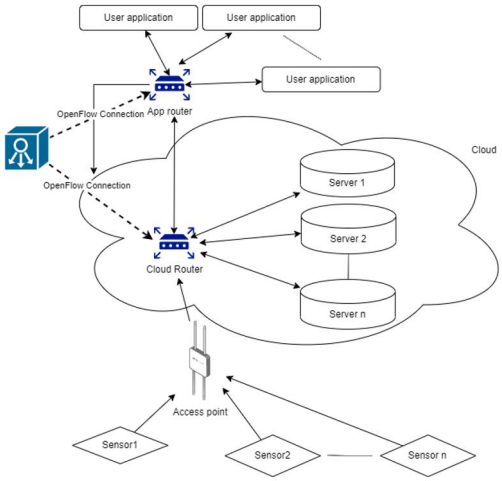


Fig. 4 Our SDN network configuration

- Connecting servers in the Cloud using cloud routers allows us to create communication channels between different servers in the Cloud. Routers are centralized entities that the servers are connected to. We find this architecture clearly presented in Ahmad Khan [4] but is implicit in other publications.
- Server to server communication: since servers are connected using cloud routers, data can flow from a server into another server. We can do this using Web-socket connection between servers so either side can send data to the other.
- Server to client communication: normally in a network, we have client to server communication, where the client sends a request to the server and the latter grants the request. However, in our case, we can have server to client communication. This can be done just like the server-to-server communication using socket.io which is a library that enables bi-directional communication between web clients and servers.

We adapt the architecture of Figure 3 to the SDN architecture, see Figure 4. The Controller will have two routers to take care of. The first router is the *Cloud router*, which interconnects the servers in the Cloud, implementing the Network layer. The second router is the *Application router* to which the cloud router connects, and which interconnects the entities in the the Application layer. We also have an *Access point* that connects the sensing layer with the cloud layer, but we do not program this one, since it is mainly charged with forwarding the data to the cloud router. These are logical devices that can be implemented by several physical devices.

Many papers in the literature mention a single controller for Wide-area SDN. In El-Garoui et al. [14] and Dias et al. [12], authors use the same controller as us (Ryu controller) to control multiple routers in their wide area SDN. The constraints on the physical placement of the servers and of the application entities will depend on factors such as the type of controllers and routers used, for example hierarchical controllers allow a more distributed placement. These are implementation concerns.

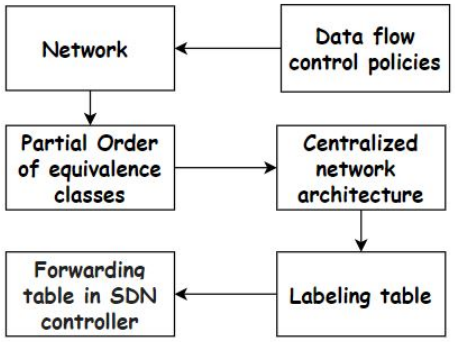


Fig. 5 Summary of our method

5.2 Labeling tables, forwarding tables and data flow control policy enforcement

Our method starts by following the principles stated in Section 4 and the references given there and is summarized in Figure 5.

We start with a network representing an application layer configuration of directly connected application entities. We identify the equivalence classes of entities in the network and the result is a partial order of equivalence classes with sources and sinks. We then assign labels to the equivalence classes, which will become the labels of the entities in each class. The label associated to an equivalence class is the set of all the data /emphcategories that the entities in the equivalence class can contain, including the categories in the labels of the equivalence classes it dominates in the par-

tial order [31][45]. These labels can be simply obtained by set union proceeding from sources to sinks. For example, if the label of a source equivalence class contains the category *BobPulse*, then all equivalence classes that dominate it will also contain this category in their labels.

The initial network will not be in the form of the *centralized cloud-based configuration* of Figure 2 since application entities will be shown as communicating directly and not through the Cloud. So, the next step, an addition to the method described above and in our previous work, is to create the cloud infrastructure. This will be done by assigning at least one *storage entity* (in practice, a server or database), to each equivalence class of entities. Hence in our architecture, data are sent simultaneously to application entities and to their associated storage entities for permanent storage. The partial order of equivalence classes will be unchanged, with storage entities added to equivalence classes. The collection of these storage entities forms the Cloud and implements the Networking Layer of the IoT.

At this point, we note that we can eliminate labels based on categories and use only entity names in labels, thus going back to the initial model presented in Sect. 4. For example, if a category such as *BobPulse* originates from an entity (a sensor) named A, then each occurrence of *BobPulse* in labels can be replaced by the name A. The partial order and the label inclusion relation are the same whether we use category names or entity names in labels. These new labels based on entity names will directly give the routing information needed to configure the SDN routers. They are compiled in labeling tables in the following simple way. For each entity such as B, we say that $A \in Holds(B)$ *iff* $Label(A) \subseteq Label(B)$. A labeling table will have a line for each entity B in the network and a column Holds containing the set of As such that $A \in Holds(B)$.

For the controller, $A \in Holds(B)$ means that data in entity A can be forwarded to entity B. The programming of SDN routers is then immediate. Forwarding tables contain the command *forward* if a packet should be forwarded from an entity to another. For each router we implement a forwarding table that only includes the entities that are connected to it. Recall that we assume an architecture where every entity has a unique *IPAD*.

We assume that we deal with routers with arbitrary large capacities. Average routers in use today can have a maximum of 250 entities connected to them [1], but this number can be increased by connecting routers sequentially (in cascade). Many modern routers adapt automatically if a port is connected to another router. These technical details are ignored here because they depend on the technology available.

Of the several columns a forwarding table may have, we take into consideration only the columns *Match Rules* and *Action*. Each packet will have a source and a destination header. If in the labeling table $A \in Holds(B)$ then the controller will create in the router a flow entry using the*IPAD (A)* source *(IP src)* and *IPAD (B)* destination *(IP dst)* in match rules and define the forward action for such a pair since it is an authorized flow. When a packet arrives to the router, the latter will compare the *IP src* and *IPAD dst* in the packet headers. If there is a forwarding rule, the router will perform it. Otherwise, the packet will be dropped. If a packet arrives to a router and the destination entity cannot be found connected to this router, the router will forward this packet to next router in the configuration. This will prevent overloading routers and will eliminate unnecessary delays. In this way, the partial order of equivalence classes, which is essential for data flow security, will be implemented.

6 Example

As an example, we consider a very small health system. It is chosen very small so that all aspects of our method can be shown in detail, with all figures fitting in the pages. A similar example was proposed in [32] but is reformulated here. It belongs to a class of systems having the following configuration: there are *sensors* for patients' blood pressure and pulse. There are wards, each of which has *doctors* and *nurses*, and patients are assigned to *wards*. There is also a*Reanimation department* and a *Chief of Medicine department*, each with a workstation. Entities other than sensors are application entities. There are the following data categories: *Press* and *Pulse* data for each patient, and *Stat* (statistics) data for each ward.

The security policies or requirements to be implemented are:

- The sensors should have highest integrity but also low secrecy, since their Pressure and Pulse data are needed by all other entities.
- The *Chief of Medicine* department will have the lowest integrity, since it uses data collected from all other entities, but also the highest secrecy, since it contains highly sensitive data for all patients and*Wards*.
- The*Wards* and *Reanimation department* take data from the sensors, process them and forward the results to the*Chief of Medicine* department, thus should have intermediate levels of integrity and secrecy.
- Conflicts: a) Patient data should be known only in each patients' own *Ward* and in *the Reanimation* and the*Chief of Medicine* departments. In addition, b) Each Ward keeps its own statistics that should be known only to it and to the *Chief of Medicine*.

