Preprints.org

Article

Not peer-reviewed version

Implementation of a Partial Order Data Security Model for the Internet of Things (IoT) Using Software Defined Networking (SDN)

Abdelouadoud Stambouli * and Luigi Logrippo

Posted Date: 12 January 2024

doi: 10.20944/preprints202401.0970.v1

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

Preprints.org is a free multidiscipline platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

Implementation of a Partial Order Data Security Model for the Internet of Things (IoT) Using Software Defined Networking (SDN)

Abdelouadoud Stambouli [1,*] and Luigi Logrippo [1,2,*]

[1] Université du Québec en Outaouais; AS: staa16@uqo.ca; LL : Luigi@uqo.ca
[2] University of Ottawa.
* Correspondence: AS: staa16@uqo.ca; LL: Luigi@uqo.ca

Abstract: Data security in the Internet of things (IoT) is often implemented by means of encryption, which can be burdensome for some entities. We propose in this paper a solution based on routing, by which data are forwarded only to entities that are intended to receive them. An IoT network can be seen as a partial order of equivalence classes of entities, and each entity can be labeled according to the position of its equivalence class in the partial order. The partial order can be constructed according to requirements of secrecy (or confidentiality), integrity and conflicts. Routing tables among entities can be compiled by using the labels. The method is demonstrated in this paper for Software defined networking (SDN) routers and controllers. 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. A small 'hospital' example is shown for illustration. Procedures for network reconfigurations are discussed. We also demonstrate the method for the normal case where different partial orders coexist among a set of entities. The method proposed does not impose an overhead on the normal functioning of an SDN network, since it requires calculations only when the network must be reconfigured, because of administrative intervention or policies.

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

---

1. Introduction and motivation

In this paper, we see the Internet of things (IoT) as an evolving set of entities that contain data and can modify their data contents by reading or writing other entities.

Data security, information security and data privacy are major concerns in the IoT, see Alaba et al. [3], Suo et al. [31], Qiang et al. [24]. The importance of data flow security for data privacy was discussed by Landwehr in [20]. This paper presents a method for directing data flows in the IoT, by using Software defined networking (SDN) in such a way that common data security requirements are satisfied. Data security aspects that are considered here are *secrecy* (also called *confidentiality*), *integrity* and *conflicts*, as defined in Sect. 2.

Many approaches for data security in the IoT are based on data encryption, where the responsibility is for the entities to encrypt-decrypt data so that only certain entities have access to them; however, encryption, decryption and related operations can be burdensome for some devices, such as small sensors. Other approaches are based on data labeling, combined with policies stating that only entities labeled in certain ways can read or write entities labeled in certain other ways (see Sect. 10). Data labeling is a general-purpose data flow control method, which can be used to specify constraints on reading and writing among entities. In this paper, we show that SDN routing tables can be compiled for arbitrary networks by using labels constructed according to an efficient method.

© 2024 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

This approach is not exclusive with respect to encryption, but the combined use of the two methods is left to future research.

In Sect. 2, we give a short account of previous research on data flow control for security. In Sect. 3, we introduce Software defined networking. Sect. 4 describes our method in principle. Sect. 5 presents a concrete 'hospital' example. Sect. 6 discusses the topic of network reconfigurations. Sect. 7 shows how networks with multiple flows can be implemented in our method. Sect. 8 presents how our method was simulated and tested. Sect. 9 deals with efficiency and scalability. Sect. 10 provides a literature review. Sect. 11 concludes the paper.

2. Preliminaries

In classical papers by Denning [10] and Sandhu [28], which where preceded by the work of Bell-La Padula [6] and followed by many others concurring with their basic ideas, a concept of *secure information flow* was introduced, by which data is expected to flow according to the order relations in a lattice of labeled entities. We adapt the basic definitions from this literature as follows. For entities *x* and *y* we say that there is a *Channel* from *x* to *y* if entity *x* has permission to write data on *y* or entity *y* has permission to read data from *x*, and graphically we represent this as an arrow from *x* to *y*. Terms such as *receiving* or *pulling* can also be used instead of *reading* and *sending* or *pushing* instead of *writing*.

The *CanFlow* or *CF* relation is defined as the transitive, reflexive closure of the *Channel* relation, and graphically corresponds to directed paths between entities. Any network of entities that is represented as a directed graph in this manner is a *preorder* that can be reduced to a *partial order* of equivalence classes of entities, where two entities *x* and *y* belong to the same equivalence class iff *CF(x,y)* and *CF(y,x)* [21]. In other words, a set of entities is in the same equivalence class if the data that is in any one of them can reach any other through a directed path; further, the set of the resulting equivalence classes forms a partial order.

We define the *label* of *x*, *Lab(x)* as the set of the names of all entities that can flow to *x*. So, if *x*, *y*, *z* are the names of all entities that can flow to *w* beside *w* itself, then $Lab(w)=\{x,y,z,w\}$. Entities in the same equivalence class have the same label.
It follows that *CF(x,y) iff* $Lab(x) \subseteq Lab(y)$, and so labels, with the inclusion relation between them, define the flow relation between entities in the partial order.

Following the usual terminology in data security theory, we call entity names *categories*, and we note that each category also defines a data *provenance*. Thus, the label of an entity represents the data categories that the entity *can know*. It is important to note that, given a set of reading and writing permissions, as it could be represented in an access control matrix or in a graph, the labels can be computed efficiently by using well-known graph component algorithms [30].

