Giuseppe Massimiliano MILOTTA

was born in Catania, Italy, in 1986.
He received his master degree at
Università degli studi di Catania in 2017
after a collaboration with
Télécom Paris Tech in 2016.
He finished the last year of PhD
in Information Engineering at
Università Mediterranea di Reggio Calabria
in 2020. He is now cooperating with the
CNIT research unit of Catania.
His main research activities focus on
SDN and the IoT's world,
with a particular interest in the UAVs,
security and group communications.

DOTTORATO di RICERCA in INGEGNERIA dell'INFORMAZIONE
SCUOLA di DOTTORATO
Università degli Studi *Mediterranea* di Reggio Calabria

The new applications populating the Future Internet will increasingly rely on the exchange of data among groups of devices, dynamically established according to their profile and habits. This will definitely challenge traditional group communication solutions that lack the necessary flexibility in group management and do not support effective control policies on involved endpoints. Indeed, today, Internet can support the following data delivery schemes: unicast, multicast, broadcast, and anycast, according to the way in which the endpoints of the information exchanges are identified. However, there are several reasons exist discouraging network operators to actually offer all such data delivery schemes to end users.
This work introduces new disruptive network-layer solutions based on the recent literature.
Among them, in the first part of this work, the Sociocast is presented, which has been theorized as enabler of flexible interactions among groups of devices tied by social relationships in the SDN context.
While, in order to overcome the problems arising by the centralized nature of MQTT, broker bridging, in the second and last part of this work, an optimization technique is proposed.

ISBN 978-88-99352-45-5

Progetto grafico
Centro Stampa d'Ateneo

COLLANA DELLA SCUOLA DI DOTTORATO DELL'UNIVERSITA' DEGLI STUDI *MEDITERRANEA* DI REGGIO CALABRIA

Enhancing Group Communications through the Social Internet of Things Giuseppe Massimiliano MILOTTA

DOCTORAL SCHOOL OF MEDITERRANEA UNIVERSITY OF REGGIO CALABRIA

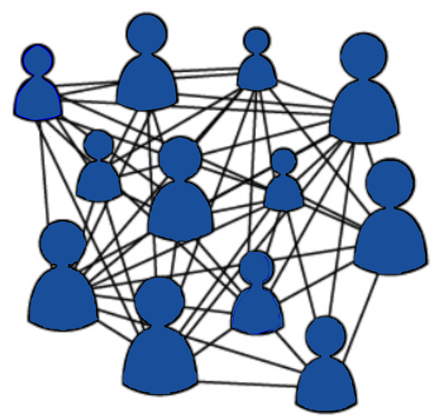
DRINF
DOTTORATO di RICERCA in INGEGNERIA dell'INFORMAZIONE

CSdA Centro Stampa d'Ateneo

DIIES
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE, DELLE INFRASTRUTTURE E DELL'ENERGIA SOSTENIBILE

Giuseppe Massimiliano MILOTTA

Enhancing Group Communications through the Social Internet of Things



Supervisors: Prof. Antonio IERA
Prof. Giacomo MORABITO
Coordinator: Prof. Tommaso ISERNIA
S.S.D. ING-INF/03
XXXIII Ciclo

Collana
Quaderni del Dottorato di Ricerca in
Ingegneria dell'Informazione
Quaderno n°49

DOCTORAL SCHOOL
UNIVERSITA' *MEDITERRANEA* DI REGGIO CALABRIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE, DELLE INFRASTRUTTURE E DELL'ENERGIA SOSTENIBILE (DIIES)

PHD IN
INFORMATION ENGINEERING

S.S.D. ING-INF/03
XXXIII CICLO

ENHANCING GROUP COMMUNICATIONS THROUGH THE SOCIAL INTERNET OF THINGS

CANDIDATE
Giuseppe Massimiliano MILOTTA

ADVISORS
Prof. Antonio IERA

Prof. Giacomo MORABITO

COORDINATOR
Prof. Tommaso ISERNIA

REGGIO CALABRIA, January 2021

Finito di stampare nel mese di **Gennaio 2021**

Edizione Centro Stampa d'Ateneo

Quaderno N. 49
Collana *Quaderni del Dottorato di Ricerca in Ingegneria dell'Informazione*
Curatore *Prof. Tommaso Isernia*

ISBN 978-88-99352-45-5

Università degli Studi *Mediterranea* di Reggio Calabria
Salita Melissari, Feo di Vito, Reggio Calabria

GIUSEPPE MASSIMILIANO MILOTTA

ENHANCING GROUP COMMUNICATIONS THROUGH THE SOCIAL INTERNET OF THINGS

Contents

List of Figures

List of Tables

Chapter 0: Introduction

In this chapter, I would like to introduce my thesis work, especially what are its goals and how it is structured, but also to give some reminders and look over some topics that will be examined in the following chapters.

0.1 Abstract

The new applications populating the Future Internet will increasingly rely on the exchange of data among groups of devices, dynamically established according to their profile and habits (e.g., common interest in same software updates and services). This will definitely challenge traditional group communication solutions that lack the necessary flexibility in group management and do not support effective control policies on involved endpoints (i.e., authorized senders and intended receivers). Indeed, today, Internet can support the following data delivery schemes: unicast, multicast, broadcast, and anycast, according to the way in which the endpoints of the information exchanges are identified. However, there are several reasons exist discouraging network operators to actually offer all such data delivery schemes to end users. As a result, application developers can rely on unicast communications only, and more complex group-based data dissemination policies are implemented as part of specific applications and services and through additional patches to the basic Internet implementation. And yet, group-based communications are crucial in several Internet of Things (IoT) application scenarios. To address the cited issues, the idea of introducing new disruptive network-layer solutions has emerged from recent literature. Among them, in the first part of this work, the Sociocast[1] is presented, which has been theorized as enabler of flexible interactions among groups of devices tied by social relationships in the SDN context.

[1] This work led to the publication of two articles: [64]and[65].

While, in order to overcome the problems arising by the centralized nature of MQTT, broker bridging (which has been is now supported by most MQTT implementations), in the second and last part of this work, an optimization technique is proposed. However, broker bridging does not address the performance issues occurring when publishers and subscribers are connected to different brokers.

In this work a technique is investigated exploiting the Social Internet of Things (SIoT) concept to decide the most convenient broker for each publisher/subscriber. A hybrid technique which integrates experimental results obtained in an emulated testbed and analytical derivations will be introduced to evaluate the performance of the proposed approach.

0.2 Original Contributions of this work

My original contributions to the works characterizing this thesis are listed below:

- Study and implementation focused on the integration in the SDN environment of Sociocast.
- Experimentation in SDN and analysis of different IoT use cases and scenarios: smart industry, smart home, WSN management, smart mobility.
- Study and implementation of a clustering algorithm based on the model of Karger's algorithm.
- Implementation and experimentation, in an emulated smart campus scenario, of a technique which uses the above-mentioned clustering algorithm and the Social Internet of Things to enhance MQTT bridging.

0.3 Organization of this work

In the first chapter the background concepts, needed to fully understand the two discussed solutions, are given. Starting from the definitions of SDN and SIoT with an in-depth study on IoT and MQTT.

In the second chapter, the concept of Sociocast is introduced as a solution based on Software Defined Networking (SDN) for its implementation at the network layer in Internet of Things. The Sociocast performance is studied and compared to methods running at the application layer that provide similar features. Experimental results, as well as the used tools description are also given.

In the third chapter, a MQTT bridging optimization technique is introduced based on a social clusterization of the network elements characterizing the network topology. The performance of this approach and the comparison with a benchmark are also discussed, as a brief description of the tools used during the experimentation phase.

In conclusion, a summary of all the results and considerations regarding the Sociocast and the MQTT bridging optimizer, and possible future works will be discussed in the last chapter.

Chapter 1: Background

This chapter will provide all the definitions and discuss all the properties of the subjects necessary to have a full view and a better comprehension of the major topics of this work.

In particular, in the first paragraph 1.1, is discussed in depth the Software Defined Networking (SDN) approach, giving a definition, analyzing its three states of abstraction and the various open-source SDN controllers available today.

In the second paragraph, 1.2, the Social Internet of Things paradigm will be analyzed, starting from a bit of history and fun fact on IoT to better understand its natural evolution and escalation up to now days. Moving to the need for a social bound between "things" and, especially, how the SIoT paradigm works and can enhance the IoT world.

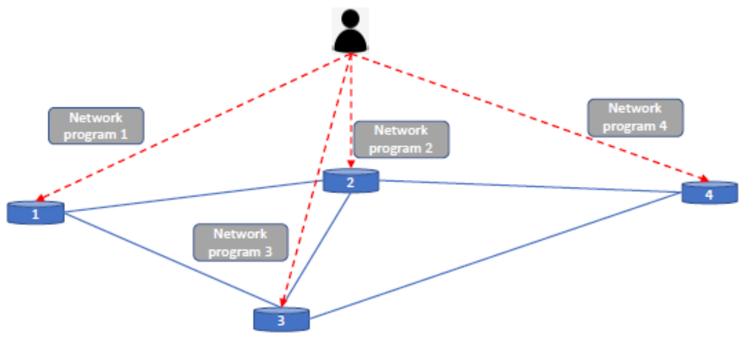
In conclusion, to better understand the Chapter 3 of this work, the MQTT protocol will be discussed in paragraph 1.3.

1.1 Software Defined Networking

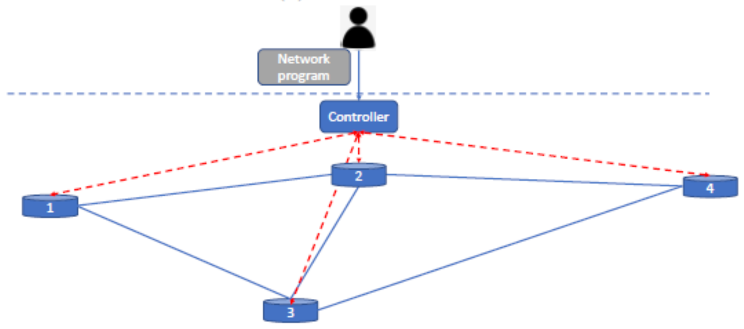
1.1.1 What is SDN?

The overview of the SDN paradigm will start by answering a simple yet fundamental question: "What is a Software Defined Network?"
Simply, a Software Defined Network is a network which operates as specified by means of a software program.
One possible objection to this answer is that, in the years the boundary separating what is software and what is hardware inside network devices has been going down in the protocol stack and, in some cases, such boundary is even fading away (consider the NetFPGA project, for example [1]). According to such a view, which is firmly rooted on an incontrovertible truth, SDN would just be a fancy name given to a trend going on for several decades.

This view can be easily refuted by observing that the above mentioned trend would allow what can be called a Network of Software-Defined Devices.



(a) Without SDN.



(b) With SDN.

Figure 1: Programmable networks without (a) and with (b) SDN.

According to such a view a hypothetical network programmer can indeed define the behavior of the network through software. However, in order to do so, he/she needs to write and deploy a different program for each network device as sketched in Figure 1.

The Software Defined Network idea, instead, implies a totally different way of programming the network as depicted in Figure 1b which relies on a unique application programming interface (API) for the entire network. This is indeed a completely different paradigm which opens the path to a realm of new possibilities.

In the effort of defining a taxonomy, in [2] four conditions are given for identifying a software defined network:

- The control and packet forwarding planes are decoupled thus, network devices become simple (packet) forwarding elements;
- Forwarding decisions are flow-based rather than destination-based. A flow is defined by a set of packet field values acting as match (filter) criterion;

- The control logic is moved to an external entity called SDN Controller or Network Operating System (NOS);
- The network is programmable through software applications running on top of the NOS which interacts with network elements.

While the above requirements give the most comprehensive description of the key SDN features to date, they may be all specific of the SDN general concept or reflect the characteristics of the current SDN implementations.
For example, is the flow-based requirement a key feature of the SDN paradigm or is it just a heritage of OpenFlow? Therefore, according to the philosophy perfectly contained in the Van der Rohe's sentence "*less is more*", it is reported in the following, the broadest definition which can be found in the words of Nick McKeown who has played a key role in software defined networking development [3]: "*A Software Defined Network is a network in which the control plane is physically separate from the forwarding plane and a single control plane controls several forwarding devices.*"
SDNs have several advantages over traditional networks; in fact, they are [4]:

- Agile: Abstracting control from forwarding allows the administrators to adjust traffic flow dynamically.
- Centrally managed: Network intelligence is (logically) centralized in SDN controllers which have a global view of the network.
- Programmatically configured: SDN allows network administrator to configure, manage, secure, and optimize resources dynamically by means of automated SDN programs.
- Open Standards-based: SDN makes easy the network design and operation since instructions are provided by SDN controllers instead of specific devices and protocols.

1.1.2 SDN Architecture

The actual realization of the SDN concept requires addressing a large number of technical issues which boil down to

- revisiting network architectures which have been considered untouchable for a very long times;
- defining new abstractions of the resources and operations of the network as a whole;
- designing consolidation mechanisms able to control the operations of individual network devices in such a way that the desired behavior of the network as whole emerges.

The rest of this paragraph will focus on the architecture implications of software defined networking, while in the following sections we will address the other two items.

In Figures 2 and 3 we show the innovation on the architecture introduced by SDN at the both the levels of individual network devices and the entire network, respectively.

In the right part of figure 2 it is shown the architecture of traditional routers. We can clearly distinguish two subsystems. One contains the interface cards whereas the other contains the controller cards. The interface cards subsystem is responsible for packet forwarding which is executed according to the content of a table called forwarding table. The content of such table is copied by the routing table which is contained in the controller card subsystem and is calculated by executing the distributed routing algorithm denoted as routing control in the figure. The controller card subsystem exposes a user interface which can be used by the network administrator to manage the network device. Such interfaces offer a limited number of options regarding the control of the network behavior. Furthermore, it is clear that, while there are common features, the interfaces offered by routers produced by different vendors are different and therefore, the know-how developed for configuring one of them cannot be used to configure the others.

In SDN this monolithic architecture is broken into two pieces so that forwarding and control functions are clearly (even physically) separate. Accordingly, all network devices become just switches whereas the control functions are concentrated in appropriate machines running a control program. In fact, switching devices treat received packets according to what decided by the control program. The interactions between the network devices and the control plane happen through an open interface called Southbound API. More specifically, according to the OpenFlow naming the switches are called OpenFlow Switches whereas the control plane is called OpenFlow Controller. The interactions between OpenFlow Switches and OpenFlow Controller happen according to OpenFlow protocol.

In modern SDN solutions the Controller consists of a Network Operating System (NOS) that provides several Network applications with access to the abstraction of the network resources. The interface between the NOS and the network applications is called North-bound interface

According to such an approach the network layer disappears from the intermediate network devices which, instead, execute the physical and link layers only, as shown in Figure 3.

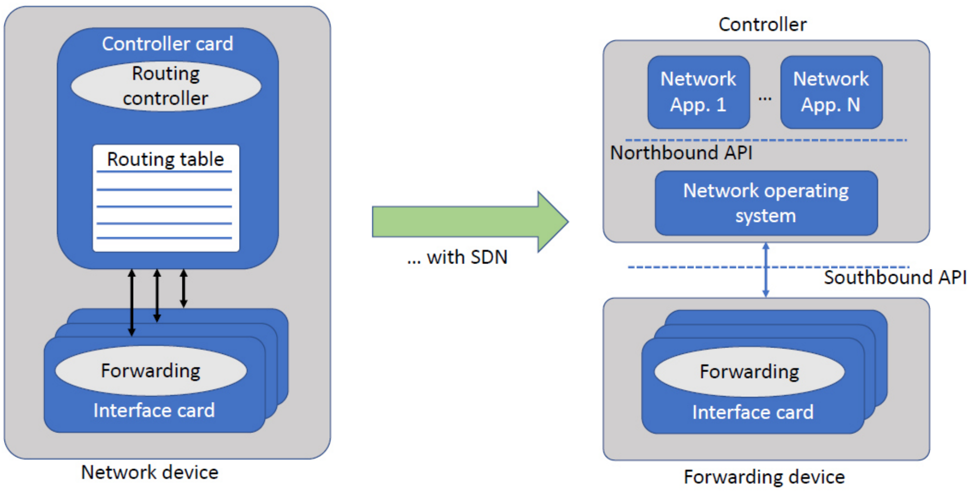


Figure 2: Impact of SDN on the architecture of network devices.

1.1.3 The three fundamental abstractions of SDN

The SDN paradigm has been developed around the core objective of enabling a simple control of a complex system such as a communication network [5]. Such philosophical approach has put a the center of the SDN design the definition of appropriate abstractions.

In fact, defining a solution requiring the low level configuration of each network device would be impossible in a network with more than a few network devices. Quoting the analogy with software programming provided in [5], it would be like requiring the programmer of a complex distributed software to decide at which memory address to store each variable utilized in the software. This is, indeed, what happened with early programming language, however, modern languages offer high level abstractions of computer resources, e.g., (virtual) memory, network, processing, which can be exploited by the programmer without knowledge of the way such resources are actually managed by the specific hardware components.

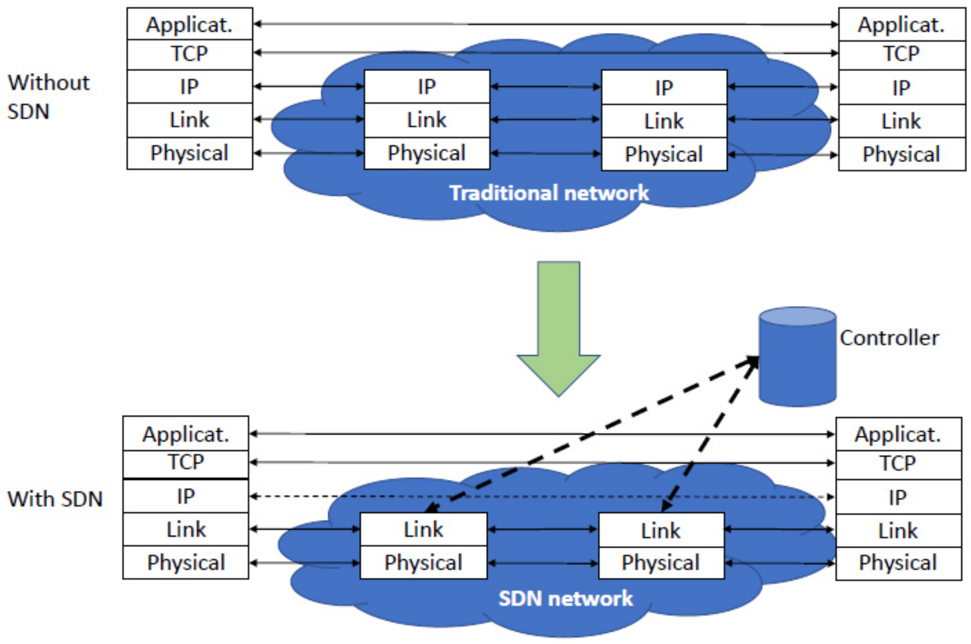


Figure 3: Impact of SDN on the network protocol stack.

Such approach has the advantage of decoupling the development of new programs from the implementation of the functions hidden behind the above mentioned abstractions. In fact, they can evolve independently as long as the interfaces are maintained, which fosters the astonishing rapid progresses characterizing the software arena.

Therefore, in the SDN development it has been crucial to identify the core abstractions around which the rest of the paradigm should be designed.

The three fundamental abstractions of SDN are

- **The forwarding abstraction**, which should encompass any possible forwarding behavior while hiding details of the underlying data plane operations;
- **The distribute state abstraction**, which should consider all operations required to collect the state information about data plane devices in order to form a global view of the network;
- **The specification abstraction**, which should allow a network application to express the desired network behaviors without being responsible for implementing those behaviors.

In the following sections of this chapter each of the above abstractions will be analyzed, with special reference to their definition in OpenFlow, which is the most known SDN solution, to date.

1.1.3.1 Forwarding abstraction

In OpenFlow forwarding is flow-based: all possible packets are partitioned in subsets called flows and each forwarding element, i.e., the OpenFlow Switch, treats incoming packets as specified for the flow they belong to.

Accordingly, the forwarding abstraction deals with two questions:

1. how to classify incoming packets in ows, and
2. what can be done on the incoming packets.

Regarding the first issue, we observe that a flow, f, is defined through a set of $n > 0$ rules, $r_1, r_2,\dots,r_n$. A packet belongs to flow f if it satisfies all of the above n rules.

In principle rules may consider any feature of the packet. However, in most SDN implementations including OpenFlow, the format considered for the packets is compliant with TCP/IP and rules are based on the content of the fields of the headers of levels 2-4 of the protocol stack.

Despite such choice imposes a certain degree of rigidity in the definition of the rules, it has two fundamental advantages, though.

First, it simplifies the design and realization of the hardware of the forwarding devices. As a consequence, it has been immediately possible to realize SDN forwarding devices capable of operating at line rate [6]. This has been crucial in encouraging industry in adopting the SDN approach.

Second, it makes implementing the switching/routing operations typical of TCP/IP extremely easy, as we will see shortly. This has been crucial, instead, to encourage network managers in adopting the SDN approach.

Such rigidity is being relaxed in new OpenFlow versions, yet, it has always been a clear OpenFlow design directive to make the transition from traditional networks as smooth as possible and thus, we expect that the way rules are defined will remain rooted over TCP/IP layering and packet formats in the next future.

There are, however, certain networking environments where flexibility is way more important than data rates and compatibility with traditional networking solutions. This is the case, for example, of wireless sensor networks (WSN). In fact, in WSN data rates are usually low and there are no well established protocol stacks to be compliant with. Other features, such as the possibility to treat data packets depending on the values they are carrying, are more important instead. Accordingly, solutions with more flexible ways of defining rules have been proposed for WSN. SDN-WISE is a notable example of such category [7].

