Pós-Graduação em Ciência da Computação

Daniel Bezerra

SOFTWARE-DEFINED CLUSTER: An SDN-Based Cluster Architecture for IoT

Recife
2018

Daniel Bezerra

SOFTWARE-DEFINED CLUSTER: An SDN-Based Cluster Architecture for IoT

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação.

Área de Concentração: *Internet* das Coisas
Orientadora: Judith Kelner
Coorientador: Rafael Roque Aschoff

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

B574s Bezerra, Daniel
Software-Defined Cluster: an SDN-Based Cluster architecture for IoT / Daniel Bezerra. – 2018.
53 f.: il., fig.

Orientadora: Judith Kelner.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Redes de computadores. 2. Internet das coisas. I. Kelner, Judith (orientadora). II. Título.

004.6 CDD (23. ed.) UFPE- MEI 2018-135

Daniel Bezerra

Software-Defined Cluster: An SDN-Based Cluster Architecture For IoT

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 11/09/2018.

ACKNOWLEDGEMENTS

Thank God for being the source of my strength and knowledge.

Thanks to all my family, especially my parents and sister, for being my daily joy and my source of energy. For guiding and supporting me through good and bad times. I appreciate efforts each one of you has done on my behalf. Thank you for each motivation word you have given me, they always encouraged me to move on and continuously grow in life.

To my friends, thanks for the patience, motivation, and support during these years. Thank you for helping me through troubled times and sharing the moments of fun and happiness.

To my friends and co-workers in the GPRT research group, thank you for all the knowledge I gained through the years. Each one of you added something new to me, in either being a better professional or a better person. I'll carry what I learned with you forever.

To all the teachers who contributed to my run, especially professor Djamel and Judith, for both always being attentive advisers and for having accompanied me since before the beginning of this work. Thanks for teaching and helping me to become a better professional.

ABSTRACT

The Internet of Things (IoT) represents an emerging paradigm that has gained relevance in both academia and industry. However, there are still adversities to be exploited in IOT applications, e.g., issues related to security, heterogeneity, and interoperability. Some IoT protocols use a broker to allow communication between heterogeneous devices and simplify network management. However, the broker is a single point of failure in the network. To circumvent this problem, some brokers support the construction of a cluster, allowing communication between devices with high availability. However, this solution depends on proxies or load balancers to manage the clients' connections, i.e., there is still a single point of failure. This work proposes and implements the Software Defined Cluster (SDC), an SDN application that offers an alternative to clustering. SDC manages the connections between devices and brokers, through SDN, allowing high availability communication that avoids the central point of failure in IoT networks.

Key-words: Internet of Things. Clustering. Broker. IoT Protocols.

RESUMO

A Internet das Coisas (IoT) representa um paradigma emergente que ganhou relevância tanto na academia quanto na indústria. No entanto, ainda existem adversidades a serem exploradas em aplicações IOT, por exemplo, questões relacionadas à segurança, heterogeneidade e interoperabilidade. Alguns protocolos de IoT usam um *broker* central para permitir a comunicação entre dispositivos heterogêneos e simplificar o gerenciamento da rede. No entanto, o *broker* é um ponto único de falha na rede. Para contornar esse problema, alguns *brokers* suportam a construção de *clusters*, permitindo a comunicação entre dispositivos com alta disponibilidade. No entanto, essa solução depende de *proxies* ou balanceadores de carga para gerenciar as conexões dos clientes, ou seja, ainda há um ponto único de falha na rede. Este trabalho propõe e implementa o *Software Defined Cluster* (SDC), uma aplicação SDN que oferece uma alternativa ao clustering. O SDC gerencia as conexões entre dispositivos e **brokers**, por meio do SDN, permitindo uma comunicação de alta disponibilidade que evita o ponto único de falha nas redes IoT.

Palavras-chaves: *Internet* das Coisas. *Clustering*. *Broker*. Protocolos IoT.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
DDS	Data Distribution Service
DTLS	Datagram Transport Layer Security
HA	High Availability
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
MQTT	Message Queue Telemetry Transport
OMG	Object Management Group
ONF	Open Networking Foundation
Pub/Sub	Publish/Subscribe
QoS	Quality of Service
REST	Representational State Transfer
SDC	Software-Defined Cluster
SDN	Software-Defined Networking
SPoF	Single Point of Failure

CONTENTS

1 INTRODUCTION

The Internet of Things (IoT) is a paradigm that consists of pervasive physical devices that can connect and interact among themselves. "Things" are set to share data and exchange information according to well-established protocols to reach a goal (ATZORI; IERA; MORABITO, 2010; CHEN; HU, 2013).

Today, the Internet contributes positively to human activities in business, education, communication, science, and other areas. With things connected, the environment sensing, the information gathering, and the data analysis and storage will turn the Internet into an even greater tool for information generation and propagation. As a result, the number of connected things is expected to grow exponentially shortly (WHITMORE; AGARWAL; XU, 2015; QIN et al., 2016; CHEN; HU, 2013; GRIECO et al., 2014). Ericsson states that IoT is expected to surpass mobile phones as the largest category of connected devices in 2018 (ERICSSON, 2016). This addition of IoT devices into the Internet will also have an impact on web traffic. According to Cisco, global IP networks will support up to 10 billion new devices and connections, increasing from 16.3 billion in 2015 to 26.3 billion by 2020 (CISCO, 2016).

Connected Internet objects are diverse and range from everyday devices, e.g., home appliances and automobiles, to industrial grade robots. While a smart home may provide a small diversity of IoT devices, there are resource-demanding scenarios that present a diversified range of IoT devices. A case in point is Industrial Robotics.

Overall, industrial applications, e.g., autonomous vehicles, industrial robots, and medical robots, exhibit strong scenario similarities and can stand to benefit from IoT advances. More specifically, combining the monitoring, automation, and control of applications with network technologies enables the industrial IoT to take a significant leap in productivity and trigger economic growth (ERICSSON, 2015).

One of the critical features of IoT systems is the support for interoperability between heterogeneous devices. This characteristic is in line with the overall goal of IoT "*To connect everything and everyone everywhere to everything and everyone else*" (GRIECO et al., 2014). Different devices mean different vendors and architectures, therefore, the connection among them is challenging for the absence of a common communication channel or compatible hardware. Some IoT protocols have an intercommunication entity, commonly referred to as broker, to overcome the variation among devices and allow data exchange. The broker acts as an individual that intermediates the communication among peers in an IoT scenario, independent of the type of device, technology, or message size.

As IoT is a growing technology, it still presents some issues in different areas:

- **Management:** IoT devices generate massive amounts of data that need to be processed and stored. The network must handle the heterogeneous and voluminous data from both personal and enterprise traffic as IoT becomes more widely distributed (LEE; LEE, 2015);

- **Standardization:** Standardization plays an essential role in IoT development. However, IoT rapid expanding in our society creates difficulties in the standardization processes. To standardize an IoT technology improves the usage and development of the applications and services. The main challenge is the coordination efforts needed to englobe applications, services, and devices from different countries (XU; HE; LI, 2014);

- **Energy consumption:** Due to the increase in data rates, network services, and connected edge-devices, the energy consumption from the network is increasing at high speed. Therefore, the adoption of IoT systems tends to increase the network energy consumption significantly (KHAN et al., 2012);

- **Security:** In an IoT environment, the system should be able to handle the data exchange without data loss due to congestion or delays in the network and ensure security, preventing external intervention and monitoring (KHAN et al., 2012).

There are different security levels in IoT, from the physical layer to the application layer. IoT networks follow some security constraints, such as:

- **Resilience:** If an IoT device is compromised, the IoT system must recover from failure and avoid the general network failure;

- **Authentication:** Identification and validation of the peers an IoT device is connected with;

- **Authorization:** Ensures that only authorized users can reach services and devices within the network;

- **Availability:** The system ensures that services will be available mos of the time even in the presence of attacks and power loss.

In this work, a resilience issue related to the IoT broker is investigated. Although the broker provides reliability in message delivering it is still a Single Point of Failure (SPoF) in the network architecture. As discussed further in this work, it is an entity that manages and interconnects different devices in the same network, but it is the central unit that holds the communication working. If something happens to it, the system may collapse.

A commonly applied solution is to assemble brokers as a cluster. A cluster of brokers is a workaround for the central entity issue in the architecture as well as a fault-tolerant solution with a high availability service. However, to obtain such benefits, the brokers must be configured to work with other applications, e.g., load balancer, to work correctly.

Not to mention that the bigger the architecture, the bigger the resources used to create communication with the cluster. This work proposes an alternative solution to the SPoF issue by exploring the advantages of Software-Defined Networking (SDN). Since SDN offers advantages as logically centralized network control and better end-user experience, it is a promising tool to develop the broker abstraction in the network.

This work does not propose a better or overlapping solution than clustering but a new alternative to avoid the SPoF issue.

1.1 OBJECTIVES

The main objective of this dissertation is to propose a novel solution to avoid the SPoF issue in broker-based IoT networks.

The secondary objectives consist of:

- Analyzing the limitations of current broker-based IoT networks.
- Elaborating a solution able to create a similar structure of a cluster but without the same dependencies.
- Validating the implementation of the SDN-based cluster as an alternative to the current cluster-based solution.

1.2 WORK ORGANIZATION

This dissertation is organized as follow. Chapter 2 defines the concepts used throughout the work. It details the IoT protocols and broker as well as the SDN principles. The main contributions that form the basis of this dissertation are appointed at Chapter 3. The work discusses the aspects regarding IoT security issues with and without SDN-based solutions and how SDN spreads the solution possibilities in IoT. Chapter 4 details the developed architecture in this work: The Software-Defined Cluster (SDC), an SDN-Based Cluster. In Chapter 5 the SDC behavior in an IoT environment is analyzed and compared with the traditional clustering architecture. Finally, the conclusion of this work, exhibiting the overall results of SDC is presented in Chapter 6.

2 THEORETICAL BACKGROUND

This chapter details the fundamental concepts in the context of this work. Section 2.1 provides an insight of the commonly used IoT protocols, their main features, and architectures. Section 2.2 is dedicated to present and explore message brokers and how they perform in an IoT environment. The broker cluster is detailed in Section 2.3. The scenario environment and tools used in this work are presented in Section 2.4. At Section 2.5, the general concept of SDN is detailed. Finally, in Section 2.4.3, the scenario adopted in this work is presented.

2.1 IOT PROTOCOLS

This section is divided into two parts. First, we present the Publish/Subscribe (Pub/-Sub) paradigm, detailing the operation and topology of this model, followed by a brief explanation of broker-based protocols and brokerless protocols.

2.1.1 Publish/Subscribe Paradigm

The Pub/Sub message paradigm is an alternative to the traditional client-server model to provide one-to-many message distribution. It is composed of three main components: Publishers, entities responsible for message sending; Subscribers, components that receive the messages; and Broker, the central unity of mediation.

To summarize, in Figure 1, the Publisher sends a message to the subscribers without knowledge of which subscriber may receive the message. Similarly, the subscribers express interest in receiving a particular kind of content unaware of the message sender. The Broker works as a manager by filtering received messages from Publishers and distributing them to the Subscribers.

The communication between clients and broker are through topics. A topic is a logical channel between two peers. The subscribers assigned to a topic receive all messages published in this channel, while publishers are responsible for sending messages through a

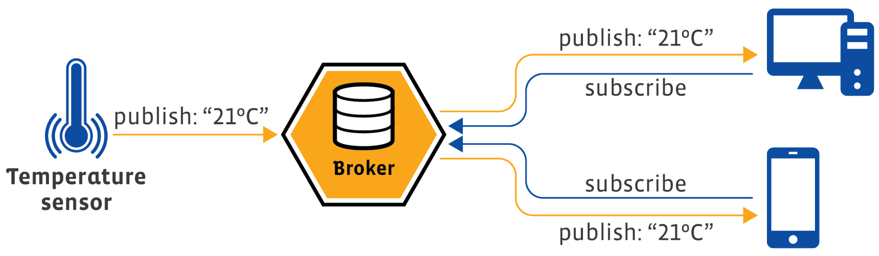


Figure 1 – Classical Publish/Subscribe architecture

previously defined topic configuration. For instance, a client that wishes to publish a data in string format defines a specific topic which will only transmit string types. Likewise, a client that subscribes to such topic only expects to receive string data types.

In contrast with the client-server model, where the end-points communicate with each other directly, the Pub/Sub messaging is loosely coupled (EUGSTER et al., 2003), e.g., the publishers and subscribers have almost no knowledge about each other. Entities decoupling may be decomposed in three dimensions: *Space Decoupling*, when both of them do not know each other's location in the network (IP address and port number); *Time decoupling*, when they do not need to exchange messages between each other at the same time; and *Asynchronous Decoupling*, when entities current work is not interrupted while publishing and consuming messages as they are not blocking events.

The Pub/Sub paradigm also provides scalability higher than the client-server approach (ENTERPRISE, 2017) as operations on the broker can be highly parallelized and processed in an event driven manner. The loosely coupled messaging also benefits scalability since clients operate independently from each other.

2.1.2 Broker-based Protocols

In a broker-based environment, every application or client is connected to the central broker. There is no direct message exchange between applications. The broker manages all communication in the network.