Following Sandhu [28], we say that the top entities in a partial order, not having any outgoing data flows, have *top secrecy*; while the bottom entities, not having any incoming data flows, have *top integrity*. Top secrecy corresponds to minimum integrity, and vice-versa. By extension, entities can be considered to have different levels of secrecy or integrity according to their position in the partial order. In addition, situations of conflicts (such as the ones represented by *Chinese Walls*) can be represented by excluding certain label combinations: e.g. if no entity is supposed to know both data of category *x* and category *y*, then labels containing both *x* and *y* are forbidden (again, this is consistent with Sandhu [28]).Papers [21-22-30] have shown that these definitions can be used not only for lattice networks, but also for any networks that can be specified by means of access control matrices or permission lists, including the widely implemented Role-based access control (RBAC).

Thus, a main difference between the theory presented in Denning [10] and Sandhu [28] and the theory used here can be expressed as follows: while the former states that secure data flows must be built by forming lattices of labeled entities, the latter theory shows that for any preorder of entities, representing an arbitrary data flow in a network at a given time, the equivalence classes of the entities form a partial order, and the entities in the partial order can be considered to have different levels of secrecy or integrity according to the position of their equivalence classes in the partial order. In the

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

former theory, a lattice structure is a precondition for secure data flows, while, in the theory we have described, any network has its own security properties, for secrecy, integrity and conflicts.

3. Software defined networking (SDN)

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, e.g. Huang et al. [17] 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 controls it through standards such as OpenFlow, 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 router.

To justify our choice of the SDN architecture, we start from the observation that global security solutions are more efficient and effective 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. The use of SDN limits our approach to centrally controlled IoT systems, but the resulting efficiency makes it valuable in such contexts.

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.

4. Our implementation method

4.1. Network configurations and graphic representation

Following on the principles presented in Sect. 2, the main idea of this paper is that the labels of entities, which can be computed automatically and efficiently from the *Channel* relation [30-22], can be used for routing data in SDN networks, from the entities where data originate, their provenance, to all the entities where they can flow. In addition, by using SDN configured according to the partial order security model, it is possible to constrain the flow of data in IoT systems in order to satisfy data security requirements. On this basis, we formulate an SDN architecture where SDN forwarding tables are set by the SDN controllers using the entities' labels. It follows from the theory presented in Section 2 that this method is general, in the sense that it can be used for any network for which the data flow relation can be represented as a directed graph of entities. Our method considers a centralized IoT architecture where all data are transferred and stored in cloud platforms and accessed by user applications.

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. [9], Hany et al. [15], and Roy et al. [26]. Further, our efficient centralized algorithms can reconfigure networks dynamically as necessary, see Section 6. 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 efficiency.

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

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

devices must be achieved through these servers. This IoT configuration 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.

The *Networking layer* is used to connect IoT objects to the Internet, it also contains the servers used to store the data collected from the *Sensing layer*. Several communications technologies and protocols can be used in this layer such as 3G/4G/5G, Zigbee, Bluetooth, WiFi 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.

We adapt the centralized IoT architecture to the SDN architecture, see Figure 1. 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 *Networking layer*. The second router is the *Application router* to which the cloud router connects, and which interconnects the entities in the *Application layer*. We also have an Access point that connects the *Sensing layer* with the *Networking layer*, but we do not program this one, since it is essentially charged with forwarding the data to the cloud router. These are logical devices that can be implemented by several physical devices of different types.

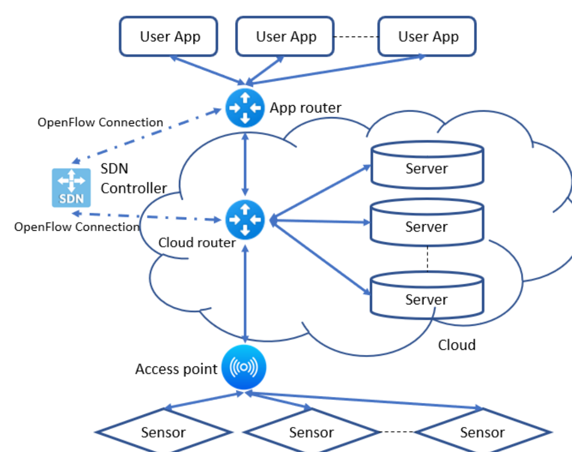

Figure 1. Our SDN implementation configuration.

Many papers in the literature mention a single controller for Wide-area SDN. In ElGaroui et al. [12] and Dias et al. [11], the 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, not discussed in this paper.

4.2. Labeling tables, forwarding tables and data flow control policy enforcement

Our method starts with a network representing an application layer configuration of directly connected application entities, defining a *Channel* relation. Following the principles presented in Sect. 2. the equivalence classes of entities in the network are identified and the resulting reduced graph is a partial order of equivalence classes. Labels are then assigned to the equivalence classes, which become the labels of the entities in each class. These labels can be simply obtained by set union proceeding from source to sink. For example, if the label of a source equivalence class contains the

name *BobPulse*, then all equivalence classes that dominate it will also contain this name in their labels. As mentioned, the necessary calculations are automatic, starting from the *Channel* relation [21].

