Pedro Ákos Horváth Filipe da Costa

Degree in Computer Science and Engineering

Practical Aggregation in the Edge

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Engineering

Adviser: João Carlos Antunes Leitão, Assistant Professor,
NOVA University of Lisbon

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2018

Abstract

The Edge Computing paradigm has earned recent popularity. As Cloud-based applications start to require more resources, solutions that reside outside the scope of the Cloud have began to be studied. However, Edge Computing presents a broad spectrum since every device that is not in a data center is considered to be a potential Edge resource. Consequently, in this thesis we focus on the challenging edge scenario that is wireless ad hoc networks. These networks are composed by wireless devices that can connect to each other without requiring any infrastructural support, which is a common scenario at the very edge of the system, in wireless sensor networks, mobile ad hoc networks, and potentially, smart cities.

In this work we look into aggregation as a distributed computation scenario, that will serve the purpose of enabling more general computations in the edge. Aggregation protocols and algorithms provide an essential part of managing and maintaining large scale systems, that we target in the form of wireless ad hoc networks. As such, we study different protocols that use different techniques to obtain either exact or estimated aggregation results. Furthermore, as we aim to implement these protocols in real scenarios, we discuss some frameworks for building distributed protocols and applications.

However, the study of the state of the art revealed that there is no support for developing distributed protocols and applications in wireless ad hoc networks for commodity devices. Therefore, we present a prototype framework that is able to cope with these limitations, and propose to leverage on it to adapt existing aggregation protocols to the wireless environment. Moreover, we plan to develop a new aggregation protocol that is able to overcome the limitations of the existing ones, by leveraging on the different properties that different protocols provide, and evaluate these protocols with well defined metrics in a test-bed composed of twenty four Raspberry Pis.

Keywords: Edge Computing, Wireless Ad Hoc Networks, Aggregation, Frameworks

Resumo

O paradigma de Computação na Berma ganhou popularidade recentemente. Com o aparecimento de aplicações baseadas na Nuvem que requerem mais recursos, soluções que residem fora da Nuvem começam a ser objecto de estudo. No entanto, a Computação na Berma apresenta um longo espetro de concretizações, visto que qualquer dispositivo que está fora de um centro de dados pode ser considerado como um potencial recurso computacional na Berma. Como tal, nesta tese focamos-nos no caso particularmente desafiante de computação em redes ad hoc sem fios. Estas redes são formadas por dispositivos com rádios que se conseguem ligar uns aos outros sem suporte da infraestrutura, que por sua vez, são ilustrativas de casos comuns no extremo da sistema, sob a forma de redes de sensores, redes moveis e potencialmente cidades inteligentes.

Neste trabalho olhamos para a agregação como um caso de computação distribuída, que irá servir o propósito de pavimentar o caminho rumo ao suporte de computações mais genéricas na berma. Os protocolos e algoritmos de agregação providenciam uma parte essencial de gerir e manter sistemas de grande escala, que estudamos no contexto de redes ad hoc sem fios. Assim, estudamos diferentes protocolos que utilizam diferentes técnicas para calcular resultados exactos, ou estimados, de agregação. Adicionalmente, como pretendemos implementar e utilizar estes protocolos em ambientes reais, discutimos algumas frameworks para construir e executar protocolos e aplicações distribuídas.

Durante o estudo do estado da arte revelou-se que não existem ferramentas que suportam o desenvolvimento de protocolos e aplicações distribuídas em ambientes sem fios para dispositivos comuns. Como tal, apresentamos um protótipo de uma framework para lidar com estas limitações, e propomos alavancar nesta framework adaptando os protocolos de agregação existentes para as redes ad hoc sem fios. Além disso, planeamos desenvolver um novo protocolo de agregação que é capaz de ultrapassar as limitações dos protocolos existentes, alavancando nas diferentes propriedades que cada um consegue garantir, e avaliar este protocolos com métricas bem definidas numa plataforma de teste constituída por vinte e quatro Raspberry Pis.

Palavras-chave: Computação na Berma, Redes Ad Hoc Sem fios, Agregação, Framework

Contents

List of Figures

Chapter 1

Introduction

Nowadays many user-facing distributed applications rely on Cloud-based infrastructures to process and manage user and application data, where client applications frequently interact with remote servers in data centers. This shift has been mostly motivated by the increase of the user base and the amounts of data that need to be manipulated by such applications. Consider, for example, social network applications such as Facebook, Twitter, or Instragram that have billions of users accessing the Cloud at once [67], or IoT devices in a smart city that are producing huge amounts of data that is uploaded to the cloud for processing.

However, solely relying on the Cloud infrastructure can have its disadvantages. The increase on the required resources of the cloud also leads to an increase in the cost for application providers; the latency experienced by end users that must constantly contact remote servers, as well as security aspects that arise from outsourcing data storage and computations [29]. These issues have motivated the need to move computations outside the data center towards the edge of the system. This has led to the emergence of the Edge Computing paradigm.

Edge Computing can be defined as performing computation outside the Cloud boundary, at devices that are closer to the source of data, which nowadays are mostly end user devices [67]. As one gets farther away from the Cloud, one encounters multiple types of devices where intermediate computations can be performed. First ISP servers, then gateways, laptops, smart phones, sensors and actuators. It is also important to note that, further from the Cloud the amount of devices increases, while the individual amount of resources per device decreases. Executing computations in such environments becomes therefore, a complex task, as one has to deal not only with a large number of devices, but also manage their limited resources.

Another important aspect that one has to consider as we move towards the edge of the system, is that infrastructure support becomes increasingly lacking. This translates, for instance, in the fact that devices will be connected by limited capability links, in particular, one can expect to find most devices being connected through wireless mediums, that offer a potentially highly unreliable communication environment. Due to this, in this work, we focus on the particularly challenging setting, where all devices communicate through a infrastructure-less wireless medium.

Furthermore, achieving general purpose computations in the edge is not a trivial task. However, we can look at aggregation computation as a first step towards general computation support in the edge. Aggregation can be computed in-network as devices exchange information and cooperate among them. TinyDB [54] showcases how this can be done in order to create a database in sensor networks. Additionally, Astrolabe [76] exhibits how aggregation can be leveraged to perform monitoring tasks in large scale systems. Finally, ZebraNet [39] presents the importance of having a support runtime for a self-managed network. All of these application examples showcase the relevance of efficient aggregation mechanisms.

Objective

In the thesis we plan to address two complementary challenges. First, we aim at developing and implementing a framework that simplifies the development and execution of protocols and applications that operate in wireless ad hoc networks. Such framework must achieve a set of non-trivial goals, in particular:

1. simplify the management and configuration of applications, avoiding the need to configure membership aspects;

2. provide fundamental mechanisms for supporting the operations of distributed and decentralized protocols such as message passing;

3. provide an efficient executing environment for protocols and applications in this scenario.

As we detail in Chapter 2 of this document, existing aggregation protocols behave differently across different settings, and present trade-off along dimensions such as precision and efficiency. To overcome this limitation, we further plan to develop a novel aggregation algorithm, particularly tailored for ad hoc networks, that can cope with different execution environments by combining techniques of other protocols in a natural way. This aggregation protocol will come to light by levaraging on the framework and the study of the operation and properties of existing aggregation protocols.

Contributions

The main contribution that are expected from this work are three fold:

1. The design, implementation, and evaluation of a framework for implementing and executing decentralized distributed protocols and applications in ad hoc network environments.

2. A set of implementations of existing distributed data aggregation protocols and the proposal of a novel aggregation protocol, that operate in a reliable and efficient fashion in ad hoc network environments.

3. An experimental comparison on aggregation protocols under distinct operational conditions, namely in scenarios where data aggregation is performed on demand by the application and in scenarios where data aggregation runs continually in background (for instance to support monitoring tasks).

Research Context

The work to be conducted in the thesis is an integral part of the research agenda of the H2020 Lighkone: Lightweight computation for networks in the edge (Project number 732505), founded by the European Commission. Due to this, a fraction of the contributions of this thesis, in particular regarding the design and implementation of a development and execution framework for ad hoc protocols and applications was already conducted, and discussed in some detail in this project.

Part of the contribution of this work appear as part as the “D5.1: Infrastructure Support for Aggregation in Edge Computing” deliverable produced by the Lighkone Consortium, in January 2018.

Thesis Structure

The rest of the document is organized as follows:

- Chapter 2 presents Edge Computing in more detail, explains the wireless medium and the variants of wireless ad hoc networks. We give context on what communication strategies are used to perform aggregations and present overlay networks as a way to build self-managed networks in decentralized systems. Additionally, we detail aggregation protocols and provide an overview on existing frameworks for building and executing distributed protocols and applications.

- Chapter 3 briefly presents our prototype framework, details the strategy for achieving the proposed aggregation protocol and discusses the evaluation plan and the scheduling of future work.

Chapter 2

Related Work

In this chapter we further detail the issues related to the Cloud infrastructure and explain why we should begin to look for solutions that reside in the Edge (Section 2.1), which provides our motivation to study infrastructure-less networks. As such, we describe the wireless medium and the challenges associated with its use (Section 2.2), describing how the wireless medium can materialize in an infrastructure-less network (Section 2.3).

Additionally, we provide an overview of (decentralized) communication strategies (Section 2.4), as they are essential to perform distributed computations. We discuss existing overlay network solutions in peer-to-peer (Section 2.5), serving as an inspiration to build decentralized and adaptive systems. We proceed by detailing aggregation as a form of simple, yet fundamental, distributed computation (Section 2.6).

Lastly, we give an overview of existing frameworks and tools to develop distributed protocols and applications (Section 2.7).