2.1.2.1 Advanced Message Queue Protocol

The Advanced Message Queuing Protocol (AMQP) is an open messaging protocol based on the Pub/Sub message pattern. It is an OASIS[1] standard and it has been commonly used in the financial sector since it requires extremely high levels of performance, throughput, scalability, reliability, and manageability for services as trading and banking systems (VINOSKI, 2006). However, the protocol has some peculiarities. As depicted in Figure 2, an AMQP broker is composed by two main components (O'HARA, 2007):

- **Queues** are the main core concept in AMQP. They store messages and send them to message consumer nodes. A queue can be programmed to store messages in disk or memory with a configurable dropping rate.
- **Exchanges** are entities that receive the messages and distribute them to the respective queues. The routing algorithm designed to allocate the messages depends on the exchange type and rules.

[1] https://www.oasis-open.org/

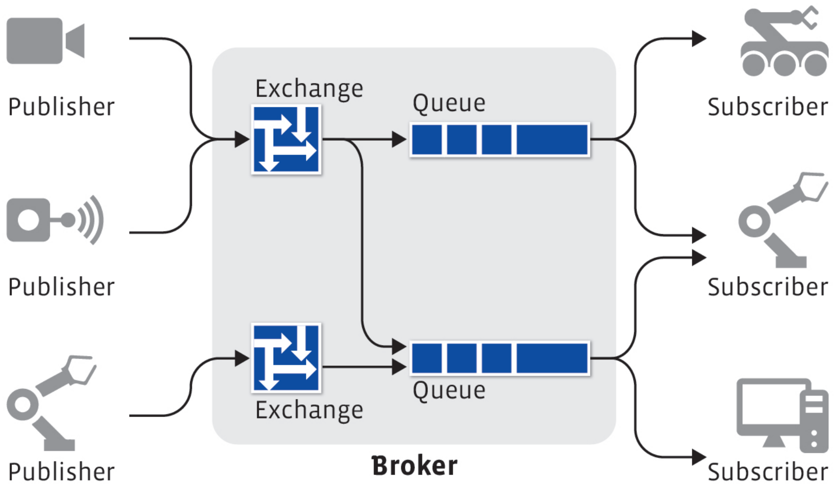


Figure 2 – AMQP broker model

2.1.2.2 Message Queue Telemetry Transport

Message Queue Telemetry Transport (MQTT) is a messaging transport protocol developed by IBM[2] and standardized in 2013 by Oasis[3]. It is a lightweight, simple and open protocol, ideal for constrained environments and small bandwidth networks with limited processing capabilities (LOCKE, 2010). These characteristics, alongside with the MQTT design, make it an optimal protocol for IoT contexts.

It works over TCP using the Pub/Sub message pattern. Similarly to AMQP, it manages messages in queues. However, MQTT does not contain exchanges. All messages are sent directly into topics which clients express their interest in publishing/subscribing.

Although MQTT works over TCP, it presents some Quality of Service (QoS) levels. As TCP already guarantees the packet delivery between peers, MQTT provides an Application Level QoS for message delivery. Depending on the determined QoS, nodes may delete the message as soon it is sent to the subscribers or store the messages to send in a future occasion when the subscriber is available. In summary, MQTT handles retransmission and guarantees the delivery of the message, regardless of how unreliable the network is. Clients are granted the freedom to set the QoS level of the messages they wish to send according to the application and network conditions b . The QoS levels are the following:

- **At most once (level 0)**: The broker/client will deliver the message once, with no confirmation.

- **At least once (level 1)**: The broker/client will deliver the message at least once, with confirmation required.

[2] https://www.ibm.com/br-pt/
[3] https://www.oasis-open.org/

- **Exactly once (level 2)**: The broker/client will deliver the message exactly once by using a four-step handshake.

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

2.1.3 Brokerless Protocols

In a brokerless environment, each client can reach one another and communicate directly. As there is no central broker to manage the connections, for an application to connect with another, it has to know the network address of the desired entity.

2.1.3.1 Constrained Application Protocol

As IoT environments may contain limited-resource devices, the Internet Engineering Task Force (IETF)[4] developed the Constrained Application Protocol (CoAP), a web transfer protocol for constrained nodes and constrained networks (SHELBY; HARTKE; BORMANN, 2014). It is considered to be a replacement for the Hypertext Transfer Protocol (HTTP) in an IoT network. HTTP uses Representational State Transfer (REST) as a communication interface between two nodes on a server-client architecture (MUMBAIKAR; PADIYA et al., 2013). However, for IoT applications, the usage of REST would imply higher power consumption and overhead. Hence, CoAP allows limited nodes to use RESTful services within their limitations.

CoAP benefits include easy integration with HTTP allowing constrained devices to communicate with each other as well as request REST services to an HTTP server through REST-CoAP proxies that can easily convert the messages. The protocol also presents a low header overhead, ideal for IoT applications, and provides service and resource discovery through multicasting. However, CoAP presents some drawbacks. Although the protocol provides a mechanism to guarantee the message delivery, it works over UDP. Depending on the application and the network conditions, UDP may degrade the message transferring among peers. CoAP also does not have a standard security feature. It leans on Datagram Transport Layer Security (DTLS) which adds an initial handshake and a little overhead on each datagram.

2.1.3.2 Data Distribution Service

The Data Distribution Service (DDS) protocol is an Object Management Group (OMG)[5] standard. It is a publish-subscribe middleware protocol for distributed application communication and integration. DDS intends to provide low-latency data connectivity, extreme reliability, and scalable architecture, which is ideal for IoT environments.

[4] https://www.ietf.org/
[5] http://www.omg.org/

Unlike the previous protocols, DDS is data-centric. Data-centricity means that the application knows what data it stores and controls how to share that data. Meanwhile, message-centricity is about direct point-to-point messaging through patterns like request/response and publish/subscribe.

In contrast with other publish/subscribe protocols (e.g., MQTT and AMQP), DDS provides a brokerless architecture for data transmission. The protocol offers different QoS levels (such as granting data reliability, durability, availability, security) to keep an efficient distribution of data in distributed systems.

2.1.4 Broker vs Brokerless

The previous sections provided information regarding protocols that work with either broker or brokerless architecture. Each one has benefits and downsides that must be taken into consideration when deploying an IoT network.

Certainly, each application and scenario has an ideal architecture. A broker-based architecture is commonly represented as a star architecture (Figure 3). As previously stated in Section 2.1.1, the broker allows space and time decoupling. These advantages from the Pub/Sub model are made possible because of the broker. The broker acts as the mediator of all message exchanges among the network peers since all clients connect to the broker. The abstraction of recognizing other peers in the network allows the connected clients to operate and swap messages identifying just the broker's IP address. Moreover, the clients do not have to act at the same time due to the broker storing messages in queues. It allows, at some level, the broker to be resilient to client errors (Zeromq, 2011), since past messages sent from the client will be kept in the queue even if the client connection fails.

Although a broker is easily deployable and scalable, the system maintenance gets more complex as more brokers and devices join the network. The broker is also known as the bottleneck of the network since it processes all exchanged messages among nodes as well as it is a SPoF in the system.

Contrasting with a broker-based network, the brokerless-based network is represented as a mesh of connections where every device in the network can talk directly with each other. The brokerless structure does not present the issues of the broker-based architecture, as the single point of failure and the bottleneck during message exchange. However, it is more complex to organize and manage (LILIS; KAYAL, 2018).

According to (RabbitMQ, 2010), three points that might be needed in a brokerless network are:

- **Discovery** is the problem of finding out the other peers in the network and what they propose to do in the system. It does not have a proper algorithm or method to be implemented, and it depends on the network and application.

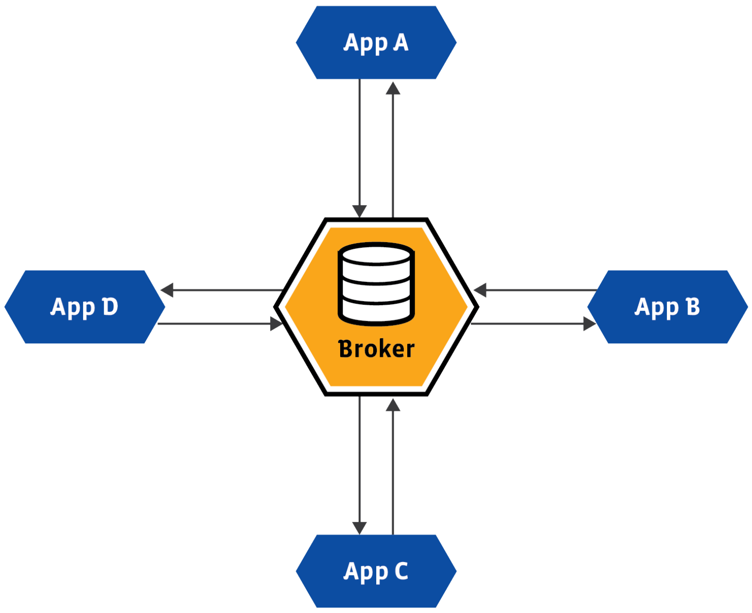


Figure 3 – Star architecture in a broker-based system

- **Availability** is the problem of dealing with the system, subsystem or equipment in a specified operable state at the beginning of an operation, when it is called for an unknown time. For instance, if a set of peers is down and did not receive the sent messages, they should be kept in the sender, or they should be discarded? The nature of the application should be considered and implemented by the system developer.

- **Management** of connections among peers can get complex and hard to manage. Levels of agreement such as "Which peer is connected to one another?" and "Which clients are allowed to connect with each other?" may be difficult to implement and manage the more you scale.

The key advantage of the brokerless system is the very low communication latency since there is no intermediary party between peers. It is an advantage of great importance for real-time systems and automation processes.

IoT protocols are commonly based on one of the discussed architectures. There is not a correct answer of "Which one is the best?", the architecture depends on the implemented application and the desired objectives. In this work, our scenario, which will be detailed in Chapter 4, works under a broker-based system. It is a factory environment composed of heterogeneous devices such as sensors, robots, and cameras, that is managed by a central entity. In the next section, the broker used in this work is presented and detailed.

2.2 MESSAGE BROKER

A broker is a network element that applications communicate with by exchanging predefined messages. It mediates communications among devices and allows them to interact with each other even in heterogeneous networks. The MQTT and AMQP protocols have a central broker, as standard, to manage the communication.

There is a considerable amount of brokers able to operate in an IoT network. Some brokers and applications worth to mention are:

1. Mosquitto (LIGHT, 2013): An open source message broker that implements the MQTT protocol.
2. ActiveMQ (SNYDER; BOSNANAC; DAVIES, 2011): Open source messaging and Integration Patterns server that supports both AMQP and MQTT.
3. eMQTT (TEAM, 2012): Distributed, massively scalable, highly extensible MQTT message broker written in Erlang.
4. HiveMQ (ENTERPRISE, 2012): An MQTT broker tailored specifically for enterprises.
5. Apache Qpid (QPID, 2013): Apache Qpid makes messaging tools that speak AMQP and support many languages and platforms.

Another broker that is widely used for both academic researches and enterprises is the RabbitMQ (SOFTWARE, 2013). It was initially developed to work with AMQP and later with MQTT.

Although there are many brokers for a developer to implement in an application or network, all of them are a central entity, i.e., they are a SPoF in the network. To avoid this issue and provide High Availability (HA), the brokers usually form a cluster. Clustering creates a connection between two or more brokers that share status and data to work as one unique entity. However, this method does not have native support by all brokers, e.g., Mosquitto doesn't support clustering.

RabbitMQ is well-disseminated, and well-recognized by both industry and academia (PIVOTAL, 2014). It is multilingual and works with established IoT protocols. Therefore, it is the broker used in this dissertation.

2.3 RABBITMQ CLUSTER

The RabbitMQ broker is a well-disseminated message middleware for IoT applications. The collection of brokers is named a cluster. A cluster is mainly formed to avoid a single point of failure and allow HA features in the network, which will be detailed further ahead.

In a cluster, RabbitMQ nodes must know the IP address of each other and resolve the hostname of each node. This allows the cluster creation, management, and communication.

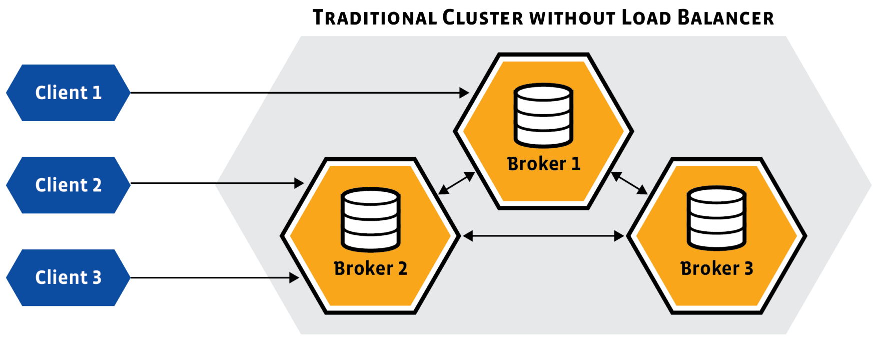


Figure 4 – The brokers exchange messages and state among them, allowing clients to connect with any broker of the cluster

The nodes also share and replicate data and state. By default, the exception for the replication is the message queue, although it can be visible from other nodes. RabbitMQ provides a HA mode which allows the queues replication/mirroring.