The network obtained in this way will not be in the form of the *centralized* cloud-based configuration 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, 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 names in labels can be replaced by entity names. For example, if a name such as *BobPulse* denotes data originating from an entity (a sensor) named *A*, then each occurrence of *BobPulse* in labels can be replaced by the name *A*. The label inclusion relation remains the same. 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 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, for each *B*, the set of *A*s 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 includes the entities that are connected to it.

We assume that we deal with routers with arbitrary large capacities. Average routers in use today can have up to 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 need 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 this is an authorized flow. When a packet arrives to the router, the latter will compare the *IP src* and *IP 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 the next router in the configuration. This will prevent overloading routers and will eliminate unnecessary delays. In this way, the specified partial order of equivalence classes will be implemented.

5. Example

5.1. The basic configuration and its implementation

As an example, we consider a very small health system. In generic terms, its configuration is as follows: 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: *Pressure* 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. Thus, they must be at the bottom of the partial order [21], and this is where they will end up given that their labels contain only one data category.
- 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

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

patients and *Wards*. It must be at the top of the partial order [21], and this is where it will end up given that its label contains all data categories.

- The *Wards* and *Reanimation* departments 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*. These conflicts are represented by forbidding labels containing conflicting data categories, such as, in our example, *SamPress* in *Ward2.*

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 2. In the figure, a rectangle represents a *sensor* (three in 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 the label (a set of data categories) in braces. For readability, labels have been simplified with respect to the theory presented in Sect. 2, but still show the data categories allowed in each entity. As in Sect, 2, arrows represent directional channels for receiving (reading, pulling) and sending (writing, pushing), so for example in Figure 2 entity *C* can send data to *K*, or equivalently *K* can receive data from *C*. There is an arrow, or a path of arrows (a data flow), between an entity and another iff the label of the first entity is included in the one of the second.

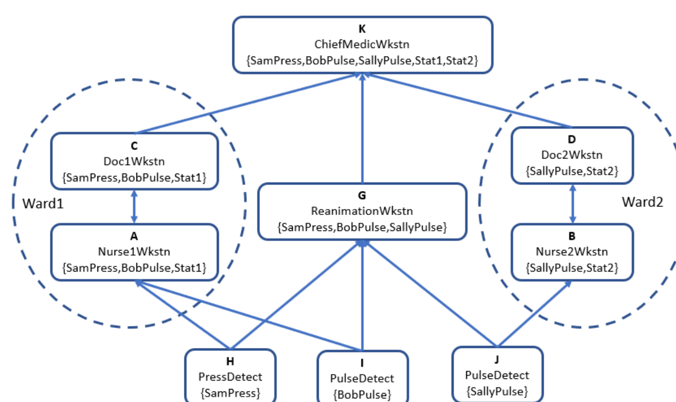

Figure 2. Hospital example.

According to IoT terminology, the three sensors are the *Sensing layer* and the rest is the *Application layer*. Figure 2 shows a 'direct' Channel configuration without the Cloud or the Networking layer. It can be checked that the security policies above are implemented by the choice of label sets, which was done by the security administrator at the time the network was configured, explicitly or implicitly by defining the *Channel* relation. For example, the blood pressure of *Sam* can only be known in *Ward1*, in the *Reanimation* or *Chief of medicine* departments.

The partial order of this architecture is shown in Figure 3. 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. This network serves as a reference point for representing the sequential and hierarchical relationships in our design. In Figure 4, we present the labeling tables for the App router in this configuration. It can be checked that such routers implement all and only the flows in Figure 2. The simple SDN implementation configuration of this example is shown in Figure 5.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

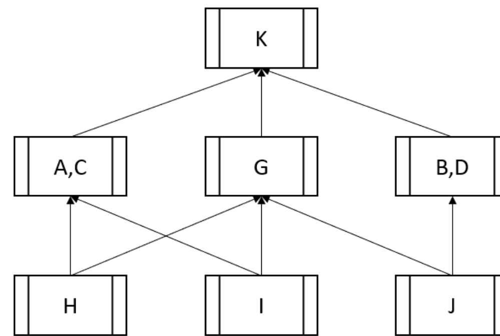

Figure 3. Partial order for figure 2.

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

Figure 4. Labeling tables for the configuration of Figure 2.

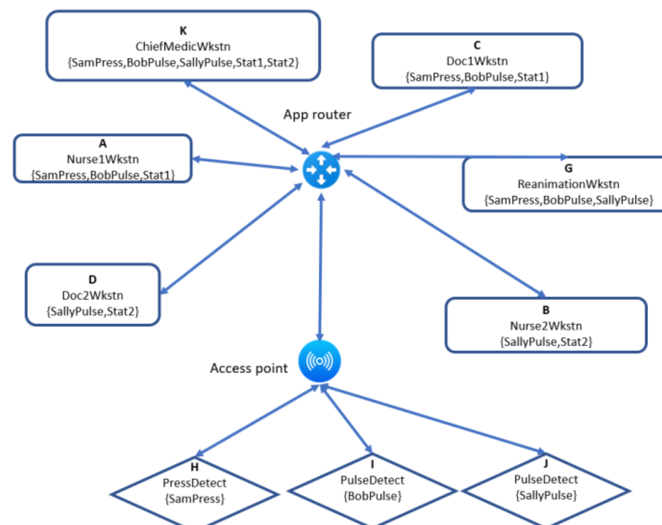

Figure 5. A simple implementation configuration of Figure 2.