2.1 From the Cloud to the Edge

The Cloud gained significant momentum in recent years. Cloud providers such as Amazon and Google have millions of user using their Cloud services to deploy applications and store data. In order to do so, the Cloud environment presents a multitude of commodity servers working together in one data center or across multiple data centers scatered throughout the world, which gives the possibility to tackle large scale computation problems with solutions like MapReduce [23], and provide backend services for large scale applications, such as Facebook.

However, the Cloud is not perfect. Data privacy presents itself as one of the main problems of the Cloud as discussed in [29]; furthermore, issues regarding data management, resource allocation, and scalability still persist [25]. With the increasing number of applications and users accessing such applications, these issues start to become more pronounced. Applications and devices producing large amounts of data (e.g, daily-life devices; IoT devices; sensors; and others) present a significant overhead in data transfer and data management for the Cloud. Even though the Cloud infrastructure is said to be elastic, there is the possibility that the Cloud resources become saturated, as it is estimated that the total amount of data produced by people and devices will reach 500 zettabytes by 2019 [21, 67], which can, therefore, lead to an increase in latency experienced by the users, or full disruption of applications operation.

With this in mind, we must look for solutions that are outside of the Cloud, hence we must begin to look towards the edge of the system and on how and what can be leveraged to relief the Cloud from doing all the work. Outside the Cloud there are mini-datacenters; routers and gateways that can have mini-servers; end user devices, such as laptops and smarphones; IoT devices; sensors and actuators; among other computational resources. We notice that farther from the Cloud the devices become less powerful, but in higher numbers while having much lower latency for end users. Given this scenario, the same computation and storage opportunities that were possible in the Cloud become infeasible, thus a new computational paradigm is required: the Edge Computing paradigm.

However, there is no unified vision on what Edge Computing is, due to the fact that every computation that can be executed outside the Cloud can be considered to fall in the scope of Edge Computing, and any network resource that is outside a data center, can be viewed as an Edge Device. As such, Edge Computing materializes in various and different forms. Through Fog Computing [18, 55, 81], which tries to extend the Cloud's infrastructure closer to data sources, and end users. Through intelligent IoT or sensor networks that are able to pre-process data cooperatively without the need of the Cloud [6], and even in other forms as described in [40, 73].

Consequently, the Edge Computing paradigm involves any possible computations near data sources, as such, we define our Edge Devices as neither being sensors, which have limited resources, nor being as powerful as servers. Instead, we focus on Edge Devices that have some computational power, can be very disperse and have no infrastructural support (e.g, no access to routers or access points).

This implies that wiring devices together might not be possible, thus, we must consider wireless as a possible means for supporting all communication between devices.

2.2 The Wireless Medium

The wireless medium is a shared medium were devices can communicate with each other through radio waves using the same frequency. The radio waves are generated by transceivers, or radios, which are usually omnidirectional and can reach every device that is within its transmission range. Consequently, when a message is sent from a wireless device, the message is delivered to every other device that is within that range, this is commonly refereed to as *one hop broadcast*. This abstraction allows to build point-to-point and multicast primitives within the range of a sender by having receivers, for whom the message is not addressed, to simply drop that message.

Since the wireless medium is a shared medium between any device that intents to transmit on the same frequency, wireless communication suffers from contention, which is a usually related to collisions. When two devices transmit a message at the same time and are within range of each other, a collision will most likely happen. Consequently, none of the messages will be heard by the receiver causing message loss.

To better understand the consequences of collisions, consider three devices A, B and C, as in Figure 2.1. Where A can only communicate with B; B can communicate with A and C; and C can only communicate with B. Now assume that, A and C intend to send a message to B. As A and C are unaware of each other they will both send the message to B simultaneously, causing a collision between the two message and making B unable to receive either of them [66]. This can be viewed as the *hidden terminal problem*.

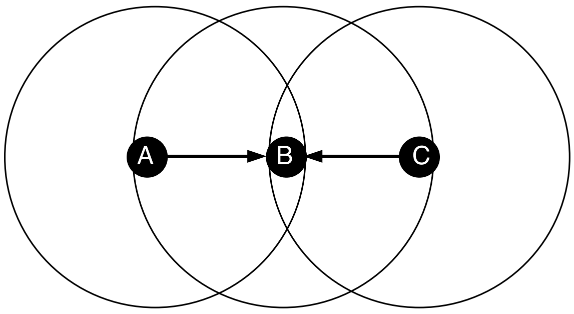

Figure 2.1: Hidden Terminal Scenario

Extending this scenario to another with larger numbers of devices, where devices need to forward messages between them, we can incur in the *broadcast storm problem* [61]. Nodes will continuously transmit, or retransmit, messages that will collide and consequently, saturate the network, rendering any form of communication impossible.

A strategy to minimize the effects of the hidden terminal problem and broadcast storm is to have nodes performing some form of access control to the wireless medium (we discuss how this is performed in detail further ahead). Unfortunately, this can lead to the *exposed terminal problem* [66]. This happens when a node decides not to transmit even if its transmission does not interfere with another ongoing transmission, which can consequentially lead to significant decrease of the network capacity, potentially disrupting the execution of protocols and applications.

2.2.1 MAC Layer Protocols

To address the contention and collision issues in wireless networks, many medium access control (MAC) layer protocols have been proposed [24]. We will discuss the most commonly used, which are the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), and Time Division Multiple Access (TDMA).

CSMA/CA In CSMA/CA [83] when a device wants to transmit, it first listens (senses) the medium to see if no other device is transmitting. If the medium is occupied, then the device waits for a some time and senses the medium again. If the medium is free, it proceeds to send a Request-To-Send (RTS) frame and waits for a Clear-To-Send (CTS) frame from the intended receiver. By using the RTS/CTS mechanism CSMA/CA avoids the hidden terminal problem, if no CTS frame is heard, it means that a collision has occurred due to a hidden terminal. In this case, the sender will wait some time and restarts the mechanism to access the medium (by sending the RTS frame). When the CTS frame is received, the device proceeds to transmit the data frame and waits for an ACK frame as a positive acknowledgement that the data frame was received. If the ACK frame is not received, then a retransmission must take place.

There is a exception when using CSMA/CA, when the data frame's destination is the (physical layer) broadcast address. In this particular case, the RTS/CTS mechanism cannot be used and ACK frames are not sent after a successful transmission [74].

TDMA The TDMA [80] protocol tries to ensure collision-free transmissions, by defining a schedule of non-overlaping time slots for each device. TDMA divides a time period into fixed-length slots where transmissions are possible, when two devices have the same slot they must coordinate to change their slots in order to avoid collisions. Needless to say that finding the optimal time slots for every device in a network in a distributed maner, is a NP-complete problem [26], thus TDMA can be highly inefficient in some cases, computing sub-optimal schedules that can lead to a low utilization of the wireless medium.

Furthermore, the wireless environment presents more challenges than contention and collisions. A wireless link is usually more unstable than a wired link, since it is more affected by the physical environment [1]. This can lead to anomalies such as high bit error due to noise, which must be taken into consideration when leveraging the wireless medium. Additionally, links might not always be symmetric [44], due to different transmission power of radio devices, that might make it hard to reason about the behaviour of protocols and applications. Another aspect to consider is the variety of available technologies that use the wireless medium. Each one has different reach, different data rates, different frequency ranges, availability on devices, and other limitations.

2.2.2 Wireless Technologies

There are many wireless technologies that allow us to build a wireless networks [31, 59, 63, 69]. For completeness we briefly discuss three main technologies currently used for wireless networking that use the 2.4Ghz frequency range. WiFi (Wireless Fidelity), which is the most common on commodity devices, Bluetooth, used for connecting peripheral devices, and ZigBee that is used in wireless sensor networks.

Wifi

The WiFi [69] technology is the one most seen in nowadays commodity hardware. It can be easily found in our laptops, mobile phones, tablets, gaming consoles, and many others. WiFi has a range of around one hundred meters and a data rate of approximately 500 Mb/s.

Bluetooth

Bluetooth [31] is the technology most commonly used to connect peripheral devices, such as keyboards, computer mouses, headphones, among others. It is designed to work in short range having a reach in the order of ten meters. This technology was also designed to be energy efficient, and having low data rates (around 1 Mb/s, however this can differ with version and specification). It is usually seen operating in a master/slave configuration, where a device can only be, either a master or a slave. However, each master can only have up to seven active slaves devices.

ZigBee

ZigBee [63] is one of the main technologies used in sensor networks and IoT devices, having a reach that varies between ten meters and one hundred meters and a very low data rate transmission (around 256 Kb/s). As sensor and IoT devices are energy constrained, ZigBee is also designed to be highly energy efficient. A ZigBee network is composed by three types of devices: *coordinators*, *routers*, and *end devices*. Every network must have a coordinator that builds and configures the network, routers are then used to relay information between nodes, whereas end devices operate as data producers that must be connected to the coordinator, either directly or through routers.

2.2.3 Discussion

Given these three technologies at our disposal, we will focus on WiFi. WiFi has less restrictions allowing to create decentralized (without coordinators) networks with more ease. It is the most commonly found in commodity devices, as such it will also be the most prevalent in the edge of systems. Furthermore, WiFi presents the highest data rate of all three technologies presented which will allow to perform computations with more ease. This will be an important aspect to perform efficient aggregation in the edge.

Additionally, resource constrained devices that are restricted to the use of Bluetooth or ZigBee can be incorporated in edge systems by having them connect directly to one of the WiFi equipped devices, generating hierarchical networks.

2.3 Wireless Networking

Wireless networks enable multiple wireless devices to communicate, providing abstractions from aspects as the relative positioning of the devices. The most common form a wireless network is based in infrastructure, where a set of devices connect to an access point or a router to access the Internet or other infrastructure networks (e.g, a local area network).