After the cluster is deployed, a client can connect to any cluster node at a time (Figure 4). In HA configurations, the queue mirroring is transparent to the client. In case of failure, the client should be able to reconnect to other broker nodes. Therefore, the client needs to know the address of every cluster member to complete the recovery process. To avoid to hard-code node addresses in the client's applications, it is recommended to build a DNS service or a Load Balancer to manage the connections and ensure flexibility.

The default cluster configuration does not allow queues to share data among nodes even though they are visible to the other brokers. In HA mode, the queues are mirrored across all nodes to ensure data redundancy. Each mirrored queue is defined as a *master queue* while the replicates are the *mirrors*. Every operation for a queue is first executed in the master queue and then propagated to the mirrors. This replication is transparent to the clients.

Although clients are connected to just one node, the replication of data and states among the nodes does not distribute the load across them. As the focus, in HA mode, is to enhance availability and resilience, all cluster brokers participate in all system operations. With the queue message sharing, the client reconnection after a node fails will not inflict in data loss.

2.4 SIMULATION TOOLS

In this section, we present the simulator used to create a heterogeneous IoT scenario and the framework utilized for the communication of the devices into the simulation.

2.4.1 ROS

The Robot Operating System[6] (ROS) is a flexible framework for writing robot software. ROS is a modular environment able to integrate several devices, providing a flexible, heterogeneous and collaborative architecture. According to (QUIGLEY et al., 2009), the principles of ROS can be summarized as:

- Peer-to-peer: ROS allows the communication among several participants using a peer-to-peer topology. To provide a lookup mechanism for the participants to find each other and establish a connection between peers, ROS implements the main entity known as *master*;
- Multilingual: ROS was designed to language-neutral. It currently supports four different programming languages: C++, Python, Octave, and LISP;
- Tools-based: To manage the complexity of ROS, there is a large number of tools to build and run the ROS components. Some tasks managed by the tools include navigating the source code tree, support for get and set configuration parameters, visualize the peer-to-peer connection topology, and so on;
- Thin: All algorithm development occurs in standalone libraries that have no dependencies on ROS. The ROS performs modular builds throughout the source code following a “thin” ideology;
- Free and Open-Source: ROS source code is available and it is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects.

As depicted in Figure 5, ROS presents well-defined structures in the architecture. **Nodes** are the software modules responsible for computation. Nodes communicate with each other through **Messages**, which are structured data types. Messages are transferred through **Topics**. If a node needs to send data, it publishes such data on a topic. Likewise, if a node wants to receive information from another node, it subscribes to a topic. As topic-based publish/subscribe works as a "broadcast" messaging among nodes, it is not appropriate for synchronous communication. For this reason, ROS provides **Services** that work as a request/response communication between nodes. To communicate among themselves, nodes need to register to a **ROS Master** that manages the connections and provides the lookup mechanism, as previously detailed.

2.4.2 Gazebo

Gazebo is a simulation tool designed to accurately reproduce the dynamic environments a robot may encounter. It can simulate robots in both indoor and outdoor environments and

[6] http://www.ros.org/

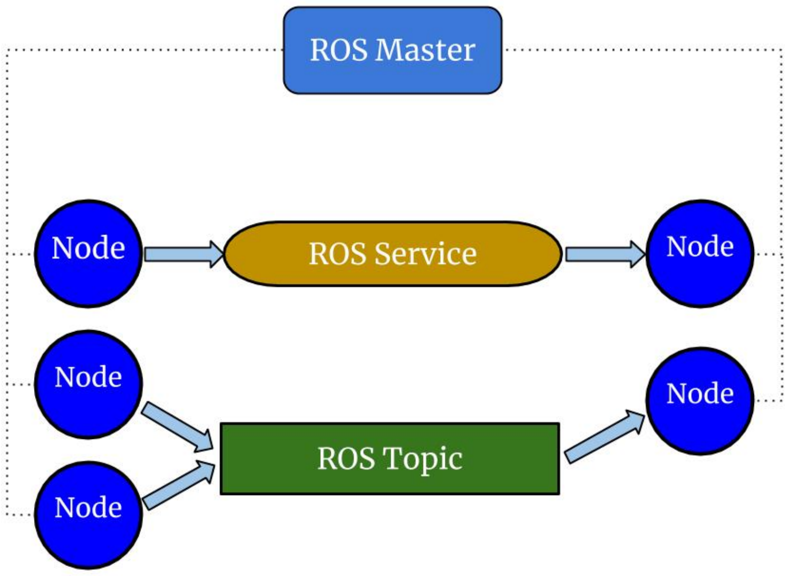


Figure 5 – ROS messaging structure

provides realistic sensor feedback. All simulated objects have mass, velocity, friction, and numerous other attributes that allow them to behave realistically when pushed, pulled, knocked over, or carried (KOENIG; HOWARD, 2004).

2.4.3 ARIAC Scenario

The goal of the Agile Robotics for Industrial Automation Competition (ARIAC) is to enable industrial robots to be more productive, autonomous, and responsive to the needs of shop floor workers[7].

In the competition, the teams would have to complete a set of tasks using the simulation provided. The agile aspect of the competition addresses four aspects:

- Failure identification and recovery, where the robot can detect failures in a manufacturing process and automatically recover from those failures.
- Automated planning, to minimize (or eliminate) the up-front robot programming time when a new product is introduced.
- Fixtureless environment, where robots can sense the environment and perform tasks on parts that are not in predefined locations.
- Plug and play robots, where robots from different manufacturers can be swapped in and out without the need for reprogramming.

In summary, the tasks provided by the competition are composed by a set of pieces that should be organized on a tray and be sent for assembling. The task consists of types, number, position, and orientation of each piece to be assembled.

[7] https://www.nist.gov

Figure 6 depicts the environment used in ARIAC. In the scenario, there are five different types of pieces to be assembled (pulley, disk, piston, gasket, and gear), a robotic arm to move the pieces, eight bins to dispose of the pieces, a conveyor belt to provide pieces that are not available in the bins, two autonomous ground vehicles (AGVs) to discharge the assembled kits, and a few sensors.

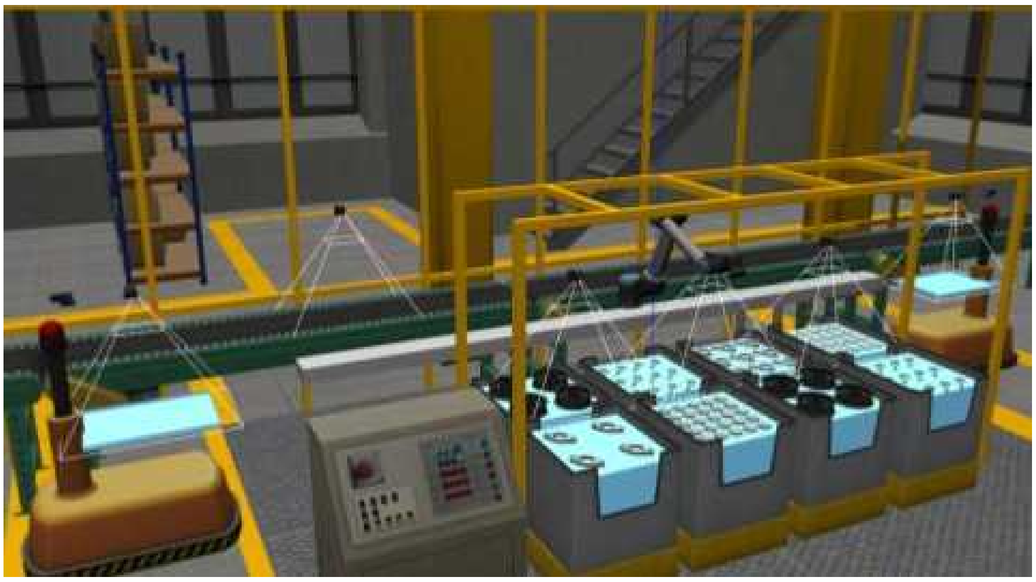


Figure 6 – ARIAC Scenario: Assemble kits of parts

There are two main files to start the simulation, a competitor configuration file, and a trial configuration file. The former holds information about the sensors as positions, orientations, amount, and types of sensors. The latter keeps the detailed tasks that should be performed and scenario configurations during simulation runtime, i.e., pieces located at the bin or belt, faulty parts, and kit orders.

At the end of each trial, the following performance results are displayed:

- **Total Score:** Represents the number of pieces deposited and delivered correctly in the kits. It combines the type, orientation, position of the pieces to reach a final score.
- **Total Process Time:** The total time taken from beginning to end of the trials, not necessarily when the tasks are completed.
- **Part Travel Time:** Represents the time that pieces are being moved by the robotic arm.
- **Time Taken 1:** Time for assembling the first order.
- **Time Taken 2:** Time for assembling the second order.

2.5 SOFTWARE-DEFINED NETWORKING

Networks have developed to the point of taking an essential part in our daily life in diverse areas, e.g., industry, education, and business. Although this growth has positive effects on the advancement and consolidation of protocols and techniques, it also narrowed the chances of a big impact from new researches. The introduction of new network research works becomes complex for either the enormous installed base of equipment and protocols or the absence of real-world realistic settings for protocol and technology deployment. These issues cause the ideas from the research community to go untried and untested (MCKEOWN et al., 2008).

To solve the issues related to the development of network research, the SDN technology was conceived. It is defined as an emerging network architecture where network control is decoupled from forwarding and is directly programmable (FUNDATION, 2012). It is divided into three layers (Figure 7):

- **Application Layer**: The layer in which developers deploy their applications. The main intention is to improve them through resources provided by SDN.
- **Control Layer**: The layer responsible for traffic decision throughout the network. The SDN applications interfere with the network structure and forwarding as it is configured according to the applications themselves.
- **Infrastructure Layer**: The network devices are located in this layer. Each device has configurations about the network traffic, from the above layer, and they are responsible for following the set of instructions and forward the data as preset.

Applying an SDN has advantages such as:

1. Logically Centralized Control: The control and data planes are decoupled, allowing the network intelligence to be logically centralized. As a result, programmers can build highly scalable and flexible networks which may vary according to the application needs.
2. Techniques Evolution: SDN provides APIs that allow developers to implement network services, e.g., forwarding and routing. Such traditional techniques can even be improved through programming.
3. Hardware abstraction: As the network control is decoupled from the network devices, their hardware can be simplified, cut costs, or specialized to reach a bigger performance.

According to (FUNDATION, 2012), SDN provides a dynamic network architecture that transforms current network backbones into service-delivery platforms. It makes the network more efficient by adapting the infrastructure to match the application specifications,

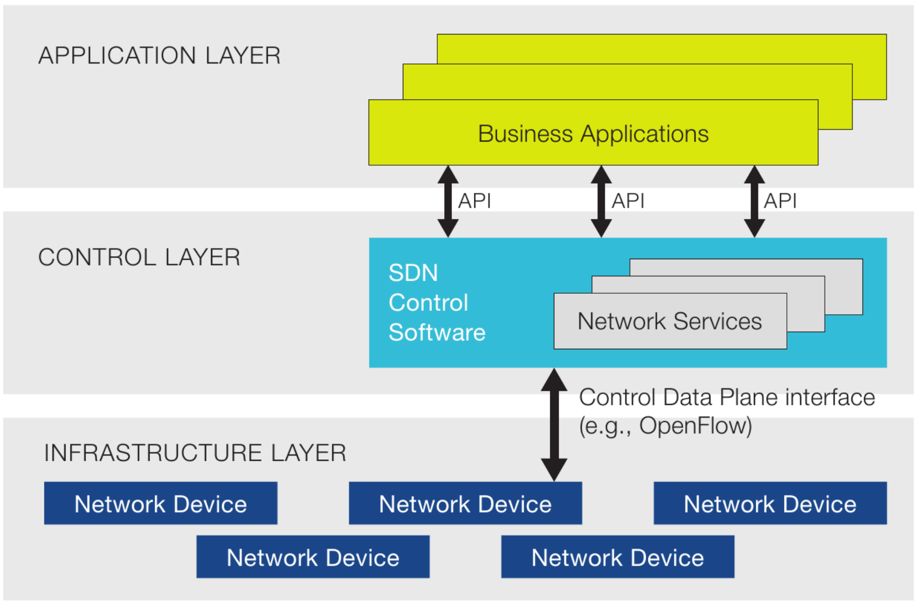


Figure 7 – SDN Architecture. Source: (FUNDATION, 2012)

while it can abstract obstacles for the network developments, e.g., network device's hardware. In this work, we apply SDN for forwarding control and routing decision. The scenario devices access to the broker will be transparent and controlled by the SDN control.

2.5.1 The Openflow Protocol

According to (FUNDATION, 2012), OpenFlow is the first standard implementation designed for SDN. It was created in the University of Standford as a solution for the need to conduct tests in real networks (MCKEOWN et al., 2008). As the current network infrastructure is settled with a variety of equipments and protocols, there is almost no practical way to test new protocols in realistic scenarios.

Realizing this issues, the authors created a protocol inspired by the principles of SDN, i.e., the decoupling of routing, controlling, and clients applications. The development of OpenFlow was initially focused on deploying it in the university campus for research purposes. However, industry embraced SDN and OpenFlow focusing on increasing network functionality at the same time they reduce costs and hardware complexity (LARA; KOLASANI; RAMAMURTHY, 2014).