5.2. Introducing the Networking Layer

To the configuration of Figure 2, we now add the Cloud and the *Networking layer*, to fully implement the configuration of Figure 1. As mentioned, the Networking layer is provided by storage entities, see Figure 6. The partial order of equivalence classes implemented by this configuration is shown in Figure 7.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

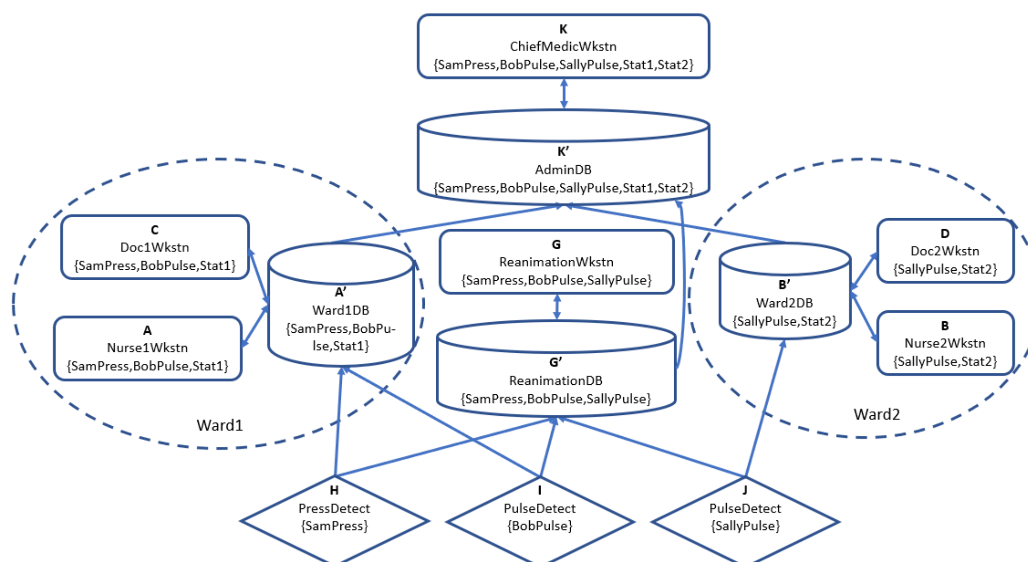

Figure 6. Cloud configuration for Figure 2.

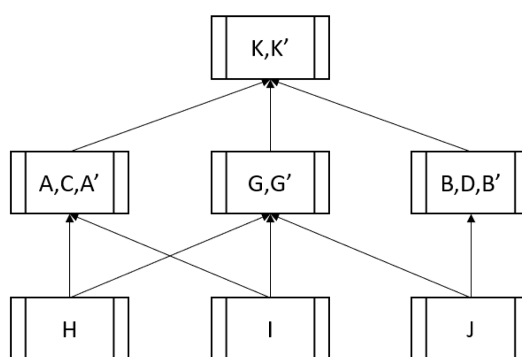

Figure 7. Partial order for the centralized architecture.

Now, flows between application entities must pass through the Networking layer, and so storage entities (such as databases, servers, etc.) have been added to the Cloud, hence at least an equivalent storage entity (i.e., with the same label) is associated to each equivalence class of application entities. Storage entities 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. In this configuration, we have deleted any entity-to-entity channels that are not transiting by a storage entity. Note that the required data flows between application entities (and no additional ones) are still obtained by transitivity.

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 8.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

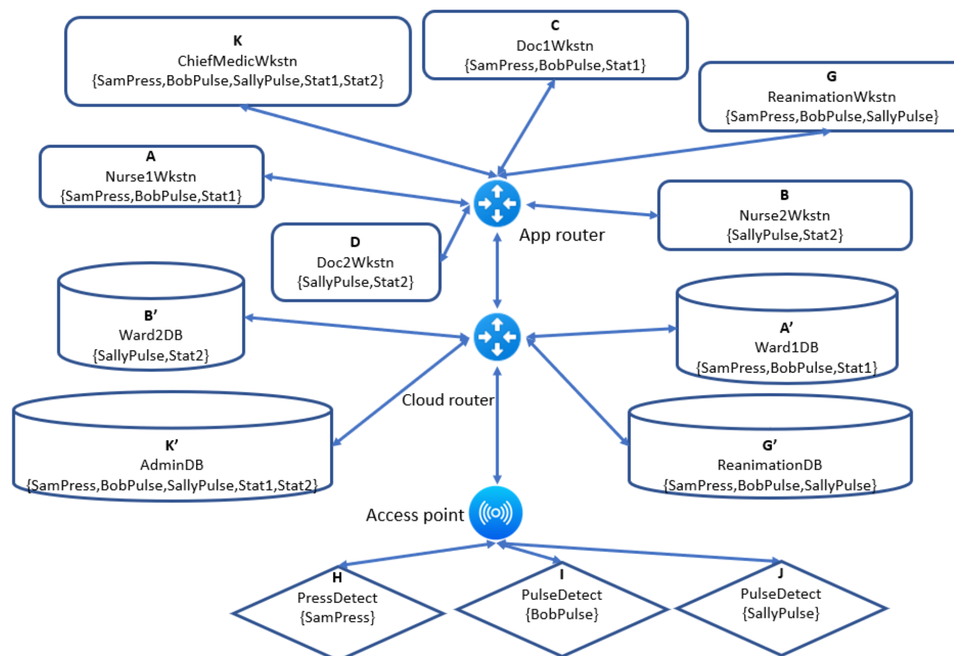

Figure 8. An implementation configuration for Figure 3.