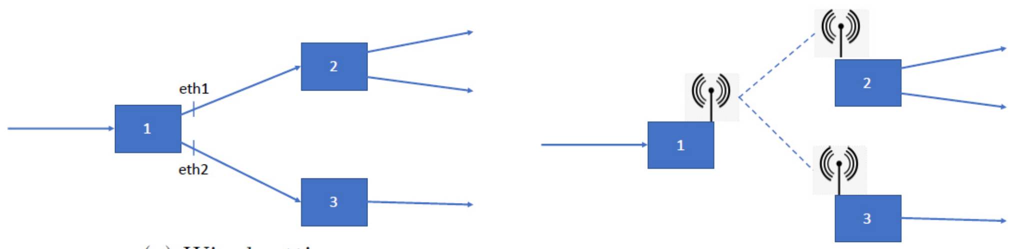
For what concerns the classification of packets in flows, we observe that depending on the way in which rules are given, it is possible that a packet might satisfies the rules of several flows. If this is the case, it is important to define the policy which should be considered to classify the packet.

In traditional SDN solutions a packet is assigned to a unique flow. If this is the case, then a priority scheme should be defined.

Other solutions, allow to assign packets to several ows and treat them accordingly. More specifically, suppose that a given packet satisfies the rules given for flows f1 and f2. If the forwarding operations for packets of the two flows are

- the same, the packet will be forwarded as specified for the two flows;
- different, the packet will be duplicated and one copy will be forwarded as given for flow f1 whereas the other copy will receive the treatment given for flow f2.

Returning to the second issue regarding the forwarding operations that can be specified for packets in a SDN network. Such operations are usually called actions and, for each flow, it is possible to define a pipeline of actions which should be applied to the relevant packet. It is obvious that the major action is "forward". In this case it is necessary to specify to which element(s) the packet should be forwarded to. This is typically done by giving the output port through which the packet should be relayed.



(a) Wired setting. (b) Wireless setting.

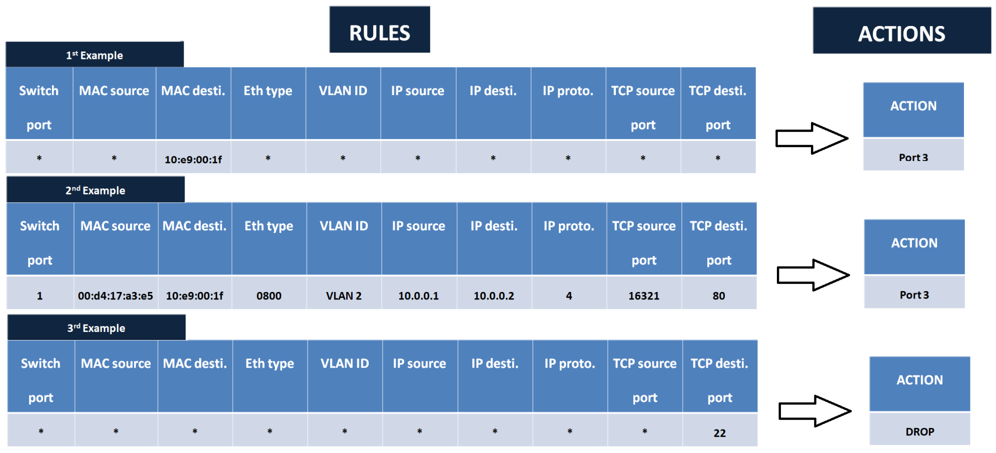
Figure 4: Forwarding in wired and wireless setting.

Besides enabling the forwarding of a packet through a specific network ports, OpenFlow allows to flood the packet through all network ports but the ingress port, to send the packet to the controller, or to include the content of the packet in the forwarding table.

Another fundamental operation, besides the obvious "forward" discussed above, is "drop", for example, using such operation it possible to allow a forwarding element to behave like a firewall.

Note that the two above operations do not change the packets. Indeed, there are operations that can modify the packet header at layers 2-4 of the TCP/IP protocol stack.

We note that through appropriate combination of rules and actions, it is possible to transform a SDN forwarding element in any desired traditional networking element. For example, in the Figure 5 we show how to set rules and actions in such a way that the forwarding elements behaves like a traditional switch, a flow switch, and a firewall.



RULES

ACTIONS

1st Example

Switch port	MAC source	MAC desti.	Eth type	VLAN ID	IP source	IP desti.	IP proto.	TCP source port	TCP desti. port	ACTION
*	*	10:e9:00:1f	*	*	*	*	*	*	*	Port 3

2nd Example

Switch port	MAC source	MAC desti.	Eth type	VLAN ID	IP source	IP desti.	IP proto.	TCP source port	TCP desti. port	ACTION
1	00:d4:17:a3:e5	10:e9:00:1f	0800	VLAN 2	10.0.0.1	10.0.0.2	4	16321	80	Port 3

3rd Example

Switch port	MAC source	MAC desti.	Eth type	VLAN ID	IP source	IP desti.	IP proto.	TCP source port	TCP desti. port	ACTION
*	*	*	*	*	*	*	*	*	22	DROP

Figure 5: Rules and actions examples in the forwarding abstraction.

The rules given to define a flow and the corresponding actions to be executed on the flow packets are usually stored in appropriate tables, called flow tables, maintained in the forwarding devices. The entries of the flow tables, which also contain some statistical information about the flow load, are set by the controller.

More specifically, incoming packets that the forwarding element cannot classify in a flow are forwarded to the Controller which replies by installing the corresponding flow entry in the flow table. From that moment all packets of the new defined flow are treated by the forwarding device accordingly.

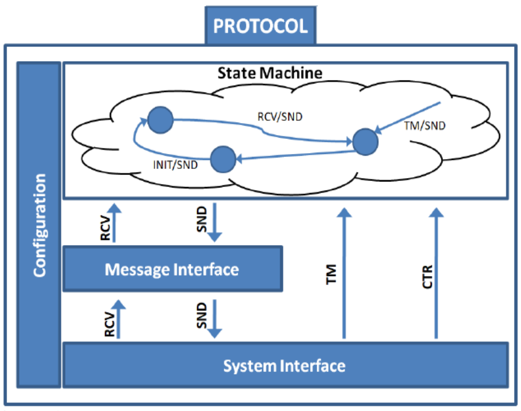
Before concluding this section, observe that in OpenFlow and early SDN solutions flow tables specify stateless forwarding rules. In fact, in such approaches all stateful operations are executed by the controllers only. There are cases, however, in which such approach requires frequent interactions between the forwarding devices and the controller. Each interactions, however, results in increased delay and additional signaling. The above problem of the stateless approach has been recognized by several researchers and several stateful SDN solutions have been proposed. Early examples are OpenState [8] and FAST [9] for infrastructured networks and SDN-WISE [7] for infrastructureless networks.

1.1.3.2 Distribute state abstraction

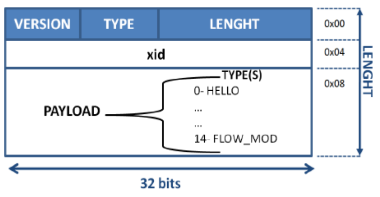
SDN network applications are responsible for setting the policies for packet handling throughout the network which is a complex distributed systems. The SDN controller is therefore, responsible for collecting information regarding the state of the network elements so as to create a consistent and comprehensive view of the current network state as well as for configuring the behavior of individual network elements in such a way that, collectively, they implement the policy decided by the network application.

The operations required for data collection and network device configurations are specified through an appropriate protocol. In the rest of this section, we will describe the fundamental features of such protocol in the case of OpenFlow.

As shown in Figure 6a, the OpenFlow protocol consists of four major components: message layer, state machine, system interface, and configuration [10].



(a) Openflow protocol overview.



(b) Format of protocol messages.

Figure 6: OpenFlow protocol.

1.1.3.2.1 Message layer

The *message layer* is the core of the protocol as it defines the format for all relevant messages. Accordingly, it should support the construction, copy, comparison, printing, and manipulation of messages.

The interested reader can refer to [10] for the specification of the structure and the semantic of individual messages, which depends on the specific protocol version.

Here we just observe that all OpenFlow protocol messages begin with the same header structure shown in Figure 6b. The *version* field identifies the version of OpenFlow utilized to generate the packet. The *length* field gives where this message will end in the byte stream starting from the first byte of the header. Third, the *xid,* or transaction identifier, is a unique value used to match requests to responses. Finally, the *type* field which indicates what type of message and, thus, how to interpret the payload, is version dependent.

1.1.3.2.2 State machine

The *state machine* defines the behavior of the protocol. It is used to describe actions such as: negotiation, capability discover, flow control, delivery, etc.

In this perspective, observe that OpenFlow has a very simplified machine model. In fact, almost all the messages are asynchronous, and thus states are not necessary. However, some complex operations require a state to be maintained. Examples of such operations include connection establishment which requires some interaction between the two end points, i.e., the OpenFlow Controller and the OpenFlow Switch, and capability negotiation that is performed before packets can be exchanged. Again, we refer the interest reader to [10].

1.1.3.2.3 System interface

The system interface specifies how a protocol relates with the outside world. Usually it determinates mandatory and optional interfaces along with their use, such as TLS and TCP as transport channels.
System interface is the part of OpenFlow protocol that provides service to other
elements in the system, or that depends on and invokes other elements in the system.

In OpenFlow there are the following system interfaces:

- **TCP/TLS interface:** TCP/TLS interface connects with protocols in the lower level protocol stack, as indeed TCP and TLS. It provides trustworthy stream oriented transmission between switch and controller, by transfer any OpenFlow message between them.
- **Switch agent interface:** Switch agent interface interacts with the system kernel of OpenFlow switches. It forwards messages from the controller to the switch's kernel to be processed. It also forward switches asynchronous messages to the OpenFlow stack for processing and then transmitting them to the controller.
- **Controller application interface:** Controller application interface combine with controller applications that run on top of the stack. It accepts messages from controller application to switch and sends them to OpenFlow stack for processing and transmission. It also transmits messages sent to the controller application from OpenFlow stack.
- **Configuration interface:** OpenFlow incorporates some parameters that can to be configured (e.g. controller IP addresses) by system

operator either before or all along their use. These parameters are set or can be modified through the configuration interface. Hence the configuration interface provides as a service to system operator the capability configure OpenFlow stack.

1.1.3.2.4 Configuration

The OpenFlow protocol is rich in elements and aspects which need to configure or set some initial values, like the default buffer size and the reply intervals to X.509 certificates.

Two important aspects of the OpenFlow configuration are:

- **Configuration Language,** which is created to have an interface to easily configure the controller and the switches. This language is implemented by using a front-end compiler which validates syntax and typing so that the configuration can be checked for validity and more advanced configuration features can be implemented.
  The main structures in the language are bindings between stages and identifiers, and between the staging of those identifiers. The types in the language are predefined, using literals for IPv4, IPv6, integers, time, protocols, strings, and OpenFlow protocol versions. These literals can be merged into predefined types, which form stages (example of stages are: Realm, Authentication, Authorization, Initialization). Each stage describes either a controller's behavior that should be adopted when making or accepting new connections, or the connection behavior for a switch.
- **Configuration Utility**. The configuration language is backed by a utility that checking grammar and typing, and then creates commands that can be run to configure the controller or the switches. This utility is characterized by four stages: **Parsing,** which occurs concurrently with **tokenizing,** will evaluate if the input is properly formatted. **Elaboration** checks the whole input matches types defined by the language, preventing incorrect input for specific fields. And finally, **evaluation,** in which the input is evaluated and convert into commands for the controller or the switch agent.

1.1.3.3 Specification abstraction, from SDNv1 to SDNv2

Before illustrating the features of the Specification abstraction, we observe that this was not included in early mainstream SDN solutions. It was only in a second development phase that the need for such a new abstraction was recognized. Accordingly, we will start by discussing how SDN would work without the specification abstraction and, thus, why it is needed. Then, we will provide details about such abstraction. Finally, since SDN controllers are responsible for the support of the specification abstraction, we will illustrate the main features of the most popular SDN controller platforms realized so far.

1.1.3.3.1 The Network Operating System

To begin with, the reader must be aware that there are crucial differences between a NOS and an "Operating System" (OS) in computer science. In fact, for computer science an OS is based on strong fundamental principles such as synchronization, mutual exclusion and others. Conversely, this is not the case for the networking where we can observe a lot of guidelines but only a few fundamental principles.

Despite the fact that networks were easy at the beginning (just think about EtherNet-IP), as time passed by and new control requirements and tools (VLAN, middleboxes, deep packet inspection, etc) were added, they have grown in complexity. Now, some questions should pop up in the reader's mind, like: "If they are so complex then how do they still work? And why nothing has changed in networking so far?". The answer is simple: every time a new requirement appears, it is applied by defining a new protocol, a new ad hoc mechanism or simply by manually configuring the network. Also note that, as IP is used everywhere and it has undergone very few changes since its first appearance, this means that any new control requirement should be IP compliant (this is the real reason why it's so difficult to change!) and should take into account every possible configuration of the network elements. This results into adding massive complexity to the control plane. SDN tries to hide all this complexity and to extract simplicity due to abstractions.

As asserted previously, the state distribution abstraction aims at hiding state dissemination and collection to the control program thanks to a global network view. As such the control program takes the global network view, which is a graph of the network, as input and gives a configuration of each device in the network as output.

The NOS is responsible for the creation of the global network view. The NOS is a distributed system that runs in servers (controllers) in the networks: by using the forwarding abstraction, it communicates with all the elements in the network, getting state information from these devices and sending control directives that should be implemented. The compound between the NOS and the forwarding abstraction is known as the SDNv1.

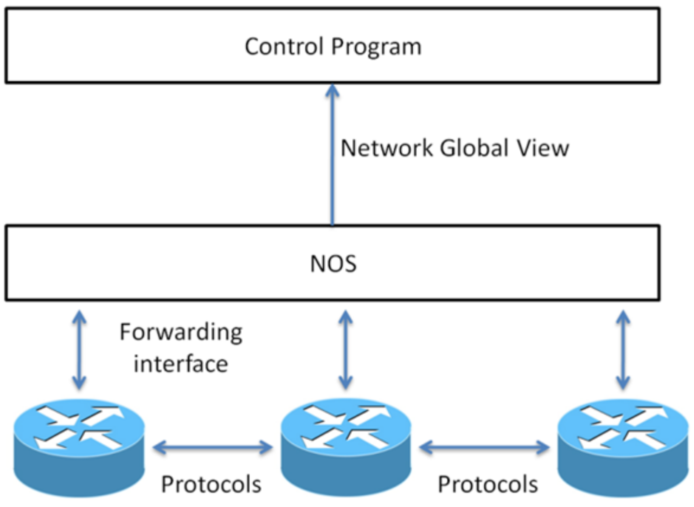


Figure 7: SDNv1 architecture.

The benefits that SDNv1 introduces are the following:

- There is no need for creating distributed control protocols, we can just define a single centralized control function.
- For the control program a network configuration becomes a simple function of the global view:
  Configuration= F(View)
  which implies that it is easy to write and apply, but also to verify and maintain.

- NOS handles the state of the network from the dissemination to the collection of the states.
- NOS must achieve eventual consistency with the real network, which means that the configuration, function of the view, is eventually correct and two clients may have two different views of the network: fortunately this happens with a negligible probability. Also, there are no consistency requirements for packets and flows and we found a strong consistency for the modification of the control function. All this makes SDN scale.

Nevertheless, something is still missing: according to the control program requirements, the NOS sends the network configuration, created by the global view, to the physical devices in the network. One possible way to do so is via OpenFlow.

With this approach, the control program has to configure each device in the network. This happens because the NOS eases the implementation of functionalities, but not their specification. There is a need for another abstraction: the specification abstraction.

1.1.3.3.2 The Specification Abstraction

The specification abstraction gives the control program an abstract view of the network, which is a function of the global view created by the NOS. The program responsible for the creation of this new view is called Network Hypervisor (NHypervisor). A simple example of how the NHypervisor works is shown in Figure 8, where all the access to the network are taken into account for an access control application.

Starting from the network abstract view, the control program creates an abstract configuration of the network.

Abstract configuration= F(Network abstract view)

This way, the model created has all the details needed to set the control program's aims without giving all the information which is necessary to implement these goals: this is known as SDNv2, Figure 8.

To summarize, in SDNv2, whenever the control application wants to install a specific behavior in the network elements, it speci_es such behavior on the abstract network view; then, the Network Hypervisor maps all the controls from the abstract view to the global view configuration; ultimately, the NOS distributes this configuration to the real devices in the network.

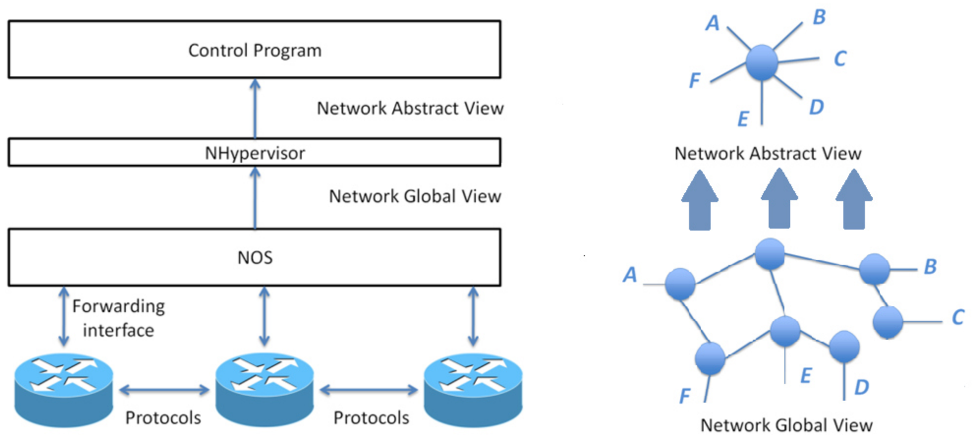


Figure 8: SDNv2 architecture

1.1.3.3.3 The SDN controllers classification

In the last few years, several SDN controllers have entered the market, and have been used by network programmers and managers with different aims and different backgrounds all around the globe. The aim of this paragraph is to classify the most popular Open Source SDN controllers available today, like OpenDayLight (ODL) [11], the Open Network Operating System
(ONOS) [12], Ryu [13] and so on, evaluating the advantages and disadvantages of each of them and trying to answer such questions as "Why should I prefer to use this specific SDN controller rather than another?".

First of all, it is crucial to understand that each platform has different usages based on what it can do and also on the scope of the project and the organization behind it.

A first classification can be done by considering the architectures so to divide the SDN controllers in two groups: *Single Instance* (or *Centralised*) and *Distributed*.

In the Centralised solutions, the SDN controllers are easier to maintain and grant lower latency between the tightly coupled southbound API, the path communication elements (PCE) and the northbound APIs; on the other hand, as the SDN network grows bigger, centralised controllers are likely to become a bottleneck. Centralized solutions include platforms such as ONOS and ODL.

By contrast, in the Distributed platforms each function can be scaled independently, by decoupling the processing of PCE, Telemetry and Southbound interface traffic, in order to avoid congestion and to allow the platform to scale more effectively. Additionally, specialised tools to handle and analyze big datasets can be used without negatively impacting southbound protocol performance. However, this achievement is paid in terms of complexity, both in deployment and in maintenance.

Such solutions includes controllers such as OpenKilda [14] and Faucet [15].

Ryu is a bit different compared to the other choices: although its core set of programs can be considered as a platform, it can be imagined as a toolbox in which the SDN controller functionality can be found.

In the following, the abovementioned NOS solutions are compared in the light of some key features:

- **Scalability and Fault Tolerance:** As stated above, scalability is a crucial characteristic and should be taken into account when classifying the SDN platforms. As the size of the network grows, it becomes impossible for a single controller to handle the load of information coming from every switch in the network; also, the probability of having fault in the system increases. For this reason, dividing the network into smaller logical islands/clusters decreases the need for a single southward looking network to scale. All the platforms with a centralized architecture, like ONOS, try to scale in this way, including native BGP routing to orchestrate traffic flows between the SDN islands. ONOS and ODL are the only ones implementing native clustering and being able to maintain a cluster. Each of these is supported by a distributed datastore that shares the current network state between the islands; also, both of them also provide a fault tolerance system with an odd number of SDN controllers. In the case of master node failure, a new leader would be selected to administrate the network.

OpenKilda approaches cluster scalability in a modular way. While Floodlight is used as a southbound interface to the switch infrastructure, responsibility for PCE and telemetry processing is pushed northward into a completely separated Apache Storm based cluster.

Both Ryu and Faucet contain no intrinsic clustering capability and require external tools such as Zookeeper to distribute a desired state. So OpenKilda, Ryu and Faucet have no inbuilt clustering mechanism, and rely on external tools, instead. This simplifies the architecture of the controllers and eases them from the overhead of maintaining distributed databases for state information. High availability is achieved by running multiple, identically con_gured instances, or a single instance controlled by an external framework that detects and restarts failed nodes.

- **Modularity, Programming Languages, and Interfaces:** The modularity of each controller is determined by the design focus and programming languages. ONOS and ODL have functions that connect code modules at the cost of centralising processing to each controller. They are both Java-based controllers which use OSGi containers for loading bundles at runtime, allowing a very flexible approach to adding functionality. Also, since Java is a well-known and widely used programming language, the development resources are abundant, with good supporting documentation and libraries available. Ryu is a Python based controller and provides a well-defined API for developers to change the way components are managed and configured. Adding functionality to Faucet and OpenKilda is possible by modifying the systems that make use of their northbound interfaces, such as the Apache Storm cluster: this allows for the added flexibility of using different tools and languages depending on the problem to overcome. The northbound API too is managed differently by these platforms. ONOS and ODL have the largest set of northbound interfaces with RESTful APIs and gRPC, making them the easiest to integrate. Ryu and OpenKilda offer limited

RESTful compared to ONOS and ODL, whereas Faucet takes a completely different path, relying on configuration files to track the system state rather than of API calls.

In conclusion, all the characteristics and peculiarities of the SDN controllers described above are summarized in Table 1.

SDN Controllers' Features			
Controller	Language	Architecture	Native Clustering
ONOS	Java	Centralised	Yes
OPENDAY LIGHT	Java	Centralised	Yes
RYU	Python	Distributed	No
FAUCET	Many	Distributed	No
OPENKILDA	Many	Distributed	No

Table 1: SDN controllers' features.