The interest results in the creation of the Open Networking Foundation (ONF), in 2011, some of its members include AT&T, Google, Microsoft, and Verizon. Currently, ONF has more than 100 members. Their main objective is to lead the advancements in SDN as well as develop and standardize related technologies such as the OpenFlow protocol.

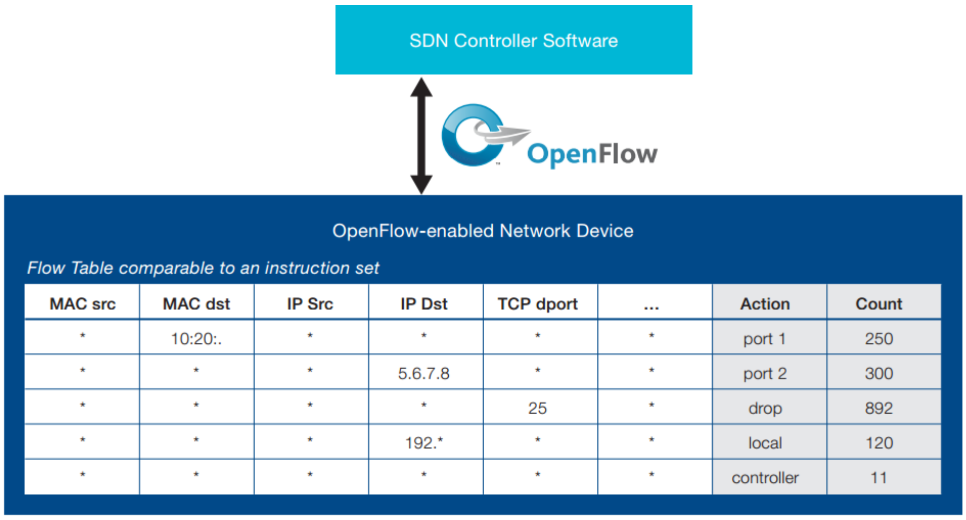
OpenFlow has been used in diverse areas and applications (e.g., network management, security features, network virtualization) and to deploy mobile systems (LARA; KOLASANI;

RAMAMURTHY, 2014).

2.5.2 Openflow Protocol Architecture

The OpenFlow is an open protocol that programs the flow table in switches and routers (MCKEOWN et al., 2008). After noticing similar functionalities and operations in enterprise switches, the OpenFlow was formed to work with flow tables without the need for companies to expose their equipment's source code.

(FUNDATION, 2012) compares OpenFlow with an instruction set of a CPU. Software applications can access, through OpenFlow, a set of basic configurations (Figure 8) that can be programmed to control the forwarding plane of network devices, such as switches and routers, both physical and virtual.


SDN Controller Software
OpenFlow
OpenFlow-enabled Network Device
Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Figure 8 – OpenFlow instruction set. Source: (FUNDATION, 2012)

There are three main components of a SDN architecture (Figure 9): A Controller, one or more SDN switches, and a secure channel. The Controller is a software that manages the network components by adding and removing flow-entries from the flow table and interfacing SDN applications and network components. An OpenFlow Switch is a piece of equipment that follows the rules set by the controller to forward data. Within the switch, there are the flow table, where are stored the flow-entries, and a secure channel that is the direct communication link between the switch and the controller.

As OpenFlow allows the network programming, an OpenFlow-based SDN architecture provides granular management of the network, enabling the response of real-time changes at the application, user, and session levels, contrasting IP-based routing that does not provide this level of control (FUNDATION, 2012).

2.5.3 Floodlight Controller

The Controller is one of the main components of the SDN architecture. It provides a global view of the network and can control access, QoS, security, and other policies. In this work, the chosen controller is the Floodlight for its modular architecture and easy deployment of new modules.

The SDN controller was released in 2012 and drew the attention of developers that wished more control in their applications network. The Floodlight Open SDN Controller is an enterprise-class, Apache-licensed, Java-based OpenFlow Controller (FLOODLIGHT, 2012b).

Floodlight is designed to work with heavily concurrent systems, such as data centers and enterprises networks (BHOLEBAWA; DALAL, 2018). It controls the network through routers and switches that support the OpenFlow protocol. According to (FLOODLIGHT, 2012b), the main features of the controller are the following:

- Floodlight supports OpenFlow, i.e., it can work with both physical and virtual OpenFlow-enabled switches.

- It is Apache-licensed, i.e., users have the freedom to use it for any purpose.

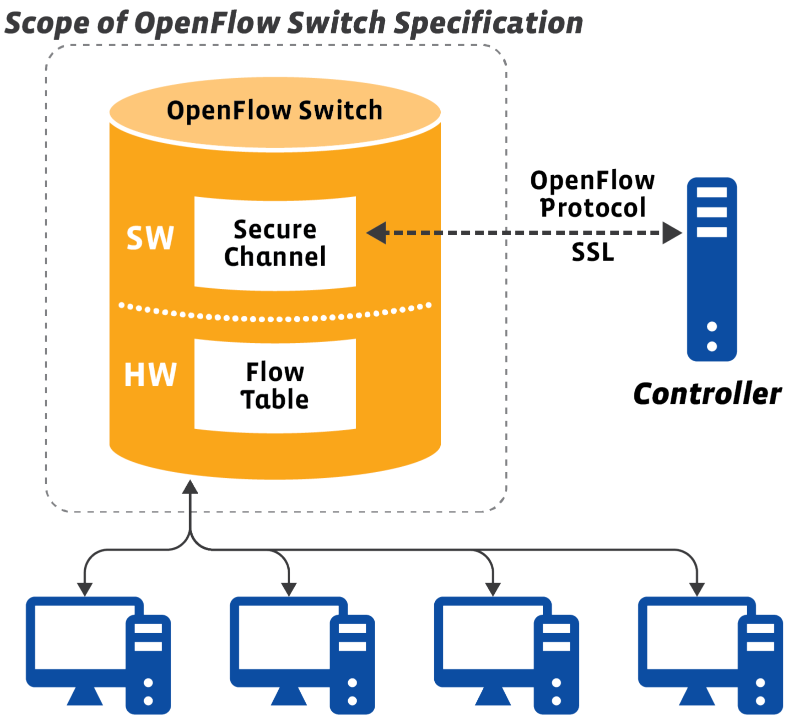


Figure 9 – OpenFlow switch. A remote controller controls the flow table through a Secure Channel. Adapted from: (MCKEOWN et al., 2008)

- Floodlight has an open-community of developers to maintain, enhance and extend the code.
- It is easy to use as they created extended documentation to build and run Floodlight modules.
- Floodlight is tested and supported by professional programmers.

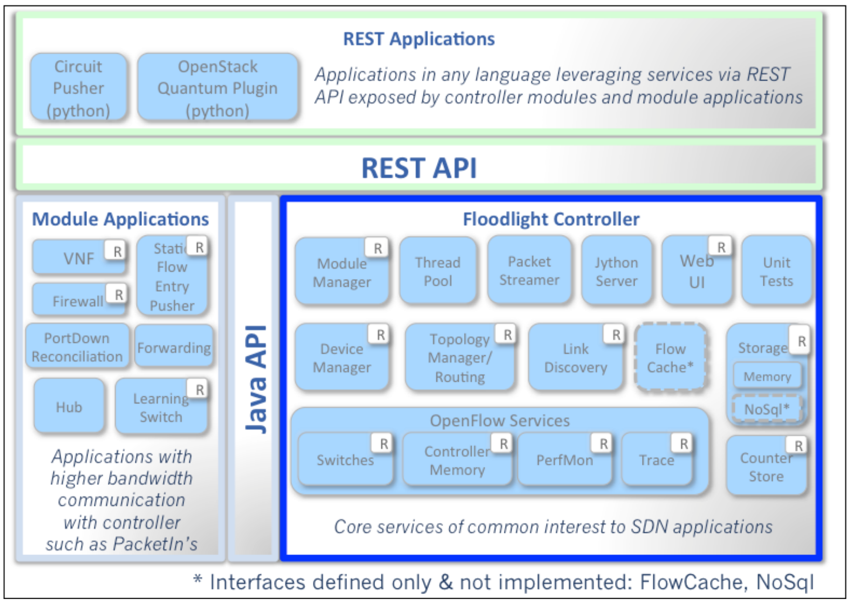


Figure 10 – Floodlight architecture. Source: (FLOODLIGHT, 2012a)

Floodlight is built to be modular. Then, besides the OpenFlow controller, it is a composition of modules built alongside the controller. As depicted in Figure 10, the Floodlight controller acts in the OpenFlow network with the common network modules, used by SDN applications (e.g., Device Manager, Link Discovery Model), to manage and control the network. At the same level of the standard modules, the Java API allows the creation of new modules to execute alongside the floodlight controller modules or extend their features. The before-mentioned features are accessible by the applications through a REST API, which is the recommended procedure to separate applications that will utilize floodlight (FLOODLIGHT, 2018). Through the REST API, one can use Floodlight to recognize the devices in the network, to apply firewall rules, or compute routes from a device to another.

2.6 CONCLUDING REMARKS

In this section, we provided information about the main technologies used in this paper. We detailed the main IoT protocols in Section 2.1, including AMQP and how it works with the Pub/Sub pattern. In Section 2.1.4, we detailed the main differences between a broker-based system and a brokerless system, highlighting that there isn't a more correct or more efficient architecture, it depends on the application and the objectives of the system. Section 2.4 defines the tools utilized in our scenario, as well as the definition of our scenario itself. Section 2.5 explains how SDN works, the benefits and drawbacks of the technology, and our motivation for using it as a tool in our work.

3 RELATED WORK

IoT is growing, but there are open issues yet to be studied before it becomes consolidated in our daily lives. A major challenge in IoT is security. The security in IoT has to be addressed due to the high possibility of security risks such as eavesdropping, unauthorized access, and data modification (MALINA et al., 2016).

In Section 3.1, we introduce the common research issues in IoT security. Section 3.2 details the current IoT solutions for security issues. Moreover, in Section 3.3 we provide an overview of how SDN has been used to overcome such security issues in IoT systems. Lastly, Section 3.4 summarizes the discussed topics and assembles concepts used in the remaining of this work.

3.1 SECURITY IN IOT

From a logical viewpoint, an IoT system is a collection of devices that work collaboratively to reach a common goal. As the broad scope of IoT networks in our daily lives keeps increasing and their deployment has different factors to consider (e.g., technologies, protocols, and design methodologies), the security threats for IoT environments follow the same growth.

Unlike conventional networks devices, most IoT technologies don't support security enforcement due to hardware constraints or limited computing power. Moreover, the network itself is highly dynamic and scalable making it difficult to define a standard security countermeasure. Also, an important requirement for an IoT network is device heterogeneity, i.e., devices from different technologies and specifications, leveraging the complexity of a security level deployment (SICARI et al., 2015).

According to (WEBER, 2010), to provide a secure service, the following security requirements are needed:

- **Resilience to attacks**: The system must avoid the single point of failure and recover from node failures.
- **Data authentication**: Exchanged messaged may be authenticated.
- **Access control**: Access control must be granted for information providers.
- **Client privacy**: Even with a look-up system, specific information from clients must not be made public.

In summary, the security measures have to provide authentication and authorization of IoT nodes (things, users, servers, objects) and data authenticity, confidentiality, and integrity (MALINA et al., 2016). As the IoT devices are hardware-constrained, the security solutions are usually implemented at network, transport and application Layers.

(VORAKULPIPAT et al., 2018) defines the current stage as the Second Generation of IoT, whose users are more familiar with IoT networks and devices and use it in a real environment situation. However, the challenges are currently towards both users and developers. The authors enhance the availability concerns towards a large number of IoT devices that rely on a centralized platform and should avoid a single point of failure. Moreover, authentication and user identification can be a challenge since real environment deployment can create various constraints, e.g., corporations security policies and system monitoring.

3.2 GENERAL SECURITY SOLUTIONS FOR IOT

A key bottleneck in IoT applications is scaling the number of devices able to connect to the network. Focusing on the publish/subscribe architecture, (SEN; BALASUBRAMANIAN, 2018) creates a broker, named Nucleus, which achieves high scalability, resilience to failure, and low bandwidth communication between the broker and the publisher/subscriber. Nucleus works by decoupling the communication service provided by brokers from the service that maintains information about the publishers and subscribers. It is a good proposal, but it is limited to MQTT and depends on the Google Kubernetes Engine (GKE), a managed environment for containers deployment.

(BHAWIYUGA; DATA; WARDA, 2017) propose the design and implementation of token-based authentication of MQTT protocol in constrained devices. The authors inserted a token authentication server in the usual MQTT network. The clients (publishers and subscribers) create a valid token when connecting in the network for the first time and the broker verifies the token validity during the beginning of the pub/sub process. Their result implies that the system can perform the authentication of valid tokens in relatively acceptable time (less than one second). The token-based authentication is a valid process, but in an IoT network, it adds more steps in the communication process since the broker communicates with the authentication server frequently.