We limit ourselves to an instance of this type of network where there are two Wards and three patients, *Sam, Bob and Sally*, each using a sensor. It is shown in Figure 6, and implements the specified security policies. In the figure, each rectangle represents what we call a *sensor* (three in

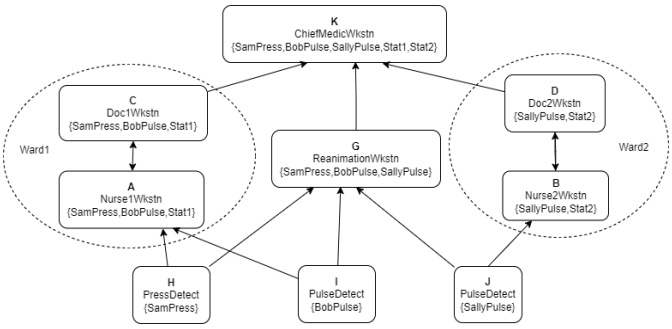


Fig. 6 Hospital example

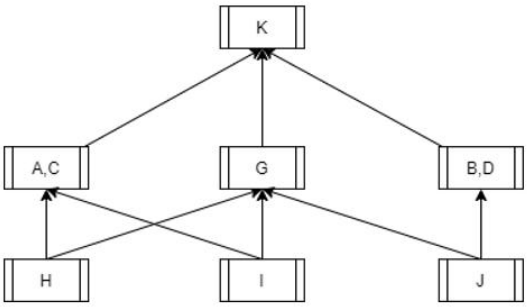


Fig. 7 Partial order of equivalence classes for the example of Fig. 6

the bottom layer) or an *application entity* (six in the layers above), and includes an upper case letter for a short name of the entity, a longer descriptive name and a label (a set of categories) in braces. Labels give the data categories that each entity can know. As earlier, arrows represent directional channels for receiving or sending, so for example in Figure 6 entity *C* can send data to *K*, or equivalently *K* can receive data from *C*.

According to IoT terminology mentioned above, the sources are the Sensing layer and the rest is the Application layer. As mentioned, the Networking layer will be provided by the storage entities that will be added in the next step.

It can be checked that the security policies above are implemented by the choice of label sets. For example, the blood pressure of *Sam* can only be known in *Ward1*, in the *Reanimation* or *Chief of medicine* departments. This configuration implements a partial order of equivalence classes, as discussed in Section 4. Note the equivalence classes *A,C* and *B,D*, since the entities in Wards have symmetric channels and thus can know the same data. The other equivalence classes are singletons. Using double-sided rectangles for equivalence classes, the partial order of equivalence classes for the network of Figure 6 is shown in Figure 7, identical to Figure 1(c). As earlier, in order to simplify the diagrams we show them transitively reduced. For example, a direct flow from *H* to *K* is allowed, and it will be in our SDN implementation.

As presented in Sect. 5, we now add the cloud layer, or network layer, to the network of Figure 6. Flows between application entities must pass through this layer, and so storage entities (such as databases, servers…) must be added to the Cloud, so at least an equivalent storage entity (i.e. with the same label) is associated to each equivalence class of application entity. Our centralized architecture is shown in Figure 8, where storage entities, which constitute the Cloud and the IoT networking layer, are identified with primes. For example, we have added an entity *G'* that allows the *ReanimationWkstn*, entity *G*, to retrieve the data received from the sensors.

When doing this configuration, we delete any entity to entity channels that are not transiting by a storage entity. None of these modifications changes the connectivity of the network, since the required data flows can still be obtained by transitivity. The partial order of equivalence classes for Figure 8 is given in Figure 9. Note that, as expected, the latter is the same as the partial order of Figure 7 for the entities that appear in both partial orders.

For the implementation configuration, the sensors are connected to access points that transfer their data to first-level cloud routers. These cloud routers forward the data to the storage entities. Finally, second-level routers are configured to connect the user endpoints to the first level of cloud routers. By adding the required routers, we obtain the configuration shown in Figure 10.

Note that all the storage entities are connected to the *Cloud router*, the application entities are connected to the *App router*, while the sensors are connected to an *access point*, which in its own turn is connected to the cloud router, just as in Fig. 3. As mentioned, no direct communication between application entities is allowed, all data must pass thought the central Cloud. However, we allow communication between storage entities in order to permit data flows to higher levels in the partial order.

This having been done, we must configure our routers; we do this by constructing the labeling tables.

The cloud router will have the function of allowing application entities and sensors (right column) to send data to the storage entities, see Figure 19. In all labelling tables that we will present, an entity name such as *A* will stand for *IPAD(A)*.

So, for example, data sent from sensor *J* that detects *SallyPulse* will arrive at the Coud router through the access point. The router will find the rows containing *J* which are the ones for storage entities *B'* and *G'* and forward the data to these entities. If the destination is not found in any row of the first router, the latter will send the data to the second router.

As a further example, note that *Label(J)= SallyPulse* and *Label(B)=Label(B')=Lab(D)=SallyPulse,Stat2*. So by label inclusion, *J,B,D,B'* can all flow to *B'*. These are all

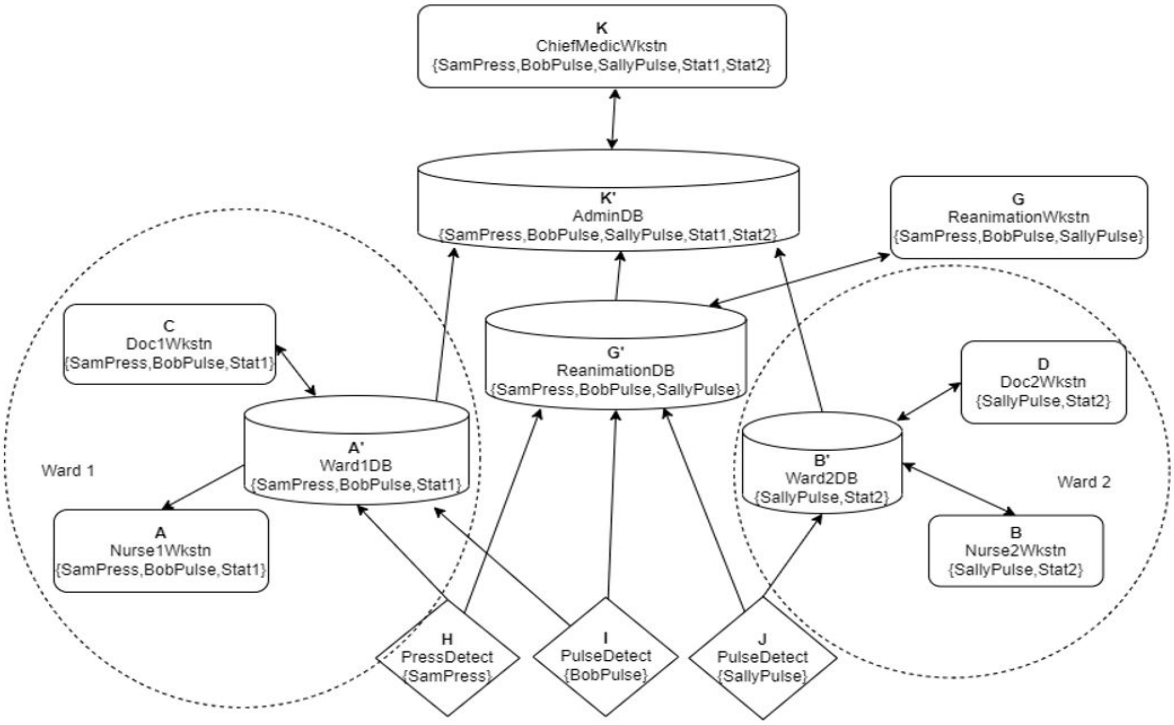


Fig. 8 IoT configuration for our case study

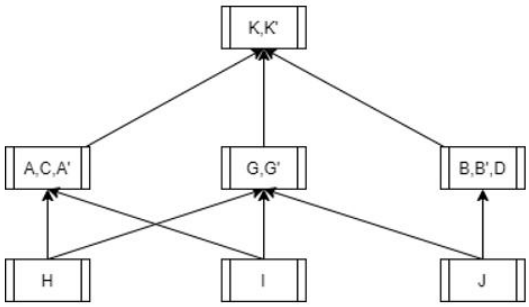


Fig. 9 Partial order for the centralized architecture

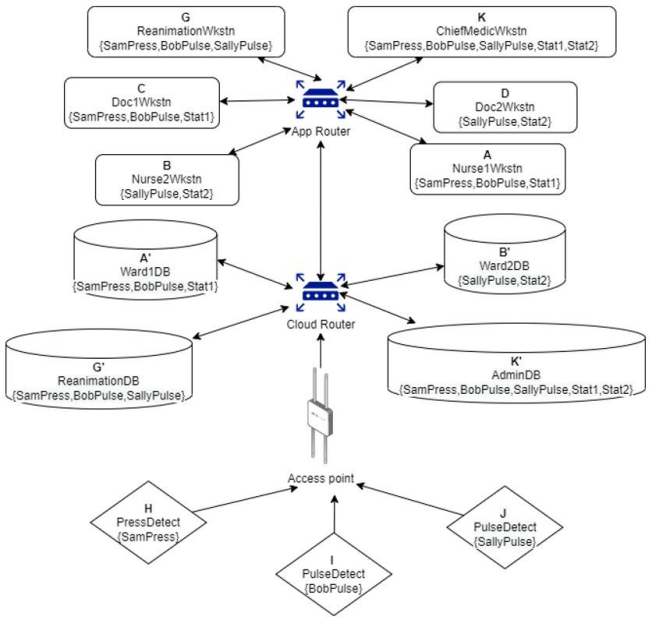


Fig. 10 The implementation configuration for the case study

and only entities whose labels are included in the label of *B'*, and so they are all and only entities whose data should be allowed to flow to *B'*, as shown in Figure 11. The table in Figure 11 can be easily constructed from Figure 9.