1.2 Social Internet of Things

1.2.1 IoT, through a bit of history to a definition

Nowadays, if we had to give a definition of Internet of Things (IoT), probably, first, we would start trying to define what a “thing” is. A “thing” can be a person with a wearable, like a smartwatch, or with a heart monitor implant, a farm animal with a injectable ID chip (biochip transponder), a car with sensors to alert the driver of other nearby cars or some sort of danger in the road, like a pedestrian crossing the street.

Simply said, a “thing” could be any object that can be assigned with an IP address and is able to transfer data over the Internet.

If we take this definition in a strict way, then we will be very surprised to discover that the first IoT devices fitting in it are much older than expected. The first one, in the year 1982 at Carnegie-Mellon’s Computer Science Department, was a Coca Cola vending machine connected to the Internet, allowing some grad students to check the stock and temperature of the drinks stored in it. Or, a toaster[2], in the early 90’, that was turned on by a remote computer.

We have to wait the last year of the old millennium to actually hear and read the term “Internet of things”. It was the 1999, when Kevin Ashton coined and used “Internet of things” in a presentation for Procter & Gamble describing a system in which sensors where used to enhance a computer. But these are the foundations on which today's concept of Internet of Things, or IoT, and its definition are based:

“A network of items—each embedded with sensors—which are connected to the Internet.”[16].

[2] John Romkey, software engineer, had built one for the showfloor of Interop 1990

The IoT, in conclusion, is an ecosystem of connected smart devices, mechanical and digital machines, people and even animals that are provided with an unique identifier and the capability to send data over the Internet without necessitating any human-to-human or human-to-computer interaction.

1.2.2 Internet of Things standards

As foretold from a report of Cisco in 2018, more than 500 billion devices are expected to be connected on the web for the year 2030. These numbers are estimated considering how the usage of IoT is exponentialy increasing, partculary in organizations and in the vast majority of industries to operate more efficiently, to better understand customers needs, to improve decision-making and overall to increase the value of the business in general.

For this reason, a branch of the IEEE, the IEEE-SA (IEEE Standards Association) which is responsible for defining the specifications and best practices based on nowadays the scientific and technological knowledge, determined over 140 standards and projects that are related to the IoT concept. One of these projects is the IEEE P2413 [17].

The scope of IEEE P2413 is to define and standardize the IoT architectural framework, address descriptions of various IoT domains, define their abstractions, and identify commonalities between different domains. Furthermore, IEEE P2413 is considering the architecture of IoT as three-layered, as shown in Figure 9: IEEE P2413 IoT architecture.

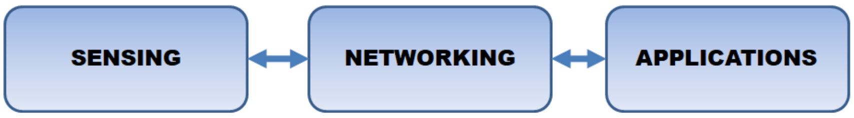


Figure 9: IEEE P2413 IoT architecture.

The goals for the IEEE P2413 are the following:

- Increasing system compatibility and interoperability to enable cross-domain interaction and platform unification
- defining an IoT architecture framework which groups all the various IoT application domains.
- increasig transparency
- reducing industry fragmentation
- leveraging the state of the arts.

1.2.3 From "smart" to "social", an evolutionary leap

Nowadays, we are familiar with devices that thanks to their abilities we call "smart objects" and which are considered the fundamental elements of the IoT. These objects are just a first small step in an evolutionary path which has been characterizing modern communication devices since the advent of the IoT in the telecommunications' world.

As stated in [18], it is possible to make an analogy with the evolution of human beings starting from *homo sapiens* to *homo socialis,* passing by *homo agens* and predict a not so far future for smart devices, imagining a similar evolution scheme, from *res sapiens* to *res socialis,* as shown in Figure 10: Evolution from *Res sapiens* to *Res socialis* where are also reported all the main features characterizing these categories.

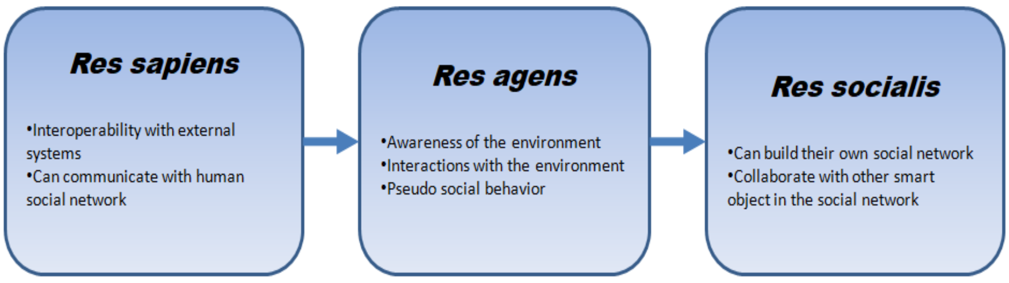

Res sapiens
•Interoperability with external systems
•Can communicate with human social network
Res agens
•Awareness of the environment
•Interactions with the environment
•Pseudo social behavior
Res socialis
•Can build their own social network
•Collaborate with other smart object in the social network

Figure 10: Evolution from *Res sapiens* to *Res socialis*

The basic idea to the concept of *res socialis* is that: smart objects, in general are associated with the services that they can offer or deliver and that a social network of smart object enhances the capability of the object to publish services and information, to find them and to discover new resources to better implement them. A smart object just has to navigate the social network of friends instead of using the typical internet discovery tools, and all the scalability issues that come with them.

The advent of this generation of social objects is related to the actual creation of a "social networks of smart objects" in analogy with the social networks of human beings.

To do so, the following conditions are needed:

- Giving a notion of social relationship among objects, and
- Giving a reference architectural model implementing a social Internet of Things based on the codified inter-object relationships,

In relation to the first issue, the definition of a social behavior established between objects has been addressed in [19] where a definition of a novel paradigm called Social Internet of Things (SIoT) has been given, in addition to all the relevant social structures occurring between the smart objects (nevertheless some of these social structures and social relationships will be explained and discussed in the section 2.2).

Regarding the second issue, giving a reference architecture for the SIoT is mandatory to fully comprehend the Sociocast approach which is explained in the next chapter. The SIoT architecture is based on the three-layered architectural model for IoT presented in Figure 9: IEEE P2413 IoT architecture. and it consists of:

- the sensing layer, devoted to the data acquisition and node collaboration in short-range and local networks;
- the network layer, which is aimed at transferring data across different networks;
- the application layer, where the IoT applications are deployed together with the middleware functionalities.

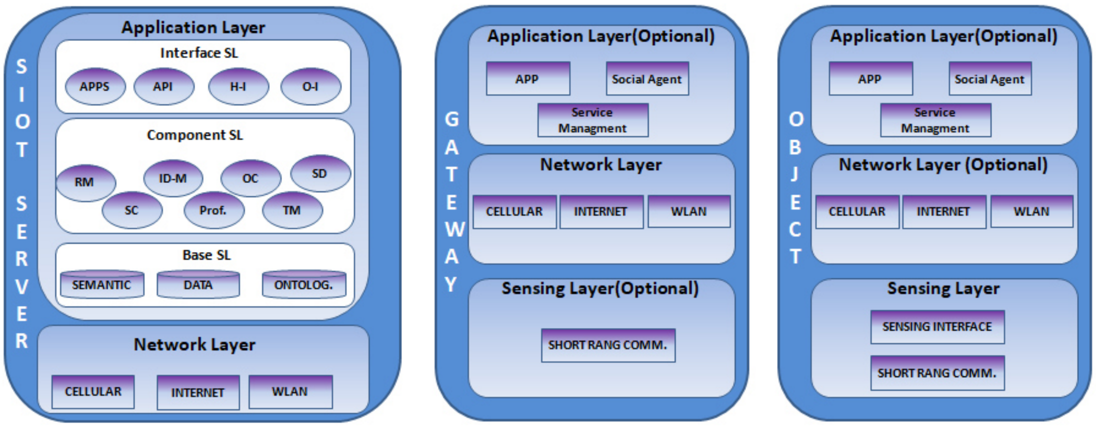


Figure 11: SIoT Three-layer Architecture.

Figure 11: SIoT Three-layer Architecture. shows the SIoT three-layered architecture, whose fundamental characteristic elements are: the SIoT Server, the Gateway, and the Object.

The SIoT Server encompasses only the Network and the Application Layers. Specifically The Application Layer consists of three sublayers:

- The Base Sublayer wich includes the database, to store and manage all the data and the relevant descriptors which record the social member profiles and their relationships. Humans' data (e.g. the owner's data) too are stored and managed inthere. In different database are stored the ontologies which are used to represent a semantic view of the social activities.
- The Component Sub-layer, incorporates all the tools implementing the core functionality of the SIoT system, such as the ID management, which assigns an ID that univocally identifies all the possible categories of objects. The profiling tool, which configures the information about the objects. While the owner control (OC) defines all the activities that can be performed by the object, the information that can be shared, as well as the type of relationships that can be established. The relationship management (RM) allows objects to begin, update, and conclude their relationships with other objects. The service discovery (SD) finds which objects can provide the required

service, whereas the service composition (SC) component enables the interaction between objects. In conclusion the trustworthiness management (TM) component understands how the information provided by the other members shall be processed.

- The third sub-layer, the Interface Sub-layer, is where the third part interfaces to objects, humans, and services are gathered. It may be mapped onto a single site or in the cloud.

In the Gateway and Objects systems, the combination of layers is not determined, instead, but rather it may vary depending on the device characteristics.

In order to explain and consider all the possible scenarios tree examples have been made depending on the grade of "smartness" of the object.

- A dummy Object (e.g., a RFID tag) is equipped with a functionality of the lowest layer, is only able to transmit simple signals to another element (e.g., the Gateway). The Gateway, in this scenario, is equipped with all the functionalities of the three layers.
- In another scenario, a device (e.g., a video camera) is able to sense the physical world information and to send the related data over an IP network. In this case the object is equipped with the functionality of the Network Layer. Consequently, there is no need for a Gateway with Application Layer functionality. An Application Layer in a server with the gateway application layer functionality would be enough.
- A smart object (e.g., a smartphone, which has enough computational power) may implement the functionality of all the three layers so that the Gateway is not needed, but for some communication facilities targeted to maintain the Internet connectivity of the object.

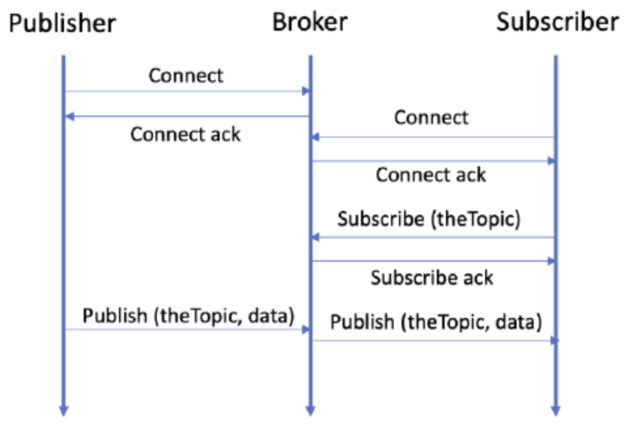
Whatever is the scenario, the Application Layer encompasses the SIoT applications, as well as the social agent and the service management agent:

- The social agent is responsible for the communication with the SIoT servers, to update its profile and friendships, as well as to discover and request services from the social network.
- The service management agent is responsible for the interfaces with the humans that can control the behavior of the object when communicating within their social network.

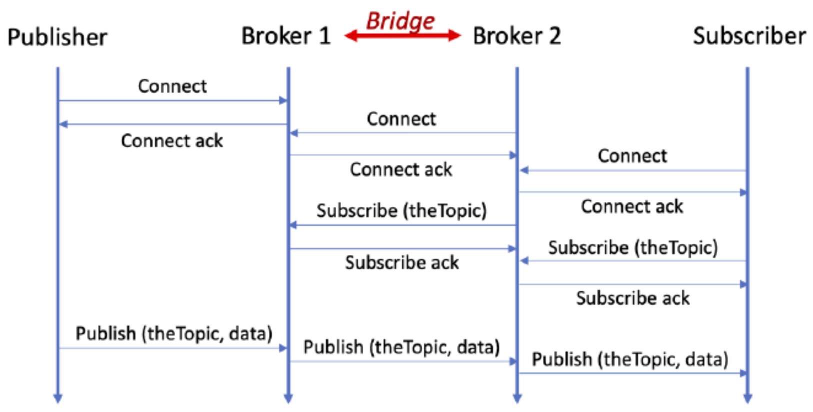
1.3 Message Queuing Telemetry Transport

The Message Queuing Telemetry Transport (MQTT) is a machine-to-machine (M2M)/Internet of Things (IoT) connectivity protocol, standardized by OASIS [20].

In Figure (1a) we represent the main interactions between MQTT entities. We can distinguish publishers, brokers, and subscribers. The broker, obviously, plays a central role as it maps the interest of subscribers for a certain topic with the messages published under that topic. A topic is a UTF-8 string that the broker uses to filter messages for each connected client and it consists of one or more topic levels, separated by forward slashes, similarly to folders/files in a file system. Therefore, as we show in Figure (1a) after establishing a connection with the broker, clients can subscribe to and publish under a certain topic, myTopic; forwarding relevant published messages to subscribers interested to the topic is, therefore, the broker's responsibility. In Figure (1b), instead we sketch the operations in the case MQTT bridging has been activated. More specifically, a client (the Subscriber) can subscribe to a certain topic, say myTopic, on Broker 2. If Broker 2 recognizes that such topic is handled by another broker, say Broker 1, then it can subscribe to such topic on Broker 1. As a result, when a client (the Publisher) publishes a message under topic myTopic on the MQTT Broker 1, such message will be forwarded to MQTT Broker 2 which, in turn, will forward it to the Subscriber. By comparing Figures 1a and 1b, it is clear that in the bridged case one extra communication hop is necessary and more messages are exchanged. This would result in increased delay and resource consumption.



(a) Standard (no bridge).



(b) Bridged.

Figure 12: MQTT interactions

Chapter 2: Sociocast

2.1 Statement of the problem

The Internet is experiencing a rapid transformation pushed by the growing need to overcome its intrinsic limitations and ossification, which challenge network practitioners and researchers. The pressing need to come to the definition of a new Internet of the future is also motivated by the multitude of Internet of Things (IoT) applications that are recently emerging in various vertical markets [21]. Such applications are increasingly characterized by group-based (i.e., one-to-many, many-to-many) communications established among large sets of devices in need for simultaneously exchanging data, e.g., in case of sensors' software updates, service advertisement, device configurations.

In human-centric communications, frequent instant messaging occurs within communities of users sharing similar interests and people massively interact with their friends, and friends of their friends, over social networks. Similarly, groups of IoT devices are likely to interact with each other, especially if they are located in the same place (e.g., sensors/actuators in the same building), are owned by the same user (e.g., consumer devices and home appliances), share similar profiles (e.g., the same brand and type), frequently meet each other (e.g., vehicles on a given road segment).

Support of interactions among devices raises outstanding challenges for network operators. First, IoT applications require the dynamic and flexible management of group-based interactions, whose scope is decided according to a given topic and to the ties existing between the endpoints involved (e.g., co-locality, similarity of devices, etc.).

Second, the communication endpoints should have the power to control data exchanges. Indeed, a control of the enabled data receivers is strongly desired by the source device, due to the potentially confidential nature of the data exchanged.

Moreover, the massive presence of group-based communications established by billions of IoT devices, expected to increase even at a higher pace in the near future, can cause network congestion and waste device and network resources, unless proper countermeasures are taken.

A solution is required to allow nodes to flexibly specify how to prioritize (filter) the nodes from which they want (or they do not want) to receive data, and the network to react accordingly, so as to prevent the threats of Denial of Service (DoS) attacks.

Conventional multicast-based approaches [22], being mainly designed to simultaneously transmit data from one or multiple senders to a group of (unknown) receivers, fail in natively achieving such objectives and in ensuring the required flexibility in group establishment and management. Clumsy patches to existing multicast solutions may further complicate their design and hinder their (already limited) deployment.

This is the reason why in [23] authors argue in favor of a novel and future-proof comprehensive solution, named Sociocast, encompassing both a communication method and a data delivery scheme, going well beyond Internet Protocol (IP)-based multicast. Sociocast is theorized as a means for identifying, in a flexible manner, the intended endpoints (senders/receivers) of data exchange sessions. Groups are dynamically created according to the mutual position of endpoints in a social network of devices and the type of relationships among them, by means of properly defined filtering rules and policies.

This work treasures the theoretical analysis in the cited vision paper and makes a significant step forward both in terms of practical design and experimental evaluation. The possibility of implementing the conceived Sociocast primitive as a network-layer solution in IoT domains, wherein switches and routers are responsible for the efficient delivery of packets issued by IoT devices. In particular, the reference network infrastructure is deployed according to the Software-Defined Networking (SDN) technology [24].

SDN has been introduced to address a typical issue in traditional IP networks, namely, the lack of programmability in network management and configuration. Due to its peculiarities, it can play a crucial role to bring the social dimension into the group data delivery procedures enforced at the network layer.

The main contributions of Sociocast can be summarized as follows.

- The design of an architectural framework encompassing all the entities and functionalities supporting Sociocast, according to a software-defined network approach.
- The definition of the main procedures for the creation of the Sociocast packets, their forwarding and filtering, and the subscription of devices to Sociocast groups.
- The performance assessment through the widely known Mininet network emulator [25], when dealing with push-based data dissemination and deploying the Sociocast network application into the ONOS SDN controller [26]. The impact of different end-point distribution patterns and different involved social relationships on the performance is evaluated by comparing our proposal to an alternative approach where the groups are created at the application layer. Results show that the Sociocast approach allows for a reduction of signaling and data packets by a factor of 10 and 5, respectively, in the scenario where the number of recipients is high and are close each other.

A large number of different applications rely on one-to-many and many-to-many data traffic exchange ranging from live video streaming, audio/video conferencing [27] and multiplayer games [28] to communications among groups of servers within data centers [29] and wide-area control in smart grids [30]. Multicasting functionality is typically leveraged in such contexts, which can be performed either at the network (IP) layer or at the application layer [22], [31] and also with the support of SDN [32], [33].

IP-based multicasting. Traditional multicast routing and management protocols, such as Protocol-Independent Multicast (PIM) [14] and Internet

Group Management Protocol (IGMP) [35], effectively establish and maintain multicast communication paths between sources and receivers to enable the forwarding of packets to a multicast group. Each group is assigned a unique class D IP address. A host can send data to a multicast group by using the local network multicast capability to transmit the packet. A multicast router, upon reception of a packet, looks up its routing table and forwards the packet to the appropriate outgoing interface. Group membership is managed at the network level through routers: when a host decides to join/leave a particular multicast group, it sends the request to the local multicast router, through IGMP [35].

IP multicast allows data to be distributed in such a way that the least amount of replicas of the same packet is placed into the network.

In its recent version, v3, IGMP allows to specify the set of senders from which a node wants to receive packets, in agreement with the Source-Specific Multicast (SSM) protocol [36]. In other words, the only packets that are delivered to a receiver are those originating from a specific source address requested by the same receiver. Hence, SSM is particularly well-suited to dissemination-style applications with one or more senders whose identities are known before the application begins.

Non-IP multicasting. The design of multicast solutions has also been investigated beyond IP. In application-layer solutions, group membership, multicast delivery structure construction, and data forwarding are exclusively controlled by participating end-hosts, thus, the support of network nodes is not needed [31].

In the clean-slate future Internet MobilityFirst architecture [37], a context aware delivery primitive is proposed, which generalizes multicast to groups established on the basis of attribute-based descriptors. The name service, in charge of resolution procedures between Global Unique Identifier (GUID) and network addresses, maintains a membership set that consisting of all GUIDs of devices that subscribed to the multicast group. The sender is responsible for   sending data to each of the returned addresses.

SDN-based multicasting. SDN can simplify multicast traffic engineering thanks to the centralized nature of the network control plane. Current multicast solutions employ a shortest-path tree to connect the source to the receivers which is built according to local information. Traffic engineering is difficult to be supported in a shortest-path tree. By utilizing the global view of the SDN controller, in [38] all the possible routes between the sources and each host of the multicast group are calculated in advance. In contrast with IP multicast, there are no de facto standards for SDN multicast routing. Different approaches targeting different optimization objectives can be targeted in a flexible manner and it is unlikely that a given approach is going to be dominant. SDN multicast is enabled by writing an application for the SDN controller that optimizes the traffic flows to meet the particular needs of the end-user [32]. The SDN controller can build the multicast tree to meet link constraints (bandwidth consumption) or path constraints (end-to-end delay) [33]. Hence, it is a valuable solution when Quality of Service (QoS) requirements need to be ensured to a multicast flow, e.g., in case of a multi-party video-conferencing service [39].

2.1.1 Weaknesses and open issues

The use of the traditional IP multicast is prone to multiple issues:

- Without the explicit join to the multicast group, a router will not forward multicast IP packets destined to end-hosts. This process implies the distribution of the consent to join the multicast group among devices, increasing the signaling overhead.
- There is no way for the sender to control who subscribes to a multicast group.
- It prevents the creation of discrimination policies based on the destinations of the information within the same multicast group. Therefore, when a limitation to the distribution of packets to some entities of the same multicast group is needed, another multicast group must be created, with a consequent increase in the number of signaling packets in the network.

- All routers must be replaced with multicast-enabled routers, which could be expensive and hardly viable for the network operator, raising interoperability issues.

The poor flexibility of the IP-based multicast discourages to pursue such an approach for the wide variety of sender-initiated dynamic group-based communications, as demanded by future IoT deployments.

On the other hand, application-layer solutions have the drawback of a definitely worse performance in terms of end-to-end latency and efficiency compared to IP multicast. This is because end-hosts have little or no knowledge of the underlying network topology.

Thanks to its programmability and global knowledge of the topology, SDN can make the creation of the multicast tree more efficient, thus improving forwarding procedures. However, to the best of our knowledge, the flexibility of SDN has not been investigated to manage dynamic group formation.