According to (PENG et al., 2016), security is a fundamental requirement of an IoT system, especially with the development of network attacking techniques. The authors focus on specific issues for IoT, e.g., protocol performance on constrained devices and data privacy in an IoT server, and developed a secure protocol named AIPS, for Another Identity-based Publish/Subscribe protocol. The main feature of the protocol is the usage of identity-based cryptography (IBC) instead of the public key infrastructure (PKI), from usual IP protocols, to accomplish identity authentication. An AIPS server manages the IBC key generation and manages the network. As their focus is Wireless Sensor Network (WSN), they apply the concept of trust zones to isolate WSNs from the Internet, which can exchange messages just through an IoT gateway. Such ideas are interesting, but there is space for improvement. As within the WSN is a distributed architecture the quality of service in an Internet-scale interconnection among devices is challenging. Also, different end devices have different characteristics and communication requirements.

In the smart home context, (ZHU et al., 2017) present a Blockchain-based Identity Framework for IoT (BIFIT). The framework is an user-centric approach to manage IoT identities and to facilitate their monitoring, i.e., the blockchain is used to correlate users and things while generating private keys for monitoring and security. As blockchain is a recent technology, this work is still in an initial state. However, the usage of this distributed and secure technology is intriguing.

Many authors, including the above-listed papers, cite the single point of failure issue in their works, but they don't propose a solution to avoid it (ZHANG; CHO; SHIEH, 2015)(ASIRI; MIRI, 2016). In the next section, with the SDN usage, a bigger range of security threats, including the SPoF, are investigated.

3.3 SDN-BASED SOLUTIONS

Many papers took the initiative to join SDN and IoT to use the best of SDN management and monitoring features in an IoT network. (HAFEEZ et al., 2016), (HAFEEZ; DING; TARKOMA, 2017), (MAKSYMYUK et al., 2017) used SDN to monitor the network to make traffic flow decisions and apply security policies. The author's scenarios are real-world environments with heterogeneous devices using different protocols; they benefit from SDN network decoupling to abstract the high-level information, e.g., protocols and technologies, to manage their systems in a network level. The management platforms use SDN as a tool to reach a common goal: to monitor and take decisions based on the network traffic.

(LI; BJöRCK; XUE, 2016) and (DEMETRIOU et al., 2017) focus on using SDN as an access control tool. The former enables fine-grained access control for IoT devices in the network by associating the client identity to its traffic, enforcing access control policies. The latter focusses on dynamically deploy fine-grained security policies adjusted according to users' requirements, devices' capabilities and networking environments. Both works have in mind a heterogeneous environment and the abstraction of hardware technologies. By using SDN to analyze the network traffic, they imply security policies to grant client access in the network.

(KATHIRAVELU; SHARIFI; VEIGA, 2015) implement a middleware able to interact with devices in a smart building environment. The Cassowary is a platform for Context-Aware Smart Buildings based on SDN. It uses an ActiveMQ broker to establish the communication between devices and the centralized SDN controller.

3.4 SUMMARY AND DISCUSSION

As previously defined, an important constraint for IoT security is resilience. The IoT system must be resilient to node failures. As detailed previously, an issue in most IoT systems is the single point of failure from broker-based systems. Many works, detailed above, bind IoT and SDN to provide new features for IoT environments exploring SDN advantages in the network. However, there are no works that investigate the SPoF issue when combining these two technologies.

Therefore, the SDC was proposed as an alternative to mitigate the SPoF issue in IoT networks. In the next section, the SDC architecture will be detailed.

4 SOFTWARE-DEFINED CLUSTER

Security in IoT environments is still an open issue, and there are great opportunities to explore and study. As mentioned previously, this work is an alternative solution for broker clustering. The central entity named broker manages Broker-based IoT network, and it represents a SPoF.

This chapter is organized as follow: Section 4.1 describes the architectural details of SDC while Section 4.2 explains its general overview. Section 4.3 describes the SDN Layer and its main purpose. At Section 4.4, the SDC Listener will be explained and detailed.

4.1 ARCHITECTURE LOGIC

The SDC approach uses SDN as a tool to forward messages to brokers and ensure HA in IoT networks. SDC architecture consists of an SDN Layer between the IoT devices and the brokers. The SDN Layer distributes the clients' connections and forwards packets to each IoT broker.

The brokers used in SDC are the same as those used in the conventional clustering. They are also RabbitMQ brokers that work under the same conditions as detailed in Section 2.2. However, instead of clustering the nodes, they are independent of each other, and they are not aware of other brokers' presence in the network. As there is no direct communication among them, their queues and their state are not visible from outsiders. Since they are individual brokers, they can't share data and maintain a HA by themselves. In SDC, the SDN Layer is responsible for managing connections, sharing data, and controlling states.

As detailed in Section 2.5, SDN manages the network and determines the data path through rules previously established. The principal operations done by SDN are the packets converting and forwarding. The SDN Switch analyses the packets information and checks where it should be delivered to. As AMQP uses TCP as the transport protocol, the rules in the flow table of the switch are related to the TCP protocol.

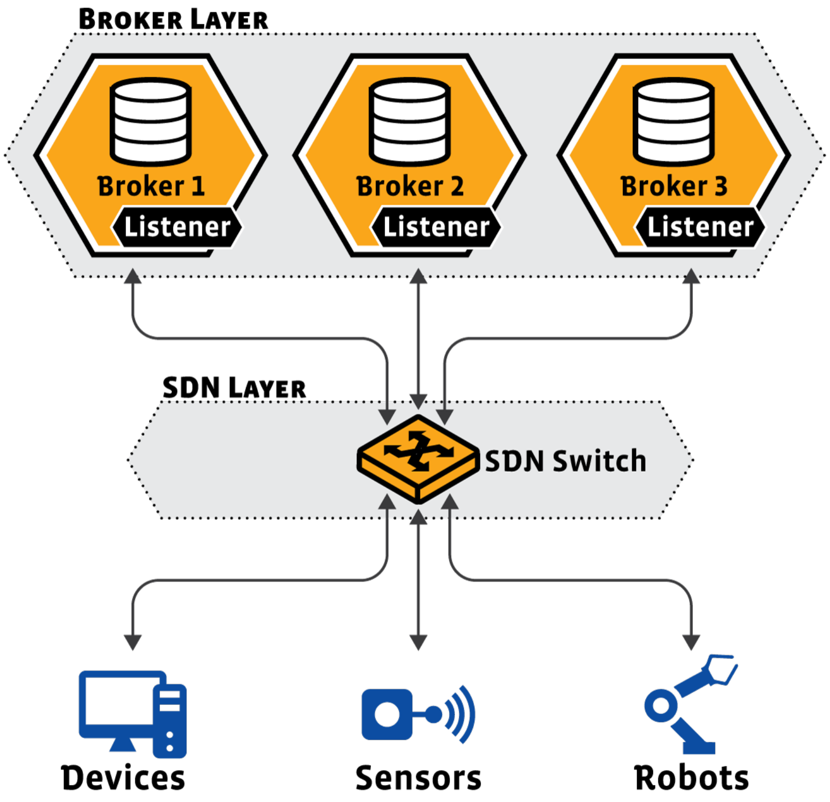


Figure 11 – SDC Architecture

4.2 GENERAL OVERVIEW

SDC consists of two main parts: The SDN Layer and the Broker Layer. The SDN manages the incoming traffic from clients and handles the connections before forwarding the packets to the brokers. Comparing with the traditional architecture, the SDN switch works as the proxy; however, it does not add one more hop in the network architecture.

Figure 11 depicts the overall functioning of SDC. Each connection from the clients (devices, sensors, and robots) through the SDN switch, is modified, at packet level, to be sent to the brokers. The switch is configured with a virtual IP, which works as a reference path for all clients. As each TCP flow is a stateful connection between client and server, it cannot be redirected. Flows are characterized by a tuple of five elements {src-IP, dst-IP, src-port, dst-port, protocol} ("src" meaning source and "dst" meaning destination). Before sending the packets to the brokers, the client connects to the virtual IP and the SDN application changes both IP and port fields and then sends the packets to the brokers. The broker receives the packets in the destination port, processes the request, and sends the response back to the virtual IP at the switch. Then, the SDN switch changes the IP and ports to the intended client.

To maintain the HA among the brokers, an application at the broker layer listens to the traffic and replicates the incoming requests to other brokers. The set SDN rules do not modify the replicated packets. Therefore, the flow between listener and brokers are just forwarded by the SDN.

4.3 SDN LAYER

After a client expresses the desire to connect with the broker, the first layer that the request reaches is the SDN layer. It is responsible for distributing the new connections and for applying rules to message flows. These two tasks are detailed in this section.

4.3.1 Handling New Connections

When a client wants to establish a new AMQP connection with a broker, it creates a TCP socket that stays open until the connection is not active anymore. In a network which client and broker connect through a proxy, the management of new connections is the responsibility of the proxy. As SDC does not have an intermediary entity, besides the usual network switch, between client and proxy, the SDN Layer manages the new connections and distributes them following a round-robin algorithm.

For each new client that requests a connection, the redirection module transmits the flow to a broker in the cluster. The module is developed with the Java API, extending the standard Forwarding Module of the Floodlight controller.

After the first connection request from the client, the SDN Layer registers that the stated client is connected directly with the chosen broker and the destination of the transmitted messages within this connection do not need to be verified again.

4.3.2 SDN Rules for SDC

After choosing the destination of the incoming new connection, the messages must be forwarded to the chosen destination broker. The SDN Layer is responsible for manipulating the messages between client and broker. The forwarding module was extended to define three main SDN rules in SDC. The first rule is the definition of a virtual IP that acts as a reference address for clients and brokers to communicate. Although it is not an actual physical address, all peers in the network can connect to it. The new IP is created through the SDN API and stored at the switch flow table.

The second and third rules are similar. Both of them modify the packets whose destination are the brokers or the IoT clients. More specifically, the second rule changes the traffic from clients to broker while the third rule focuses on traffic from brokers to clients. Clients willing to connect to a broker send a request to the virtual IP. When the packets arrive at the switch with the following configuration:

$src-IP :< Client-IP >; dst-IP :< Virtual-IP >; src-port :< Client-port >; dst-port :< Virtual-port >; protocol : TCP$

When the switch identifies that the client wants to connect to a broker, the second rule is applied, changing the destination and source information to the following:

$src{-}IP :< Virtual{-}IP >; dst{-}IP :< Broker{-}IP >; src{-}port :< Virtual{-}port >$ $; dst - port :< Broker - port >; protocol : TCP$

The path from broker to a client is also modified but this time by the third SDN rule which receives the packets as:

$src{-}IP :< Broker{-}IP >; dst{-}IP :< Virtual{-}IP >; src{-}port :< Broker{-}port >$ $; dst - port :< Virtual - port >; protocol : TCP$

And forwards the packets to the client as:

$src{-}IP :< Virtual{-}IP >; dst{-}IP :< Client{-}IP >; src{-}port :< Virtual{-}port >$ $; dst - port :< client - port >; protocol : TCP$

With such changes, each client assumes that they are connected with a broker located at the virtual IP address, and each broker believes the clients are located at such address.

To exemplify, consider an environment that a client has an IP of 10.10.0.1 and the broker is located at 10.10.0.2. Through SDN the application creates a virtual IP 10.10.0.6 at the virtual port 6 of the switch. If the client desires to communicate with the broker, it sends the data to 10.10.0.6 as the destination address. The SDN application changes both source and destination information to forward to the broker. Then, when the broker sends the response to the switch to 10.10.0.6, the configuration tuple is altered again, and then forwarded to the client.

4.4 BROKER LAYER - THE SDC LISTENER

When the message arrives at the broker, the data must be shared with the other brokers of the cluster to maintain the HA. The responsibility for managing the data replication is the Listener, a module located at the broker machine that sniffs the broker interface and replicates the incoming traffic.

The Listener interprets all requests from clients, such as exchange declaration, queues declaration, and message publishing, and replicates the request to other brokers. It uses the RabbitMQ API to propagate the requests from a broker to another. As it is just a sniffer, it does not consume queues or blocks the AMQP port.

As the listener injects data into the network when sending the clients requests to other brokers, the SDN application must be aware and not apply any rule in the flow table for that kind of traffic.

4.5 TECHNICAL DETAILS

To clarify some technical specifications, the components of SDC are implemented with a different set of tools. The SDN controller is the Floodlight controller, detailed in Section 2.5.3. The brokers are RabbitMQ instances (Section 2.2). The Listener is a sniffer implementation using Libtins, a high-level, multiplatform C++ network packet sniffing and crafting library (FONTANINI, 2014).

4.6 SUMMARY

In this chapter, the SDC architecture was presented, including the applied SDN rules and the Listener module. In the next chapter, the SDC will be analyzed and compared with the traditional RabbitMQ cluster.

5 SDC VALIDATION

In this chapter, the SDC is analyzed regarding its capacity to sort connections among brokers, replicate queues and exchanges, manage the client-broker connections, and replicate queue data.

Section 5.1 describes the scenarios used to validate the SDC. Section 5.2 details the client connection distribution by the SDN switch. The Queues and Exchanges replication are explained in Section 5.3 while the data replication is explained in Section 5.4. The bandwidth usage for both SDC and traditional cluster is evaluated in Section 5.5. Section 5.6 details how SDC handles broker failures. The difficulties and challenges during SDC development are listed in Section 5.7.

5.1 TEST SCENARIO

An SDN-based scenario was built to validate SDC. All scenarios were designed into the Mininet (OLIVEIRA et al., 2014), a network emulator that creates a realistic virtual network with switches, hosts, links, and other network elements in a single machine. Mininet also allows the implementation of OpenFlow and users can create a prototype of an SDN network.