As required, 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 Figure 6. No direct communication between application entities occurs, all data passes through the *Cloud Router*. However, communication between storage entities is allowed 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. In all labelling tables, 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 *Cloud 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. Note for example that *Label(J)={SallyPulse}* and *Label(B)=Label(B′)=Lab(D)={SallyPulse,Stat2}*. So by label inclusion, each of *{J,B,D,B′}* can flow to *B′*. These are all 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 9. The table in Figure 6 can be easily constructed from the partial order of Figure 7.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

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	I, J, H, 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

Figure 9. Labeling table for Figure 5.

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*. Algorithms exist to perform the transformation from Figure 6 to Figure 7, as well as for the calculation of the routing tables.

At first sight, the final configuration of Figure 7 seems to have no relation with the partial order of equivalence classes we started from (Figure 2), 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, although not along the same paths. This means that the initially given policies of secrecy and integrity, as well as conflicts, are properly implemented. For example:

1. In Figure 2 we have a flow *H→A→C→K*. By looking at Figures 6, 7 and 8, we see that the data of *H* can go to *A*, *C*, and *K* through the *Cloud Router* and server *A′*.
2. Similarly, in Figure 2 we have the flow *J→G→K*. Data can go from *J* to *G* through *G′* and from *J* to *K* through *B′* and *G′*.
3. On the other hand, unwanted flows are clearly impossible. The reader can check easily that there is no way for data in *H* to end up in *D*, or data in *C* to end up in *D*. Conflict requirements are satisfied.

The second example shows that our solution uses multicasting, by which the same data can arrive at destination through multiple paths. Multicasting is a well-known technique in networks and methods exist to eliminate duplicate packets received from several sources. It makes our solution fault-tolerant to some extent, e.g., in the second example the failure of one of *B′* or *G′* will not affect the path *J→K*.

Clearly, this example can be scaled up by introducing many more sensors, many more wards, many more workstations, etc. To make this possible, the entities would have to be parameterized or indexed, such as *PulseDetect1*, *PulseDetect2*, etc. Evidently, real-life networks will not be able to be shown in the simple graphic format used here. Graphic interfaces, showing high-level representations that can be manipulated by administrators, should be the subject of future research.

6. Network reconfigurations

In the IoT, the network topology may be continuously transformed or reconfigured with entities and communications channels being created or removed. This can occur for many reasons, notably

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

by intervention of a system administrator, or by effect of policies. For example, in some systems there are reconfigurations 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. The routing tables must be updated at each such event, but the implementation configuration will remain the one of Figure 1 or 8. It is a useful property of the partial order model that a partial order exists for any network, and so our method can be used to construct new routing tables after each network reconfiguration (note that, on the contrary, if a lattice-structured network is reconfigured, the result may not be lattice-structured). Reconfigurations may affect only some routing tables.

In our partial order model, the reconfigurations that matter are the changes in the domination relation, because these are the only ones that can change the data flows. Reconfigurations that do not change this relation are adding or removing channels that are implied by transitivity. Consequently, we consider three types of reconfigurations: introduction or removal of entities, or label changes. The label of an entity can be specified in one of two ways:

- Explicitly: in this case, the label indicates the intended contents of the entity and, by inference, its *Channel* and *CF* relations
- Implicitly: in this case, the *Channel* or *CF* relations of the entity are given and, by inference, its intended contents and label.

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 certain 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 decisions should be left to the designers of specific systems. Standard data structure theory proposes methods that can be used for optimizing the reconfiguration methods, and we leave this to further research.

- *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, see above.
    - 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: 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 the 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 the *App Router*. The new entity will access the same data entities as the other entities of its 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 a corresponding storage entity for this new entity, with the same label. For the implementation configuration, the new entity must be connected to the *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.

- *Entity removal and entity failure:* This will change the implementation configuration since the removed entity will not be included in the new implementation configuration. It should be kept in mind that removal of an entity does not make it 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 will 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 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.

- *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 has no effect on the implementation configuration: the entity remains in its place. The labeling tables will have to consistent with the new labels.

It should be stressed that these are not manual operations, and we have described, in general terms, the algorithms to perform them.

7. Networks with multiple flows

In the example of Sect. 5, we have only considered the existence of a single data flow in the network. Usually however, several separate data flows are present in networks. 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. 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

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

requirements are reversed, and labels containing combinations of patients' diagnostic data are allowed only for certain equivalence classes of entities.

We say that the example of Sect. 5 deals with *Consultation* data that flow from patients towards the medical staff as we have seen. We add to this *Diagnostic* data that travel in the opposite direction and have their own requirements in term of secrecy, which lead to a different partial order. The network with the representation of the two different flows is shown in Figure 10. 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 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(SamPress,BobPulse,SallyPulse,Stat1,Stat2),Diagnostic(SamDiagnos,SallyDiagnos,BobDiagnos)*. This means that *ChiefMedicWkstn* participates in the two flows, and that for each flow, *ChiefMedicWkstn* has access to data of the corresponding labels.

This example shows an application 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* and *Bob* data and sends data to both but is expected not to send *Sam* 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* [6], where trusted subjects are described as “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.

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.