Once the data reaches the *App Router* the same treatment is done, we check which row of the labeling table applies according to the provenance of the data, we send the data to each designated entity and we drop the rest. By this table, the data sent to *B'* will also be available to *B, D*and *K* and the data sent to *G'* will also be available to *G* and *K*.

At first sight, the final configuration of Figure 10 seems to have no relation with the partial order of equivalence classes we started from, the only similarity being in the fact that the sensors are at the bottom layer in both. However, by the contents of the routing tables, the data flows between entities are the same. This means that the initially given policies of secrecy and integrity, as well as conflicts, are properly implemented. Clearly, this example can be scaled up by introducing many more sensors, many more wards, many

Sensor's Labeling table	
Entity	**Holds**
H	H
I	I
J	J
Labeling table of the cloud router	
Entity	**Holds**
B'	J, B, D, B'
A'	H, I, A, C, A'
G'	I, J, H, G, G'
K'	All
Labeling table of the App router	
Entity	**Holds**
G	G'
D	J, B, D, B'
B	J, B, D, B'
C	H, I, A, C, A'
A	H, I, A, C, A'
K	All

Fig. 11 Labeling table for the study case

more workstations, etc. In order to make this possible, the entities would have to be parameterized or indexed, such as *PulseDetect1, PulseDetect2, etc*. The method will remain the same, but the diagrams might be too large to be shown on a page. Evidently, practical implementations of our method will not be able to be shown in graphic format, except perhaps for some high-level representations.

To summarize our implementation method, we propose Algorithm1 that highlights how we go from the initial system architecture that can be considered as a logical topology into an actual physical one that can be configured and implemented in a real context.

7 Network transformations: creating, removing and moving entities

As all networks, IoT networks are subject to transformations, which correspond to changes in secrecy levels and changes in channel configurations. Concepts for network transformations are presented in [31]. For completeness, let us review some notions here.

Transformations can occur for many reasons, notably by intervention of a system administrator, or automatically by effect of policies. For example, in many systems there are transformations that are determined by policies expressed in terms of time, such as that at certain times, certain entities may change their permissions (i.e. labels), or disappear altogether, while others may be created. In our partial order model, the transformations that matter are the changes in the domination relation, because they are the ones that can change the data flows. Transformations that do not change data flows are adding or removing channels that are implied by transitivity. We consider three types of transformations: introduction or removal of entities, or label changes.

Algorithm 1 Implementation algorithm

Require: Logical topology (graph that describes the data flow among entities, see Fig. 6)
Ensure: Physical topology (topology that shows how the system can be configured in a real context, see Fig. 10)
for all the workstations of the network **do**
  Create an equivalent storage entity.
end for
for all entities E in the Input **do**
  for all E that are not sensors **do**
    if E does not have a bidirectional connection with any storage entity **then**
      add a bidirectional connection between E and it equivalent stroge entity .
    end if
  end for
  for all sensors **do**
    remove all connections between the sensors and entity E.
    add a connection from the sensor into storage entity equivalent to E.
  end for
  save the new topology as Input.
  for all entities X in the Input **do**
    add the same entity X in the Outputrouter.
  end for
  for all the entities X in the Output **do**
    if entity X is a sensor **then**
      Place X in the sensing layer.
    else
      if entity X is a storage device **then**
        Place X in the cloud.
      end if
      Place X in the application layer.
    end if
  end for
end for
create a cloud router $R1$ and app router $R2$ and an access point AP.
add a connection from the access point AP into the cloud router $R1$.
add a bidirectional connection between all the remaining routers ($R1$ and $R2$).
for all the entities C in the cloud **do**
  add a bidirectional connection between C and a cloud router $R1$.
end for
for all the entities A in the application layer **do**
  add a bidirectional connection from A to app router $R2$.
end for
for all the entities B in the sensing layer **do**
  add a connection from B to the access point AP.
end for
Return Output

The following sections will explain in detail the several types of transformations that can occur. We assume that transformations can be treated one at the time.

For implementation efficiency, it should be considered that each network will have different update needs. For example, some networks may have very frequent label changes, but much less frequent additions or removals: in this case, the algorithms and data structures will have to be optimized for performing quick label changes, and it may not matter if they perform less well for the other operations. Adding backward links in the labeling tables will help speed up cer-

tain searches, but will also increase the amount of memory required for the tables. Further, the labeling tables may have to be kept sorted according to some criteria, to speed up searches. Such implementation decisions should be left to the designers of specific systems. Standard data structure theory proposes methods that can be used for optimizing the basic methods we propose below, and we leave this to further research. For the purpose of this paper, we do not assume any specific organisation of the labeling tables and we note that all the operations mentioned below can be performed by using simple searches, sorts, insertions and substitutions. Efficiency and scalability will be discussed in Section 10.

In each of the cases below, we distinguish between the changes that occur in the implementation configuration and changes that occur in the labeling table. As mentioned, the two are not related in obvious ways, in fact the implementation configuration changes only in the cases of entity addition or removal.

7.1 Addition of new entities

Three types of entities can be introduced: sensors, storage entities and application entities. In each case, we assume that the new entity comes with a label, assigned by users or administrators, that implicitly specifies the intended contents of the entity and the position of the new entity in the network.

Adding a sensor : sensor's labels contain only the names of the sensors themselves, along with the names of other equivalent sensors, if any. In the implementation configuration, the new sensor must be attached to the appropriate access point. In the labeling tables, a line must be added for the new entity, containing in the *Holds* column the name of the equivalent sensors. Further, the name of the new sensor must be added to the *Holds* lists of all entities that should receive data from it.

Adding a storage entity (server or database) : if it is decided to add a new storage entity into the cloud layer, the change to the implementation configuration is the appearance of this entity attached to the cloud router. Concerning the labeling table, this new entity will have to belong to one of the already existing equivalence classes. This one already must have at least one storage entity (otherwise it will be disconnected from all other entities). Then the new entity must be added to the labeling table with the same *Holds* list as the other entities in its equivalence class; it must also be included in the *Holds* lists of all the entities in its equivalence class.

Adding an application entity: two main cases arise, according to whether the new entity belongs to an existing equivalence class or whether instead it will be in a new equivalence class (in other words, whether it has an existing label or a new one).