All the following experiments are executions of the Ariac environment. There are two main programs to run the scenario, the Ariac scenario itself and the competition program that is planning logic to assemble the kits. The Mininet emulates the network that allows communication among nodes in the AMQP network. Besides that, it emulates both network and SDN devices, as previously stated. The experiments are run at the same machine with 16GB of RAM and Intel Core I7 Processor.

According to Figure 12, it is composed of two client machines, an SDN switch connected to an SDN controller, a conventional network switch, and two brokers. The Ariac scenario runs at the **Ariac** host, and the competition program runs at the **GPRT** host. The SDN switch responsible for managing the connections is **S1** that is connected to controller **C0** through a secure link. Both brokers **Rabbit** and **Hare** are unaware of each other's existence in the network. However, they will exchange information through SDC via switch **S2**.

The configuration represented in Figure 13 was created to compare the SDC solution with the traditional clustering. It is similar to the SDC setup, but there is the presence of a **proxy** machine, that is absent at SDC. One unique host represents the **cluster**, but it must be interpreted as a connection between two RabbitMQ brokers that work in unison.

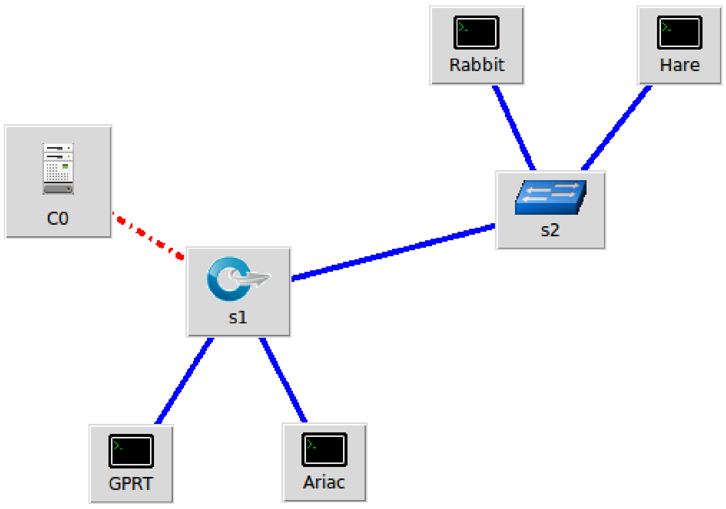


Figure 12 – SDC test scenario

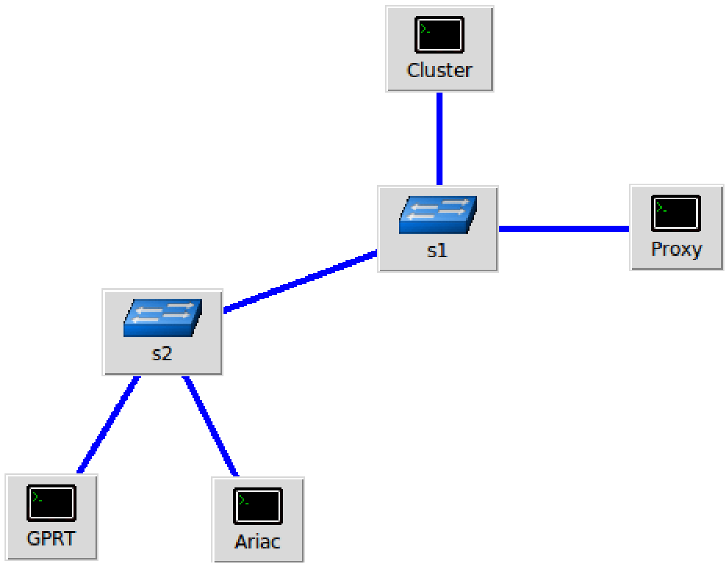
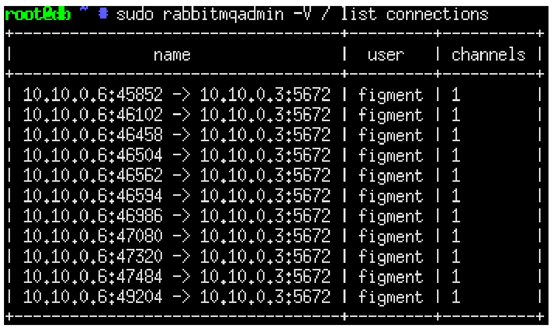


Figure 13 – Traditional clustering Scenario

5.2 DISTRIBUTING CLIENT CONNECTIONS

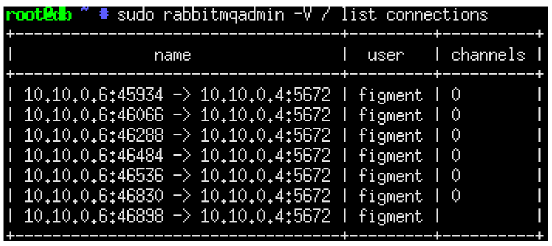
Clustering allows clients to connect to distinct brokers of the cluster and still exchange messages with each other. In the traditional clustering scenario, the proxy is responsible for distributing the client connections among the available brokers. At SDC, the distribution and management of connections are the responsibility of the SDN network.

The SDN switch follows a simple round-robin algorithm to assign the connections to each broker. Figure 14 depicts the SDC connections at each broker. Note that the source IP is always the virtual IP 10.10.0.6, and the port is the original socket port, characterizing the connection origin. Although both cluster methods provide similar connection results, SDC does not depend on a proxy, DNS or another name resolution feature to connect client and broker.



```
root@db ~ # sudo rabbitmqadmin -V / list connections
+-----------------------------------+---------+----------+
|               name                |  user   | channels |
+-----------------------------------+---------+----------+
| 10.10.0.6:45852 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46102 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46458 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46504 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46562 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46594 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:46986 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:47080 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:47320 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:47484 -> 10.10.0.3:5672 | figment | 1        |
| 10.10.0.6:49204 -> 10.10.0.3:5672 | figment | 1        |
+-----------------------------------+---------+----------+
```

(a) List of connections at broker Rabbit



```
root@db ~ # sudo rabbitmqadmin -V / list connections
+-----------------------------------+---------+----------+
|               name                |  user   | channels |
+-----------------------------------+---------+----------+
| 10.10.0.6:45934 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46066 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46288 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46484 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46536 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46830 -> 10.10.0.4:5672 | figment | 0        |
| 10.10.0.6:46898 -> 10.10.0.4:5672 | figment |          |
+-----------------------------------+---------+----------+
```

(b) List of connections at broker Hare

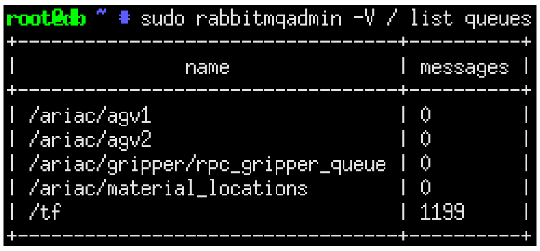
Figure 14 – Through the RabbitMQ management plugin, it is possible to fetch informations about the brokers current state

5.3 QUEUES AND EXCHANGES REPLICATION

In AMQP, the queues and exchanges are the main components of the broker. The clients declare both of them to the brokers and request a bind between them. The declarations and requests are done through the broker's API. As detailed in the previous chapter, the Broker Layer listener executes the replication to other brokers in SDC. It listens to incoming AMQP messages and uses the broker's API to transfer data. It is worth to mention that the Listener does not consume the queue of the broker that is running in the same host.

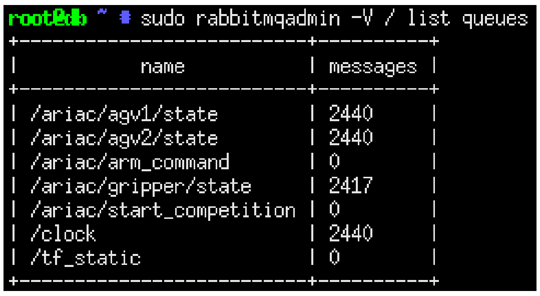
The module in charge of managing information about queues, exchanges, and nodes in a RabbitMQ traditional cluster is the Mnesia, an Erlang distributed database (MATTSSON; NILSSON; WIKSTRÖM, 1999). The SDC Listener works to provide a similar service as Mnesia when regarding queue and exchange replication. The Listener acts as a sniffer in the Broker network port (for AMQP it is 5672 by default) and interprets incoming messages. Any TCP message from clients is read and converted to new information to the other brokers of the network.

The result of queues and exchanges replication with SDC Listener is depicted in Figure 15. Before the Listener actuation, the declared queues are unique due to the client's distribution, detailed in the previous section. Figure 15(a) and Figure 15(b) represent the broker's state prior to the Listener's operations. After each broker connects with their respective clients, the Listener will replicate the exchanges and queues to other brokers, as depicted in Figure 15(c). At this point, both brokers have the same exchanges and queues; the next step is to replicate the published data at each queue.



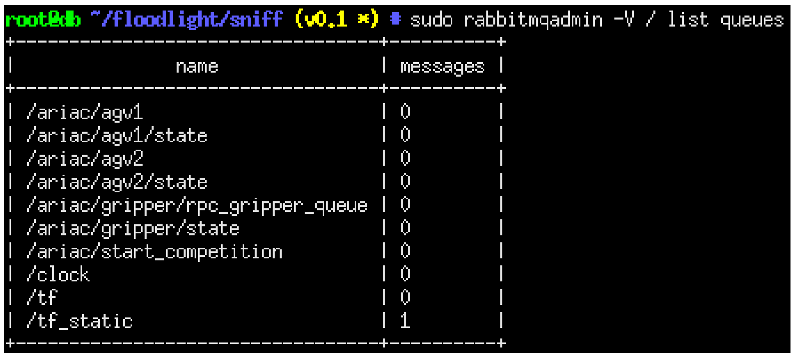
```
root@db ~ # sudo rabbitmqadmin -V / list queues
+-----------------------------------+----------+
|               name                | messages |
+-----------------------------------+----------+
| /ariac/agv1                       | 0        |
| /ariac/agv2                       | 0        |
| /ariac/gripper/rpc_gripper_queue  | 0        |
| /ariac/material_locations         | 0        |
| /tf                               | 1199     |
+-----------------------------------+----------+
```

(a) List of queues at broker Rabbit



```
root@db ~ # sudo rabbitmqadmin -V / list queues
+---------------------------+----------+
|           name            | messages |
+---------------------------+----------+
| /ariac/agv1/state         | 2440     |
| /ariac/agv2/state         | 2440     |
| /ariac/arm_command        | 0        |
| /ariac/gripper/state      | 2417     |
| /ariac/start_competition  | 0        |
| /clock                    | 2440     |
| /tf_static                | 0        |
+---------------------------+----------+
```

(b) List of queues at broker Hare



```
root@db ~/floodlight/sniff (v0.1 *) # sudo rabbitmqadmin -V / list queues
+-----------------------------------+----------+
|               name                | messages |
+-----------------------------------+----------+
| /ariac/agv1                       | 0        |
| /ariac/agv1/state                 | 0        |
| /ariac/agv2                       | 0        |
| /ariac/agv2/state                 | 0        |
| /ariac/gripper/rpc_gripper_queue  | 0        |
| /ariac/gripper/state              | 0        |
| /ariac/start_competition          | 0        |
| /clock                            | 0        |
| /tf                               | 0        |
| /tf_static                        | 1        |
+-----------------------------------+----------+
```

(c) List of queues at broker Rabbit after the Listener actuation

Figure 15 – After the Listener's actuation, the queues that were exclusive of each broker are replicated and now are accessible by every broker.

5.4 QUEUE DATA REPLICATION

The most important task to maintain the high-availability in the broker system is the data replication. It allows distinct clients to connect at different brokers and still consume the same content. The Listener replicates all incoming data to the other broker of the cluster. It interprets the messages and, through RabbitMQ's API, creates a communication path between brokers.

Efficient data replication is fundamental for the HA environment to avoid delays in the message exchange and data loss. To evaluate SDC, the Ariac competition scenario is run in both clusters. The order is the assembling of two parts located in the bins. To clarify, the scenario is composed of six cameras, one robotic arm, and two AGV vehicles. Each one of these devices is sending messages constantly to the cluster. The scenario was run 25 times and the Total Process Time (TPT) of the assembly was acquired.

Figure 16 depicts the mean time taken for the Ariac to conclude the tasks. The TPT between both clusters is almost the same since SDC takes an average of 26s to run the tasks while the RabbitMQ architecture is around one second faster than SDC. Since the Traditional cluster is consolidated and optimized, it was already expected for it to execute the data sharing more effectively. However, as SDC is an alternative and novel approach for clustering, the similar but slower time is acceptable.