However, in the context of large-scale IoT or sensor networks deployments, it is hard to have all devices in the reach of an access point or wireless router. In these scenarios, we have to resort to infrastructure-less networks, where devices interact directly among themselves. These networks are usually refereed to as *ad hoc networks*.

Since we target systems that operate with no access to infrastructure and where the number of devices can grow at any point in time (in an organic fashion), ad hoc networks are a key aspect. We now discuss these networks in more detail.

2.3.1 Wireless Ad Hoc Networks

A simple ad hoc network, is composed by a set of nodes that are interconnected arbitrarily forming a multi-hop network (i.e, a network where not all devices are directly reachable by a radio transmission of any of the other devices). These nodes can only communicate with their direct neighbours using one hop broadcast or point-to-point communication primitives. As such, there is usually no routing involved in these networks and routing among devices must be performed at application level. Additionally, these networks might not even use IP addresses to identify nodes, and can rely solely on MAC addresses for communication.

Given the capability of the nodes to interconnect in a arbitrary fashion, an ad hoc network is highly dynamic and have gained some popularity in use cases such as disaster relief operations [32], military operations [70], monotiring and sensing applications [82], and even in supporting communication among vehicles [28].

However, these networks can be materialized in various forms given the network's objective and the devices that compose it.

Wireless Mesh Networks

Wireless mesh networks build upon ad hoc networks to extend an existing network infrastructure [3]. A mesh network is composed by two types of devices, *mesh routers* and *mesh clients*. Mesh routers are assumed to be static and are responsible for forwarding packets in transit between devices. While mesh clients connect to the routers and only send and receive messages, being typically devices with lower amounts of resources. With this, a mesh network is capable of performing routing, enabling mesh clients to access infrastructural resources, such as file servers, application servers, and Internet gateways.

However, in typical mesh networks, routing requires explicit support. To ensure (effective) routing connectivity, mesh routers may need to be specialized hardware containing multiple antennas to leverage the wireless medium full spectrum [64]. Moreover, when considering highly dynamic networks, with nodes joining and leaving, these networks might experience high control overhead for maintaining routing information up-to-date.

Mobile Wireless Ad Hoc Network

In the literature [2, 27, 30, 34, 62], when wireless ad hoc networks are mentioned, they are usually referring to a mobile wireless ad hoc network (MANET), where nodes that materialize the network are mobile. This means that nodes will change neighbourhoods frequently, leading mobile wireless networks to be highly dynamic.

This in turn, makes routing between the nodes even more complicated. As routes frequently change it is not fesable to hold information such as the minimum number of hops between pairs of nodes, and other techniques have been proposed [9, 42]. Furthermore, some also consider that the nodes composing the network might be resource constrained, as such, extending the network's life should be considered when incorporating routing in MANETs [58].

Vehicular Wireless Ad Hoc Network

These networks come as a particular case of MANETs. Vehicular wireless ad hoc networks [28] (VANETs) assume that the network is mostly composed of vehicles that communicate with each other and with (fixed) stations that are positioned near the road to get useful information or access resources outside of vehicles computers.

These networks suffer from the same problems as MANETs related to routing, however, the network's objective is fundamentally different. These networks may service vehicles to be more autonomous or to inteligently collect information from sensors scatered throughout an area [7]. Although, VANETs are composed by vehicles and, even though present an interesting edge computing scenario, they are outside the scope of this thesis.

Wireless Sensor Networks

We can perceive these networks as beeing wireless ad hoc networks composed of vast numbers of specialized sensors that collect data from the environment. These sensors are described as being very small and inexpensive [4], being perfect for mass deployement. Moreover, sensor networks can be deployed remotely on harsh conditioned environments, such as underwater [5] or in volcanic regions [78]. However, due to their size, sensors tend to be very resource constrained and extending their battery life tends to be the main focus of research on sensor networks.

A typical sensor network architecture is composed by a sink, or gateway, node that can be connected to an infrastructure, and a number of sensors that are connected to the sink or to each other, forming a multi-hop wireless network. Therefore, the goal of each sensor is to report their collected data to the sink node, which is then responsible for processing that data.

2.3.2 Routing in Wireless Ad Hoc Networks

As previously discussed in Section 2.2, wireless networks present a different set of challenges from the ones commonly found on wired networks. A key difference between wireless and wired networks can be found on the routing strategies used in these environments, particularly when considering wireless ad hoc networks.

In wired networks routing among nodes is supported by the IP layer that is well defined. Routing relies on specialized hardware (switches and routers). The latter rely on link-state protocols [19] that assume that wired links are not only stable but also possible to monitor.

In sharp contrast, in wireless ad hoc networks there are no dedicated hardware assisting in routing activities. This requires nodes to coordinate among themselves to achieve multi-hop communication. In some cases however, routing is avoided by transmitting messages through flooding of the network, and then have each node inspect the contents of the message to decide if it should process it or drop it.

Nevertheless, routing is useful in the context of wireless ad hoc networks. For instance, a simple way to compute aggregation is to have all nodes report their values to a single node in the network that can then compute the aggregation result. Other protocols and applications might also benefit from the ability to send messages to any node in the system.

There have been extensive research performed in the context of the ad hoc network routing. In particular, solutions have been proposed where nodes have no mobility, while other solutions focus on the challenging problem of routing in MANETs.

For completeness we now briefly describe some of these works, whose techniques can be leveraged in our own work for designing protocols that leverage on the ability to send messages among specific nodes in the network.

Ad-Hoc On-Demand Distance Vector Routing

Ad-Hoc On-Demand Distance Vector Routing [62] (AODV) is a routing protocol designed for large scale and highly dynamic wireless ad hoc networks. It is a reactive routing scheme, in the sense that routes are established when a node wants to contact another node to which a route is unknown.

The algorithm is based on sending a message throughout the whole network in search of the destination node. These messages contain a sequence-number to overrule older messages, a hop-counter to determine the shortest path, and the source node's ID (IP address). Nodes forward these messages, incrementing the hop-counter and updating their routing table (storing the source ID and the hop-counter); when these messages reach the destination node, it replies to the message with the lowest hop-counter; when the reply reaches the source, a route has been established.

The routes established are temporary, as they are invalidated from time to time. This allows AODV to handle link changes, but on the other hand, it might also cause a high amount of re-transmissions in order to re-establish routes that are frequently needed.

Optimized Link State Routing Protocol

In contrast, Optimized Link State Routing Protocol [35] (OLSR) is a proactive routing protocol that relies on periodic routing table updates. The protocol has neighbouring nodes coordinate to decide on *multipoint-relay* nodes that cover two-hop neighbour nodes which are then tasked with the forwarding of data packets.

The algorithm begins by each node sensing their direct neighbours and then locally broadcasting their list of neighbours. The nodes, now have information about their two-hop neighbours and can decide on what nodes will he use as multipoint-relays. The nodes then, locally broadcast their list of multipoint-relays nodes (notice that it is the same message as before, but only containing a subset of neighbours). The multipoint-relay nodes will then broadcast the message throughout the network, while the nodes that are not present in the list will only processes it. Nodes will update their routing table with the received information about multipoint-relay nodes which will then be used to establish routes.

The use of these multipoint-relay nodes, allows OLSR to minimize the effects of a full network broadcast, thus having a low risk of saturating the network. This is very well suited for dense wireless networks, however, as the network size increases, so will the control packets necessary for maintaining the routing information. Furthermore, in networks that are more sparse and wide, the protocol will have difficulties in maintaining up-to-date information in the routing table, since nodes might observe frequent changes on their neighbours.

Better Approach to Mobile ad hoc Networking

Better Approach to Mobile ad hoc Networking [38] (BATMAN) is a proactive routing strategy similar to OLSR, although it tries to improve on the shortcomings of OLSR (e.g, maintenance of routing tables, bigger control messages). As such BATMAN relies solely on announcements that provide routing information.

Each node periodically announces a packet containing it's ID (IP address), the ID of the node who sent it (at the start, the originator node), and a sequence number to overrule older announcements. When a node receives an announcement, it changes the sender ID to its own, and re-transmits the announcement. Nodes count how many announcements have been received from each node by each link (direct neighbour) to determine the best (probable) next-hop for forwarding messages towards that node.

The algorithm presents no more control messages than the announcement messages that are kept with constant size. Nevertheless, the amount of effective re-transmitted messages can grow uncontrollably when the number of devices grow. Additionally, if all nodes transmit their announcements simultaneously a large number of collisions can happen. These negative effects an be somewhat mitigated by selecting only a few nodes to retransmit announcements.

Greedy Perimeter Stateless Routing

Greedy Perimeter Stateless Routing [42] (GPSR) is a reactive routing protocol that uses the positioning of nodes to determine where to forward a packet. However in order to obtain the position of neighbour nodes, GPSR relies on a proactive mechanism that has nodes periodically (locally) announce their positioning.

The protocol combines two techniques in order to establish routes between nodes: *greedy* routing and *perimeter* routing. Greedy routing has nodes forward packets to the node that is closest to the destination node. When a greedy decision cannot be made (i.e, there is no (known) neighbour closer to the destination), GPSR resorts to perimeter routing, where the node will make an heuristic decision to forward the packet around the perimeter of the area that lies between the current node and the destination that is unoccupied by other devices.

The protocol introduces minimal control overhead of routing packets. However, nodes are required to have a way of determining their position (GPS or other means) which might be impossible in some scenarios.