These issues have motivated the theorizing of a new communication method and data delivery scheme [23], able to better fit the nature of upcoming group-based communications: Sociocast.

This is introduced as a novel and flexible solution that allows group-based communications in the IoT enhanced with the notion of social ties. It inherits the strengths of IP multicast, in that it lets network nodes disseminate packets in an efficient manner: Sociocast packets are assigned an IP address to facilitate their forwarding. In addition, the proposal in [23] enables a mutual control of the end-points: not only the receiver can filter different senders, as in SSM, but also the sender can (implicitly) decide which node should belong to the set of intended receivers, by specifying the features (in terms of social relationship) of such receivers. The above mentioned capabilities are disruptive when compared to conventional IP-based multicast. Sociocast relieves the burden of group management from network nodes and of explicit join procedures from devices.

2.1.2 Advantages of a “social-oriented" approach

The use of social links to support network functionality is not new in the telecommunications landscape.

Several routing protocols in wireless ad hoc [40], mobile opportunistic and delay-tolerant networks [41, 42, 43, 44], have been designed to build upon the key concepts of social network analysis, i.e., small world phenomenon [45] and centrality. The former one, a.k.a. community, captures the fact that actors within a social network are separated from each other by an average number of fairly limited hops; while the latter one shows that some nodes in a community are the common acquaintances of other nodes.

In the aforementioned works, the knowledge of social characteristics (e.g., node centrality, in-betweenness) is used to make better forwarding decisions and to assist the relay selection when delivering data to the intended destination(s).

Many of the studied approaches involved unicast or multicast communications [46, 47, 48]. The issue of data broadcasting in a Mobile Social Network, where mobile social users physically interact with each other, is analyzed in [49].

The objective of this work is to exploit similar concepts but under a different perspective. We do not aim to improve forwarding decisions by leveraging social network properties, but rather to better disseminate data at the network layer within dynamically created groups of socially connected devices.

The proposal has the potential of a real game changer in the view of the creation of the future Internet of Things, by providing superior advantages compared to what has been done so far in literature.

In fact, social bonds not only ensure minimum separation distances between actors, which are crucial for an efficient and fast data propagation, but may also enable data exchange within trusted groups and the creation of groups that include actors belonging to different communities. In Sociocast this translates into the possibility of an efficient and flexible group end-points discovery, an intrinsic possibility of implementing policies for creating trusted groups of end-points directly at the network level, and the ability to effectively and simply deal with the problem of interoperability among different IoT platforms.

Obviously, in order to do this we need to start from a paradigm that can provide for the establishment of pseudo-social ties between devices (to operate at the network layer). This is already available in solutions of \social networks of IoT devices", such as the Social Internet of Things (SIoT) [19] for example.

However, they need to be moved from the application layer, wherein they have

been initially conceived, down to the control plane of the network layer. In so doing, group establishment and data exchange among members of such groups

can be managed in a tighter way, with inherent flexibility and efficiency in terms of network resource usage.

2.2 Sociocast

2.2.1 Objectives and design principles

In this Section is described how it’s possible to achieve a real implementation of the Sociocast concept by relying on the capabilities of the Software Defined Networking paradigm. The resulting solution is an enabler for group communications based on social notions at the network layer.

Social ties among devices. Devices are likely to interact with other devices having similar profiles and habits, e.g., those located in the same place, owned by the same user, produced in the same company branch.

Such ties are well captured by the SIoT paradigm in [19], where a few basic types of social relationships, defined according to user-defined policies, are introduced: co-ownership object relationship (OOR), created between devices that belong to the same owner; co-location object relationship (CLOR), created between stationary devices located in the same place; parental object relationship (POR), created between devices of the same model, producer and production batch; co-work object relationship (CWOR), created between moving devices that meet each other at the owners' workplace; social object relationship (SOR), created as a consequence of frequent interactions between moving devices. The framework is quite flexible and other types of relationships can be easily added on a per use-case basis.

Applications requiring data dissemination to a social group of devices are, for instance, software updates: a given software patch needs to be safely delivered to all the devices or sensors of the same brand, model, batch. For this, POR relationships should be exploited. Similarly, some data need to reach all other devices belonging to the same owner in case of personal bubbles: the OOR relationship is appropriate in this scenario. Business services may be advertised to all devices that either are currently in the same area (CLOR) or often visited the same place (SOR).

Targeted data delivery schemes. Sociocast aims to enable:

- a given sender to disseminate data in a push-like manner to specific nodes, which are friends over a social network of devices, according to properly defined filters and policies (i.e., the social relationship type);

- a node to subscribe to specific social-based topics (i.e., to receive data from friends of a given type);
- a node to prioritize (and not to receive) data from particular senders, e.g.: to enforce QoS; to identify the more suited and trusted communication endpoints for security reasons; to save resources.

Deployment options. To target the aforementioned objectives, the envisioned framework has (i) to enable nodes to indicate in an agile manner the features of the end-points of data flows (i.e., the set of intended recipients and/or the authorized senders) based on the distance in a social network of devices, (ii) to properly and dynamically identify them, (iii) to forward data packets accordingly.

A straightforward approach to accomplish the first two features could be one that relies on an application-layer solution. For instance, the intended set of receivers can be specified by a given sender at a high-level, e.g., through meta-data. Then, the resulting request can be sent to a purpose-built proxy which is in charge of mapping such data onto IP addresses of the receivers, similarly to [37]. Despite the virtue of simplicity, such an approach has the drawback of poor performance in terms of efficiency in the usage of network resources, since data forwarding to each intended destination is performed at the underlying network layer in a myopic manner.

Thus, our interest is on a network-layer approach, according to which the features of the intended set of receivers of a given data packet (or of a sender of unwanted data packets) can be translated into a network-layer IP address, hence treated (forwarded/dropped) by network nodes, accordingly. The approach proposed is inspired by the traditional IP-based multicast, with which it shares a few aspects, such as the routing of packets with a multicast address (a Sociocast address, in our case). However, multicast lacks the flexibility necessary to implement the aforementioned critical functionalities for the future Internet of billions of devices, while it meets the requirements of the end-users and those of the network operators.

By overstepping the agnosticism about Sociocast traffic at the network layer, the following advantages are expected:

- data forwarding can occur in an efficient manner, e.g., by reducing the number of duplicated packets, and saving bandwidth accordingly;
- filtering procedures can be enforced in-network, as requested by potential data recipients, to limit the massive amount of generated traffic;
- network operators can benefit from traffic reduction, which is particularly crucial for their infrastructures expected to be largely overwhelmed in the near future.

Programmable packet treatment. Recent advancements in networking technologies make the deployment of Sociocast at the network layer even more viable. We identify SDN as the key enabler for Sociocast. Thanks to its programmability, which reduces the complexity of network elements, SDN can inject forwarding/dropping rules and properly manipulate headers of packets to make more efficient their forwarding.

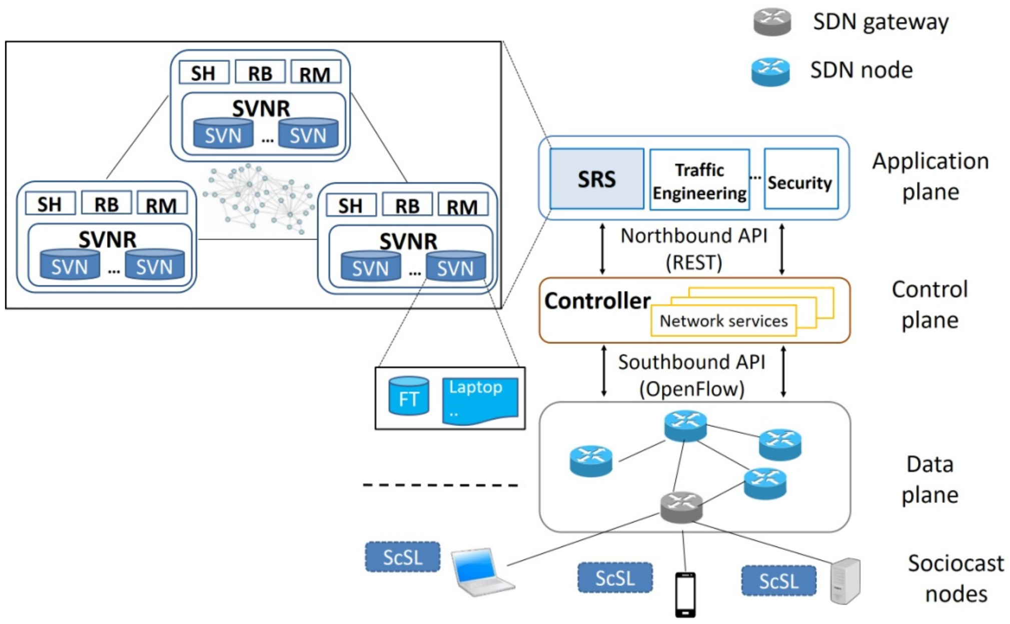


Figure 13: Sociocast architectural framework

Such policies can be defined in a network application, with no need to modify the data plane of the underlying network infrastructure.

2.2.2 The architectural framework

The main entities of the envisioned framework are: the Sociocast nodes, the SDN network (encompassing both switches and controller), augmented with the notion of Sociocast, and the Sociocast Relationship

Service, as shown in **Errore. L'origine riferimento non è stata trovata.**and detailed in the following.

The Sociocast nodes The Sociocast nodes are the endpoints of a Sociocast communication. They are legacy IoT devices (e.g., smartphones, sensors) augmented with the Sociocast Support Layer (ScSL) running on top of the transport layer, through which they are enabled to create, send and/or receive Sociocast packets. The ScSL exposes the Sociocast Application Programming Interfaces (APIs) to the applications that want to use the Sociocast communication configuration for data delivery. It is through this layer that Sociocast packets are created and received by the end-devices.

The SDN network The SDN network is composed of three different planes, according to the legacy deployment. The data plane encompasses the SDN switches, which are SDN-enabled network nodes connected to each other and interacting with the SDN controller. Among them, the SDN gateways are the ingress/egress nodes of the SDN network. SDN nodes interact with the SDN controller through the OpenFlow(OF) southbound interface.

The control plane includes the SDN controller, which oversees the SDN nodes, according to specific orchestration policies defined at the application plane, and tracks the graph of the network topology in the Network Information Base (NIB). According to information in the NIB and policies defined by network applications, it injects rules in the so-called flow tables of SDN nodes to enable the forwarding of Sociocast packets through OF messages [50].

The Sociocast Relationship Service The Sociocast Relationship Service (SRS) is implemented at the application plane, next to conventional SDN applications, and it provides the following main functionalities:

1. establishing social relationships among nodes. Without loss of generality, we inherit concepts and methodologies regarding the policies for the establishment of the social links between nodes from the well-accepted SIoT paradigm [19];
2. keeping track of the established social relationships;

3. providing interfaces towards the SDN network and navigating the social network so to identify the nodes that belong to the set of the potential recipients/authorized senders of a Sociocast packet.

Herein, a major element is the Social Virtual Node (SVN), which represents the digital counterpart of a physical device. It stores some meta-data providing information about the nature of the device and a list of friends, which is organized in a table named Friends Table. For each friend in the table, the SVN records the type(s) of friendship(s), defined according to the SIoT paradigm and the trust level associated with each friend.

The Social Virtual Node Repository (SVNR) stores all SVNs associated to the physical devices in a given area. Indeed, one SVNR is responsible for providing the objects in a given area with the described services; more SVNRs are then interconnected in a distributed system. The following modules are associated to the SVNR.

- *The Relationship Manager (RM)* is responsible for the relationships' lifecycle management, i.e., detecting, creating, updating and deleting relationships[3].
- *The Relationship Browser (RB)* navigates the Friends Table to find potential recipients of a Sociocast packet, according to their position in the social network. Policies for the social network navigation are discussed in [51].
- *The Sociocast Handler (SH),* whenever queried by the SDN controller, provides for the members of a Sociocast group, after querying the RB module, through a Representational State Transfer (REST) API. SVNRs, along with relevant functionalities (i.e., RM, RB and SH), can be deployed as a peer-to-peer system, for instance building upon the one described in [51].

[3] For a detailed description of relationships management, the reader is referred to [51].

The design choice is aimed at providing an implementation of SDN-based group communications based on a de-facto global IoT resource directory, which is distributed and without a single player in control of the system. Digital representations of physical IoT devices will run in distributed servers and can create autonomously social-like relationships with each others. Based on such a distributed resource directory, interactions (both point-to-point and point-to-multipoint) between IoT resources belonging to different platforms can be straightforwardly enabled. Each SVNR (or group of SVNRs) could, in fact, contain the images of the devices belonging to a given platform, it can be owned and maintained by the owner of the platform (or even the owner of the group of IoT devices), and interacts in a peer-to-peer fashion with other SVNRs constituting the SRS.

2.2.3 Sociocast in action

In the following, are described the main steps for the creation of a Sociocast packet. Then, it's illustrate the Sociocast data delivery, according to a push-based dissemination, publish/subscribe procedures to Sociocast groups, as well as filtering according to Sociocast rules.

2.2.3.1 Creating a Sociocast packet

A Sociocast packet is created whenever a device needs the services offered by the Sociocast framework, which are intended to: (i) disseminate data in a push-like manner; (ii) indicate the subscription to a Sociocast group; (iii) or to filter/prioritize data from particular senders. Whenever a packet is created, it has to indicate which one of these three types of services is requested. The above are the types of Sociocast services. The above-mentioned types of Sociocast services are those supported in the current implementation, but the set of Sociocast services can be easily extended in the future.

Let us consider a device, say A, which creates a packet with data to be sent to a Sociocast group. The application in A makes a request to the ScSL via the available APIs, providing the following information: (i) the type of requested Sociocast service; (ii) the social relationship (e.g., OOR, CLOR) according to which the Sociocast group has to be formed; (iii) the social distance (number of hops over the social network), which represents the scope of the Sociocast group.

The ScSL reacts to the incoming request by creating an IP packet with the following header fields:

- Source IP address: the IP address of the source device.
- Destination IP address: a fixed IP address, identified in the following as IPSC, assigned to Sociocast that allows SDN gateways to identify Sociocast packets.
- Sociocast Tag: a 2-bytes field that is carried inside the transport-layer destination port and is used to uniquely identify the type of social relationship and other appropriate filters (e.g., number of hops, possible application of Sociocast, etc.). The encoding is as follows:

  -METADATA: device metadata available for future applications.

  -RELATIONFILTER: type of relationship (e.g., OOR, SOR, CLOR, etc.).

  -FEATUREGROUP: type of Sociocast services needed by the application (e.g., GroupCreation, SourceFiltering, Pub/Sub).

  -RADIUS: maximum distance, in number of hops, from the source.

  Figure 14: Examples of Sociocast Tag configuration. shows some examples of Sociocast Tag configuration.

Being Sociocast packets identified through conventional layers 3 and 4 header fields, legacy matching rules can be applied, with no need to resort to OF experimenter fields [52]. Such design choices would facilitate the deployment of Sociocast, which candidates itself as a short-term solution to be exploited by network operators.

For the sake of simplicity, the encoding described above refers to the case the IPv4 is used. Similar considerations hold for IPv6 packets, for which matching fields can be handled by OF since version 1.2 [52].

For those constrained IoT devices belonging to Low power and Lossy Net- works (LLNs), 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks) header compression methods can be used [53] over the link interconnecting the devices to the SDN gateway. For the IPv6 headers, compression methods may also affeect source and destination addresses, and they vary according to the fact that the source is communicating with nodes either within or outside the WPAN. In the latter case, a 50 percent compression ratio can be still achieved by letting the full destination address, carrying the Sociocast address, be transmitted.

TCP header compression for IoT scenarios [54] is still an open issue at the standardization level [55], not part of RFC 6282 [53]. The compression foresees to avoid sending the port numbers in each packet, which however does not affect the Sociocast communications as the port number with the Sociocast TAG is reconstructed at the gateway. Indeed, decompression occurs at the SDN gateway letting Sociocast packets travel with conventional IP header fields in the SDN network. Similar operations are performed at the SDN gateways the destinations are attached to, if the latter ones belong to a WPAN.

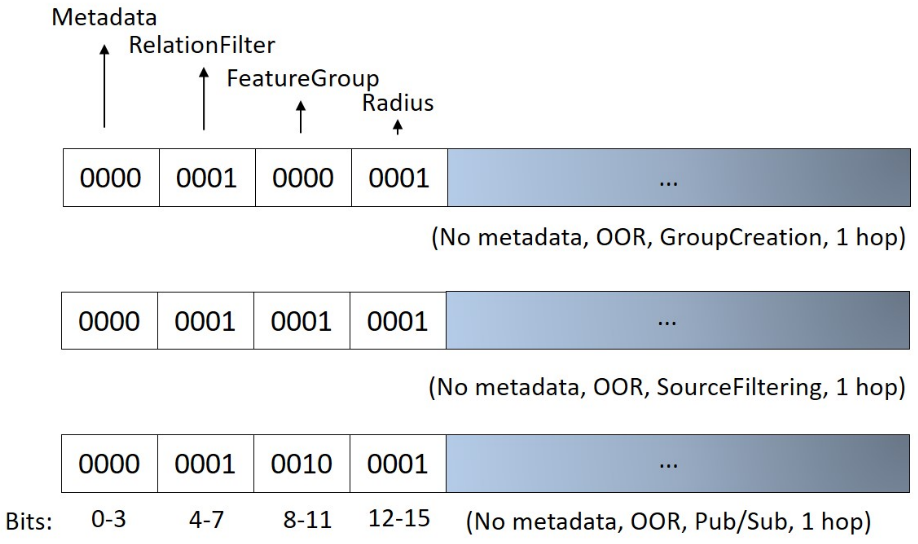


Figure 14: Examples of Sociocast Tag configuration.

2.2.3.2 Push-based data dissemination

Once the Sociocast packet is created with data to be disseminated, it is sent by the source device and treated in the network through the following steps.

1. The Sociocast packet reaches the SDN gateway, which the source device is connected to. Since, initially, a forwarding rule is not set in

the flow table of the SDN gateway, the GoToController rule applies for it. Hence, a OF Packet In message is issued to be transmitted to the SDN controller.

2. Upon reading the header of the Sociocast packet[4], the SDN controller realizes that a Sociocast group must be created (Feature field set to GroupCreation). Thus, it issues a request to the SRS, to retrieve the set of the devices, which are meant to act as recipients of the Sociocast packet.
3. The SH triggers the browsing of the social network, as specified before, and returns to the SDN controller the addresses of the set of devices of the Sociocast group.
4. The SDN controller retrieves from the NIB the SDN nodes in the shortest paths towards the intended receivers of the Sociocast group. Then, it builds the routing paths by ensuring that SDN nodes belonging to the path towards multiple receivers receive a single rule and forward the Sociocast packet only once. Hence, it injects forwarding rules in the flow table of the involved SDN nodes accordingly, by sending OF Flow Mod messages. In particular, the SDN gateways which the Sociocast destinations are attached to, will be instructed by the SDN controller with a rule that: (i) matches the Sociocast-related header fields that identifing the Sociocast communication and (ii) foresees to forward the packet to the correct physical port after changing the destination Sociocast IP address with the IP destination (unicast) address as action. This is to ensure that all devices belonging to the Sociocast group correctly receive the Sociocast packet.

   Other SDN nodes, instead, are instructed to forward the Sociocast packet to the physical correct ports by matching the Sociocast fields values.

[4] The entire Sociocast packet is sent by the SDN gateway, hence a Packet Out is transmitted by the controller, back to the SDN gateway [52].

Once the Sociocast group is created, subsequent Sociocast packets transmitted by the source device may be handled by the SDN gateway with no need to contact the SDN controller, but rather the may be forwarded according to rules already available in the flow table. According to the legacy SDN implementation, a timeout is applied to rules injected by the controller into SDN nodes, so as to prevent a rule to stay in the table for too long and to unnecessarily occupy space in the flow table [52]. Within our framework, such a timeout can be set to reflect the lifetime of and the frequency of interactions within the Sociocast group, as well as the mobility patterns of nodes.

2.2.3.3 Publish/subscribe

Sociocast can be exploited to support publish/subscribe interaction model as well. In fact, a device can subscribe to receive packets published by devices identified by their position in the social network. For example, assume that device B wants to subscribe to receive packets generated by its friends of type OOR. If this is the case, it will generate a Sociocast packet with the FeatureGroup field set to Pub/Sub and the RelationFilter field identifying an OOR.

Such an information will reach the SDN controller which will perform the following operations:

1. It sends a query to the SH and receives the identities of the devices with position in the social network consistent with the request by device B.
2. It adds this information in a pending interest table which tracks all subscriptions received by devices. Whenever a device begins to disseminate data, the SDN controller will check whether there are devices that have subscribed to its updates (e.g., B).
3. If this is the case, the SDN controller will instruct the SDN nodes in the path to B to forward the data packets to it.

2.2.3.4 Source filtering

Sociocast allows a device to select those that are entitled to send packets to it, on the basis of their position in the social network. Such feature can be used both in a proactive and a reactive way. More specifically,

- *Proactive:* a device might decide to receive packets by its friends only, for security reasons or to save energy, computational and communication resources.
- *Reactive:* the computational or communication load for a device may become too high, e.g., because of a DoS attack. If this is the case, the device might decide to accept packets by a subset of devices, based on their position in the social network. In this way Sociocast can be exploited to realize a firewall whose policies change depending on the current load.