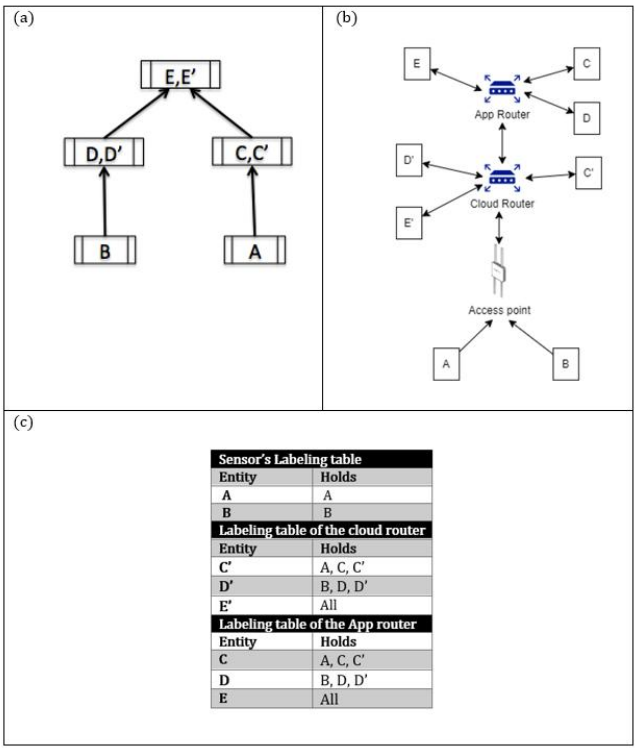
a) The first case is easily treated. For the implementation configuration, the new entity will be connected to *App router*. The new entity will access the same data entities as the other entities of its equivalence class. For the labeling table, a new entry must be created for the new entity, its name must be added to the *Holds* lists of all entities in its equivalence class, and the *Holds* list of the new entity must be the same as the *Holds* lists of these entities. The name of the new entity should be added to the Holds lists of all entities that dominate it in the partial order (that should receive data from it).

b) The second case is the case of addition of an application entity with a new label, that creates a new equivalence class. In this scenario we must add at least one corresponding storage entity for this new entity, with the same label. For the implementation configuration, the new entity must be connected to *App router* and the new storage entity must be connected to the *Cloud Router*. For the labeling tables, new entries must be created for each of the two new entities. The *Holds* lists of these two entities must be identical, and must contain the names of all entities from which they should receive data. The names of these two entities must be added to the *Holds* lists of the entities where they should send data.

Example for case a): adding a new entity *SpecialistDocWkstn*, this specialist can be consulted by *Doctor 2* in the case of an emergency, meaning that the new entity will access the data of *doc2Wkstn* through the existing storage entity named *Ward2DB*. The label of this entity will be *SallyPulse,Stat2*. Example for case b): suppose that we have the partial order presented in Figure 12(a), its implementation configuration is shown in Figure 12(b) and the labeling tables contained in the routers are shown in Figure 12(c). Entities *A* and *B* are sensors, entities *D', C'* and *E'* are storage entities (databases) and the rest are workstations. Now, we add a new independent entity *F* into the application layer with its associated storage entity *F'* . This new entity will receive data from sensor *A*, process them and send the results to entity *E*. This addition will affect both labeling tables, some lines are added to both tables in order to include the new entities. The changes are shown in Figure 13.

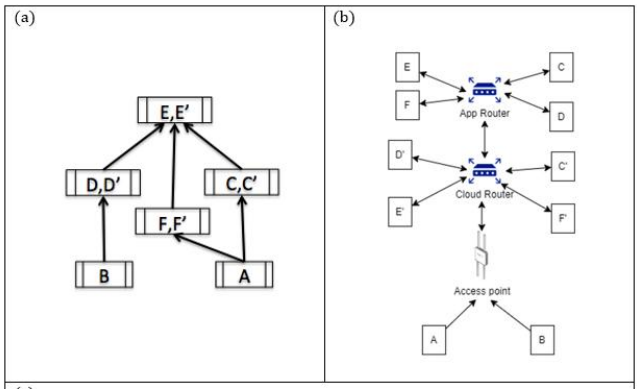
7.2 Entity removal and entity failure

This will change the implementation configuration, since the removed entity will not be included in the new implementation configuration. As in the case of addition, we have three cases: removing a sensor, removing a storage entity, and removing a workstation. In the two latter cases, it should be kept in mind that removal of an entity does not make it



Sensor's Labeling table	
Entity	Holds
A	A
B	B
Labeling table of the cloud router	
Entity	Holds
C'	A, C, C'
D'	B, D, D'
E'	All
Labeling table of the App router	
Entity	Holds
C	A, C, C'
D	B, D, D'
E	All

Fig. 12 (a) Partial order of the example. (b) Corresponding implementation configuration. (c) The labeling table of the two routers



Sensor's Labeling table	
Entity	Holds
A	A
B	B
Labeling table of the cloud router	
Entity	Holds
C'	A, C, C'
D'	B, D, D'
E'	All
F'	A, F, F'
Labeling table of the App router	
Entity	Holds
C	A, C, C'
D	B, D, D'
E	All
F	A, F, F'

Fig. 13 (a) The new partial order for the example of Fig.10. (b) its new corresponding implementation configuration. (c) The new labeling table of the two routers

necessary to find alternate paths in a network, since labeling tables contain the *IPADs* of all potential receivers of a data item. In fact, the system will keep working properly even if nothing is done in the case of entity removal: simply, data may continue to be sent to a non-existent entity. In this sense, we claim that our system is tolerant to entity failure, an important property in the IoT.

Removing a sensor: the sensor must be removed from the implementation configuration. All occurrences of the name of the sensor must be removed from the labeling tables.

Removing a storage entity: the fact that every equivalence class of entities must have a storage entity implies that a storage entity can be removed if and only if the there remains at least one storage entity in its equivalence class. The name of the storage entity must be removed from the labeling tables, however these tables should already contain references to other equivalent entities, so nothing else needs to be changed. In practice, the different storage entities in an equivalence class may have different contents, and if so some contents may have to be copied, but we leave this as an implementation issue.

Removing an application entity : in our example this would be removing a workstation. In this scenario, we need also to check the equivalence classes. We have two cases:

a) If the equivalence class that contains the entity to remove has at least another application entity in it, we only remove the intended entity and we leave the corresponding storage entities for the other application entities. The name of the removed entity must be removed from the labeling tables.

b) Otherwise, we remove the intended entity and all the equivalent storage entities since none of them is required any more. The names of all such entities must be removed from the labeling tables.

7.3 Label changes

Changing the label of an entity is equivalent to removing the entity and then adding it with the new label, and so it can be done by combining the two procedures. This change does not need to have an effect onthe implementation configuration, since the entity can remain in its place. The labeling tables will have to correspond to the new labels.

Label changes may be requested by administrators or may occur by the effect of policies in order to create or remove data transfer channels or increase or decrease the secrecy or integrity of entities; these changes normally lead to new partial orders. Here is a simple example. Suppose that entity A has *label* A and entity B has *label* B. Neither of them dominates the other in the partial order, and so there is no data flow between A and B; both A and B have maximum secrecy and integrity. An administrator may decide to create

a flow from A to B and to do so it can change the label of$B$ to A,B. This simultaneously decreases the integrity of B and the secrecy of A, while the secrecy of B and the integrity of A remain unchanged. In the initial labeling table, each of A and B will contain only its name in the *Holds* column; in the final labeling table, the *Holds* of A will be unchanged, while A is added to the *Holds* of B.

So security concerns have to be taken care of in transformations, because these can lead to violations of security constraints, possibly by indirect data transfers through entities that may change labels while keeping previously acquired data. This problem was mentioned in [31] and a systematic study of it belongs to other work. Certain labels may have to remain forbidden through transformations. Remedial action, such as data purging, may have to be imposed. In our example, the security requirements mentioned at the beginning of Sect. 6 must be kept invariant through transformations (until these are revised, of course). If entity B's label is changed to include *SamPress* together with *SallyPulse*, then Sam's pressure can flow to entity B which is in *Ward2*, violating policy Conflicts a). This can be prevented by allowing labels including *SamPress,SallyPulse* only in the equivalence classes of entities *ChiefMedicWkstn* and *ReanimationWkstn*. If a workstation in one ward is relabeled to be moved to another ward, then all data that has so far flown to the workstation and should not flow to the second ward must be purged from the workstation. Sensors will have to keep maximum integrity and so normally their *Holds* lists will contain only their name, together with the names of other equivalent sensors if they exist. How exactly to implement all this can vary from an organization to another.

8 Networks with multiple flows

In the example of Sect. 6, we have only considered the existence of a single data flow in the network. Usually however, several separate data flows are present in a network. Each one of these flows will have different security requirements and will need to be controlled separately, hence it will have its own partial order. In [32] we have shown an e-commerce example where there are two data flows, one to carry orders and another in the opposite direction to carry billing data. We modify our hospital example to add a downward flow that we call *Diagnostic*, from the *Chief of medicine* towards the patients. For this new flow, the secrecy-integrity requirements are reversed, and labels containing combinations of patient's diagnostic data are allowed only for certain equivalence classes of entities, as shown in Figure 14. Note that, for consistency with Figure 17, in this figure we show the least secret entity at the top.

We continue to deal with a centralized network. Hence, storage entities must be added to the network. This will result in the network of Figure 15.