2.3.3 Discussion

All of the previously mentioned wireless networks are particularly tailored for different edge scenarios. Wireless mesh networks are usually employed to extend existing infrastructures or to facilitate the access to resources in settings such as companies and office spaces, where there is (some) lack of infrastructure. In this context, routing might be a desirable feature to facilitate the interactions with these resources, that can be Internet gateways, File Servers, or even printers. We note that these networks are not designed to allow distributed computations to occur across the devices that compose it. Moreover, particularly in scenarions where all devices belong to a company, one could leverage these networks to take advantage of idle resources in these devices as proposed in the context of peer-to-peer systems [8, 71].

Mobile and vehicular wireless ad hoc networks focus on the challenges that arise form the fact that devices are not stationary, which introduces additional dynamics, for instance, regarding the stability of the neighbours of each device. The focus of these networks is usually to provide effective routing schemes among devices to support some form of cooperation among devices, such as pass information or sharing context-sensitive data.

Wireless sensor networks focus on an extreme edge computation scenario, where large numbers of devices, typically resource constrained and heterogeneous in nature, interact among each other to propagate sensed information towards a sink node that exists somewhere in the network. Many times, these networks are supported by low cost wireless technologies such as ZigBee. In this context however, we note that some of the processing performed by the sink node could be performed in the network, if there were some nodes in it that had additional memory and computational capability (and eventually a source of power).

Altogether, these networks build upon a wireless ad hoc network to provide additional functionalities and to address specific challenges found in each of these scenarios.

In this work we aim at bringing computations towards the edge of the system, which implies that computations should be performed in-network, as devices exchange information among them. Contrary to the ad hoc networks presented above, whose focus lies on enabling any pair of nodes to exchange information through routing in adverse scenarios, we envision systems where devices interact naturally to achieve a common goal. Moreover, we aim at building systems that can grow organically without support from infrastructure and with self-organizing capabilities.

To achieve these goals we will operate directly at the ad hoc network level, without assuming any form of routing, nor any particular interaction pattern from devices. More specifically, routing should be handled at the application level such that it can operate specifically to provide the minimal functionality required by protocols or applications with minimal cost and overhead in the wireless medium.

In the following, we discuss decentralized communication strategies and self-organizing overlays that where originally proposed in the context of peer-to-peer systems, and that can benefit us in achieving the vision described above.

2.4 Decentralized Communication Strategies

A basic requirement to perform distributed computations is the ability to communicate and exchange information with other nodes, to either, coordinate, or transfer relevant data. As previously discussed, we target highly decentralized systems where nodes do not have access to infrastructure and hence, have to communicate using decentralized approaches.

We now study some of these mechanisms, that can broadly be characterize as being deterministic or random in relation to the patterns of message exchange among a set of nodes in a network.

2.4.1 Deterministic Communication Patterns

A deterministic communication pattern has nodes communicate in a predefined pattern, in other words, all messages transmitted by a node will take the same path in the network, to reach their destination. We can define two main deterministic communication patterns: *flooding/broadcast* and *topology dependent*.

Flooding/Broadcast

In a flooding/broadcast [49] pattern, a message sent by a node has, usually[1] the objective of reaching all nodes. In this approach, nodes send the message to all their neighbours and, receivers of that message, keep forwarding it, repeating the same pattern.

Topology Dependent

A topology dependent [36] pattern has nodes leveraging an underlying (logical) topology to propagate their messages to a particular element in the network (such as a sink node), with the objective of minimizing the communication cost.

Frequently used topologies include: *i)* ***single-path tree***, where nodes are arranged in a tree, and information can easily flow towards the root node, or be broadcasted downstream to all or a subset (i.e, branch) of elements in the tree; *ii)* ***multi-path tree*** is similar to the previous one with the exception that it contains redundant paths along the tree; *iii)* ***ring*** topology where nodes are arranged in a ring, allowing simple communication to relative portions of the ring; and finally *iv)* ***cluster-based*** topologies, where nodes are organized into clusters, which has nodes within the cluster communicating frequently, whilst communication between clusters is performed in a higher hierarchical level (i.e, cluster representative nodes communicating with each other).

2.4.2 Random Communication Patterns

Random communication patterns are, as the name implies, patterns that have nodes exchanging messages through random local decisions. There are two main random patterns: *gossip-based patterns* and *random walks*, that we now briefly define.

Gossip-Based Patterns

Gossip-based [45] patterns rely on having nodes exchanging messages by selecting subsets of other nodes in direct reach at random, usually in a per message basis.

The effective size of the subset selected for transmitting a message, depends on the goal of the communication step, as well as from operational aspects of the system deployment, such as the total number of nodes in the system or the number of reachable nodes of the node propagating a message.

The exchange of information among nodes usually follow a pair-wise model (where two nodes interact directly), and can be implemented using the following strategies: in *i)* ***push*** strategies, nodes that have relevant information, immediately send it to the subset of

[1]There are flooding techniques with limited horizon, where a message will be forwarded only for a pre-defined number of hops

receivers selected at random; in contrast, *ii)* ***pull*** strategies have nodes selecting random subsets of other nodes in their vicinity, and ask for relevant information through a pull request. Nodes that have relevant information can reply to these requests directly; finally, there are *iii)* ***hybrid*** approaches that combine the two previous ones, usually using push to quickly disseminate relevant information, and pull to recover lost updates.

Random Walks

Random walks [56] consists of a node choosing a random neighbour to propagate a message. This neighbour will forward this message following the same pattern (picking another random neighbour). This process continues until some criteria is met (e,g. maximum number of hops, reach target node or originator).

This mechanism allows messages to transverse the network through a random path, which allows a low communication overhead. Additionally, it is well suited to obtain random samples of information distributed across many nodes.

There are also variants of this approach where the selection of the node to whom to forward the message is not entirely random, but biased by some application specific criteria [11, 79].

2.5 Self-Managed Overlays in Peer-to-peer Networks

Since we aim at designing systems that are decentralized and have self-management properties, we need a subtract that simplifies the coordination among devices. Moreover, as presented before, some communication strategies rely on concrete topologies. One way to achieve both aspects is to rely on (logical) *overlay networks*.

Peer-to-peer systems effectively operate on top of logical networks. These networks are called overlay networks because they operate above another network layer, typically the physical network. An overlay network presents four main properties [46] of interest: *i)* ***Connectivity***, the overlay should be connected (i.e, there is at least one path from each node to every other node); *ii)* ***Degree Distribution*** nodes in the overlay should have a uniformly distributed number of neighbours; *iii)* ***Average Shortest Path***, the average path length between all nodes should be low, to provide efficient communication; and finally, *iv)* ***Clustering Coefficient***, which relates to the density of neighbourhoods, and how failures and message propagation affects these dense neighbourhoods (e.g, a high value of clustering coefficient may indicate that a loss of a few links may cause two dense neighbourhood areas to become partitioned).

Although these networks do not usually consider wireless environments, they provide interesting management mechanisms that should be considered in the context of the work presented here.

2.5.1 Overlay Solutions

There are two main classes of overlay networks: *structured* and *unstructured* overlays. As we present further ahead in the document, there are decentralized aggregation protocols that operate over both classes of overlays.

Structured overlays are logical networks that have a known topology (e.g, a ring, a tree, among others), on the other hand, unstructured overlays present a random topology. Structured overlays tend to achieve a higher performance for particular objectives (e.g, resource location or application level routing), but usually have high maintenance overheads. In contrast, unstructured overlays have a lower maintenance cost, but might not be as efficient as structured overlays, being more suitable for cooperative dissemination of information.

Overlays are built and maintained by distributed algorithms. These algorithms are associated with membership protocols. However, achieving a global membership (logical connections to all nodes) is a challenge in large scale systems, since maintaining up-to-date information about every node in the systems leads to a lot of control overhead. Consequently, these protocols rely on partial memberships (connections to logical direct neighbour nodes usually, a subset of all nodes) to achieve global connectivity.

For completeness, we provide some examples of membership protocols that might help us developing large scale edge computing distributed infrastructures.

Chord

Chord [72] is a structured overlay that provides the abstraction of a *distributed hash table* (DHT) with the purpose of delivering efficient resource location. For this, Chord relies on *consistent hashing* [41] to assign uniformly distributed keys (that map resources) to nodes, which are then arranged in a ring structure.

Consistent hashing requires that all nodes are able to communicate with all other nodes, thus requiring a global membership. Chord is able to relax this property by, having nodes store only connections to a subset of nodes, which are called *fingers*. These fingers are connections to nodes that are scattered in the DHT, in other words, a node stores the connections to the two direct successor nodes in the ring, plus nodes that are at an increasing number of hops in the ring (i.e, a node i has fingers $i+1$, $i+2$, $i+4$, $i+8$, and so on). With this, Chord is able to resolve the consistent hashing requirement by redirecting the request for a key to the (known) node whose identifier is closer to the intended key, which will consequently forward it, until it reaches the node responsible for the key.

The management of the finger table of each node is based on a periodic stabilization mechanisms, where each node uses the routing of the overlay to update its finger entries.

Hybrid Partial View

Hybrid Partial View, or HyParView, [49] is a protocol that builds and maintains an unstructured overlay for message dissemination using gossip techniques. To do so, HyParView maintains two partial views (i.e, partial memberships), a small *active view* that contains the nodes used to disseminate messages, and a larger *passive view* that is used to replace failed nodes from the active view.

The active view only changes in reaction to nodes joining or leaving the membership, hence, it is managed by a *reactive* strategy. When a node wants to join the membership, it must first contact a node within the membership to request entry. This contact node will trigger random walks that will update the passive views of (some of the) other nodes, and define the new node's active view. Moreover, every time a message is disseminated through the active view, the links are effectively tested, and if there is a node that has failed, the active view replaces the failed node with a node from the passive view.