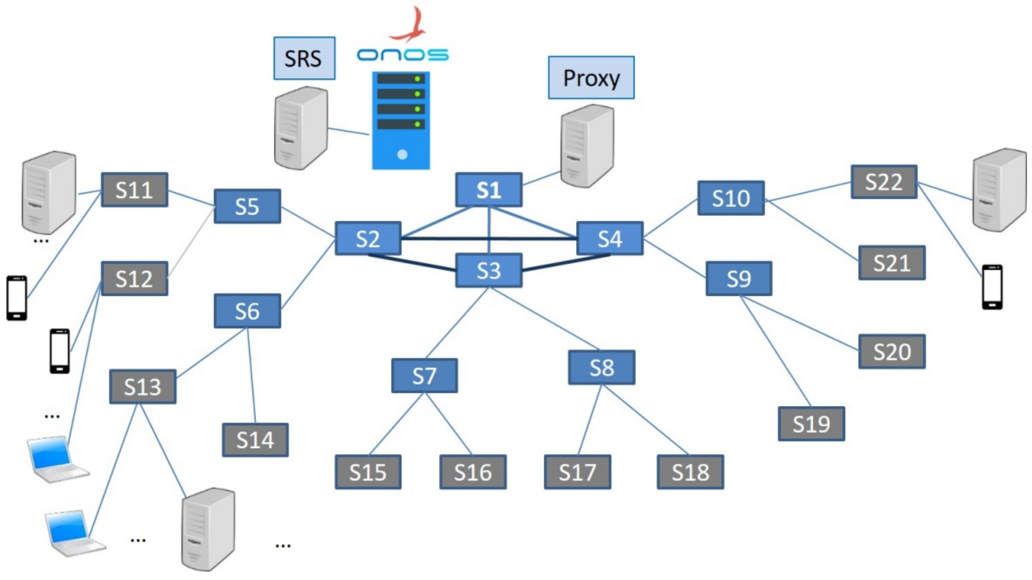


Figure 15: Reference topology.

A device, say C, wishing not to receive packets from nodes with certain social properties sends a Sociocast packet by specifying in the FeatureGroup field SourceFiltering. Once the packet reaches the SDN controller, the latter one will query the SH, which will reply with the list of authorized IP addresses.

Accordingly, the SDN controller will insert entries in the flow table of the SDN gateway which C is attached to, whit the aim of specifying, on the one hand, the forwarding rule for packets destined to it sent from authorized senders and, on the other, the dropping rule for those which are not allowed.

2.3 Experimental setup

In this paragraph it's described the environment for the performance evaluation. More specifically, Section 2.3.1, and 2.3.2 described the tools utilized for the performance evaluation and the reference topology, while in Section 2.3.3 are discussed the scenarios.

The benchmark utilized for comparison purposes is presented in Section 2.3.4, whereas the considered performance metrics are identified in Section 2.3.5.

2.3.1 Tools description

The SDN controller ONOS [26] and the network emulator Mininet [25] are the main tools used for the performance evaluation. Both of them need to be described in more details to better understand how the experimental playground has been created.

2.3.1.1 ONOS

At the beginning of SDN, there was only a SDN controller, NOX [58], but since then, the number of SDN controllers has grown significantly. Indeed, as stated in Chapter 1, whitin this plethora of SDN controllers some are proprietary, while many more are open-source and implemented in different programming languages.

In particular the SDN controller ONOS, which is a Java-based controllers, uses OSGi containers for loading bundles at runtime, allowing a very flexible approach to adding functionality and also, since Java is a well-known and widely used programming language, the development resources are abundant, with good supporting documentation and libraries available.
As Figure 16 shows, in the ONOS controller architecture can be found three different layers:

- *Northbound APIs and protocols.* An exclusive characterization of ONOS is the intent framework, which allows a control application to request a high-level service without having to know the implementation details. State information is provided to control applications across the northbound API, either synchronously, via query, or asynchronously, via listener callbacks.

- *Distributed core.* The state of the whole SDN network (links, hosts, and devices state) is maintained in the distributed core. ONOS is deployed as a service on a set of interconnected servers. The ONOS core provides the mechanisms for service duplication and coordination among different instances, providing the applications, in the upper layer, and the network devices, in the downer layer, with the abstraction of logically centralized core services.
- *Southbound APIs and protocols.* The southbound layer hides the diversity of the network, allowing the distributed core to be both device and protocol agnostic.

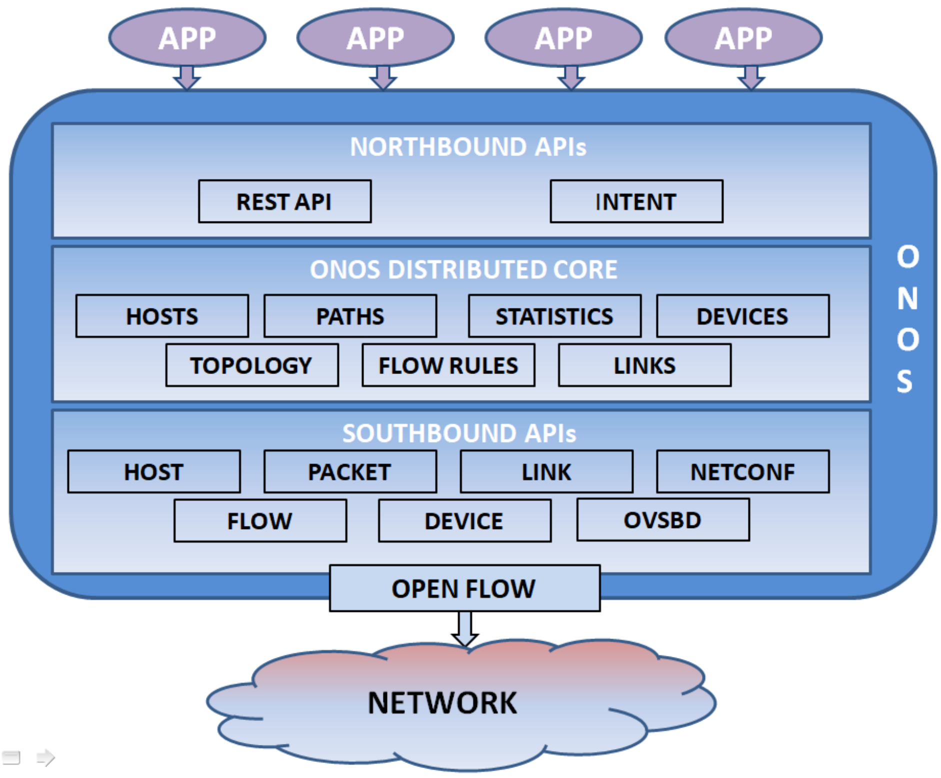


Figure 16: ONOS Architecture.

2.3.1.2 Mininet

2.3.1.2.1 What is Mininet?

"Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking." This definition is taken from Mininet official webpage [56], and highlights that Mininet is a very powerful tool when working with SDN; indeed, it provides a simple network testbed for developing OpenFlow applications.

The emulated hosts in Mininet behave just like they were "real", which means that it's possible to either run any program installed in the undergoing Linux Operating System or a specific code for an application or, again, for example, to login with SSH.

This concept is extended to all the network elements: not just hosts, but also switches, links and controllers. They are "real", even though they are created with software, instead of being an hardware part. This allows for the emulation of networks that are a copy of real ones and vice versa, for the creation of a Mininet network to test a future real one and use the same code and applications for both of them. Mininet works via Linux command line and Python API.

2.3.1.2.2 How Mininet Works

Mininet uses some Linux features, such as process abstraction, which allows for the virtualization of the computing resources and makes the system look like a set of "containers". Each container has a fixed share of the processing power and a virtual link that helps the creation of links with realistic speed and delay. Using this process based virtualization, Mininet is able to run many hosts and switches (up to 4096 on a single kernel!). Let's now analyze each emulated network element:

- **Emulated Hosts:** A host emulated in Mininet is a group of user-level processes moved into the network namespace, a virtualization feature that provides individual processes with separate network interfaces, routing tables, and ARP tables.

- **Emulated Links:** The data rate of each link is guaranteed by the Linux Traffic Control which, through a series of schedulers, allows you to create traffic flows with the specified characteristics. Note that each emulated host has its own virtual Ethernet interfaces and each pair of virtual Ethernet behaves like a cable connecting the two interfaces.
- **Emulated Switches:** Mininet uses Open vSwitch in kernel mode to move packets between interfaces.

2.3.2 Tools usage and reference topology

The focus of the performance evaluation is to assess Sociocast in the case of push-based data dissemination towards a group of devices.

To this purpose, we built an emulation playground. In particular, the Mininet network emulator [25] has been used, which allows fast prototyping and experimental evaluation of OF-enabled networked systems. The experimental setting consists in the network topology depicted in Figure 15: Reference topology.. A full-mesh interconnects the core SDN nodes, which are the roots of a three-layers fat-tree topology. Up to 21 devices are attached to each SDN gateway (not all the devices are shown in the Figure). ONOS has been considered as a reference SDN controller in the context of this work, due to its scalability properties and its highly modular architecture [26].

The ONOS controller interacts with an external SRS, which establishes social relationships among emulated devices, and manages them.

The ONOS controller and the Mininet network emulator are both running on the same virtual machine, while the SRS runs in a different one. Both these
virtual machines are located in a physical server with an Intel Xeon(R) CPU E5-2630C v3 1.80 GHz x32 processor and 377,8 GiB of memory.

2.3.3 Social relationships settings and traffic patterns

The performance of the proposed solution has been evaluated with a set of representative IoT test configurations properly designed to take into account

different numbers and distributions of nodes in the emulated topology, different physical distances between sources and destinations, and different types of service. This is aimed at making the obtained results as generalizable as possible and having a clear idea of the potential and limits of Sociocast in multiple scenarios. Each of the test configurations has been mapped onto a use case characterized by the exploitation of a particular type of social relationship between the devices involved. In this way, helpful guidelines can be provided about the suitability of the proposed solution in the context of different application scenarios and, at the same time, of the effectiveness of communications based on each of the possible social-like relationships established among IoT devices. Details are given in the following. Table 2: Summary of the main social relationships settings. also summarizes the major features characterizing each scenario, which are: the types of social relationship invRel.) (shortened as Rel.), the number of destinations (shortened as DSTs) for each communication, their distance from the source (shortened as SRC), and their position with reference to the considered network topology.

2.3.3.1 Scenario A: Smart industrial plant.

Group communication needs: an industrial plant is equipped with several connected devices (sensors and actuators) and one of these (randomly selected) belonging to the emulated topology issues a Sociocast packet destined to all the

devices connected to the same gateway. The group can be created, for instance, for the dissemination of alarms, for group configuration and reconfiguration, for functional testing.

Involved relationship type: CLOR.

End-point distribution profile: all endpoints clustered in the same area.

2.3.3.2 Scenario B: Smart home monitoring.

Group communication needs: a randomly selected device in the emulated topology, resembling a smartphone of a user currently at office, acts as a sender and issues a Sociocast packet to create a group of recipients made up of all the smart devices connected to the (same) home gateway, which is different from the one the user's smartphone is attached to. The group can be created, for instance, to notify devices to configure a warm welcome for the user.

Involved relationship type: OOR (ownership).

End-point distribution profile: sender in a location and all destinations clustered in a different (potentially) remote location.

2.3.3.3 Scenario C: Wireless Sensor Network (WSN) management.

Group communication needs: a randomly selected device in the emulated topology acts as a sender and issues a Sociocast group creation destined to all the devices of the same brand, uniformly distributed in the topology to disseminate a new configuration for the device, a software update, or a new driver version.

Involved relationship type: POR (parental).

End-point distribution profile: uniform distribution of end-points.

2.3.3.4 Scenario D: Smart mobility.

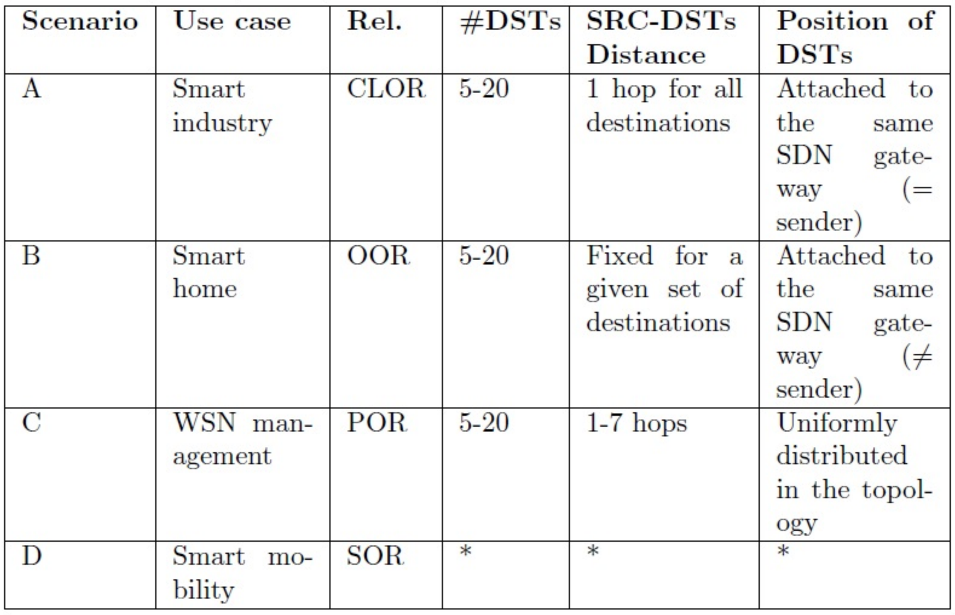
Group communication needs: we assume mobile devices (e.g., smartphones, laptops) carried by people moving in a smart city/smart campus and interacting with other devices met either in the neighborhood or close offices/classrooms. The type of the data exchanged within the group includes: information related to mobility applications, tourist information, data for the implementation of any Intelligent Transportation Systems application.

Involved relationship type: SOR.

End-point distribution profile: variable location of end-points in the group.

As to the creation of the relationships, these have been set in deterministic way except for the SOR. In particular, different groups of devices linked with POR and OOR relationships are created so as to have from 5 to 20 recipients for each simulated communication. However, the

CLOR relationship has been created among devices that are connected to the same gateway as the co-location has to be assured. As to the SOR relationships used in Scenario D, these are established among devices in the emulated topology according to their physical distance and follow a simple probabilistic model. The principle adopted is such that the closer the devices, the higher the probability that the two devices have establish a SOR relationship. Accordingly, devices attached to the same SDN gateway (i.e., an Access Point) have the highest probability to establish it. These devices are characterized by sharing the same path to reach the root node (s1 in Figure 15: Reference topology.), which is made up of 4 SDN nodes. We base on this number to define the notation to denote the relevant probability to create a SOR among them: $p_{soc,4}$. Following the same principle, devices sharing three, two, or one SDN nodes in the path to reach s1, establish a relationship with probability $p_{soc,3}$, $p_{soc,2}$, and $p_{soc,1}$, respectively. The higher j the higher the probability $p_{soc,j}$, with $j \in \{1,2,3,4\}$. The setting of $p_{soc,j}$, used in the performed simulations is reported in Table 3: Probabilities of SOR establishment.; different configurations have been considered to evaluate the impact of different numbers of friends and their distribution in the considered topology.



Scenario	Use case	Rel.	#DSTs	SRC-DSTs Distance	Position of DSTs
A	Smart industry	CLOR	5-20	1 hop for all destinations	Attached to the same SDN gateway (= sender)
B	Smart home	OOR	5-20	Fixed for a given set of destinations	Attached to the same SDN gateway (≠ sender)
C	WSN management	POR	5-20	1-7 hops	Uniformly distributed in the topology
D	Smart mobility	SOR	*	*	*

Table 2: Summary of the main social relationships settings.

Sim-ID	#Destinations	$p_{soc,1}$	$p_{soc,2}$	$p_{soc,3}$	$p_{soc,4}$
1	4.5	0.1	0.2	0.3	0.4
2	5.5	0.2	0.3	0.4	0.5
3	8.4	0.3	0.4	0.5	0.6
4	11.2	0.4	0.5	0.6	0.7
5	12.7	0.5	0.6	0.7	0.8
6	15.3	0.6	0.7	0.8	0.9
7	18.4	0.7	0.8	0.9	1

Table 3: Probabilities of SOR establishment.

2.3.4 Benchmark scheme

The performance has been compared against an application-layer solution, which we refer to as multiple unicast (labeled in the plots as M-Unicast). Note that also for this benchmark scheme, we are focusing on the push-based data dissemination scenario. The choice of this benchmark is meant to quantitatively estimate the benefits of the Sociocast proposal against an application-layer solution. In the latter one, the network layer is agnostic about the communicating group, but it offers the same features in terms of sender initiated and dynamic Sociocast group creation, hence ensuring a fair comparison. Specifically, the source node contacts a proxy in charge of interacting with a SIoT-like platform to get the set of intended destinations belonging to the Sociocast group. The latter one is described through attributes/meta-data defined at the application layer, similarly to the information encoded in the tags in Sociocast packets. After retrieving the list, the proxy forwards it to the source node which sends the packet to the destinations through multiple unicast exchanges. In other words, the controller sets up distinct routing paths for each destination and some links can be shared by multiple paths towards destinations belonging to the same group. Without losing generality, we assume that the proxy is attached to the root node of the topology (i.e., s1 in Figure 15: Reference topology.).

2.3.5 Metrics

The following metrics have been considered to evaluate the performance of the compared schemes in the creation of a Sociocast group and data exchange among its members:

- the *number of OF signaling packets* exchanged between SDN nodes and controller to build routing paths towards the intended Sociocast destinations. The metric only refers to the control packets exchanged to process incoming requests from sociocast nodes at the SDN gateway, namely Packet In, Packet Out and Flow Mod. The background (periodic) signaling exchanged between the controller and the SDN nodes is not considered;
- the *number of data packets* exchanged into the network to reach all the intended destinations of the communicating group, once it has been creaated; the metric considers the number of transmitted packets per-link and they are represented by either Sociocast or M-Unicast packets.

For the benchmark scheme, the request packets issued by the source towards the proxy as well as the signaling messages required to instruct the relevant SDN nodes towards it are also considered.

The above metrics have been measured through the well-known Wireshark network protocol analyzer[5].

Comparison experiments have been conducted when varying the number of destinations (or relevant probability settings) and are averaged over 20 runs.

[5] Please notice that the analysis of the signaling incurred for the creation of social relationships among devices is outside the scope of this work and is peculiar of the conceived SIoT implementation. We remand to [57] for more details.

2.4 Results analysis

In this paragraph are shown the performance results. More specifically, in Section 2.4.1 it's discussed Sociocast in terms of generated signaling packets; whereas Section 2.4.2 focuses on data packets.

2.4.1 Signaling packets

The first set of results aims at analyzing the control plane signaling footprint incurred by the proposal and the benchmark scheme. Figure 17: Sociocast Vs. Multiple Unicast. Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario A. Figure 18: Sociocast Vs. Multiple Unicast. Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario B. and Figure 19: Sociocast Vs. Multiple Unicast: Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario C. report the number of exchanged OF packets when varying the number of destinations of the Sociocast group under for the scenarios A-C, whereas the results for scenario D are shown in Figure 20: Sociocast Vs. M-Unicast OF packets exchanged to create the sociocast group. It can be clearly observed that for the M-Unicast approach the metric significantly increases with the number of destinations, in all the considered scenarios. Such a trend is due to the fact that the end-to-end communication path towards each single destination needs to be discovered with the support of the SDN controller. In other words, an SDN node receives a number of M-Unicast packets to forward equal to the number of destinations it allows to reach. For each of them, it contacts the controller by generating a Packet In message and waits for the corresponding Packet Out and Flow Mod with instructions about the forwarding behavior.

For a given number of destinations, the highest number of OF packets are exchanged in case of Scenario C. In the latter one, indeed, the destinations are spread over the topology and the routing path towards them may involve several SDN nodes (and gateways). Scenario B follows with a lower number of exchanged OF packets. In Scenario A, instead, only a single SDN gateway is in charge of Sociocast packet forwarding. It is the only SDN node transmitting and receiving OF packets.

In the Sociocast solution, the controller is in charge of building routing paths towards them so to avoid the forwarding of the same Sociocast packet over the same link.

Hence, unlike the benchmark scheme, in our proposal, those SDN nodes which belong to the paths towards different destinations receive only a single Sociocast packet to forward and a single Flow Mod from the controller. The gain of Sociocast w.r.t. M-Unicast in terms of exchanged OF packets gets more remarkable as the number of destinations increases. For instance, in Scenario C, it passes from a factor of around 6 for five destinations to a factor of more than 14 for twenty destinations.

It is worth observing that, in Sociocast, a single Flow Mod message may convey multiple rules to be injected into an SDN node. In particular, Table 4: Size (in bytes) of the Flow Mod packet for Scenario A. reports, for Scenario A, the size of the Flow Mod message, as measured at the SDN gateway, which the source and the destinations are both attached to. For the Sociocast proposal, the size reasonably increases with the number of destinations to accommodate the action rule for each of them. The rule specifies the physical output port as well as the change of the IP address from
Sociocast to unicast. For M-Unicast, each Flow Mod carries a single rule, since its injection is issued per each M-Unicast packet traversing an SDN node. The size increases of less than a factor of 3 for the Sociocast approach compared
to M-Unicast, in the case of twenty destinations.

Despite the larger size of Flow Mod packets, it can be easily inferred that, overall, the OF signaling footprint of the proposal, in terms of number of ex-changed bytes, is significantly lower than M-Unicast. Also, the proposal better scales with the size of the Sociocast group.

Similarly to the benchmark scheme, the proposal experiences the largest signaling in Scenario C, wherein multiple SDN nodes, involved in forwarding Sociocast packets to destinations, spread over the topology, need to be instructed.

Similar considerations hold for Scenario D, Figure 20: Sociocast Vs. M-Unicast OF packets exchanged to create the sociocast group. Also in such a case, the proposed Sociocast solution is less sensitive to the simulation settings (i.e., size of Sociocast group and its configuration in terms of proximity of destinations w.r.t. the source) than the benchmark.

2.4.2 Data packets

Results in Figure 22: Sociocast Vs.M-Unicast exchanged data plane packets when varying the number of receivers in different scenarios. shed further light into the efficiency of the compared schemes in delivering the data packets. Similarly to the OF signaling, also the number of exchanged Sociocast packets increases with the number of destinations; the highest values are experienced for Scenario C and the lowest ones in Scenario A.

As a general remark, it can be observed that the proposal is less sensitive to increases in the number of destinations when compared to the benchmark. This happens because the controller builds the routing paths to avoid that packets are redundantly transmitted over a given link shared by more destinations.