We say that the example of Sect. 6 deals with *Consultation* data that flow from patients towards the medical staff as we have seen. We add to this *Diagnostic* data that travels in the opposite direction and has its own requirements in term of secrecy, which leads to a different partial order. The network with the representation of the two different flows is shown in Figure 16. We have now two sets of labels, one with the flow identifier *Consultation*, the other with the flow identifier *Diagnostic*. There are also some new added entites: *BobWkstn, SamWkstn*, and *SallyWkstn* respectively L, F, and E which represent the patient applications that will allow them to consult the *Diagnostic* data flow. So some entities will have two labels. For example, the labels of ChiefMedicWkstn is as follows: *Consultation(Sam Press,BobPulse,SallyPulse, Stat1,Stat2),Diagnostic(Sam Diagnos,SallyDiagnos,BobDiagnos)*. This means that *Chief MedicWkstn* participates in the two flows, and that for each flow, *ChiefMedicWkstn* has access to data of the corresponding labels.

This example shows the usefulness of the concept of *trusted entities* that can access data belonging to different flows but are trusted to deliver the right data to the rightful entities only. One such entity is the *ChiefMedicWkstn*. This entity knows both *Sam's* and *Bob's* data and sends data to both but should not send *Sam's* data to *Bob* or vice-versa. The concept of trusted entity is well established in security theory and is present in the *Bell-La Padula* model [7], where trusted subjects are defined to be "guaranteed not to consummate a security-breaching information transfer even if it is possible". Trusted entities can be thought of as split in different parts, one for each flow to which they belong, with controlled internal communication between the parts. Each part will be governed by the label associated with its flow. A classical example is found in combat situations, where commanders receive data from the field and send orders in the opposite direction; they are trusted to keep the two data flows separate, namely not to send any sensitive field information together with the orders.

In order to implement this model, we need again to create a network where all the data is saved in the Cloud. For this purpose, we add storage entities for the newly created entities for the patients. These can be small storage spaces allocated through the patient's account created during the registration on the hospital servers.

Figure 17 represents the resulting network. We have two sets of labels, one set for each flow, respectively named *Consultation* and *Diagnostic*. For each flow, the labels associated with that flow are used.

The main difference with respect to the one-flow example is that the controller will have a forwarding table for each data flow. In the case of *Consultation* data flow, the labelling tables for the two routers will be the same as the one for the one-flow example. To understand the labeling table for the new *Diagnostic* flow, it is useful to see its partial order, this is

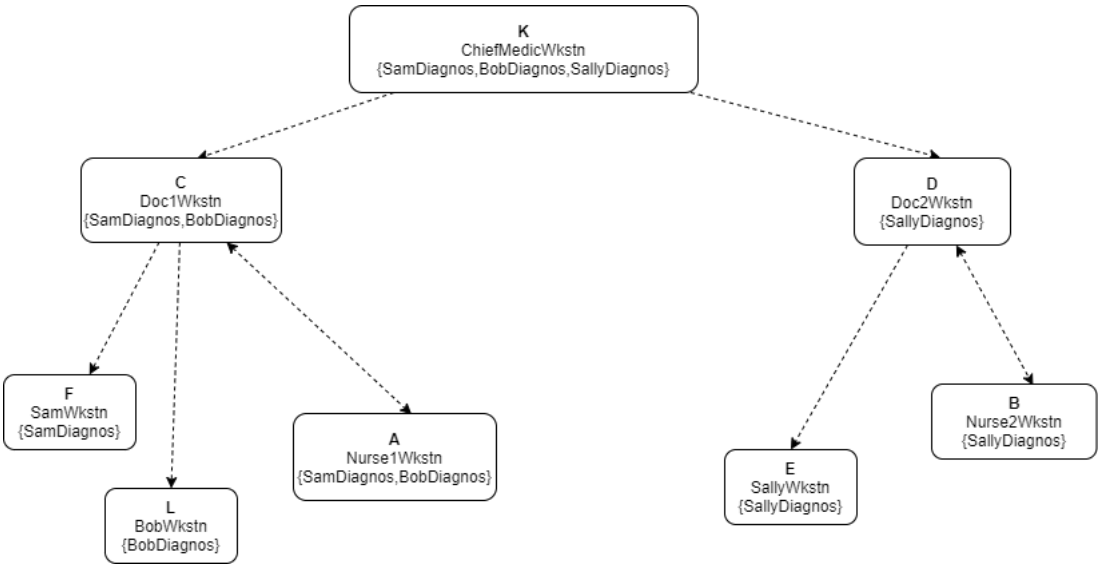


Fig. 14 The Diagnostic flow

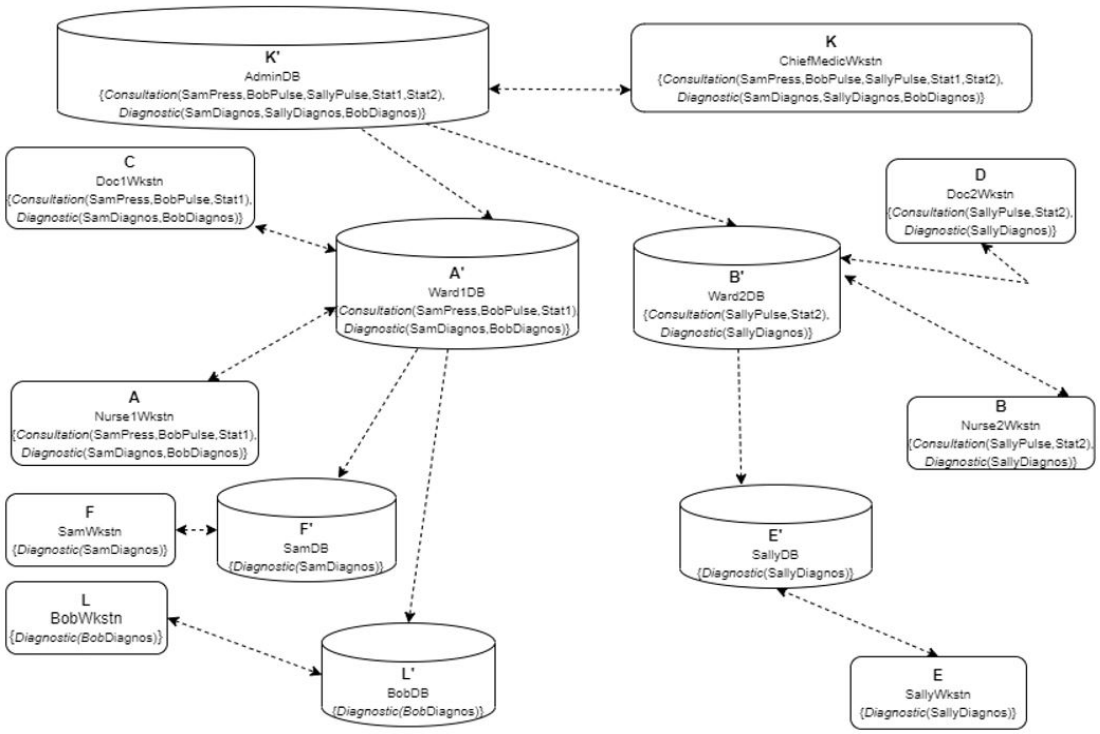


Fig. 15 The network of the Diagnostic flow

shown in Figure 18 (contrary to Figure 9, but in agreement with previous similar figures, here we have put the most secret entities at the top). The labeling table is given in Figure 19.

The new implementation configuration is presented in Figure 20. As earlier, we have two routers that interconnect the network entities: one to interconnect the storage entities and one to connect the workstations.