The passive view, on the other hand, is maintained by a *cyclic* strategy, which has nodes periodically exchange information about both their passive and active views and swap out nodes from their passive view with the received information. This allows nodes that failed to eventually be removed from all correct nodes' passive views.

Kelips

Kelips [33] consists of groups of nodes that form a DHT to provide both effective resource location and (logical) routing in an efficient and robust manner.

In Kelips, each node is assigned to a group determined by a consistent hashing function, and maintains information about the nodes that constitute the group (e.g, IP address). This information is maintained up-to-date among nodes of a group by a gossip mechanism. Additionally, nodes keep information about contact nodes, that reside outside of their groups.

As stated before, membership information in Kelips is maintained up-to-date through a gossip protocol that periodically disseminates control information both, intra-group and inter-group. When a node selects a subset of nodes to disseminate information, it chooses a subset of nodes from his group and another from its contact nodes, thus providing other nodes in the same group and in another group with potential contacts. Furthermore, when a node wants to join the overlay, similarly to HyParView, it must contact a node that is already in the overlay. This contact node, will then help set up the new node, that will start gossiping and populate its view (the group information) with nodes in his assigned group and gathering information about other groups via gossip.

Cell Hash Routing

Cell Hash Routing [10] (CHR) is a DHT built for wireless ad hoc networks where nodes are considered to be mobile (MANET). Hence, instead of nodes forming the DHT, CHR defines (logical) zones, termed *cells*, that are the basic unit of routing in the DHT.

Cells are geographic zones where all nodes, that are within it responsible for the same keys of the DHT. As nodes move in and out of each cell, they change the keys to the ones that their current cell is responsible. However, this can easily lead to cells becoming empty and having no nodes to attribute keys to. To address this issue, when trying to find a resource in an empty cell, the request is redirected to the cell closest to the destination cell. This mechanism is applied by levaring on the OLSR protocol's perimeter forwarding which is used to build routes in CHR.

2.5.2 Discussion

Overlays are widely used in peer-to-peer networks where wired connections and routing infrastructure between the nodes are assumed. A common problem in overlay networks is the *topology mismatch* [46]. This happens when the overlay topology is highly divergent from the physical topology connecting nodes, which might make operating on top fo the overlay highly inefficient [47]. Simply using an overlay network in a wireless ad hoc network will incur in a topology mismatch. This will pose additional challenges, due to the lack of a efficient routing infrastructure to enable each node to efficiently contact an overlay neighbour. In such case, nodes may be forced to forward messages until they reach their destination, and thus waste significant network resources.

Although, even in the presence of a topology mismatch, this can be an interesting approach since, one could leverage the forwarding of messages across multiple nodes, to send additional information to other nodes, hence minimizing the waste of resources. Moreover, the overlay could be built and maintained while performing aggregations, collecting summarized information about the system to enable a better management of the (logical) neighbouring relations among nodes.

Additionally, overlay networks present interesting properties for high density physical neighbourhoods, providing the ability for nodes to only actively communicate with a subset of their physical neighbours. Another possible approach, for scenarios with high density neighbourhoods, is to construct the overlay network by manipulating the schedule of the periods in which each node has its radio active. Neighbouring nodes would therefore, activate their radios in the same time windows.

2.6 Aggregation

While it is important to support general purpose computations in the edge, in the context of this thesis we focus on building aggregation protocols at the edge. Aggregation protocols can serve as a building block for more general purpose computations. Aggregation can also help in reducing the amount of communication among nodes and can be used to monitor the network's state. However, given the distributed nature of edge computing scenarios in general, and wireless ad hoc networks in particular, we must carefully evaluate the properties of distributed aggregation computations.

Aggregation computations can be defined as computing an aggregation function [36] over a set of input values, where each device (or node) holds one of these values. Traditional aggregation functions such as Count, Sum, Average, Min and Max, present different properties which we must consider to understand the challenges of computing such functions in distributed environments. To this end, we will consider the definitions given in [36], which presents aggregation functions as having two properties: *Decomposability* and *Duplicate Sensitiveness*.

Decomposability: Decomposability is a property of an aggregation function that can be defined by composing other functions. However, we can distinguish functions that are self-decomposable and decomposable. Self-decomposable functions are functions that have commutative and associative properties, much like Min, Max, Sum, and Count. In more detail, these functions can be computed by recursive application.

Decomposable functions are functions that can be composed by applying some function to a self-decomposable function. The Average is an example of decomposable functions, which can be obtained by having pairs of values $(x, 1)$ where x is the input value and 1 is the count of values, summing these pairs as $(x + y, 1 + 1)$, and finally, computing the division of both computed sums.

Duplicate Sensitiveness: An aggregation functions is considered to be duplicate sensitive if it's result is sensitive to the presence of duplicates. For example, Sum and Count will output unfaithful values when duplicates are present, while Min and Max will output the same values regardless of duplicate values. In other words, this depends on the function being idempotent or not.

Given these properties, we notice that, on one hand, Min and Max are easier to implement and can be computed with algorithms operating with fewer guarantees, while on the other hand, Sum, Count and especially Average, require additional care and algorithms that address additional aspects.

2.6.1 Aggregation Computational Schemes

As presented before, each aggregation function has its own particularities. However, when considering large systems, obtaining exact values from the computation of an aggregation function might not be a requirement (e.g, obtaining the order of magnitude of the size of a system might be enough in some cases), while in other cases, it is crucial to obtain the exact values, at least in a small region of the system, to make a decision (e.g, the temperature of the soil to trigger irrigation on a specific area).

As such, we can define two main categories of techniques to compute aggregation functions, the ones that reach exact values, and another that calculates approximations and estimations.

2.6.1.1 Exact Value Computations

Exact value computations strive to obtain the exact value of the aggregation function that is being computed. In that regard, every input value present in the network must be taken into account. As a consequence, all nodes within the system must transmit their values at least one time (as long as there are no message losses).

Therefore, we describe two techniques that reach the exact value: *Full Dissemination* and *Hierarchical Aggregation*.

Full Dissemination Full dissemination techniques come as a naive approach to compute distributed aggregation functions. Every node in the system floods the network with a request for values, and every other node responds with their value. The computation is then performed in the initiator node.

Since every node is disseminating their values, any aggregation function can be computed at each node. However, this approach will cause a high message overhead in the network, which can easily lead to its saturation.

Hierarchical Aggregation In hierarchical aggregation techniques, nodes are disposed in a logical hierarchical topology (typically a tree), where the top of the hierarchy can be viewed as a special node (*sink* node). This node is responsible for building the hierarchy, after this, the nodes that are at the bottom of the hierarchy will send their values to upper levels. Upper level nodes, compute an intermediate aggregate with the values received from lower levels, and pass the result to upper levels. This process continues until it reaches the sink node.

These techniques, reduces the amount of messages transmitted in the network, as messages pass through pre-defined paths. Nevertheless, the cost of maintaining the hierarchy might be high, and on the presence of faults, hierarchical aggregation will reach unfaithful results.

2.6.1.2 Estimation Computations

Estimation computations rely on probabilistic methods to reach the approximate result of an aggregation function. They focus on reducing message size, reducing communication, and in some cases, ensuring fault tolerance. We describe three main estimation computation techniques: *Sampling*, *Data Representation Computation*, and *Iterative Approaches*.

Sampling Sampling techniques tend to focus on probabilistic counting techniques to determine an estimation (usually to determine the network size). These techniques commonly use random walks, or random selection of nodes within the system, to collect samples of input values that can be used to infer or compute an approximation of the global aggregation result.

Although, these techniques rely on probabilistic methods, and as such, they can easily incur in situations where the sampling is biased. This can happen due to an imbalance in contributions to the aggregation result across the network.

Data Representation Computation Data representation computation relies on auxiliary data structures to compute estimates. These data structures represent the values in the system, although the representation of values differs from protocol to protocol. Some protocols focus on representing values in an histogram to answer more complex aggregation functions (median, mode, among others), while others represent values through bit masks as a way of compressing them, or in a subset of representative values.

Nevertheless, these techniques require estimation functions to extract the aggregation result from the data structures. Furthermore, the compression techniques used are typically resource heavy, and might incur in high computational overheads that might be unacceptable in some cases (sensor networks).

Iterative Approaches Iterative approaches exploit mathematical properties of aggregation functions to compute estimations. These approaches rely on the principle of "mass convergence", which dictates that as long as the sum of the aggregated values in the network remain constant, it is possible to reach the correct approximation [43]. Consequently, nodes exchange information in order to compute partial aggregate results. The more exchanges happen the more approximate the result will be from the exact value.

However, these techniques can be very sensitive to duplicate messages and message loss, since it translates to gain or loss in "mass" which will violate the principle of "mass convergence".

2.6.2 Relevant Aggregation Protocols

We now present some of the exiting distributed aggregation algorithms that illustrate the classes of solutions discussed above.

Tiny Aggregation

Tiny Aggregation [53] (TAG) was developed to support aggregation queries in wireless sensor networks using TinyOS [50]. It performs computations using an hierarchical approach on top of a topology dependent communication pattern and provides an SQL-like query language. As such, it provides basic aggregation functions such as Count, Sum, Average, Min, and Max. TAG also allows queries to use the Group By operator, enabling aggregation functions to be captured over subsets of input values given certain nodes properties (e.g, the total sum of all input values of nodes provided their geographical area).

The protocol is composed of two phases: *distribution* and *collection*. The distribution phase consists in creating the routing topology (typically tree-based) from the *sink* node (root node) to all other devices. To do this, the root node broadcasts a query request message; every node within range that receives this message will mark the message originator node as his parent and adjust their radio wake up interval according to the parent node. Nodes keep forwarding the aggregation request until all nodes are assigned a parent.