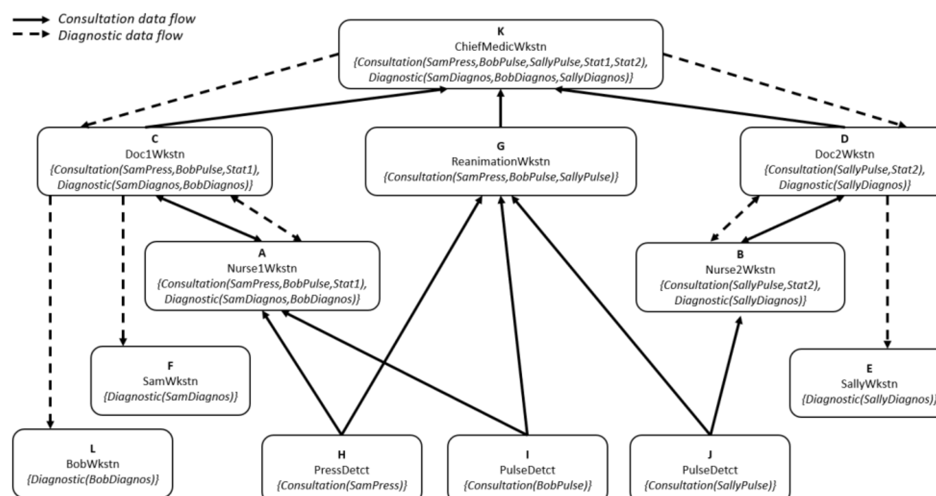

Figure 10. Two-flow network for the hospital example.

Figure 11 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.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

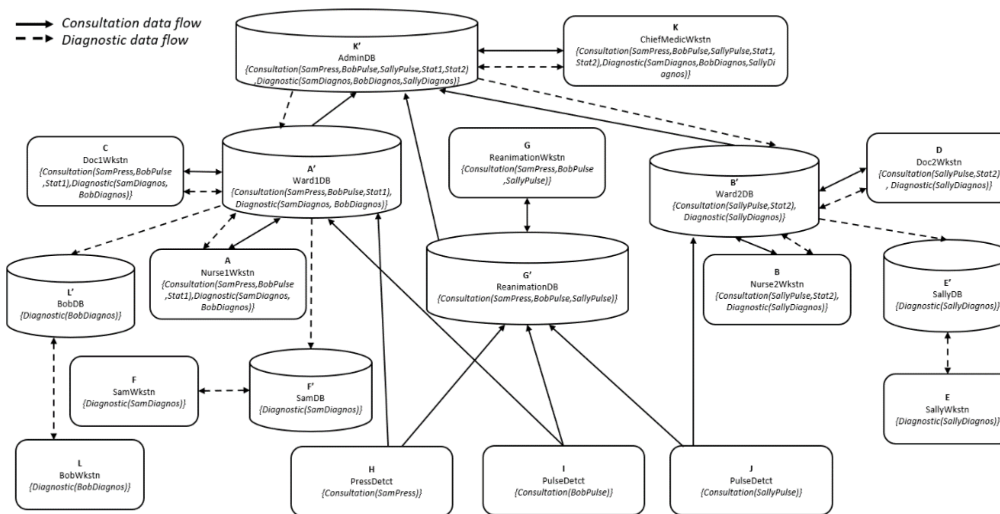

Figure 11. Centralized two-flow network.

The main difference with respect to the one-flow example is that the controller will have two forwarding tables, one 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.

The new implementation configuration is as described earlier, we have two routers that connect the network entities: one to connect the storage entities and one to connect the workstations.

Each router will have a forwarding table for each flow, each incoming packet will contain its flow identifier together with its label, and the controller will use the flow identifier to switch to the appropriate forwarding table for each packet.

The partial order for the new *Diagnostic* flow is shown in Figure 12 and the labeling tables in Figure 13.

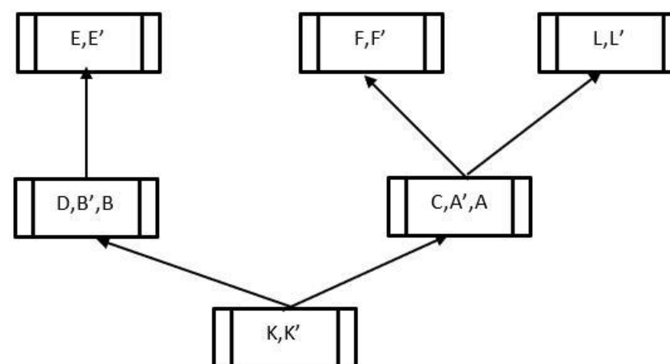

Figure 12. Partial order for the Diagnostic Flow.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

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'

Figure 13. Labeling table for Figure 8.

8. Simulation and implementation of the controller

The SDN implementation of our hospital example 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 behave 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.

For the choice of controller, several reasons led us to use the Ryu controller of Asadollahi et al. [5-27]. First, we considered the comparison study documented by Ola et al. [23]. 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 Python-based, it is easier in Ryu to develop new network management and control applications in comparison with other controllers. And finally, Asadollahi et al. [5] and Islam and Refat [18] 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. The case of multiple flows was also tested. Parameters such as performance of the controller, scalability, etc. have been tested in other SDN-related work already mentioned, see Asadollahi et al. [5], Islam, Refat [18].