Although such examination is valid, for further analysis between SDC and the tra-

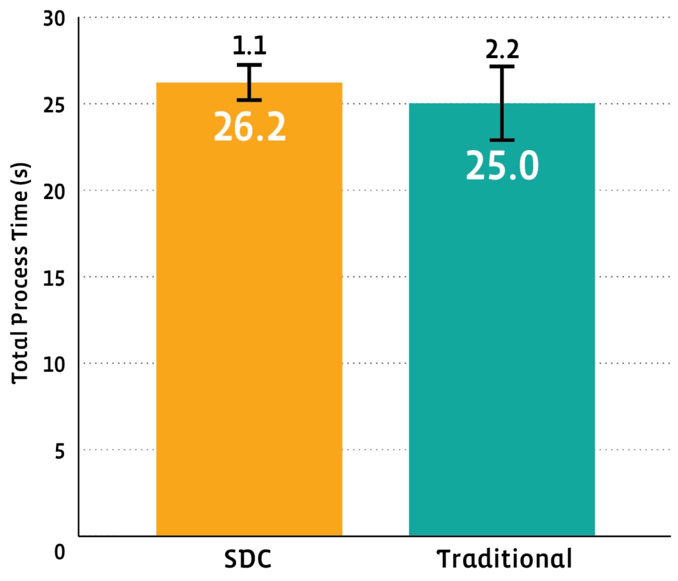


Figure 16 – Comparing the TPT for SDC and Traditional cluster

ditional cluster, it must be taken into consideration that there are components in both clusters that differ from each other and may unbalance the result. For instance, the modules responsible for message replication (Listener and Mnesia in SDC and Traditional, respectively), perform an essential role in the clustering process but their execution and processing are different from each other.

5.5 BANDWIDTH USAGE

During the experiments, the bandwidth in the highest rate links in the network was captured and analyzed. In SDC, since the client communicates directly with the broker through the SDN switch, the bandwidth was captured, using tcpdump (JACOBSON; LERES; MCCANNE, 1989), in the broker-client link for each broker during the same 25 experiments analyzed in the previous section. In the traditional architecture, the bandwidth is captured at the proxy, since all AMQP traffic goes through its interface.

According to Figure 17, SDC provided a lower bandwidth at each broker interface since the SDN switch divides the connections between the cluster nodes. As SDC does not have a proxy, all broker links are dedicated to the clients and other cluster nodes. On the other hand, the traditional cluster is bound to the proxy to communicate with the clients, leveraging the average bandwidth.

Since SDC Round Robin algorithm for associating cluster nodes and clients is not optimized, the client load for each broker may not be equally balanced. For instance, at run 25, in Figure 17, both broker's average bandwidth are similar, and the client load is equally balanced. In contrast with this behavior, run 5 presents a great discrepancy between hare and rabbit brokers. Although it does not affect the cluster functioning, it needs improvement.

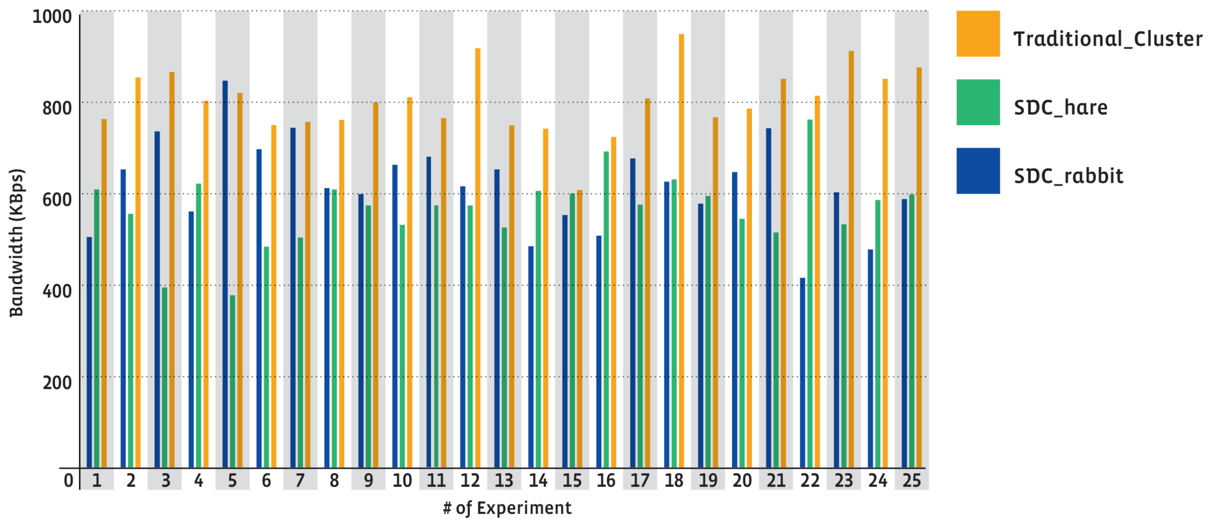
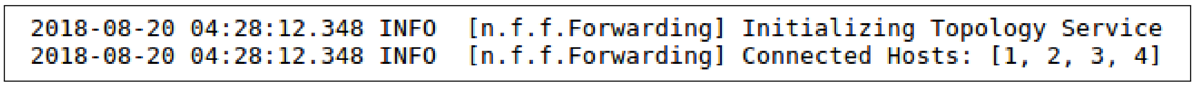


Figure 17 – Bandwidth analysis for both clusters

5.6 HANDLING BROKER FAILURE

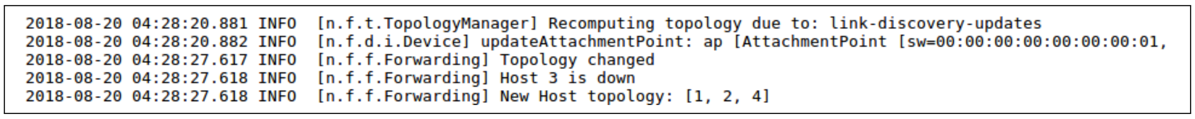
As detailed in previous chapters, Software-Defined Networks enable the network management and provides features to control the on-going traffic in the network. SDC uses the topology manager, a Floodlight module, to handle broker failures and distribute new client connections between active brokers.

Once SDC starts, the controller knows the clients and brokers in the network. The topology manager logs all peers in the network (Figure 18(a)) and distributes client connections accordingly. If a broker node fails, the topology manager recomputes the network topology and connects new clients to the remaining brokers (Figure 18(b)).



```
2018-08-20 04:28:12.348 INFO  [n.f.f.Forwarding] Initializing Topology Service
2018-08-20 04:28:12.348 INFO  [n.f.f.Forwarding] Connected Hosts: [1, 2, 3, 4]
```

(a) SDN log in the start up state listing all known hosts



```
2018-08-20 04:28:20.881 INFO  [n.f.t.TopologyManager] Recomputing topology due to: link-discovery-updates
2018-08-20 04:28:20.882 INFO  [n.f.d.i.Device] updateAttachmentPoint: ap [AttachmentPoint [sw=00:00:00:00:00:00:00:01,
2018-08-20 04:28:27.617 INFO  [n.f.f.Forwarding] Topology changed
2018-08-20 04:28:27.618 INFO  [n.f.f.Forwarding] Host 3 is down
2018-08-20 04:28:27.618 INFO  [n.f.f.Forwarding] New Host topology: [1, 2, 4]
```

(b) Topology manager updating after noticing that broker at port 3 is down

Figure 18 – The Topology Manager is an SDN built-in module that computes hosts, links and ports of the SDN network

If a broker goes down with active connections, the client must start the reconnection by itself. The Ariac scenario is a competition environment which tasks must be completed as soon as possible, then, the clients in the scenario do not try to reconnect after a broker failure. Figures 18(a) and 18(b) are a log output from a simple star scenario depicted in Figure 19. Host **h1** connects with **b1** through SDC. To simulate a node failure, **b1** interface is turned down, forcing **h1** to reconnect to **b2**, since it is the only active broker in the architecture.

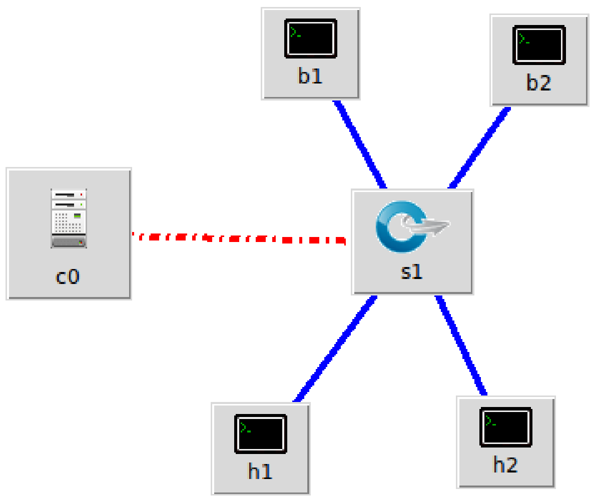


Figure 19 – SDC in an star architecture

5.7 DIFFICULTIES AND CHALLENGES

While building the SDC components, a series of challenges came up. The first difficulty was the Ariac scenario conversion to AMQP. Originally, it is built to work with the ROS framework, but it does not have a native cluster structure to serve as a baseline for SDC. Then, Ariac communication was converted to AMQP and the chosen broker to work with was the RabbitMQ.

Another difficulty during SDC implementation was the clients' connection distribution. Clients should always connect to the same IP address. As the conventional cluster uses a proxy to manage the connections to the server, SDC also needs a similar behavior in the SDN to work correctly. Fortunately, the floodlight controller supports the creation of virtual IPs, allowing a proxy-like behavior through the SDN switch.

The most challenging issue during the development of this work was the data replication among brokers. As AMQP works under TCP, a stateful point-to-point protocol, the traffic between client and server could not be replicated just using the SDN features. A series of possible solutions were tested and some of them even implemented, e.g., NAT and SDN native load balancer. The quickest and most effective solution was to create the Listener, a network sniffer that replicates the data among brokers. This issue was the most time consuming during the execution of this work.

6 FINAL REMARKS

SDN has become an efficient tool to build IoT solutions in diverse areas. In this work, SDN was used to create a solution for the SPoF issue in IoT networks. The main contribution of this dissertation is the Software-Defined Cluster, an SDN-based cluster that is an alternative for conventional clustering architectures.

There are many works that explore the benefits of SDN in IoT networks. As detailed in Section 2.5, SDN allowed a broader range of investigations in IoT in distinct areas, e.g., security and management. However, there are a few that mention the SPoF issue and suggest a solution for it. Therefore, the SDC is a novel SDN cluster solution that focuses in mitigating the SPoF in broker-based IoT networks.

SDC uses the SDN layer to manage client connections and connect them to the brokers. In contrast with the traditional clusters that use a proxy to redirect the connections, SDC binds hosts and brokers through a virtual IP, allowing a direct connection between them. From the client's perspective, the broker is always on the virtual address; however, it is located in another address and can be any cluster node.

At Section 5, it was demonstrated that SDC provides similar results, comparing to the traditional cluster, in the test scenario. The exchanges and queues were replicated successfully in the cluster brokers through the Listener. The messages were also replicated and clients connected to different brokers could communicate through SDC. Regarding the data transfer, the Ariac scenario provided similar process time in both clusters.

During the experiments, SDC demonstrated to be better at dividing the bandwidth utilization in the system. As clients and brokers communicate directly with each other, i.e., do not use a proxy or other intermediary to manage the connections, there is not a common link in the system where all packets flow through, allowing a smaller bandwidth usage between the end-points.

Although traditional clustering strategies are still efficient and an acceptable solution for maintaining HA, they have their own flaws, e.g., SPoF. The SDC provides a distinct approach to reach a similar goal but avoiding this flaw. It is worth to mention that SDC is not a replacement for the current cluster solutions but an alternative. As many other solutions in IoT, the usage of SDC in an IoT environment depends of the main objective to be reached by the IoT application. Using SDC introduces new features in an Internet of Things scenario and allows the usage of the SDN advantages in the network.

A contribution produced from this dissertation was externally published in the paper: An IoT protocol evaluation in a Smart Factory environment (BEZERRA et al., 2018) submitted and accepted, yet not published, by the Brazilian Robotics Symposium (SBR) together with the Latin American Robotics Symposium (LARS).

6.1 SDC LIMITATIONS

The results detailed in this work are from an initial version of SDC. Therefore, there are optimization points in the architecture as well as parts that limit the overall functioning of the system.

As previously commented, the algorithm to distribute the client connections in SDC is simple and does not analyze available bandwidth or any other load metrics to choose the cluster node. For instance, the Round Robin implemented in the HAProxy examines the server weight and backend load to determine which server a new client should connect to.

The Listener developed to replicate queues, exchanges and data is a limiting factor in SDC. It uses the RabbitMQ API to generate a new connection to other brokers and replicate messages. In contrast with SDC, the Traditional cluster uses Mnesia, a distributed database system to exchange messages. For SDC's Listener to reach the level of the consolidated database, it would need some upgrades to improve in both scaling and efficiency.

6.2 FUTURE WORK

As future work, the following updates may be added in SDC:

- **Optimizing the data replication among brokers:** The module in charge of replication is the Listener. As previously stated, the Listener is a quick and effective solution developed only for the replication purpose. The ideal solution could be adding a distributed database as Mnesia to manage the connections among brokers;
- **Allowing SDC to be more scalable:** SDC works with RabbitMQ that is a well-scalable broker, i.e., it can handle a great amount of clients even when organized as a cluster. SDC still requires some experiments to evaluate how much and how efficiently it can scale;
- **Optimizing the client distribution per broker:** The Round Robin algorithm used by SDC is simple and limited since it does not take into account network metrics. An improvement would be the implementation of a better algorithm that allows a better bandwidth utilization as well as a better usage of the brokers' capabilities.
- **Developing SDC to operate with non-clustering brokers:** As detailed in Section 2.2, some brokers do not support clustering. SDC would allow these brokers to exchange information and work as a cluster, even though they are unaware of each other's existence.