In the collection phase, each parent node must wait for the values of his children. Although, since the children have adjusted their wake up interval with the parent, the parent node expects that all of his children report their values within that interval.

When the parent node receives the values from all his children nodes, it computes a partial aggregate result with his own value an forwards it to his parent node. This process continues until the root node receives replies from all its children.

Given the hierarchical nature of TAG, it trades-off robustness for communication efficiency. A single fault could disconnect a large portion of the network and incur in gross aggregation errors. TAG tries to mitigate these problems, by having each node keep a list of neighbours allowing them to switch parents, snooping messages by utilizing the wireless medium, and nodes keeping cached values of their children's previously reported values that can be used if no reply is received from one of the children nodes.

However, in highly dynamic and unstable networks, these solutions might only lead to additional control overhead.

Directed Acyclic Graph

Similarly to TAG, Directed Acyclic Graph [60] (DAG) is an aggregation protocol that relies on a hierarchical topology to perform hierarchical computations. Although, the routing topology offers multiple paths among nodes instead of a single path. It support the same aggregation functions as TAG, with the exception of the Group By functionality, since it does not offer an SQL-like interface.

When an aggregation request is received, the sink node broadcasts it to the network. The aggregation request performs similarly to the one described in TAG, however, adding a list of parents to the message. This list allows for child nodes to choose a grandparent, where their input value will be used to compute a partial result, thus allowing nodes to have multiple parents.

By forming a multi-path routing tree, DAG is able to mitigate the problems that TAG presents. Nevertheless, this comes at the cost of larger messages and more complex control, and even so, it is not guaranteed that every node will have multiple parents.

Furthermore, DAG can still lead to the same issues presented in TAG (gross aggregation errors), when a grandparent node suffers from a fault, all his grand-children's input values will be lost.

Push-Sum Protocol

The Push-Sum protocol presented in [43], relies on gossip communication combined with an iterative approach to compute aggregation functions, such as Average, Sum, and

Count. The protocol operates by having nodes exchange messages with their neighbours in a pair-wise fashion.

In push-sum, each message contains a pair (v_i, w_i), where v_i and w_i is the local value, currently computed, and the weight of the value at node i, respectively. The initial value of a node is its input value and the initial weigh depends on the aggregation function to be computed. The protocol exchanges messages in the following way:

A node splits it's last computed value and weight in half, sends one half and keeps the other. When a node receives a message it simply adds the received value and weight to the local ones. The approximate aggregation result value can be calculated at any given time by dividing the computed value and the weight (v_i/w_i).

The more iterations the protocol performs, the more accurate approximation can be calculated. To calculate different aggregation functions the protocol allows the manipulation of the initial weights of values used by each node. For Average the weights are set to one in all nodes; for Sum, the initiator node, has weight one, while all the other nodes have weight set to zero. Count differs from Sum by having the value set to one in all nodes (maintaining the weight set to one in the initiator and zero in all other nodes).

Push-sum has the advantage of not requiring to maintain a topology due to its random communication pattern, provided by the gossip communication strategy. Nevertheless, a node never knows when the approximation is good enough, consequently, termination in this protocol is an issue. Moreover, the protocol is only correct if the principle of "mass convergence" is kept, thus, the links between nodes must be reliable (i.e, messages cannot be lost or duplicated).

Distributed Random Grouping

Distributed Random Grouping [20] (DRG), is an aggregation protocol tailored for wireless sensor networks that leverages the broadcast nature of the wireless medium to create random local groups that calculate local aggregates. The algorithm is followed by iterative steps, until the aggregate value converges. The algorithm allows to calculate Average, Max, and Min, but does not specify how other functions could be calculated. In each step, nodes can be presented as having three states: *idle*, *member*, and *leader*.

A step of the protocol consists of an idle node, choosing with some probability p to become a leader of a group. When one does, it sends a message to all reachable neighbours, to notify them that he is the leader. All other idle nodes that receive this message send an acknowledgement message (JACK), to the leader node, containing their local aggregated value (at the beginning their input value). Once the leader receives the JACK from his neighbours, it calculates the aggregated value with his own value, and broadcasts again, to his neighbours the aggregated value, thus updating the local group aggregated value.

In order to converge, groups must overlap overtime, propagating the previously computed aggregate from one group to another. Nevertheless, this algorithm presents the same advantages and drawbacks of Push-Sum. There is no need to maintain a topology, but losing a JACK message or an updating message, implies “mass” loss, which leads to the computation of incorrect aggregation values.

Flow Updating

Flow Updating [37] performs gossip-based communication using an iterative approach based on the concept of *flows* from graph theory, where each flow is a representation of a differential between the approximation computed by two nodes. The flow from node i to j is f_{ij}, and the flow from j to i is f_{ji}. Flows have a symmetric property, and as such, $f_{ij} = -f_{ji}$. Flow Updating levarages this property to calculate estimations about the real aggregation result, while in some cases, obtaining the exact result. It is tailored to compute the Average aggregation function.

The key idea of this algorithm is, instead of exchanging “mass” like Push-Sum or DRG, nodes exchange flows and estimates which are based on the original input value of each node that is kept unaltered. With this, the protocol is able to recover “mass” in the event of message losses.

At the start of the algorithm, each node sets the flows and estimates of their neighbours to zero, as it has no knowledge about them. Given this, it calculates his local estimate by subtracting all flow values to his original value. It further computes the flows for each neighbour by updating the currently held flow to a neighbour with the difference between the previously calculated local estimate and the previously held local estimate calculated by that neighbour (at the start zero). Then, the nodes send to each of their neighbours a message containing the flow computed to that neighbour and the calculated local estimate. When a node receives these messages, it updates the flow to the originator by storing the inverse of the received flow ($f_{ij} = -f_{ji}$) and the received estimate.

Periodically (after receiving the updates), each node can recalculate the local estimate with the updated values for the flows of each neighbour and the estimate values received, and recompute the flows to each neighbour. Nodes exchange this information until there are no more changes (the flows are zero), obtaining the aggregated result in all nodes eventually.

This protocols presents a significant advantage in relation to Push-Sum and DRG with the ability to sustain message loss and not lose “mass”. However, it is unclear how the algorithm can be generalized to perform other aggregation functions. Moreover, in the presence of node failures or link changes, the algorithm does not specify how the computation should proceed (i.e, the author assume a static topology among nodes).

Extrema Propagation

Extrema Propagation [12], combines a data representation computation technique with a gossip communication strategy. With this, Extrema Propagation is able to compute an estimation of the real aggregation result of the Sum aggregation function. Each node produces a vector with k random numbers that follow a known distribution (e.g, Exponential) and exchange this vector with their direct neighbours.

When a node receives a vector, it calculates the pointwise minimum[2] of his local vector and the received one, keeping the result as his local vector. Each node proceeds by exchanging vectors until there are no changes for T rounds (T is a configurable parameter). When the algorithm terminates, each node will have the same vector containing the minimum numbers in the system for each position in the vector. The algorithm terminates by calculating the maximum likelihood of the minimum vector, obtaining an estimation about the aggregate result based on the mathematical properties of the initial distribution of values computed by nodes.

An interesting factor about this protocol, is that the error of the estimation depends only on the size of the exchanged vector. The bigger the vector, the lower the error bound. This could come as an advantage in very large systems, where messages need to be kept small. Nevertheless, in smaller systems it is hard to justify the effort of generating random numbers that follow a known distribution, to obtain an estimate value, when the input values of all nodes fit in a single message (that is relatively small) and, contrary to this solution, able to reach an exact value.

Q-Digests

Q-Digest [68] leverages on data representation computation to provide the ability to compute more complex aggregation functions as median and mode. Each node computes a data structure, named quantile digests (q-digests) and propagates these data structures (in a compact form) along a spanning tree to reach a sink node. A q-digest is a data structure that consists of a set of buckets that represent the frequency of values within a range (i.e, an histogram of classes of values).

Each node contains a set of values within some range and the frequency of their values. The objective of the q-digest is to compress this data. To do so, these values are represented in a binary data tree. Each leaf node represent a value within a range. Values flow up the tree when their frequency is not representative enough, following properties that relate the total count of values in a sub-tree to a compression level.

In other words, one could see each step of the tree as a bucket of values, the higher it is in the tree, the more values are represented by the bucket. The q-digest, once computed will be represented as the set of buckets that are most representative (given the total count of values).

The q-digest are propagated through the spanning tree where parent nodes are tasked to merge the received q-digests with their own. This is done by simply adding the counts of the received buckets to their own and recomputing the q-digest. The algorithm terminates, when the sink node computes the final q-digest, obtaining a full histogram of the most representative values in the network.

[2] the minimum number at each position of a set or function

This protocol, by leveraging on the q-digests, presents a compression mechanism that allows to compute more complex queries. Although, q-digests require that there is some knowledge about the values that the network is producing, so that it is possible to build compatible q-digests (that have the same initial range of values) to merge them efficiently, avoiding to generate a single huge bucket where all values belong. However, such scenario might be impossible in some cases.

Randomized Reports

Randomized Reports [13] is a probabilistic polling (sampling) method to determine an estimate of the network size, as such, it only supports the Count aggregation function. It relies on a node flooding the network with a request message and setting a timer T.

Each node that receives the message, replies to the originator with some probability p. When T expires, the originator node counts the number of replies and estimates the size of the network by scaling the obtained count by $1/p$.

This is a very simple algorithm that is able to mitigate the effects of flooding messages, by not having all nodes respond to the request message. Nevertheless, this algorithm only support the Count aggregation functions, and thus presents limitations for other aggregation functions, such as Average, where more information is required.