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

After implementation, the data flow in the network was simulated. We use the *ping* command to see if the results match the security requirements expressed by our partial order. Figure 14 and Figure 15 show examples of the results obtained.

In Figure 11, we can see some results for a defined scenario. We have a connection established between entities in the case of authorized data flow *J* to *B*, where *J* represents *PulseDetect(Sally)* and *B* represents *Nurse2Wkstn*.

```
mininet> J ping B
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=17.1 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=10.9 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=5.81 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=3.22 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=7.20 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=4.15 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=7.28 ms
^C
--- 10.0.0.2 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6008ms
```

Figure 14. Simulation result of an authorized flow.

Another scenario, presented in Figure 12, shows an unauthorized flow where there is no data flow from *J* to *A*, where *J* represents *PulseDetect (Sally)* and *A* represents *Nurse1wkstn*.

```
mininet> J ping A
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
11 packets transmitted, 0 received, 100% packet loss, time 10223ms
```

Figure 15. Simulation result of an unauthorized flow.

The multiflow solution was also tested according to this method.

9. 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 as any SDN network.

As we have seen, labels can be assigned by administrators or may be calculated from *Channel* or *CF* relations, which may mean from capability lists or access control matrices. In [30] Stambouli and Logrippo presented a method for finding the labels based on such information. They showed 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 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, or *domains*. In practice, many slices or domains can be smaller than the mentioned 10,000 entities and reconfigurations may affect only some of them.

If, on the other hand, a *CF* relation must be calculated from labels, this also can be done efficiently by set membership tests, using the relation *CF(x,y) iff Lab(x)⊆Lab(y)* [22]. Finally and most important,

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

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.

10. Related work

We have drawn inspiration from the work by Etalle et al. [13], 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. [29], 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. We have seen that, following Sandhu [28] only one simple label is required in our method to express both secrecy and integrity. In a recent paper, Burke et al. [7] propose MLS-Enforcer, a Software-defined networking (SDN) controller that enforces multi-level 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.

Some papers propose the use of different types of access control and data flow control policy models in the IoT, for example Rong-na et al. [25] 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 expresses provenance to the extent that our labels can express provenance.

Al-Haj and Aziz [4] present 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 [10-28] alongside with the RAVEL architecture (Wang et al. [32]). 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.

Fernandes et al. [14], and Celik et al.[8] use data flow tracking within the system to enforce fine-grained security policies using a combination of dynamic and static analysis. We take another approach which is labeling data. This can be simpler, compatible with existing infrastructure, efficient, but can have limited granularity. Assigning labels to data packets is a straightforward process that does not require significant computational resources or expertise. This approach can be compatible with existing IoT systems, since many IoT systems already use some form of data labeling, such as metadata or tags, to identify and manage data. It can also be more efficient than tracking the flow of data in systems with limited resources. Labeling data can have limited granularity, and its appropriateness depends on the context: for example, labeling an entire data packet with a single label can be an efficient approach if the packet contains multiple data items of similar sensitivity.

Haotian et al. [16], focuses on smart home environments and their approach may be less applicable to other IoT domains. Our approach can be applied to various IoT domains beyond smart homes and proposes a solution for securely sharing data between different IoT domains. Our

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

approach also provides comprehensive analysis of the requirements for IoT data flow configuration and presents a detailed architecture for implementing it using SDN.

The vast majority of papers dealing with SDN security 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. [2–19]. Several of the proposed security solutions use cryptographic algorithms that may be difficult to implement in sensors. As mentioned, our approach does not require cryptography, although cryptography can be used to increase security. Several of the reviewed papers are short and present only ideas of solutions. Many do not concentrate on data security. None of these papers presents a data flow control method for security based on SDN routing, and thus a direct comparison of results is not possible.

Surely, some of the techniques proposed may be compatible with our approach and, in combination with it, may lead to efficiency and security improvements; this will be the subject of further research.

11. Conclusions

We have shown the feasibility of using SDN routing in IoT contexts for implementing data security requirements of secrecy, integrity, and conflicts, as we have defined them in Sect. 2. In the implementation method we propose, data will be forwarded only to entities meant to receive them, other data will be dropped.

Previous research [30-21-22] has shown that, for any network, it is possible to efficiently calculate labels for the entities in such a way that data flows are determined by the label inclusion relationship. In this paper, it was shown how the labels can be used to construct forwarding tables for SDN controllers that will control data transfers, accordingly, thus ensuring data security. We have proposed a network organization based on cloud concepts with application entities and data entities (or servers, databases). We have demonstrated our method by using a simple 'hospital' example, which was simulated using standard SDN simulation tools. We envisage future systems where security administrators will be able to configure secure data flows by composing graphs such as the ones presented in our figures, at various levels of abstraction or granularity. Reconfigurations can also be done on such graphs, and labels and routing tables can be computed and updated automatically using our method.

Routing is taken care of by external SDN routers that will not burden the IoT devices. The use of our method will not affect the normal execution of SDN, except for the mentioned recalculation of labels and routing tables when reconfigurations are done.

With respect to the literature reviewed in Section 10, 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 reconfiguration; finally, we propose an implementation and a simulation of our SDN-enabled networks.

It should be pointed out, however, that our approach is feasible only for networks where the centralized planning we have envisaged is possible. In more decentralized systems, SDN may not be feasible, and it may be necessary to use conventional methods based on encryption agreed among entities. And even in the case of centralized control, encryption may be necessary to protect from covert channels, since data are transmitted in clear. The combined use of our method with encryption will be the appropriate solution in many cases, depending on the organization of the network, its applications, and the expected security threats and risks. We leave this to future research.