This is not the case for the M-Unicast solution where forwarding decisions are separately taken for each data packet, according to the address of the intended destination.

When referring to Scenario A, the M-Unicast approach always sends twice as many data packets as the proposal. This is an obvious consequence of the fact that, after receiving the destinations list, for the M-Unicast approach there are two packets, for each destination, traveling into the topology. One packet travels from the source to the SDN gateway, and the other one from the SDN gateway to the corresponding destination. This does not apply for the Sociocast approach, where there is only the data packet from the SDN gateway to each destination.

Improvements get larger for other scenarios.

In Scenario B, more SDN nodes are involved in the routing path, despite the fact that all the destinations are connected to the same SDN gateway. Hence, more data packets travel into the network, especially for the M-Unicast solution. Such a trend is more remarkable in Scenario C, due to the larger spread of destinations over the topology. A similar trend is observed for Scenario D in Figure 21: Sociocast Vs. M-Unicast exchanged data plane packets in the topology. Note that in Figure 20: Sociocast Vs. M-Unicast OF packets exchanged to create the sociocast group and Figure 21: Sociocast Vs. M-Unicast exchanged data plane packets in the topology. the different simulation runs (from 1 to 7) correspond to the different Sim-IDs of Table 3: Probabilities of SOR establishment..

Not surprisingly, improvements of Sociocast w.r.t. M-Unicast are greater in Scenario B compared to Scenario C. Indeed, in Scenario B the path towards all intended destinations is the same from the source to the SDN gateway. Hence, in Sociocast, the SDN controller judiciously issues rules that prevent from forwarding duplicated packets over the same links.

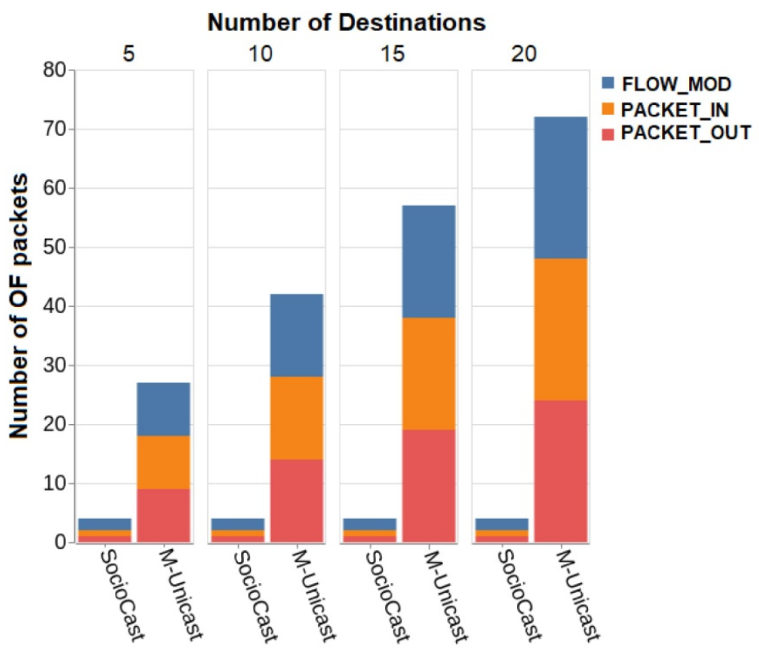


Figure 17: Sociocast Vs. Multiple Unicast. Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario A.

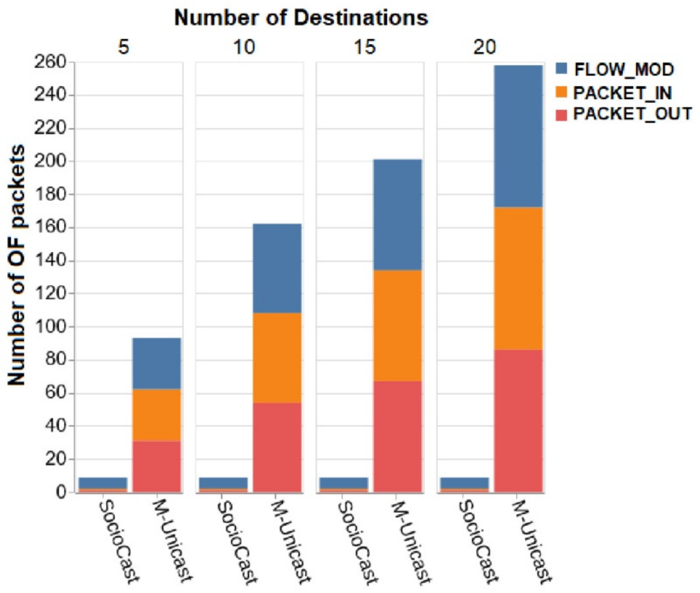


Figure 18: Sociocast Vs. Multiple Unicast. Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario B.

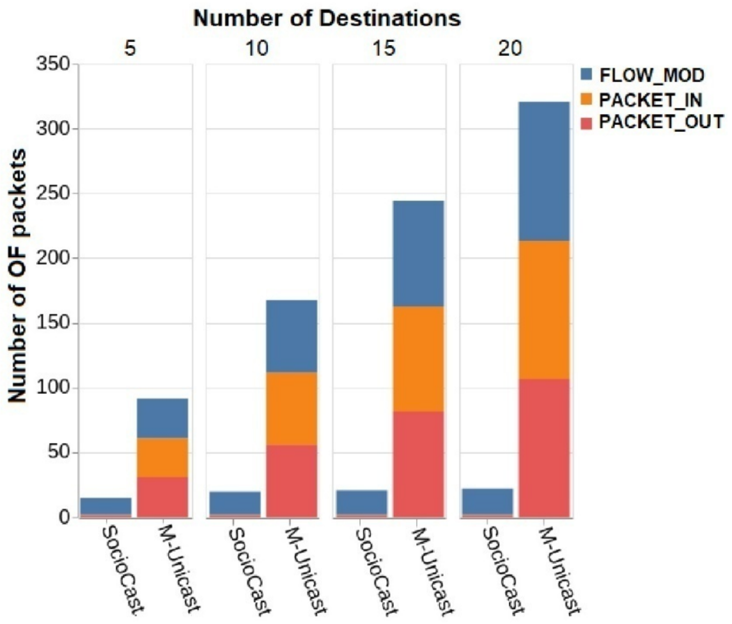


Figure 19: Sociocast Vs. Multiple Unicast: Exchanged OF packets for Sociocast group creation when varying the number of destinations, scenario C.

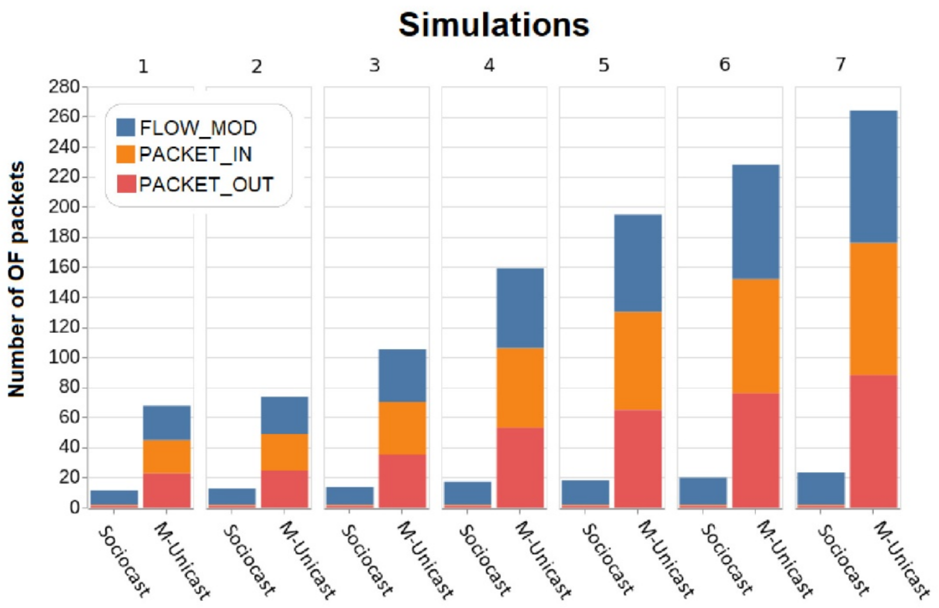


Figure 20: Sociocast Vs. M-Unicast OF packets exchanged to create the sociocast group.

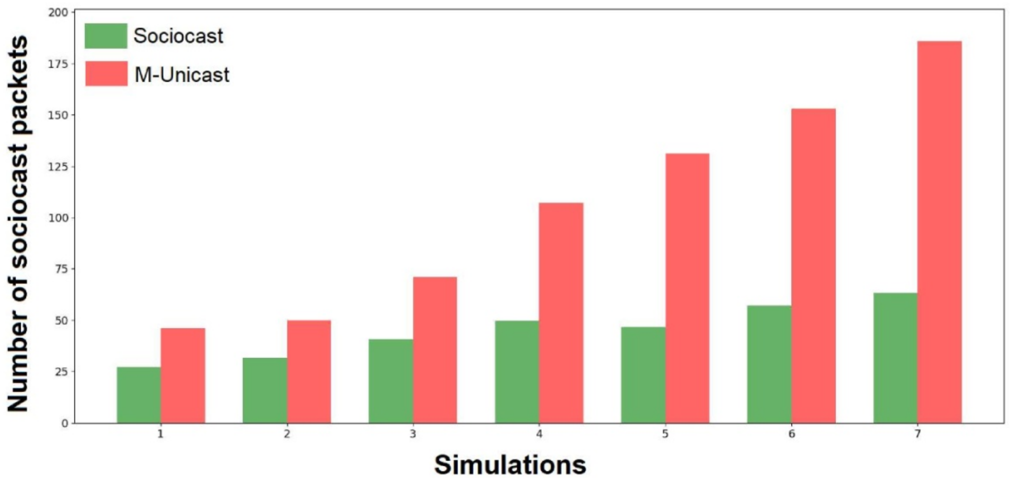


Figure 21: Sociocast Vs. M-Unicast exchanged data plane packets in the topology.

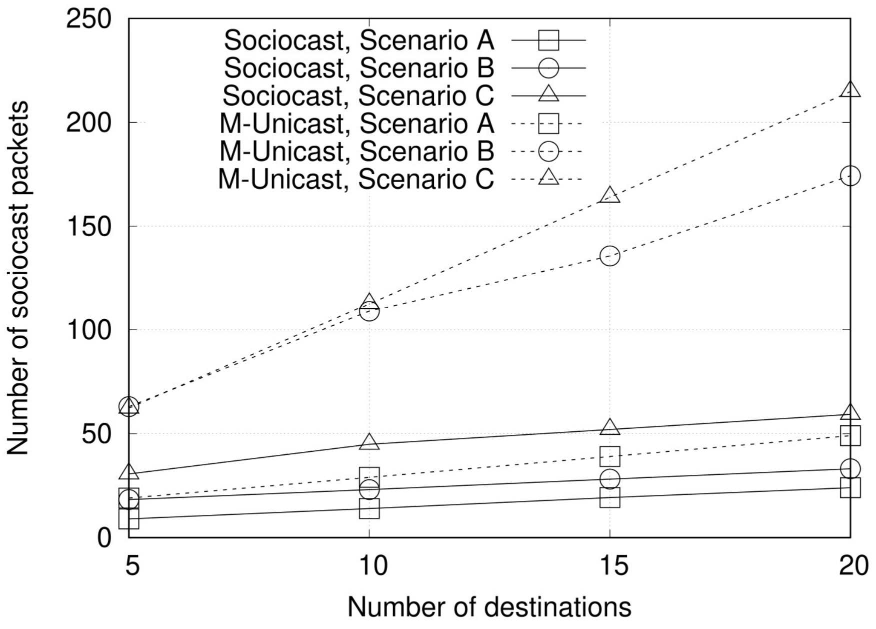


Figure 22: Sociocast Vs.M-Unicast exchanged data plane packets when varying the number of receivers in different scenarios.

#Destinations	M-Unicast	Sociocast
1	172	172
5	172	252
10	172	332
15	172	412
20	172	492

Table 4: Size (in bytes) of the Flow Mod packet for Scenario A.

Benefits of the Sococast are definitely large when big groups of destination devices are clustered together, as witnessed by results referring to Scenario B: the OF signaling is reduced by a factor higher than 10 and the number of exchanged data packets shrinks by more than a factor of 5 (for twenty destinations). The lower gains for Sociocast packets w.r.t. OF signaling are due to the fact that Sociocast also resorts to multiple unicasts forwarding in the last hop from the SDN gateway towards the intended destinations, to ensure successful reception at the application layer. It can be further easily inferred (although not shown in results) that improvements get even larger as the distance between the source and the set of destinations increases. Overall, the proposal is especially suited for push-based data dissemination to large Sociocast groups highly clustered and far from the source, which well resembles the case of multiple devices of a smart home (e.g., appliances) to be remotely configured by the user's smartphone.

In the other cases, the gains are also significant and always higher than a factor of 2. The achieved encouraging results motivate us to further explore this fertile research area which has large room for improvements. The effectiveness
of the proposal in handling other Sociocast features, like source filtering and publish/subscribe, needs to be practically explored.

Chapter 3: MQTT Algorithm

3.1 Statement of the problem

The Message Queuing Telemetry Transport (MQTT) is a standard publish-subscribe protocol for the transport of messages between devices. MQTT is efficient for low-bandwidth and unreliable networks, furthermore, thanks to its simplicity, it can be executed by devices with low capabilities in terms of computing, communication, and energy resources. As a consequence, MQTT has become one of the most popular protocols for the Internet of Things (IoT).

In MQTT, the broker plays a central role. A subscriber can inform the MQTT broker about its interest in a given topic. The broker is responsible for forwarding the messages received by the publishers under such topic to all interested subscribers.

MQTT has several interesting features that make it a suitable solution for the interactions with smart objects in the IoT:

- its execution is not demanding for what concerns computing and communication resources;
- there are several open source implementations for a wide range of computing platforms;
- it can support a large range of interaction modes spanning from thing-to-server and thing-to-thing;
- its interaction patterns are simple to use and is supported by most programming languages this makes application development extremely easy.

3.1.1 Weaknesses and open issues

MQTT has a major problem that prevents it from becoming a unifying solution for the IoT. In fact, MQTT is basically a centralized approach: publishers and subscribers should interact with the same broker in order to interact with each others.

There are two types of consequences of such centralization.

1. The broker might become a bottleneck as the processing of subscription requests and publish messages overcome its computing and communication capabilities.
2. Interactions between clients connected to different brokers becomes difficult if not impossible.

To address such problems, some MQTT implementations (such as Mosquitto), which will be discussed in the next paragraph, enable broker bridging. In other terms, they allow a MQTT broker to subscribe to certain topics which are under the responsibility of other brokers. Accordingly, it is possible to build a network of brokers and research efforts have been devoted to the optimization of such network, e.g., [59].

In principle, by fully exploiting bridging two MQTT clients can interact independently of the brokers they are connected to. However, it is obvious that interactions between clients connected to different brokers result in low performance. In fact, delays will be longer because there is a further communication hop at the overlay layer for a published message to reach interested subscribers. Furthermore, messages that are forwarded from one broker to others in order to deliver the message to all interested subscribers consume communication resources which would be left available for other uses if all the clients involved were connected to the same broker.

In this context, the objective of this chapter is to introduce a technique which exploits the Social Internet of Things (SIoT) concept [19], so as to determine how to choose the most convenient broker for each MQTT client. Indeed, the proposed approach creates clusters of clients depending on their position in the SIoT social graph and takes advantage of the evidence that most interactions are local in social networks.

3.2 Algorithm description

This section describes how it's possible to achieve the clusters creation on the basis of on the clients position in the social graph. In details, first we describe the design principles used for the correct clusterization of the social network; then in paragraph 3.2.2 is described the algorithm used to create the clusters. Finally, in the last paragraph is reported an actual example of how the algorithm works.

3.2.1 Objectives and design principles

It is important to specify what are the mandatory requirements and design principles of the clusters that we want to create. In details, the specifications requested are the following:

- **Number of clusters.** Since, in the reference topology, we are considering two MQTT brokers bridged together, we just need two clusters, one cluster per broker.
- **Size of the cluster.** We want to avoid congestion and bottle necks**:** our clusters should, therefore, be as balanced as possible, in terms of the number of elements composing them. Hence, there is a need to avoid scenarios, where given N elements composing the social graph, N-1 elements characterize a cluster while the other one is alone in the second cluster. The best possible situation occurs when the clusters contain the same amount of elements.
- **Properties of the clients**. Two elements pertaining to the same cluster should have a social relationship that bound them together. Such condition translates in a more frequent exchange of messages than that involving elements untied by a social bound.
- **Clustering of the social network**. Since our aim is to optimize the bridging between the two MQTT brokers, we, indeed, want to reduce the interactions and the message exchanged between them. Therefore, we want that the number of social

relationships between clients belonging to different clusters is a as low as possible. In a nutshell, we want to partition the social graph whit a minimum cut, which separates the two clusters.

Considering all these design principles, makes the choice of which client should go into which cluster an NP hard problem.

Several solutions have been proposed, or are well known in the literature, to solve this kind of problems, in particular, one the most used in the scenario of dividing a graph with a minimum cut, is the Karger's algorithm [60].

3.2.2 Karger's Algorithm description

The Karger's algorithm is adopted to estimate a minimum cut of a connected graph, and, as we will discuss later, to do so it adopts, as part of its logic, a degree of randomness, and for this reason it is a randomized algorithm.

The main concept on which the algorithm is founded is the "edge contraction". In graph theory, an edge contraction occurs when given two nodes A and B connected with one or more edges, all these edges are removed and the two nodes are merged together, like shown in Figure 23: Edge contraction of A and B. This operation reduces the total number of nodes in the graph by one. All the other edges connecting the nodes A and B with the rest of the graph are "reattached" to the merged node, effectively producing a multi-graph.

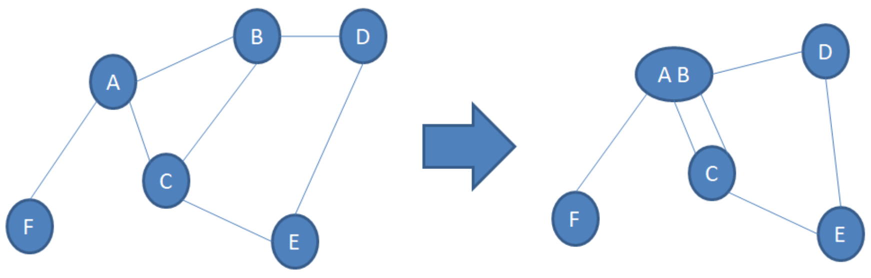


Figure 23: Edge contraction of A and B

In general, the Karger's algorithm use this concept to iteratively contracts randomly chosen edges until in the graph remain only two nodes; which represent a cut in the original graph. By repeating this basic algorithm a sufficient number of times, a minimum cut can be found which represent a optimal solution for the problem.

3.2.2.1 A modified version of Karger's algorithm for the social graph

Applying the Karger's algorithm in a social graph, especially in our scenario, is not as straightforward and simple as someone could imagine. This is so because of two main reasons:

1. Since the aim of the algorithm is to find the minimum cut, several solutions found at the end of a cycle, are the one that divides the graph in two parts one of which contains a single element. Unfortunately, these kind of solution are against the design principles according to which we want to create our clusters.
2. Since it is a randomize, the algorithm, can spent a lot of time and resources to find a solution which is suitable for our scenario and respecting our constraints.

To solve these issues, a modified version of the karger's algorithm is proposed. In order to emphasize the characteristics that we need, we have to give different weights to the edges of the social graph before the contraction, in this way we are trading a part of the randomness in the algorithm in exchange for better performances and a fasten way to find balanced clusters.

The weights (which have been given to the edges before choosing which one to contract) are representative of the number of nodes in the edge and the number of nodes in all the edge in the social network, this implies that some nodes may be taken into account multiple time if they appear in different edges. In detail the weights are created by the law:

$$W = \frac{Number\ of\ nodes\ in\ the\ ege}{Number\ of\ nodes\ in\ alle\ the\ edges}$$

After assigning a weight to each edge, the edge with the lowest one is selected for the contraction. If more edges have the same weight, one among these edges is chosen at random.

3.2.3 Clustering algorithm in action

After taking in account how the algorithm works and what the requirement needed for creating the clusters that we want, are, it is time to describe step by step how the algorithm works when applied to a small social graph (for the sake of simplicity), for example the one shown Figure 23: Edge contraction of A and B.

1. In the very first step, all the edges of the network are assigned with a weight, which, indeed, is the same since all the edges have only two nodes in this early stage. For this reason subsequently, all the edges have the same probability to be chosen for the contraction.
2. One edge at random is selected randomly (e.g., the edge AB), a new social topology is created from its contraction, Figure 23:

Edge contraction of A and B; at this point new weights are created and assigned to the edges of the new topology.

3. Based on the new given weights, the edge with the lowest one is selected for the new contraction. Since n our scenario the two edges CE and DE have the same weight, one at random is selected and contract, for example the edge DE.
4. This procedure of edges selection based on their weight and edges contraction is repeated, as shown in Figure 24, until only two nodes remain, this two nodes represent each a cluster containing the contracted nodes; cluster 1 with elements [A, B, F] and cluster 2 with [C, D, E].
5. The found solution is an optimal one and for this reason the algorithm explained above is repeated several times, starting obviously from the same social topology. Then all the possible solutions are compared and the one that respect the best our constrains is chosen.