Random Tour

Random Tour [56] consists in utilizing random walks to collect samples on the network's size in peer-to-peer networks. An initiator node, begins the process by sending a sample message to a randomly chosen neighbour, containing a tag. This tag contains a counter X with the value $1/d_i$ (where d_i corresponds to the degree[3] of the node i) and the node's ID. When a neighbour node j receives the sampling message, it increments the counter X by $1/d_j$ and forwards the sampling message to another randomly chosen neighbour. Once the message is received again by the initiator node i, the node estimates the network size by calculating $d_i X$.

The Random Tour algorithm is able to achieve good estimates of the network size, but when the network is very large (e.g, up to 100,000,000 nodes) it will take a long time to reach the estimated value at the cost of a significant number of message exchanges.

Sample & Collide

Sample & Collide [56], similarly to Random Tour, leverages on random walks to perform probabilistic counting to determine an estimate of the network size in peer-to-peer networks. The algorithm is inspired by the "Inverted Birthday Paradox" [13], as it acquires random samples from peers, and then calculates estimates based on how many random samples are required before two samples return to the same peer.

[3]number of neighbours of a node i

The algorithm executes as follows: First, the initiator node i sets a timer with value T and sends a sample message containing T, to a randomly chosen neighbour. Upon receiving the sampling message, a node j, computes a uniformly distributed random number U between [0,1] and decrements T by $-log(U)/d_i$ (where d_i is the degree of node i). Then the node verifies if T is less or equal to zero; if it is, the node is sampled and returns its ID to the initiator node; otherwise, it will forward the message to a randomly chosen neighbour with the updated timer.

The algorithm terminates when some node has been sampled for a configurable amount of times l, and an estimate of the network size is calculated through a Maximum Likelihood method.

The algorithm, when compared to Random Tour, achieves lower accuracy with fewer message exchanges. As such, the algorithm is best suited for larger systems where estimations can have lower accuracy. Nevertheless, the algorithm presents the same limitation as Randomized Reports (can only compute the Count function).

2.6.3 Discussion

The presented algorithms provide different trade-offs to the distributed aggregation computation problem. Some present very efficient methods to compute exact aggregation results, but require methods to adapt in adverse scenarios, such as TAG and DAG. Others rely on probabilistic methods to compute estimates about the result, while requiring less communication efforts in the same scenarios (Random Tour, Sample & Collide, Flow Update). Extrema Propagation, achieves the estimate values with the same communication efforts as probabilistic methods, but requires more computational power (as does Q-Digest). Additionally, probabilistic methods take more time to compute the aggregation function, or have issues with termination (Push-Sum, DRG).

As such, we can identify three metrics to evaluate an aggregation protocol: the *cost* in communication and computation that the protocol requires in order to compute the aggregation result; the *precision* of the obtained result, if it is exact or approximated, and how good is the approximation (error bound); and finally, the *time* it takes to compute the aggregation result in the network. Nevertheless, none of the presented algorithms excels at all three dimensions.

Furthermore, we also identify another issue with these protocols, which relates to how well do protocols adapt to different environments. While some protocol require that message are not lost (Push-Sum), others, on the presence of faults are unable to terminate, or generate highly inaccurate responses. This can lead to suboptimal decisions, when aggregation results are being used to make management decisions regarding an edge system.

2.7 Frameworks for building Distributed Applications

Most of the presented solutions in the previous sections have all been implemented on simulators, specialized hardware, or on wired networks. However, when looking at edge computing scenarios, and on the particular edge scenario that relies on wireless ad hoc networks, we must understand how these solutions work in practice.

To this end, we need a set of tools that will help us build and execute some of the distributed algorithms discussed, among other support distributed protocols, in order to evaluate how well they fit in a real life edge scenario. Frameworks for building distributed applications have been proposed in the past, although, they have been developed for different use-cases (e.g, group communication, sensor networks).

2.7.1 Isis and Horus

Isis [14–16] and Horus [75] are both toolkits that provided abstractions for programmers aiming at building distributed applications for clusters of computers using group communication [77].

Both Isis and Horus are protocol kernels, where each protocol is composed by a set of layers that are implemented by micro-protocols. When a message is sent or received it must pass through all layers.

These layers can be arranged in any order as long as they respects their (individual) dependencies. Each protocol must implement interfaces that support the upwards and downwards interactions with other protocols.

Isis focus lies on providing fault-tolerance mechanisms to the programmer, however Isis assumes a model were nodes crash but no network partitions can occur. Horus, on the other hand, builds further on the Isis programming model to support network partitions, while focusing on important group communication primitives, such as membership management, message ordering, view synchrony among others. Furthermore, Horus also implements special protocols that allow messages to bypass layers within the stack in order to improve performance.

Both of these frameworks are implemented in different languages, namely C and C++, and support low level programming. Although, as their focus lies on complex and costly group communication abstractions, there is a lack of support for wireless network primitives.

2.7.2 APPIA

Appia [57] is a protocol kernel that is tailored for building Quality of Service (QoS) protocols by combining micro-protocols, to support application that need to use different communication channels (e.g, multimedia applications). The QoS can be viewed as a stack of layers. Each layer is implemented by a micro-protocol in which, each instance is termed a session.

The micro-protocols are event based, and interact with each other through events. Appia implements a single threaded event scheduler that handles all events in the Appia system. This event scheduler is responsible for handling and delivering each event to the micro-protocols that should process the event, respecting their order in the stack.

Appia is implemented in Java and, as such, leverages on the inheritance mechanisms of Java Objects to allow programmers to customize their protocols and events easily just by extending top level objects defined by the Appia framework.

However, Appia was developed to support view synchrony protocols and group communication, consequently, it considers a wired environment and does not support wireless primitives, such as one hop broadcast. Furthermore, Java has no support for low level network primitives, and their network primitives consider TCP and UDP traffic only, which are not suited to build one hop broadcast primitives in wireless ad hoc networks.

2.7.3 TinyOS

TinyOS [50] is a lightweight operative system designed to provide tools and abstractions for programmers building sensor network applications. It is based on a set of reusable components that fit together to support some specific application. The programming model is event driven and is based on split phases, asynchronous *events*, and computational *tasks*.

Each application built in TinyOS is composed by a set of *components wired* together. A component defines interfaces that provide three abstraction: *command*, *event*, and *task*. Commands are explicit requests for a component, events are asynchronous responses to commands or other interrupts (e.g, hardware signals, message arrivals), and task are computational tasks triggered by either commands or events. In other words, commands and events are interactions inter-components, while tasks are interactions intra-component.

All these abstractions are processed asynchronously and handled by TinyOS's event scheduler, that by default implements a FIFO policy. These components interact with each other by wiring them together, which requires providing the complete set of components that an application uses at compilation time.

Furthermore, TinyOS already provides several components that include abstractions for sensors, single hop networking, ad hoc routing, power management, timers, and non-volatile storage.

When compiling a TinyOS program, the binary generated will be a single process application that runs on a few specialized hardware, which makes sense in a sensor perspective, since sensors have limited resources and usually run a single application. However, if we consider commodity hardware that runs a Linux based operative systems for example, this is not ideal, as we may want to have a set of services or applications running independently for different purposes.

2.7.4 Impala

Impala [51] is a middleware system that is intended to act as a lightweight operative system for sensors. The key concern of Impala is to ease the deployment and updates of sensor network applications, thus, it privileges modular applications. Moreover, Impala also presents another interesting feature, which is the ability to change application on the fly, adapting to the conditions of the device or the environment.

The applications are built by a set of protocols that are event driven. Each protocol is implement as a set of event handlers. Impala further offers a user library containing network utilities, timer utilities, and device utilities.

Impala is able to have multiple application loaded, however only one can be active at each time. All applications share the same storage, and, as such, must agree upon the basic storage organization.

To manage applications and the reception of events, Impala is composed by three main components: the *application adapter*, the *application updater*, and the *event filter*.

The application adapter is responsible to change between applications and manage their life cycle. The application updater is responsible for keeping track of the versions of the modules used by the applications, in order to perform remote updates.

The event filter captures and dispatches events between the device and the application. Impala defines five types of events: *i)* ***Timer***, which signal timers that have expired; *ii)* ***Packet***, that represents network messages; *iii)* ***Send Done***, is a notification about a successful, or unsuccessful, transmitted packet; *iv)* ***Data***, represents new information sensed by sensors; and, *v)* ***Device*** that signals a device failure.

Impala was implemented and tested in a Linux based system, with the objective of later being ported to sensors in ZebraNet [52]. As such, Impala seems to cover all the conditions that we are looking for to build distributed protocols that support our setting. However, there is no code available online to be used. Thus making Impala impossible to use in practice.

2.7.5 Discussion

As we have discussed in this section, some frameworks exist to build distributed protocols. Although, some do not address or take into consideration the wireless setting, like Isis, Horus, and Appia. Others, like TinyOS and Impala, are either too specific or impossible to use.

This motivates us to build our own framework and set of tools to provide the desired abstractions and execution support for ad hoc wireless networks. We detail our prototype further ahead in the document.

2.8 Summary

In this Chapter we presented the limitations of Cloud Computing, which motivated us to look beyond it for solutions. Hence, we detailed the Edge Computing paradigm that presents a broad spectrum of different scenarios.

We chose to focus on the particular edge scenario of wireless ad hoc networks in commodity devices, and presented the challenges that are inherent to the wireless medium. We studied existing wireless ad hoc networks, and realised that none is notably suited for efficient Edge Computing.