The main difference with respect to the one-flow example is that the controller will have a forwarding table for each data flow. In the case of *Consultation* data flow, the labelling tables for the two routers will be the same as the one for the one-flow example. To understand the labeling table for the new *Diagnostic* flow, it is useful to see its partial order, this is shown in Figure 18 (contrary to Figure 9, but in agreement with previous similar figures, here we have put the most se-

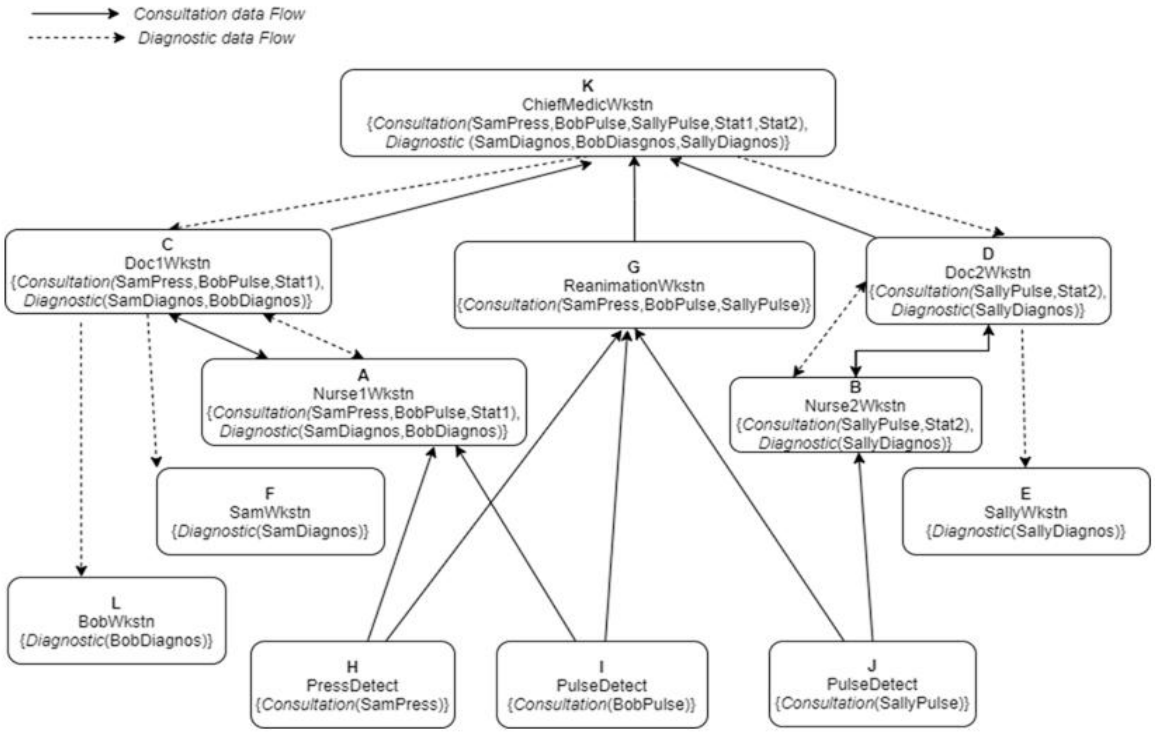


Fig. 16 Two-flow network for the hospital example

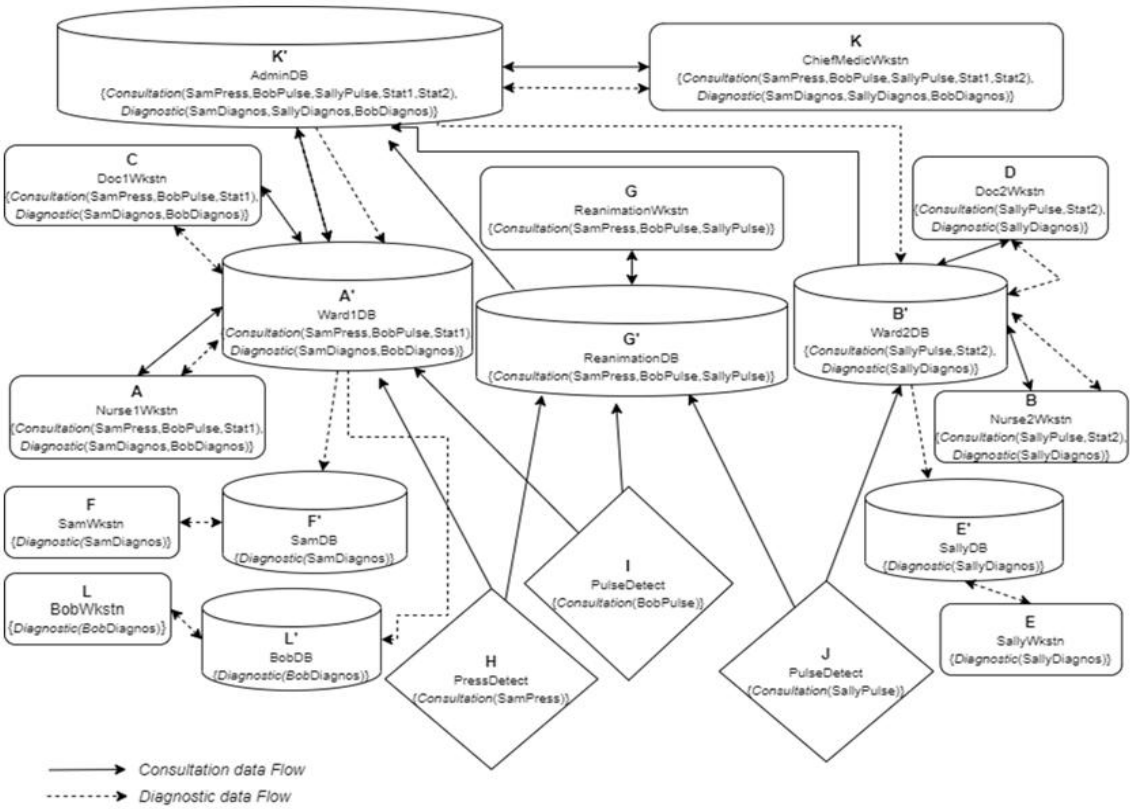


Fig. 17 Centralized two-flow network

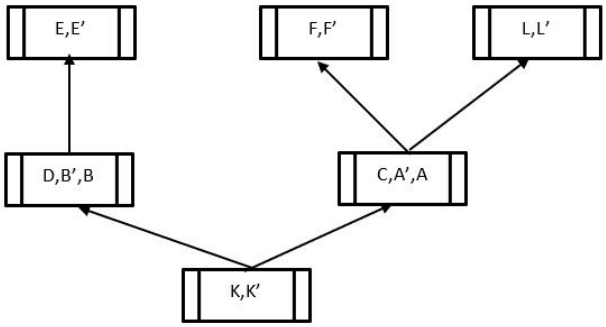


Fig. 18 Partial order for the Diagnostic data flow

Labeling table of the cloud router	
Entity	**Holds**
K'	K, K'
B'	K, B, D, B', K'
A'	K, A, C, A', K'
E'	K, D, B, E, E', K', B'
F'	K, C, A, F, F', K',A'
L'	K, C, A, L, L', K', A'
Labeling table of the App router	
Entity	**Holds**
K	K', K
C	K, A, C, K', A'
A	K, C, A', A
D	K, B, B', D, D'
B	K, D, B', B, K'
E	K, D, B, E', E, K', B'
F	K, C, A, F', F, K' A'
L	K, C, A, L', L, K', A'

Fig. 19 Partial order for the Diagnostic data flow

cret entities at the top). The labeling table is given in Figure 19.

9 Simulation and implementation of the controller

The SDN implementation of our hospital case study has been tested by using the Mininet network emulator. Mininet is a network emulator that runs a collection of end-hosts, switches, routers, and links on a single Linux kernel. It uses virtualization to make the system look like a complete network, running the same kernel, system, and user code. Mininet hosts behaves just like real machines that can run arbitrary programs. The programs can send packets through what appears to be a real Ethernet interface, with a given link speed and delay. Packets get processed by what looks like real Ethernet switches or routers as in our case. In summary, Mininet's virtual component (hosts, switches, links, and controllers) are created using software rather than hardware, and their overall behavior mimics to the one of discrete hardware elements. It is usually possible to create a Mininet network that resembles a hardware network, or a hardware network that resembles a Mininet network, and to run the same binary code and applications on either platform. Mininet is particularly adapted to simulate SDN networks, also is efficient and easy to use.