Furthermore, we have provided a generic method only, we have shown how the general data flow could be organized. For the proposed method to become practical, it will require the creation of a suitable administrative model: this is the subject of our ongoing research. IoT networks can be very complex, and their design must take into consideration many different requirements, including security requirements that have not been considered here.

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).

References

1. https://www.lifewire.com/how-many-devices-can-share-a-wifi-network-818298. (Consulted on Nov. 10, 2023).
2. M. Ahmad Khan. A survey of security issues for cloud computing. Journal of Network and Computer Applications.71 (2016), 11-29.
3. 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.
4. A. Al-Haj, B. Aziz. Enforcing Multilevel Security Policies in Database-Defined Networks using Row-Level Security. Intern. Conf. on Networked Systems (NetSys 2019), 1-6.
5. S.Asadollahi, B. Goswami,. M. Sameer. Ryu controller's scalability experiment on software defined networks, IEEE Intern. Conf. on Current Trends in Advanced Computing (ICCTAC 2018), 1-5.
6. D.E. Bell, L. La Padula. Secure computer system : unified exposition and Multics interpretation. Mitre Corp. Report MTR-2997 Rev. 1, March 1976.
7. Q. Burke, F. Mehmeti, R. George; K. Ostrowski, T. Jaeger, T.F. La Porta, P. McDaniel. Enforcing Multilevel Security Policies in Unstable Networks. IEEE Trans. on Network and Service Management 19(3) (2022), 2349-2365.
8. B. Celik, L. Babun, A. Kumar, H. Aksu, G. Tan, P. McDaniel, A. Uluagac. Sensitive information tracking in commodity IoT. 27th {USENIX}. Security Symposium ({USENIX} Security 18, 2018), 1687-1704.
9. S.Christos, P. Kostas, K. B.Gyu, B. Gupta. Secure Integration of Internet-of-Things and Cloud Computing. Future Generation Computer Systems 78 (2013) 964-975.
10. D. Denning. A lattice model of secure information flow. Commun. ACM 1(5) (1976), 236-243.
11. 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 .
12. 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.
13. 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, 162-179.
14. E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, A. Prakash. Flowfence: practical data protection for emerging IOT application frameworks. USENIX Security Symposium. (2016). 531-548.
15. A.Hany, W. Gary.Intersections between IoT and distributed ledger. Advances in Computers (Elsevier). Vol. 115 (2019), 73-113.
16. C. Haotian, Z. Qiang, D. Xiaojiang, L. Lannan. PFirewall: Semantics-Aware Customizable Data Flow Control for Smart Home Privacy Protection. arXiv preprint arXiv:2101.10522, 2021.
17. D. Huang, A. Chowdhary, S.Pisharody. *Software-Defined networking and security. From theory to practice.* CRC Press, 2019.
18. M.T. Islam, N. Islam, M.A. Refat. Node to Node Performance Evaluation through RYU SDN Controller. Wireless Pers. Commun. 112 (2020), 555–570.
19. K. Kalkan, S. Zeadally. Securing Internet of things with software defined networking. IEEE Comm. Magazine, Sept. 2018, 186-192.
20. E. Landwehr. Privacy research directions. Comm. ACM 59(2) (2016) 29-31.
21. L. Logrippo. Multi-level models for data security in networks and in the Internet of things. Journal of Information Security and Applications 58, 102778 (2021).
22. L. Logrippo, A. Stambouli. Configuring data flows in the Internet of Things for security and privacy requirements. 11th International Symposium on Foundations and Practice of Security. Montreal, 2018. Springer LNCS 11358,115-130.
23. S. Ola, E. Imad, K. Ayman, C.Ali. SDN controllers: A comparative study. 18th Mediterranean Electrotechnical Conference (MELECON 2016), 1-6.
24. G. 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.
25. X. Rong-na, L. Hui, S. Guo-zhen, G. Yun-chuan, N. Ben,. S. Mang,.Provenance-based data flow control mechanism for Internet of things. Transactions on Emerging Telecommunications Technologies, 32(5), (2021) e3934.
26. W.Roy, S. Bill, J. Scott. Enabling the Internet of Things. Computer (48) (2014), 28-35.
27. RYU Project Team. *RYU SDN Framework*- RYU project team, 2014.
28. R.S. Sandhu. Lattice-based access control models. IEEE Computer 26(11), 1993, 9–19.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 12 January 2024 doi:10.20944/preprints202401.0970.v1

29. J. Singh, T. Pasquier, and J. Bacon. In Intern. Conf. on Recent Advances in Internet of Things (RIoT'15), 2015, 1-6.
30. A. Stambouli, L.Logrippo. Data flow analysis from capability lists, with application to RBAC. Information Processing Letters (Elsevier) 141 (2019) 30–40.
31. H. Suo, J. Wan, C. Zuo, J. Liu. Security in the Internet of things: a review. 2012 Intern. Conf. on Computer Sc. and Electr. Engg., IEEE, 648-51.
32. A. Wang, X. Mei, J. Croft, M. Caesar, B. Godfrey. Ravel: A database-defined network, Proceedings of the Symposium on SDN Research (2016), 1-7.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.