REFERENCES

ASIRI, S.; MIRI, A. An iot trust and reputation model based on recommender systems. In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. [S.l.: s.n.], 2016. p. 561–568.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 54, n. 15, p. 2787–2805, Oct. 2010. ISSN 1389-1286. Available at: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>.

BEZERRA, D.; ASCHOFF, R.; SZABO, G.; SADOK, D. F. H. An iot protocol evaluation in a smart factory environment. In: *SBR-LARS 2018 ()*. [s.n.], 2018. Available at: <http://XXXXX/187373.pdf>.

BHAWIYUGA, A.; DATA, M.; WARDA, A. Architectural design of token based authentication of mqtt protocol in constrained iot device. In: IEEE. *Telecommunication Systems Services and Applications (TSSA), 2017 11th International Conference on.* [S.l.], 2017. p. 1–4.

BHOLEBAWA, I. Z.; DALAL, U. D. Performance analysis of sdn/openflow controllers: Pox versus floodlight. *Wireless Personal Communications*, v. 98, n. 2, p. 1679–1699, Jan 2018. ISSN 1572-834X. Available at: <https://doi.org/10.1007/s11277-017-4939-z>.

CHEN, Y.; HU, H. Internet of intelligent things and robot as a service. *Simulation Modelling Practice and Theory*, v. 34, p. 159 – 171, 2013. ISSN 1569-190X. Available at: <http://www.sciencedirect.com/science/article/pii/S1569190X12000469>.

CISCO. *Cisco Visual Networking Index Predicts Near-Tripling of IP Traffic by 2020.* 2016. [Online; accessed 20-August-2018]. Available at: <https://newsroom.cisco.com/press-release-content?type=press-release&articleId=1771211>.

DEMETRIOU, S.; ZHANG, N.; LEE, Y.; WANG, X.; GUNTER, C. A.; ZHOU, X.; GRACE, M. Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps. In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. New York, NY, USA: ACM, 2017. (WiSec '17), p. 122–133. ISBN 978-1-4503-5084-6. Available at: <http://doi.acm.org/10.1145/3098243.3098251>.

ENTERPRISE, H. *"MQTT Essentials Part2: Publish & Subscribe"*. 2017. <https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe>. [Online; accessed 30-November-2017].

ENTERPRISE, M. H. *Broker.(2016, Feb. 8)*. 2012. <https://www.hivemq.com/>. Online; accessed 08 March 2018.

ERICSSON. *Manufacturing reengineered: robots, 5G and the Industrial IoT*. 2015. [Online; accessed 20-August-2018]. Available at: <https://www.ericsson.com/assets/local/publications/ericsson-business-review/issue-4--2015/ebr-issue4-2015-industrial-iot.pdf>.

ERICSSON. *Ericsson Mobility Report*. 2016. [Online; accessed 20-August-2018]. Available at: <https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-june-2016.pdf>.

EUGSTER, P. T.; FELBER, P. A.; GUERRAOUI, R.; KERMARREC, A.-M. The many faces of publish/subscribe. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 35, n. 2, p. 114–131, Jun. 2003. ISSN 0360-0300. Available at: <http://doi.acm.org/10.1145/857076.857078>.

FLOODLIGHT, P. Floodlight architecture. *URL: https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343549/Architecture*, 2012.

FLOODLIGHT, P. Open source software for building software-defined networks. *URL: http://www.projectfloodlight.org/floodlight/*, 2012.

FLOODLIGHT, P. *Floodlight REST API*. 2018. Available at: <https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/1343539/Floodlight+REST+API>.

FONTANINI, M. *Libtins, packet crafting and sniffing library*. 2014. [Online; accessed 20-August-2018]. Available at: <http://libtins.github.io/>.

FUNDATION, O. N. Software-defined networking: The new norm for networks. *ONF White Paper*, v. 2, p. 2–6, 2012.

GRIECO, L.; RIZZO, A.; COLUCCI, S.; SICARI, S.; PIRO, G.; PAOLA, D. D.; BOGGIA, G. Iot-aided robotics applications: Technological implications, target domains and open issues. *Computer Communications*, v. 54, p. 32 – 47, 2014. ISSN 0140-3664. Available at: <http://www.sciencedirect.com/science/article/pii/S0140366414002783>.

HAFEEZ, I.; DING, A. Y.; SUOMALAINEN, L.; KIRICHENKO, A.; TARKOMA, S. Securebox: Toward safer and smarter iot networks. In: *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. New York, NY, USA: ACM, 2016. (CAN '16), p. 55–60. ISBN 978-1-4503-4673-3. Available at: <http://doi.acm.org/10.1145/3010079.3012014>.

HAFEEZ, I.; DING, A. Y.; TARKOMA, S. Ioturva: Securing device-to-device (d2d) communication in iot networks. In: *Proceedings of the 12th Workshop on Challenged Networks*. New York, NY, USA: ACM, 2017. (CHANTS '17), p. 1–6. ISBN 978-1-4503-5144-7. Available at: <http://doi.acm.org/10.1145/3124087.3124093>.

JACOBSON, V.; LERES, C.; MCCANNE, S. The tcpdump manual page. *Lawrence Berkeley Laboratory, Berkeley, CA*, v. 143, 1989.

KATHIRAVELU, P.; SHARIFI, L.; VEIGA, L. Cassowary: Middleware platform for context-aware smart buildings with software-defined sensor networks. In: *Proceedings of the 2Nd Workshop on Middleware for Context-Aware Applications in the IoT*. New York, NY, USA: ACM, 2015. (M4IoT 2015), p. 1–6. ISBN 978-1-4503-3731-1. Available at: <http://doi.acm.org/10.1145/2836127.2836132>.

KHAN, R.; KHAN, S. U.; ZAHEER, R.; KHAN, S. Future internet: The internet of things architecture, possible applications and key challenges. In: *2012 10th International Conference on Frontiers of Information Technology*. [S.l.: s.n.], 2012. p. 257–260.

KOENIG, N.; HOWARD, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*. [S.l.: s.n.], 2004. v. 3, p. 2149–2154 vol.3.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: A survey. *IEEE Communications Surveys Tutorials*, v. 16, n. 1, p. 493–512, First 2014. ISSN 1553-877X.

LEE, I.; LEE, K. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, v. 58, n. 4, p. 431 – 440, 2015. ISSN 0007-6813. Available at: <http://www.sciencedirect.com/science/article/pii/S0007681315000373>.

LI, Y.; BJöRCK, F.; XUE, H. Iot architecture enabling dynamic security policies. In: *Proceedings of the 4th International Conference on Information and Network Security*. New York, NY, USA: ACM, 2016. (ICINS '16), p. 50–54. ISBN 978-1-4503-4796-9. Available at: <http://doi.acm.org/10.1145/3026724.3026736>.

LIGHT, R. Mosquitto-an open source mqtt v3. 1 broker. *URL: http://mosquitto. org*, 2013.

LILIS, G.; KAYAL, M. A secure and distributed message oriented middleware for smart building applications. *Automation in Construction*, v. 86, p. 163 – 175, 2018. ISSN 0926-5805. Available at: <http://www.sciencedirect.com/science/article/pii/S0926580517304867>.

LOCKE, D. Mq telemetry transport (mqtt) v3. 1 protocol specification. *IBM developerWorks Technical Library], available at http://www. ibm. com/developerworks/webservices/library/ws-mqtt/index. html*, 2010.

MAKSYMYUK, T.; DUMYCH, S.; BRYCH, M.; SATRIA, D.; JO, M. An iot based monitoring framework for software defined 5g mobile networks. In: *Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication*. New York, NY, USA: ACM, 2017. (IMCOM '17), p. 105:1–105:4. ISBN 978-1-4503-4888-1. Available at: <http://doi.acm.org/10.1145/3022227.3022331>.

MALINA, L.; HAJNY, J.; FUJDIAK, R.; HOSEK, J. On perspective of security and privacy-preserving solutions in the internet of things. *Computer Networks*, v. 102, p. 83 – 95, 2016. ISSN 1389-1286. Available at: <http://www.sciencedirect.com/science/article/pii/S1389128616300779>.

MATTSSON, H.; NILSSON, H.; WIKSTRÖM, C. Mnesia—a distributed robust dbms for telecommunications applications. In: SPRINGER. *International Symposium on Practical Aspects of Declarative Languages*. [S.l.], 1999. p. 152–163.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, Mar. 2008. ISSN 0146-4833. Available at: <http://doi.acm.org/10.1145/1355734.1355746>.

MUMBAIKAR, S.; PADIYA, P. et al. Web services based on soap and rest principles. *International Journal of Scientific and Research Publications*, Citeseer, v. 3, n. 5, 2013.

O'HARA, J. Toward a commodity enterprise middleware. *Queue*, ACM, New York, NY, USA, v. 5, n. 4, p. 48–55, May 2007. ISSN 1542-7730. Available at: <http://doi.acm.org/10.1145/1255421.1255424>.

OLIVEIRA, R. L. S. D.; SCHWEITZER, C. M.; SHINODA, A. A.; PRETE, L. R. Using mininet for emulation and prototyping software-defined networks. In: IEEE. *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*. [S.l.], 2014. p. 1–6.

PENG, W.; LIU, S.; PENG, K.; WANG, J.; LIANG, J. A secure publish/subscribe protocol for internet of things using identity-based cryptography. In: *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. [S.l.: s.n.], 2016. p. 628–634.

PIVOTAL. *RabbitMQ Hits One Million Messages Per Second on Google Compute Engine*. 2014. Available at: <https://content.pivotal.io/blog/rabbitmq-hits-one-million-messages-per-second-on-google-compute-engine>.

QIN, Y.; SHENG, Q. Z.; FALKNER, N. J.; DUSTDAR, S.; WANG, H.; VASILAKOS, A. V. When things matter: A survey on data-centric internet of things. *Journal of Network and Computer Applications*, v. 64, p. 137 – 153, 2016. ISSN 1084-8045. Available at: <http://www.sciencedirect.com/science/article/pii/S1084804516000606>.

QPID, A. Open source amqp messaging. *URL: http://qpid. apache. org*, 2013.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. Y. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5.

RabbitMQ. *Broker vs Brokerless*. 2010. <https://www.rabbitmq.com/blog/2010/09/22/broker-vs-brokerless/>. Online; accessed 07 March 2018.

SEN, S.; BALASUBRAMANIAN, A. A highly resilient and scalable broker architecture for iot applications. In: IEEE. *Communication Systems & Networks (COMSNETS), 2018 10th International Conference on*. [S.l.], 2018. p. 336–341.

SHELBY, Z.; HARTKE, K.; BORMANN, C. *The constrained application protocol (CoAP)*. [S.l.], 2014.

SICARI, S.; RIZZARDI, A.; GRIECO, L. A.; COEN-PORISINI, A. Security, privacy and trust in internet of things: The road ahead. *Computer networks*, Elsevier, v. 76, p. 146–164, 2015.

SNYDER, B.; BOSNANAC, D.; DAVIES, R. *ActiveMQ in action*. [S.l.]: Manning Greenwich Conn., 2011.

SOFTWARE, P. Rabbitmq by pivotal. *URL: https://www.rabbitmq.com/*, 2013.

TEAM, E. E. *"The Massively Scalable MQTT Broker for IoT and Mobile Applications"*. 2012. <http://emqtt.io/>. [Online; accessed 19-March-2018].

VINOSKI, S. Advanced message queuing protocol. *IEEE Internet Computing*, IEEE, v. 10, n. 6, 2006.

VORAKULPIPAT, C.; RATTANALERDNUSORN, E.; THAENKAEW, P.; HAI, H. D. Recent challenges, trends, and concerns related to iot security: An evolutionary study. In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2018. p. 405–410.

WEBER, R. H. Internet of things – new security and privacy challenges. *Computer Law & Security Review*, v. 26, n. 1, p. 23 – 30, 2010. ISSN 0267-3649. Available at: <http://www.sciencedirect.com/science/article/pii/S0267364909001939>.

WHITMORE, A.; AGARWAL, A.; XU, L. The internet of things–a survey of topics and trends. *Information Systems Frontiers*, Kluwer Academic Publishers, Hingham, MA, USA, v. 17, n. 2, p. 261–274, Apr. 2015. ISSN 1387-3326. Available at: <http://dx.doi.org/10.1007/s10796-014-9489-2>.

XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, v. 10, n. 4, p. 2233–2243, Nov 2014. ISSN 1551-3203.

Zeromq. *Broker vs. Brokerless*. 2011. <http://zeromq.org/whitepapers:brokerless>. Online; accessed 07 March 2018.

ZHANG, Z.-K.; CHO, M. C. Y.; SHIEH, S. Emerging security threats and countermeasures in iot. In: *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. New York, NY, USA: ACM, 2015. (ASIA CCS '15), p. 1–6. ISBN 978-1-4503-3245-3. Available at: <http://doi.acm.org/10.1145/2714576.2737091>.

ZHU, X.; BADR, Y.; PACHECO, J.; HARIRI, S. Autonomic identity framework for the internet of things. In: *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*. [S.l.: s.n.], 2017. p. 69–79.