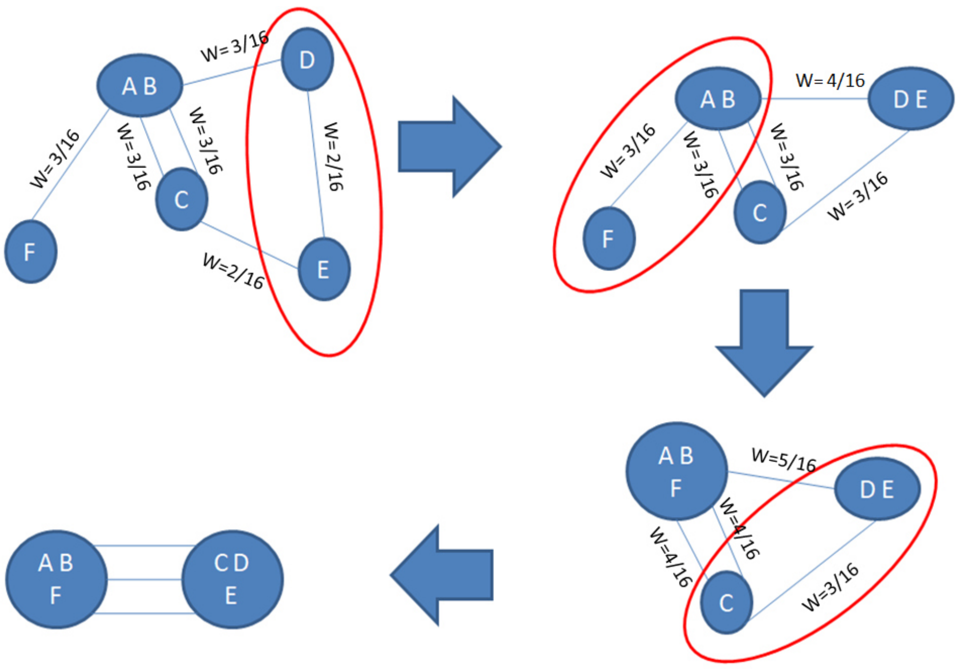


Figure 24: Algorithm in action

3.3 Experimental setup

In this section we describe the environment for the performance evaluation. More specifically, in Section 3.3.1 we describe the tools utilized for the performance evaluation, in Section 3.3.2 we discuss the proposed scenario (smart campus), whereas the algorithm settings and the traffic patterns involved are treated in Section 3.3.3. While the benchmark utilized for comparison purposes is presented in Section 3.3.4, Section 3.3.5. concludes by identifying the performance metrics considered.

3.3.1 Tools description

The network emulator Mininet [25] and Mosquitto [61] the MQTT message broker, are the main tools used for the performance evaluation. Mininet has been already introduced in Section 2.3.1.2. for this reason in the following will be discussed only Mosquitto.

3.3.1.1 Mosquitto

Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and suitable for use on all devices from low power single board computers to full servers moreover it is highly portable and available for a wide range of platforms. The Mosquitto project also provides a C library for implementing MQTT clients, as well as the very popular mosquitto_pub and mosquitto_sub command line MQTT clients.

Mosquitto is part of the Eclipse Foundation [62], and an iot.eclipse.org project [63].

3.3.2 Tools usage and reference topology

The focus of the performance evaluation is to assess the improvements brought by the modified version of the Karger's algorithm in the case of an MQTT publish/subscribe data dissemination towards a group of devices bond together by social relationships. To this purpose, we built an emulation playground. In particular, the Mininet network emulator [25] was used to recreate a small campus network topology, as depicted in Figure 25. A full-mesh interconnects the network nodes, which are the roots of a three-layers fat-tree topology. Up to 5 devices are attached to each gateway (not all the devices are shown in the Figure).

The open-source Eclipse Mosquitto broker [61] was chosen as the reference MQTT message broker in the context of this work, due to its lightweight, compatibility with almost every device and easy installation. The two bridged MQTT brokers run in two different virtual machine, both of them located in two different personal computers DELL inspirion 17 5000 series (8Gb Ram Intel Core i5-6200U). While the Mininet topology runs in a third virtual machine in an personal computer ASUS X52J (8Gb Ram, Intel Core i5 M460). These three virtual machine are connected to the same 802.11ax wireless network.

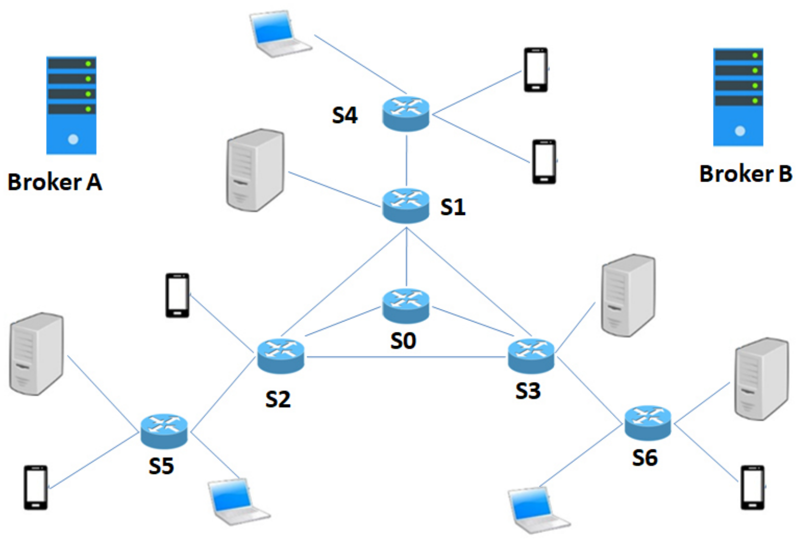


Figure 25: Reference Topology

3.3.3 Settings and traffic patterns

The performance of the proposed solution was evaluated with a representative IoT test configuration, a small smart campus scenario. This scenario is properly designed to take into account different numbers and distributions of nodes in the emulated topology and also different physical distances between sources and destinations. This is aimed at making the obtained results as generalizable as possible. The test configuration was mapped onto a use case characterized by the exploitation of a particular type of social relationship between the devices involved, the *CLOR* relationship. This particular social relationship empathies the physical distance between the devices, more specifically, the nearer the devices in the topology, the higher the probability of being friends. The probability that *i* and *j* have established a social-like relationship with each other, indeed, follows the law:

$$Pr\{i\ friend\ j\} = kD_p^{\ -\alpha}$$

Where D_p is the physical distance between *i* and *j* in terms of hops, while *k* and α are positive real numbers which characterizes the structure of the social network. Notice that as *k* and α varying different possible social graphs can be obtained from the same physical topology for example Figure 26 shows a social graph created from *k* = 0,25 and α = 0,5.

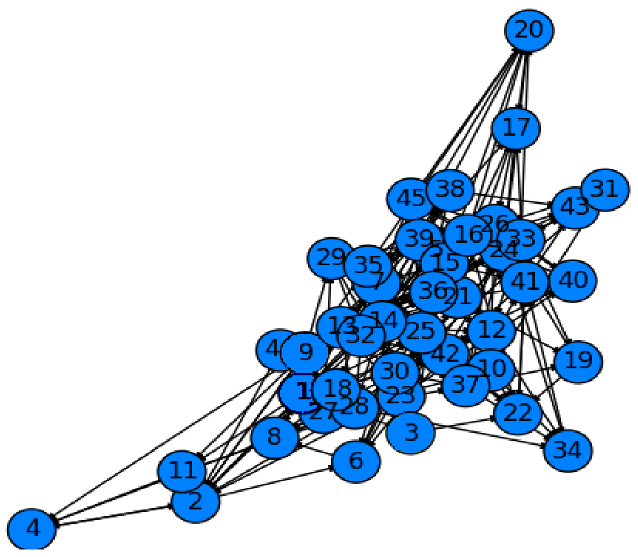


Figure 26: Social Network for *k* = 0,25 and α = 0,5

A study was conducted to determinate the social graphs peculiarities, in terms of the average of social hops and the mean of social relationships between devices, as the parameters *k* and α vary. The results of this study are shown in the figures below.

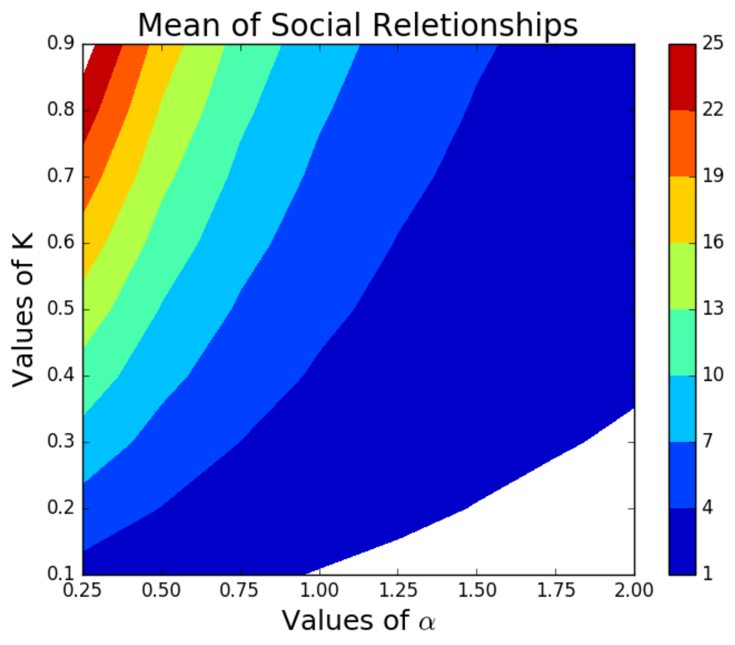


Figure 27: Average of social relationship for *k* and α

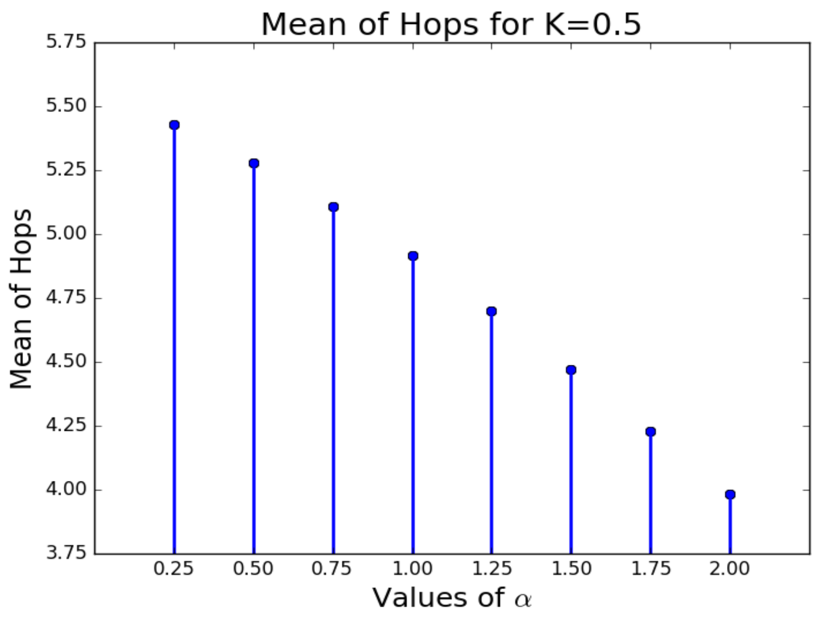


Figure 28: Average of hops for *k* and α

Of all the possible combinations of *k* and α, we focused our later study in the couples which give a moderate amount of social relationships while avoiding extreme scenarios. The considered values are reported in Table 5.

For each considered couple of *k* and α, three different social networks were created and investigated.

In conclusion, in order to take in account and study different data patterns, the probability, whit which a generic node *i* publishes something on which the generic node *j* has a subscription, is given by:

$$Pr\{i\ pub\ j\} = \eta e^{-\beta x}$$

Where *x* is the social distance between *i* and *j* in terms of hops in the social graph, while η and β are positive real numbers which characterizes the density of the data pattern.

In this way, helpful guidelines can be provided on the suitability of the proposed solution in the context of different data dissemination patterns.

α	0.4			0.6			0.8		
$\boldsymbol{k}$	0.2	0.3	0.4	0.2	0.3	0.4	0.2	0.3	0.4

Table 5: Considerated values of K and $\boldsymbol{\alpha}$

3.3.4 Benchmark scheme

The performance of the proposed approach was compared against a solution where the devices casually chose the MQTT broker to interact with. Generally speaking, a device connects to a specified MQTT broker, yet, since our scenario is representative of a small campus is possible to assume that the two brokers are placed in the same server room. For this reason the random approach is a good compromise to obtain similar clusters in comparison to the MQTT optimizer algorithm and to quantitatively estimate the benefits of the proposed solution.

Also we kept track of all the traffic pattern generated in our solution and the same schemes are applied in the benchmark, ensuring a fair comparison.

3.3.5 Metrics

Before introducing the metrics that have been considered to evaluate the performance, it's mandatory to define the random variable taken into account in the proposed approach.

Considering the generic subscriber node *i,* in the reference topology, we define the following random variables:

- D_i represents the delay encountered by a message which should be received by *i.*
- S_i which is equal to 1 if the message is generated by a publisher which is managed by the same MQTT broker of *i* while it is equal to 0 otherwise.
- X_i represents the distance in the social network between the publisher and *i.*
- Y_i represents the distance in the physical network between the publisher and *i.*

The performance metrics which have been evaluated experimentally are the following:

- $E\{D\}$, the average of the delay D_i, is defined as:

$$E\{D\} = \sum_{s=0}^{1}\sum_{x=1}^{3} E\{D|S=s, X=x\} \cdot Pr\,\{S=s|X=x\} \cdot Pr\,\{X=x\}$$

Where, $Pr\,\{X=x\}$ follows the law:

$$Pr\{X=x\} = \eta e^{-\beta x}$$

$E\{D|S=s, X=x\}$ and $Pr\,\{S=s|X=x\}$ have been evaluated experimentally.

- Additionally, $E\{D|S=s, X=x\}$ can be related to Y

$$E\{D|S=s,X=x\}= \\ =\sum_{y=2}^{ymax} E\{D|S=s,X=x,Y=y\}\cdot Pr\{\,Y=y|S=s,X=x\}$$

Where $E\{D|S=s,X=x,Y=y\}$ too was evaluated experimentally and is equal to $E\{D|S=s,Y=y\}$ and applying the Bayes's law we evaluate $Pr\{\,Y=y|S=s,X=x\}$

$$Pr\{\,Y=y|S=s,X=x\}=Pr\{\,Y=y|X=x\} \\ =\frac{Pr\{\,X=x|Y=y\}\cdot \Pr\{Y=y\}}{\Pr\{X=x\}}$$

$$Pr\{\,X=x\}=\sum_{y=ymin}^{ymax} Pr\{\,X=x|Y=y\}\cdot \Pr\{Y=y\}$$

- The last performance metric evaluated is the probability on which publisher and subscriber are served by the same MQTT broker:

$$Pr\{\,S=1\}=\sum_{x=1}^{3} Pr\{\,S=s|X=x\}\cdot \Pr\{X=x\}$$

3.4 Results analysis

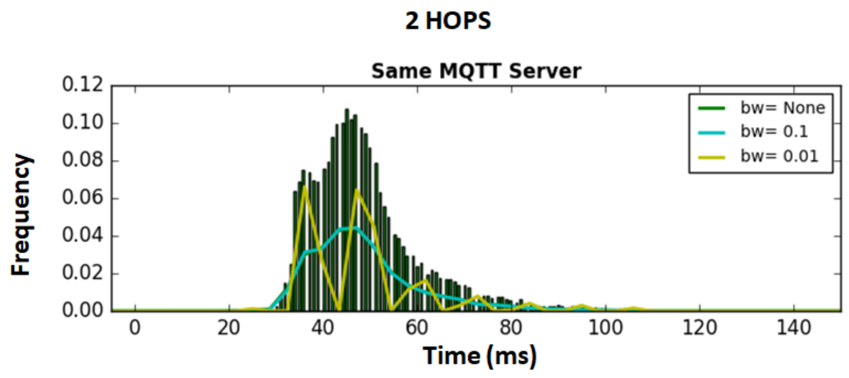
This paragraph shows the performance results based on the experimental estimated metrics. More specifically, in Section 3.4.1 it's discussed the MQTT optimizer in terms of delay in the physical network and probability on which publisher and subscriber are served by the same MQTT broker; whereas Section 3.4.2 focuses on the average of the delay compared to the benchmark.

3.4.1 Delay in the physical network

The first step of the experimental campaign was to estimate the delay inside the physical network, on the basis of different physical distances (expressed in terms of number of hops) and if the publisher and the subscriber were served by the same MQTT broker.

After choosing at random a publisher, for each of the possible cases were sent and tracked 2000 packets, for a total of 20000 packets sent.

The results of this campaign are shown from Figure 29 to Figure 32, were the probability distribution functions for each case are reported.



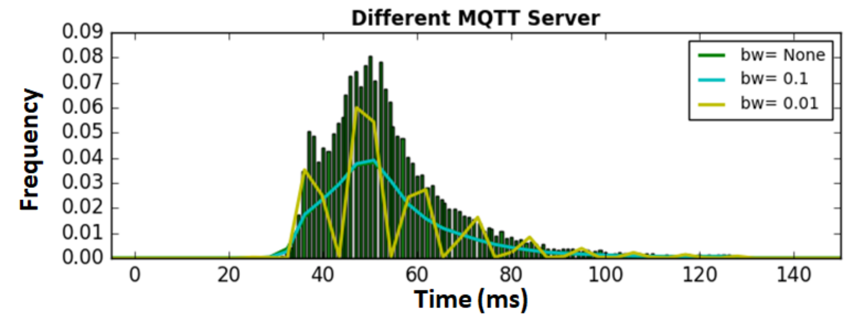


Figure 29: Pdf of delay for a distance of 2 hops

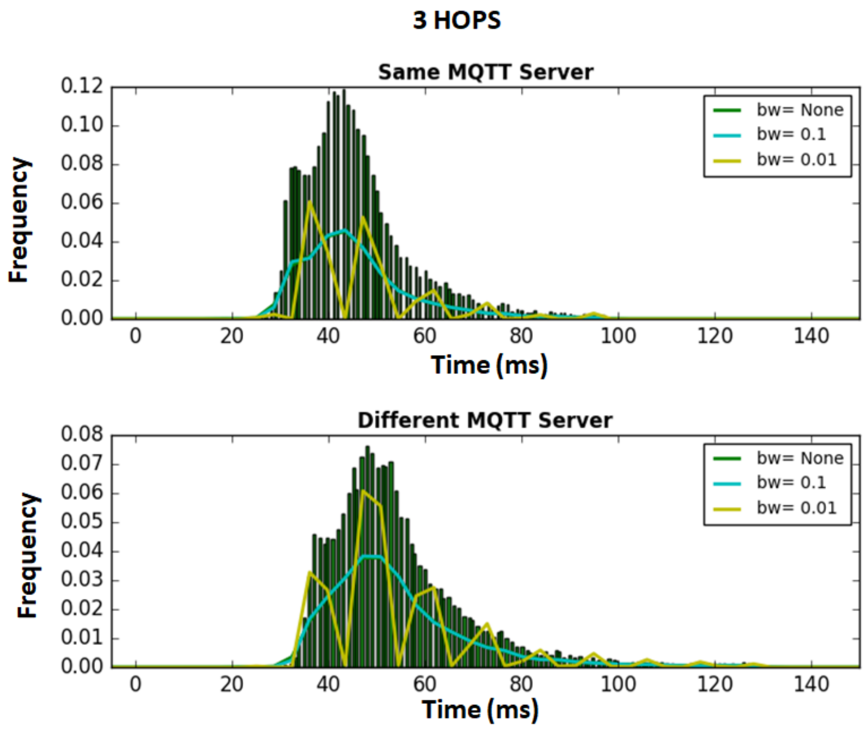


Figure 30: Pdf of delay for a distance of 3 hops

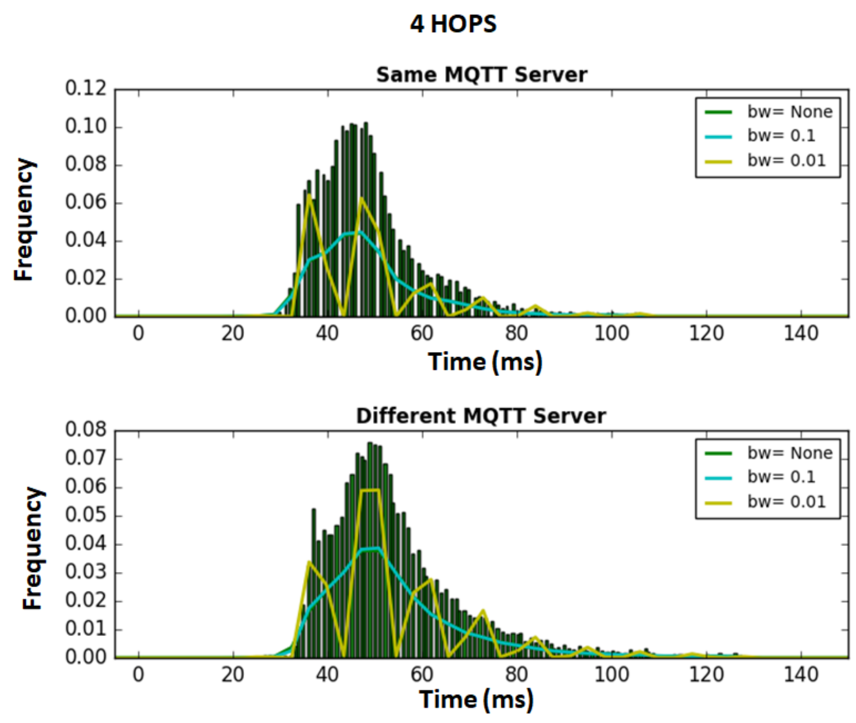


Figure 31: Pdf of delay for a distance of 3 hops

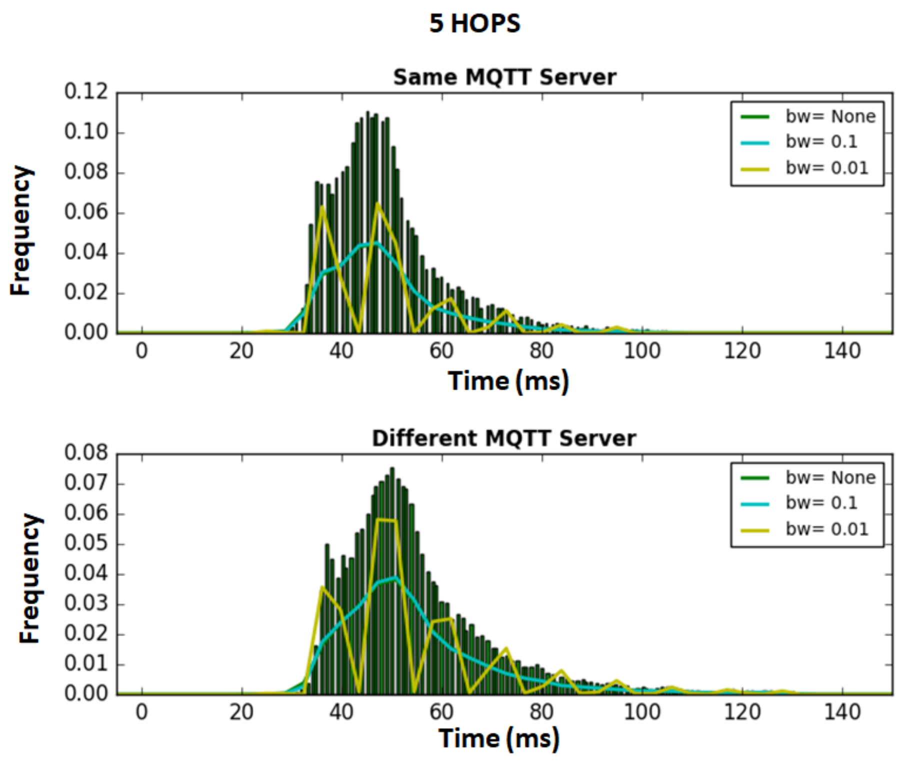


Figure 32: Pdf of delay for a distance of 2 hops

Other interesting parameters like the average and the standard deviation were estimated for all the dataset and the 95th percentile. All these are reported in following table.