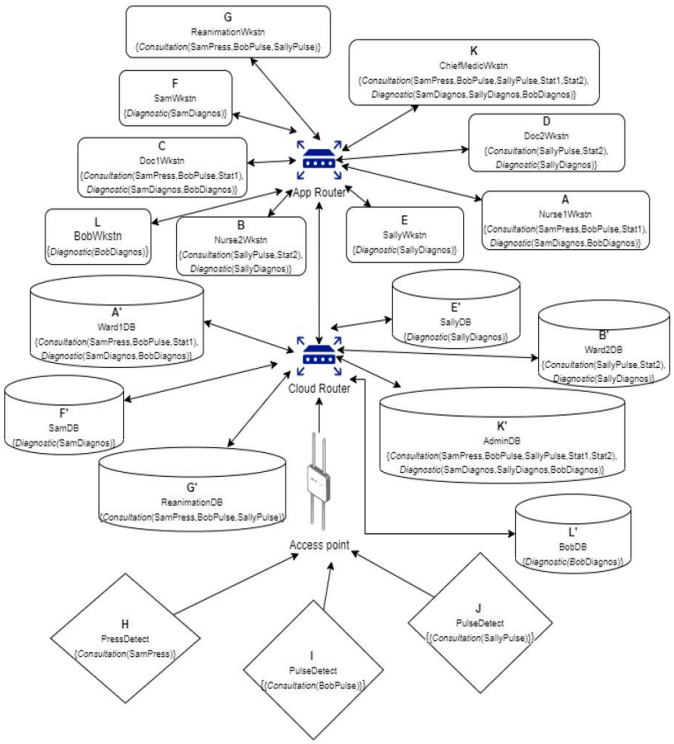


Fig. 20 Implementation configuration for the two-flow network.

For the choice of controller, several reasons led us to use the Ryu controller of Saleh et al. [40]. First, we considered the comparison study documented by Ola et al.[34]. Second, there is the fact that Ryu provides software components with well defined APIs that make it easy for developers to create new network management and control applications. Third, Ryu is the most suitable controller to use in a Mininet environment since it supports OpenFlow 1.0, 1.2, 1.3, 1.4. Fourth, because of the fact that Ryu is is Python-based, it is easier in Ryu to develop new network management and control applications in comparison with other controllers. And finally, Saleh et al. [40] and Islam and Refat [25] have reported on testing the performance of the Ryu controller in many simulation scenarios and have concluded that the controller is very suitable for prototyping and experimentation for research, experimentation, and demonstrations.

To create our implementation configuration, we have used the Python API to write a configuration Python script. First, we had to create an empty network and add nodes or entities into it. To create this empty network, we manually created a default controller called Controller c0. This default controller was replaced later with our Ryu controller.

The simulations that were done aimed to test the integrity and secrecy requirements, in other words it was tested

that by using our labeling tables and derived routing tables, data flows would only arrive to authorized entities. Parameters such as performance of the controller, scalability, etc. have been tested in other SDN-related work already mentioned, see Saleh et al.[40], Islam, Refat [25].

The two-flow configuration of Figure 20 was tested as well.

10 Efficiency and scalability

For efficiency and scalability evaluation, it is important to note that the overhead imposed by our method will occur only when the routing tables have to be updated, this means at network initialisation and whenever events such as administrative decisions or event-driven policies cause network reconfigurations; otherwise, for normal operation, the network will run at SDN speed. We have mentioned in Sect. 7 and in [31] some of the factors that should be considered when estimating the time taken by the reconfiguration process, and that detailed estimates would require consideration of the characteristics and optimizations possible for specific networks. We have mentioned that the operations involved are simple searches, additions, deletions and sorts.

In our small examples we have assumed that the labels of the entities were known, but it practice this may not be the case; labels may have to be calculated from the channel configuration, this means from capability lists or access control matrices. In [45] we have presented a method for finding the labels based on such information. It was shown there that a worst-case estimate of label calculation time is for an algorithmic complexity that is cubic on the number of entities in the network (thus excluding exponential complexity). MATLAB simulations yielding estimates were also given in that paper. It was shown in those simulations that for a network of 10,000 entities the partial order can be found and the labels calculated in about 1.5 minutes, raising to about 10 minutes for 20,000 entities and after that rising rapidly to 1,75 hours for 100,000 entities. These times can significantly improve with more efficient programs and faster computers. Research on efficient graph computations is continuously progressing. Consider also that many IoT networks can be partitioned in partially independent *slices* as they are called in 5G (Zhang [51]), or *domains* (Flauzac et al. [16][17][18]). In practice, many slices or domains can be smaller than the mentioned 10,000 entities and for many transformations it might be sufficient to reconfigure only one of them. Finally and most important, in many practical cases, policies and configurations are set up in such a way that global recalculations are unnecessary since only limited and already planned local changes will occur, with minimal overheads. Due to the many different contexts in which our method can be used, more detailed efficiency considerations, as well as the adaptation of the method to each context, are left to future research.

In our networks, if data are sent autonomously by the entities, an entity can receive a data item several times from different sources. This is normal in multicasting and standard mechanisms exist to address it.

11 Conclusion

We have shown the feasibility of using SDN in IoT contexts for implementing data security requirements of data secrecy (or confidentiality), integrity and conflicts. In the implementation method we propose, data will be forwarded only to entities meant to receive them, and it can be ensured that this property remains true through network transformations.

In previous publications [45][31][32], we have shown how Denning's and Sandhu's lattice model for data security can be generalized to a multi-level partial order model that can be found in any data network that can be represented as a directed graph. The data flows are determined by the labels of the entities, which can be given (by users or administrators, as in our example), or can be efficiently calculated from capability lists or access control lists. The data flows are from entities of low secrecy and high integrity towards entities of high secrecy and low integrity.

In this paper, we have shown how the labels can be used to construct forwarding tables for SDN routers that will control data transfers accordingly, thus ensuring data security. We have used a network organization based on cloud concepts with application entities and data entities (or servers, databases). We have also described methods to take care of the transformations of the networks, as required by administrative action or policies. We have noted in Sect. 7.2 that our networks are tolerant to entity failure. We have demonstrated our method in detail by using a simple 'hospital' example, which was simulated by using the Mininet and Ryu platforms. We have shown that the method can be used also in the case of multiple coexisting data flows. Efficiency and scalability issues were mentioned. Our solutions for transformations and multiple flows show the flexibility of our approach, as well as of the SDN architecture. Our method is centralised, in the sense that there is only one controller. Decentralization is left to future research.

As documented in Section 3, although the literature on security with SDN is plentiful, no other methods have been proposed to use SDN for data security in the way we have described, supported by partial order theory and by efficient algorithms.

Of course, we have provided a generic method only, we have shown how the general data flows could be maintained. In order for the proposed method to become practical, it will require the creation of a suitable administrative model. As

well, methods to construct complex label systems, appropriate for security applications involving many label types, will have to be developed. IoT networks are very complex and their design has to take into consideration many different requirements, not only data security-related ones. Our method intends to provide one element of solution and will have to be combined with other methods; this will be the subject of future research. Among others, although our method does not require encryption in principle, encryption will be needed, at least to take care of errors in the execution of the routing mechanisms, possibly caused by attacks on the routing tables or the by the insertion of hidden channels.

Further details on this work can be found in [44].

12 Acknowledgment

We thank Dr. Ahmed Karmouch for introducing us to the possibilities of SDN for network security and Yvon Andrianirina for technical information on SDN tools. This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

13 Compliance with ethical standards

This research was funded exclusively by Discovery Grants of the Natural Sciences and Engineering Research Council of Canada (NSERC) awarded for long-term research to co-author Luigi Logrippo. Authors Abdelouadoud Stambouli and Luigi Logrippo declare that they have no conflicts of interest. Ethical approval: This article does not contain any studies with human participants or animals performed by any of the authors.

References

1. https://www.lifewire.com/how-many-devices-can-share-a-wifi-network-818298. (Consulted on April 28, 2021).
2. J. H. Abawajy, M. M. Hassan. Federated Internet of Things and Cloud Computing Pervasive Patient Health Monitoring System. IEEE Comm. Magazine, 55(1), 48-53, Jan. 2017.
3. C. Aggarwal and K. Srivastava. Securing IOT devices using SDN and edge compu-ting. 2nd International Conference on Next Generation Computing Technologies (NGCT 2016), 877-882.
4. M. Ahmad Khan. A survey of security issues for cloud computing. Journal of Net-work and Computer Applications.71 (2016), 11-29.
5. F.A. Alaba, M. Othman, I.A.T. Hashem, F. Alotaibi. Internet of things security: A survey. Journ. Network and Computer Applications, 88 (2017), 10-28.
6. A. Al-Haj, B. Aziz. Enforcing Multilevel Security Policies in Database-Defined Networks using Row-Level Security. International Conference on Networked Systems (NetSys 2019), 1-6.
7. D.E. Bell, L. La Padula. Secure computer system : unified exposition and Multics interpretation. Mitre Corp. Report MTR-2997 Rev. 1, March 1976.
8. M. Bishop. Computer security, Art and science. 2nd ed., Addison-Wesley, 2019.
9. S. Chakrabarty, D.W. Engels, S. Thathapudi. Black SDN for the Internet of Things. 2015 IEEE 12th Intern. Conf. on Mobile Ad Hoc and Sensor Systems.
10. S.Christos, P. Kostas, K. B.Gyu, B. Gupta. Secure Integration of Internet-of-Things and Cloud Computing. Future Generation Computer Systems , 2013.
11. D. Denning. A lattice model of secure information flow. Commun. ACM 1(5) (1976), 236-243.
12. M. Dias de Assunção, R. Carpa, L. Lefèvre, et al. Designing and building SDN testbeds for energy-aware traffic engineering services. Photon Netw Commun 34 (2017).396–410 .
13. R. Djouani, K. Djouani, F. Boutekkouk, R. Sahbi .A Security Proposal for IoT inte-grated with SDN and Cloud. 6th International Conference on Wireless Networks and Mobile Communications (WINCOM 2018),1-5.
14. L. El-Garoui, S. Pierre, S. Chamberland. A New SDN-Based Routing Protocol for Improving Delay in Smart City Environments. Smart Cities. 3(3) (2020), 1004-1021.
15. S. Etalle, T.L. Hinrichs, A.J. Lee, D. Trivellato, N. Zannone. Policy Administration in Tag-Based Authorization. In: Proc. 9th Intern, Symp. On Foundations and Practice of Security. FPS 2012. Springer LNCS, vol 7743.
16. O. Flauzac, C. Gonzalez, A. Hachani, F. Nolot .SDN Based Architecture for loT and Improvement of the Security. IEEE 29th Int'l. Conf. Advanced Information Networking and Applications Workshops (WAINA) (2015), 688-693.
17. O. Flauzac, C. Gonzalez, F. Nolot. New Security Architecture for IoT Network. Pro-cedia Computer Science. 52 (2015), 1028-1033.
18. C.Gonzalez, S.Charfadine, O.Flauzac, F.Nolot. SDN-based security framework for the IoT in distributed grid. International Multidisciplinary Conference on Computer and Energy Science (SpliTech 2016).
19. A. Hakiri, P. Berthou, A. Gokhale, S. Abdellatif. Publish/subscribe-enabled soft-ware defined networking for efficient and scalable IoT communications. IEEE Communications Mag 53 (9) (2015), 48-54.
20. Z. Han, X. Li, K. Huang, Z. Feng. A Software Defined Network-Based Security As-sessment Framework for Cloud IoT. IEEE Internet of Things Journal, 5 (3) (2018),1424-1434.

21. A.Hany, W. Gary.Intersections between IoT and distributed ledger. Advances in Computers. Role of Blockchain Technology in IoT Applications (3) (2019).
22. J. Hou, L. Qu, W. Shi. A survey on internet of things security from data perspec-tives. Computer Networks 148 (2019), 295-306.
23. V.C. Hu, D.F. Ferraiolo, R. Chandramouli, D.R. Kuhn. Attribute-Based Access Con-trol. Artech House, 2018.
24. D. Huang, A. Chowdhary, S. Pisharody. Software-Defined networking and securi-ty. From theory to practice. CRC Press, 2019.
25. M. Islam, M. Refat. Node to Node Performance Evaluation through RYU SDN Con-troller. Wireless Pers Commun 112 (2020), 555–570.
26. K. Kalkan, S. Zeadally. Securing Internet of things with software defined network-ing. IEEE Comm. Magazine, Sept. 2018, 186-192.
27. K. Karmakar, V. Varadharajan, S. Nepal, U. Tupakula. SDN Enabled Secure IoT Ar-chitecture. IFIP/IEEE Symposium on Integrated Network and Service Manage-ment (IM), Arlington, VA, USA (2019), 581-585.
28. R. Kubo, T. Fujita, Y. Agawa, H. Suzuki. Ryu SDN Framework— Open-source SDN Platform Software. NTT Technical Review,12 (8) (2014) .
29. C. E. Landwehr. Privacy research directions. Comm. ACM 59(2) (2016) 29-31.
30. Y. Liu, Y. Kuang, Y. Xiao and G. Xu, SDN-Based Data Transfer Security for Internet of Things. In IEEE Internet of Things Journal, 5(1), 257-268 (2018).
31. L. Logrippo. Multi-level models for data security in networks and in the Internet of things. Journal of Information Security and Applications 58 (2021).
32. L. Logrippo, A.Stambouli. Configuring data flows in the Internet of Things for se-curity and privacy requirements. Presented at the 11th International Symposium on Foundations and Practice of Security. Montreal, 2018. Springer LNCS 11358,115-130.
33. A. Mamdouh, K. Almustafa, K. Amjad Meerja. Cloud based SDN and NFV architec-tures for IoT infrastructure. Egyptian Informatics Journal 20 (2019), 1-10.
34. S. Ola, E. Imad, K. Ayman, C.Ali. SDN controllers: A comparative study. 18th Medi-terranean Electrotechnical Conference (MELECON 2016), 1-6.
35. K. Prabhakar, N. Jisha, A. Krishnashree. SDN Framework for Securing IoT Net-works. Ubiquitous Communications and Network Computing (2017), 116-129.
36. C. Qiang, G. Quan, B.Yu, L. Yang. Research on security issues of the Internet of Things. International Journal of Future Communication and Networking, 6 (6) (2013), 1-10.
37. Z. Qin, G. Denker, C. Giannelli, P. Bellavista and N. Venkatasubramanian. A Soft-ware Defined Networking architecture for the Internet-of-Things. IEEE Network Operations and Management Symposium (NOMS 2014).
38. B. Quinn, F. Mehmeti, R. George; K. Ostrowski, T. Jaeger, T. La Porta, P. McDaniel. Enforcing Multilevel Security Policies in Unstable Networks. IEEE Transactions on Network and Service Management (2022).
39. W.Roy, S. Bill, J. Scott. Enabling the Internet of Things. Computer (48) (2014), 28-35.
40. A. Saleh, G. Bhargavi, S. Mohammed. Ryu controller's scalability experiment on software defined networks, IEEE International Conference on Current Trends in Advanced Computing (ICCTAC 2018),1-5.
41. R.S. Sandhu. Lattice-based access control models. IEEE Computer 26(11), 1993, 9–19.
42. J. Singh, T. Pasquier, and J. Bacon. Securing Tags to Control Information Flows within the Internet of Things. in International Conference on Recent Advances in Internet of Things (RIoT'15), 2015.
43. B. Smriti, R. S. Sandhu. ABAC-CC: Attribute-Based Access Control and Communi-cation Control for Internet of Things. ACM Symposium on Access Control Models and Technologies (SACMAT) (2020), 203-212.
44. A. Stambouli. Data Security In Organizational Networks And The Internet of Things, Using a Partial Order Model. PhD thesis, Université du Québec en Outaouais, fall 2021.
45. A. Stambouli, L.Logrippo. Data flow analysis from capability lists, with applica-tion to RBAC. Information Processing Letters (Elsevier) 141 (2019) 30–40.
46. H. Suo, J. Wan, C. Zuo, J. Liu. Security in the Internet of things: a review. 2012 In-tern. Conf. on Computer Sc. and Electr. Engg., IEEE, 648-51.
47. Wang, X. Mei, J. Croft, M. Caesar, B. Godfrey. Ravel: A database-defined network, Proceedings of the Symposium on SDN Research (2016), 1-7.
48. D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, J. A. Mccann.UbiFlow: Mobility manage-ment in urban-scale software defined IoT. IEEE Conference on Computer Com-munications (INFOCOM 2015).
49. R. Xie, H. Li, G.Shi ,Y.Guo B.Niu, M.Su .Provenance-based data flow control mecha-nism for Internet of things. Transactions on Emerging Telecommunications Technologies (2020).
50. M. Yassein, S. Aljawarneh, M. Al-Rousan, W. Mardini ,W. Al-Rashdan, Combined software-defined network (SDN) and Internet of Things (IoT), International Conference on Electrical and Computing Technologies and Applications (ICECTA 2017), 1-6.
51. S. Zhang. An Overview of Network Slicing for 5G.IEEE Wireless Communications, 26(3), 111-117, 2019.