Physical Distance	**Average**				**Standard Deviation**			
	Same Broker		**Different Broker**		**Same Broker**		**Different Broker**	
	100%	**95%**	**100%**	**95%**	**100%**	**95%**	**100%**	**95%**
2 hops	50.89	48.72	57.67	54.58	615.73	141.50	2366.86	227.64
3 hops	47.67	45.73	57.49	54.41	516.72	133.77	2518.71	227.41
4 hops	51.04	48.83	57.63	54.53	681.17	144.86	2223.90	230.40
5 hops	51.06	48.94	57.93	54.73	623.12	143.24	2520.48	241.20

Table 6: Average Delay and standard deviation for different distances.

3.4.2 Average of the delay

The second part of the experimentation focuses on the study of the delay in the social network based on the results obtained in the physical network. In particular, starting from the values in Table 5, a total of 27 different social networks were created and were used to test the optimization algorithm. After the creation of the clusters, for each of social network a data pattern, based on the social relationships between the nodes, was created using the probability mentioned in section 3.3.3:

$$Pr\{i\ pub\ j\} = \eta e^{-\beta x}$$

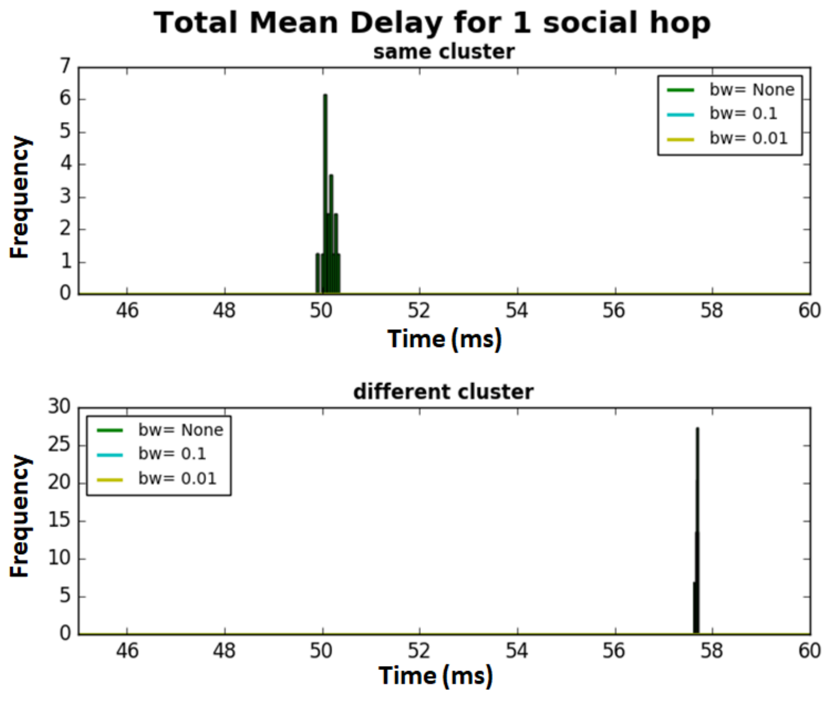


Figure 33: Average delay for a social distance of 1 hop

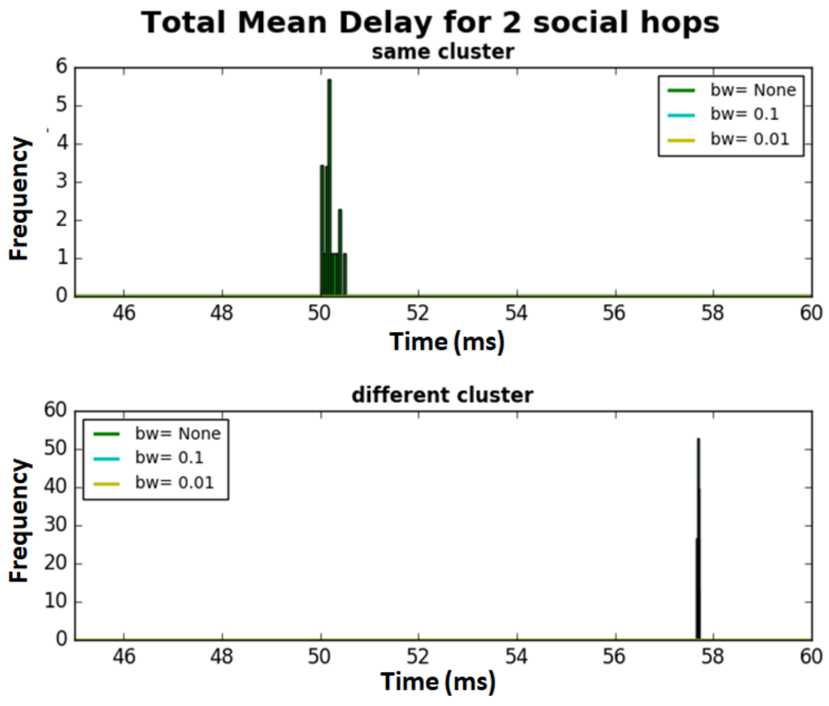


Figure 34: Average delay for a social distance of 2 hops

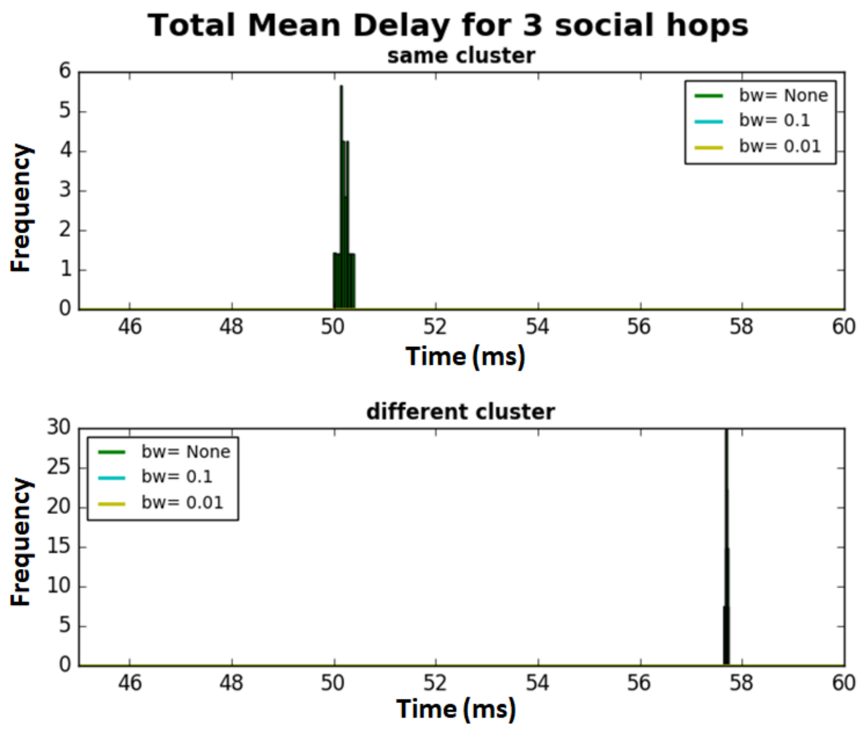


Figure 35: Average delay for a social distance of 3 hops

Due to these results we are now able to estimate the average of delay $E\{D\}$ as described in section 3.3.5. The results of the average of delay varying β from 0,1 to 3 are shown in Figure 36.

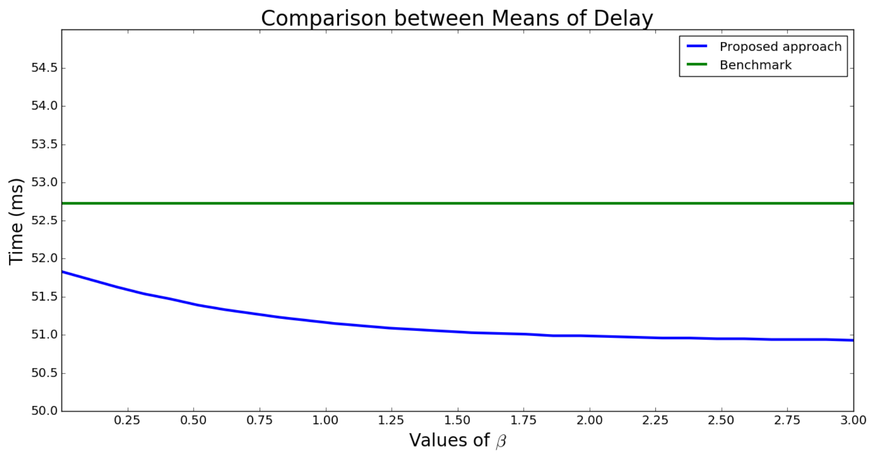


Figure 36: comparison Between the delays in proposed approach and the benchmark

Benefits of the proposed approach are evident, due to the smaller average delay when compared to the benchmark and decrease as β increases.

Chapter 4: Conclusions and future works

In this work, first, is proposed and analyzed the behavior of an architectural framework encompassing all the entities, functionalities, and procedures that support a fresh new network-layer group dissemination method, i.e., Sociocast, by leveraging a software-defined network approach.

Results achieved through an emulation testbed show the better scalability of the proposal in terms of OF signaling and data packet redundancy in comparison to an application-layer benchmark scheme, under different representative IoT scenarios.

Improvements are achieved by leveraging a purpose-built network application in the controller, which would be in charge of identifying the set of Sociocast destinations by interacting with an external SIoT platform (feature implemented at the application layer by the benchmark scheme) and responsible for smartly building routing paths towards multiple receivers so as to avoid packet duplication over links. SDN allows to manage the implementation of such functionalities at the control plane in a flexible and programmable manner, with no changes in the forwarding elements, hence making the devised framework practically viable at a low implementation cost.

As a further challenge, IoT devices belonging to Sociocast groups may move long distance between different access points. As consequence, tracking their positions at the virtual counterparts (SVN and SVNR), as well as managing the forwarding rules associated with them in the SDN nodes, become very difficult and entail proper workarounds which will be a subject matter of future investigations.

In conclusion, a technique to enhance the MQTT bridging with a social partition of the network elements was proposed and analyzed. Starting from the creation of a social network, based on *CLOR* type social relationship, rolling up to the clusterization of the network elements, thanks to a modified version of the Karger's algorithm.

Results achieved through an emulation testbed, representing a smart campus scenario, show better performances in terms of average delay when compared to a benchmark scheme without social ties between the elements.

References

[1] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad-Naous, Ramanan Raghuraman, and Jianying Luo.An Open Platform for Gigabit-rateNetwork Switching and Routing. MSE 2007, San Diego, June 2007.

[2] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky,and S. Uhlig. Software Defined Networking: A Comprehensive Survey.Proceedingsof IEEE., 103(1), January 2015.

[3] Nick McKeown.SDN and Streamlining the Plumbing.Keynote speech at COMSNET2014, January 2014.

[4] https://www.opennetworking.org/sdn-definition/

[5] Scott Shenker.The future of networking and the past of protocols. Keynote speechat Open Networking Summit 2011, October 2011.

[6] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown. Im-plementing an OpenFlow switch on the NetFPGA platform.Proceedings of the 4thACM/IEEE Symposium on Architectures for Networking and Communications Sys-tems., June 2008.

[7] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo.SDN-WISE: Design, proto-typing and experimentation of a stateful SDN solution for WIreless NEtworks. InProc. of IEEE Infocom 2015, Hong Kong, April 2015.

[8] G. Bianchi, M. Bonola, A. Capone, and C. Cascone.OpenState: programmingplatform-independent stateful openflow applications inside the switch. ACM SIG-COMM Computer Communication Review, April 2014.

[9] M. Moshref, A. Bhargava, A. Gupta, M. Yu, and R. Govindan. Flow- level statetransition as a new switch primitive for SDN.In Proc. of ACM HotSDN, August2014.

[10] http://flowgrammable.org/sdn/openflow/.

[11] https://www.opendaylight.org.

[12] https://www.opennetworking.org/onos/.

[13] https://github.com/faucetsdn/ryu.

[14] https://www.open-kilda.org/.

[15] https://faucet.nz/.

[16] IEEE, "Internet of Things," 2014.

[17] http://standards.ieee.org/innovate/iot/

[18] L. Atzori, A. Iera, G. Morabito. "From 'smart objec't to 'social object': the next evolutionary step of the Internet of things " IEEE Cmmunications Magazine- Jan 2014.

[19] L. Atzori, A. Iera, G. Morabito, M. Nitti. "The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization" Computer Networks · November 2012- 56 (16) (2012)3594-3608

[20] MQTT Version 5.0, OASIS Std., March 2019.

[21] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, In ternet of things: A survey on enabling technologies, protocols, and applications, IEEE Communications Surveys & Tutorials 17 (4) (2015) 2347-2376.

[22] C. Diot, B. N. Levine, B. Lyles, H. Kassem, D. Balensiefen, Deployment issues for the IP multicast service and architecture, IEEE network 14 (1)(2000) 78-88.

[23] L. Atzori, A. Iera, G. Morabito, Sociocast: A new network primitive for IoT, IEEE Communications Magazine 57 (6) (2019) 62-67.

[24] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodol molky, S. Uhlig, Software-de_ned networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14-76.

[25] B. Lantz, B. Heller, N. McKeown, A network in a laptop: rapid prototyping for software-defined networks, in: Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 2010, p. 19.

[26] ON.LAB, Introducing ONOS - a SDN network operating system for service providers (2014).

[27] S.-H. Shen, E_cient SVC multicast streaming for video conferencing with SDN control, IEEE Transactions on Network and Service Management.

[28] B. Knutsson, H. Lu, W. Xu, B. Hopkins, Peer-to-peer support for massively multiplayer games, in: IEEE INFOCOM 2004, Vol. 1, IEEE, 2004.

[29] X. S. Sun, Y. Xia, S. Dzinamarira, X. S. Huang, D.Wu, T. E. Ng, Republic: Data multicast meets hybrid rack-level interconnections in data center, in: 2018 IEEE 26th International Conference on Network Protocols (ICNP), IEEE, 2018, pp. 77-87.

[30] X. Li, Y.-C. Tian, G. Ledwich, Y. Mishra, X. Han, C. Zhou, Constrained optimization of multicast routing for wide area control of smart grid, IEEE Transactions on Smart Grid.

[31] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, N. D. Georganas, A survey of application-layer multicast protocols, IEEE Communications Surveys and Tutorials 9 (1-4) (2007) 58-74.

[32] S. Islam, N. Muslim, J. W. Atwood, A survey on multicasting in software-defined networking, IEEE Communications Surveys & Tutorials 20 (1) (2017) 355-387.

[33] Z.lSaeed, I. Ahmad, I. Hussain, Multicasting in software defined networks: A comprehensive survey, Journal of Network and Computer Applications 104 (2018) 61-77.

[34] A. Adams, J. Nicholas, W. Siadak, RFC 3973, protocol independent multicast-dense mode (PIM-DM): Protocol specification (revised), Tech. rep. (2005).

[35] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, RFC 3376, Internet Group Management Protocol, version 3, Tech. rep. (August 2006).

[36] H. Holbrook, B. Cain, RFC 4607, Source-Speci_c Multicast for IP, Tech. rep. (August 2006).

[37] A. Venkataramani, et al., MobilityFirst: a mobility-centric and trustworthy internet architecture, ACM SIGCOMM Computer Communication Review 44 (3) (2014) 74-80.

[38] C. A. Marcondes, T. P. Santos, A. P. Godoy, C. C. Viel, C. A. Teixeira, Castow: Clean-slate multicast approach using in-advance path processing in programmable networks, in: 2012 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2012, pp. 000094-000101.

[39] M. Zhao, B. Jia, M. Wu, H. Yu, Y. Xu, Software defined network-enabled multicast for multi-party video conferencing systems, in: 2014 IEEE International Conference on Communications (ICC), IEEE, 2014, pp. 1729-1735.

[40] D. Katsaros, N. Dimokas, L. Tassiulas, Social network analysis concepts in the design of wireless ad hoc network protocols, IEEE network 24 (6) (2010) 23-29.

[41] Y. Zhu, B. Xu, X. Shi, Y. Wang, A survey of social-based routing in delay tolerant networks: Positive and negative social effects, IEEE Communications Surveys & Tutorials 15 (1) (2012) 387-401.

[42] K. W. et al., Exploiting small world properties for message forwarding in delay tolerant networks, IEEE Transactions on Computers 64 (10) (2015) 2809-2818.

[43] K. Wei, X. Liang, K. Xu, A survey of social-aware routing protocols in delay tolerant networks: applications, taxonomy and design-related issues, IEEE Communications Surveys & Tutorials 16 (1) (2013) 556-578.

[44] M. Xiao, J. Wu, L. Huang, Community-aware opportunistic routing in mobile social networks, IEEE Transactions on Computers 63 (7) (2013) 1682-1695.

[45] D. J. Watts, Networks, dynamics, and the small-world phenomenon, American Journal of sociology 105 (2) (1999) 493-527.

[46] W. Gao, Q. Li, B. Zhao, G. Cao, Multicasting in delay tolerant networks: a social network perspective, in: Proceedings of ACM MobiHoc, ACM, 2009, pp. 299-308.

[47] W. Gao, Q. Li, B. Zhao, G. Cao, Social-aware multicast in disruption-tolerant networks, IEEE/ACM Transactions on Networking (TON) 20 (5) (2012) 1553-1566.

[48] X. Hu, T. H. Chu, V. C. Leung, E. C.-H. Ngai, P. Kruchten, H. C. Chan, A survey on mobile social networks: Applications, platforms, system architectures, and future research directions, IEEE Communications Surveys & Tutorials 17 (3) (2015) 1557-1581.

[49] J. Fan, J. Chen, Y. Du, W. Gao, J. Wu, Y. Sun, Geocommunity-based broadcasting for data dissemination in mobile social networks, IEEE Transactions on Parallel and Distributed Systems 24 (4) (2013) 734-743.

[50] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69-74.

[51] L. Atzori, C. Campolo, B. Da, R. Girau, A. Iera, G. Morabito, S. Quattropani, Enhancing identifier/locator splitting through social internet of things, IEEE Internet of Things Journal 6 (2) (2018) 2974-2985.

[52] OpenFlow switch specification - version 1.3.1 Open Networking Foundation (ONF) (September 2012).

[53] J. Hui, P. Thubert, Compression format for IPv6 datagrams over ieee 802.15.4-based networks, RFC 6282 (September 2011).

[54] A. Ayadi, D. Ros, L. Toutain, TCP header compression for 6LoWPAN, Internet Draft (draft-aayadi-olowpan-tcphc-00), work in progress.

[55] C. Gomez, A. Arcia-Moret, J. Crowcroft, TCP in the Internet of Things: from ostracism to prominence, IEEE Internet Computing 22 (1) (2018) 29-41.

[56] L. Atzori, C. Campolo, B. Da, R. Girau, A. Iera, G. Morabito, S. Quattropani, Smart devices in the social loops: Criteria and algorithms for the creation of the of the social links, Future Generation Computer Systems 97 (2019) 327-339.

[57] http://mininet.org/overview/

[58] N Gude, T Koponen, J. Pettit, B. Pfaff, M. Casado, N McKeown, S, Shenker. "NOX: Towards an Operating System for Networks" Computer-Communication Networks 2008.

[59] E. Longo, A. Redondi, M. Cesana, A. Arcia-Moret, and P. Manzoni. (2019, Oct.) Mqtt-st: a spanning tree protocol for distributed mqtt brokers. Available: https://arxiv.org/abs/1911.07622

[60] *Karger, David "Global Min-cuts in RNC and Other Ramifications of a Simple Mincut Algorithm". Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms (1993).*

[61] http://mosquitto.org/

[62] https://www.eclipse.org/

[63] https://iot.eclipse.org/

[64] L. Atzori, C. Campolo, A. Iera, G. M. Milotta, G. Morabito, S. Quattropani "*Sociocast: Design, Implementation and Experimentation of a New Communication Method for the Internet of Things"* 2019 IEEE 5th World Forum on Internet of Things (WF-IoT)

[65] L. Atzori, C. Campolo, A. Iera, G. M. Milotta, G. Morabito, S. Quattropani: "*SDN-based Sociocast Group Communications in the Internet of Things"* ITU J-FET 2020