In order to achieve reliable computations in the edge, we presented communication strategies that are used in in-network computations. We also detailed overlay networks in peer-to-peer settings as an inspirations for building and maintaining large scale networks, while offering abstractions and semantics to the network for protocols operating above them.

We continued by describing aggregation computation as a simple, yet fundamental, piece of computation and that can serve as an essential building block for more generic computations in the edge. We presented some existing aggregation protocols that perform distributed computations, and discussed their advantages and limitations.

Finally, we discussed the importance of understanding how would this distributed computations behave in real edge scenarios, and presented frameworks for developing distributed protocols and applications.

However, these tools do not suffice our goal, and as such in the following Chapter we detail our prototype framework, that we named Yggdrasil, and propose a distributed computation solution more suited for large scale wireless ad hoc networks.

Chapter 3

Planning

In this Chapter we start by briefly presenting the (current) prototype of the Yggdrasil framework (Section 3.1), as the work that has been under development at the same time as this thesis preparation. We further detail how we can leverage Yggdrasil as a tool to achieve efficient and reliable aggregation at the edge, by implementing existing aggregation protocols with support from Yggdrasil and by proposing a new aggregation protocol (Section 3.2).

We continue by detailing the evaluation plan of the implemented protocols (Section 3.3), as to suitably study these protocols according to reasonable and adequate performance metrics.

Lastly, we present the schedule for the future work to be developed in the remainder of the thesis duration (Section 3.4).

3.1 Yggdrasil

As of now we have developed a prototype framework which allows us to develop protocols with ease, on top of the wireless ad hoc networks. This framework comes as a necessity since there was no adequate solution to support the implementation of wireless ad hoc networks protocols, which could take advantage of the one hop broadcast primitive on commodity devices running Linux.

The framework, denominated Yggdrasil, provides us with the necessary tools to develop and test the protocols that we want to implement. Yggdrasil helps programmers abstract from configuring the device to support ad hoc networking and provides tools that are important when implementing a distributed algorithm, such as the management of timers, the transmission of a message and mechanisms to enable protocol interactions and foster synergy among protocols and applications (e.g, mechanisms to simplify piggy-backing of information in messages among different protocols).

In Yggdrasil, we can think of protocols being independent modules rather than stack layers. Each protocol executes within the context of its own thread, that should implement a control loop to handle events. These events can be network messages, triggered timers, events or request/replies produced by other protocols running in parallel. All these events are delivered and managed by a common runtime that all protocols share.

To build an application with Yggdrasil, one only has to register the set of implemented protocols, that are available, and the application in the runtime, and implement the event handlers in the application.

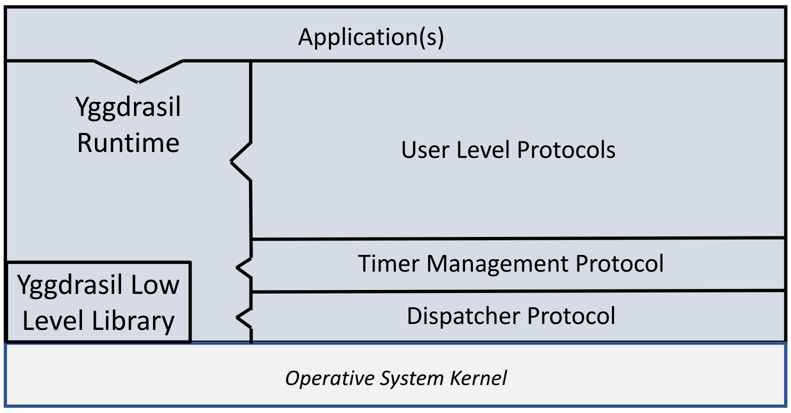

Figure 3.1: Yggdrasil Architecture Overview

In Figure 3.1 we provide an overview of the Yggdrasil architecture. Yggdrasil contains a low level library that implements system calls in order to configure the radio device properly to support ad hoc communication. The Dispatcher Protocol is used to send and receive messages from network. The Timer Management Protocol handles all timers that are set within an Yggdrasil application or protocol. Furthermore, as one can see, all components of Yggdrasil relate to the Yggdrasil Runtime, that manages all interactions between them.

Yggdrasil is written in C, and as of now, we implemented discovery protocols, a global membership protocol, three different aggregation protocols, a simple TCP server, fault detection protocols, and even some protocols that serve as tools for testing purposes, such as creating a virtual multi-hop topology across real devices.

We tested Yggdrasil using multiple (small) test applications in a fleet of twenty four RaspberryPis 3. Due to space limitation we omit further details about the framework in this document. The interested reader can find additional information in [22].

3.2 Aggregation Protocols

As previously stated, we have implemented three aggregation protocols by leveraging Yggdrasil. We have two protocols that are based on a broadcast mechanism and another that uses a tree approach that is similar to TAG [53] (with the exception of the radio scheduling). All implemented protocols are able to compute Average, Sum, Count, Min, and Max functions reaching an exact result.

The implemented broadcast aggregation protocols, rely on a similar dissemination approach as to the one employed by Extrema Propagation [12]. The first broadcast protocol operates by having each node perform a one hop broadcast periodically, containing the identifier of the node and its input value. Initially, each node only knows its own contribution (i.e, its input value). Whenever new information is received a node updates its estimates of the aggregated value, and stores the received contribution to be propagated in the future, alongside its own contribution. The protocol terminates when there are no more new contributions received by a node for T rounds (where T is a parameter).

We noticed that this protocol is not suited for large scale systems, as each propagated message will grow in size at each round. Hence, we designed a second broadcast protocol that is able to cope with this issue by having multiple contributions compresed in a bloom filter [17]. The protocol follows the same strategy as the previous one, but the messages are composed of the current aggregated value, the node's contribution (pair: input value and ID), the number of contributions that the aggregate value considers, and a bloom filter containing all nodes' IDs that have contributed to the aggregated result so far.

The message processing in this protocol is also different, and goes as follows:

1. Test if the received bloom filter is equal to the local one; if it is, do nothing; if not, continue;

2. Test if the received contribution is in the local bloom filter; if it is continue; if it is not, apply the function to the current local aggregated value and the respective input value, add the contribution to the local bloom filter, store the contribution, and continue;

3. Test all stored contributions (the node's contributions and their neighbours' contribution) against the received bloom filter, add the missing ones, applying the aggregation function to the received aggregate result and the respective input value.

4. Keep the aggregate result and respective bloom filter that has the more contributions; in case of ties, use some heuristic to determine which to keep (e.g, pick one at random).

We also plan to implement more aggregation protocols, namely Push-Sum [43], DRG [20], Flow Updating [37], and Random Tour [56]. With the addition of adapting them to the wireless environment. Furthermore, we plan to extend the existing implementation of the tree-based aggregation to deal with failures.

By implementing these protocols we will be able to better understand how they fit in the wireless environment and how could they work in conjunction. Because a large scale wireless ad hoc network may behave differently in different regions, we propose to implement a novel protocol that is able to adapt to the (local) network conditions by employing different aggregation and communication mechanisms in different parts of the network. Additionally, we plan to address the challenging task of designing and implementing continuous aggregation protocols, as a means for providing a reliable monitoring service that will be crucial to maintain large scale edge computing networks.

3.3 Evaluation Planning

To evaluate our work, we will use our test-bed composed of twenty four Raspberry Pi 3, that we have available now. However, in the case that scalability should be tested in larger scale systems, we will use simulation (e.g, NS-3 [65]).

We plan to evaluate the implemented protocols using the previously described metrics: *cost*, *precision*, and *time*. For cost, we will analyse the CPU and memory usage in each device for each algorithm, furthermore, will look into the average disseminated message size, and the total number of messages that is required to obtain the aggregation result. For precision, we will evaluate, the difference between the obtained aggregation result and the exact result. Finally, we will measure the time it takes to complete the distributed aggregation task, in a single node, and in the whole network.

We will use to a two phase testing methodology. Firstly, we will test the aggregation protocols locally resorting to a virtual topology simulating a multi hop network between the devices. However, this can lead to a high amount of interference in the wireless medium, as all devices are within range of each other. Consequently, in a second phase, we intend to deploy our devices in a building (e.g, the informatics department), to provide a more real testing environment. While this environment still has radio noise (due to access points and laptops among other devices) this will capture a setting that is closer to real deployments.

Nevertheless, the second phase requires that some remote control mechanism is employed, as to simplify the task of executing multiple experiments, potentially using different algorithms and protocols. As such, we plan to implement a remote control network, potentially resorting to variants of HyParView [49] and Plumtree [48].

3.4 Scheduling

In order to achieve our objectives, we organize the work plan in the following four main tasks and respective subtasks:

Task 1: Yggdrasil Development

Task 1.1: Optimization for existing prototype

Task 1.2: Development of evaluation tools

Task 1.3: Addition of other features

Task 2: Aggregation Protocols

Task 2.1: Implementation of existing aggregation protocols

Task 2.2: Development and implementation of new aggregation protocol

Task 2.3: Optimization of the new aggregation protocol

Task 3: Experimental Evaluation

Task 3.1: Yggdrasil performance test

Task 3.2: Comparative study of existing aggregation protocols

Task 3.3: Evaluation of new aggregation protocol

Task 4: Writing

Task 4.1: Paper reporting evaluation of existing aggregation protocols and design and evaluation of a new protocol (potentially for SRDS 2018)

Task 4.2: Paper reporting design and evaluation of Yggdrasil (potentially for Middleware 2018)

Task 4.3: Thesis writing

Task 4.4: Paper with summary of the work for Portuguese national conference (Inforum 2018)

The time plan for these task is presented in Figure 3.2, we also note that the time plan is ambitious due to the research context of thesis, which has to produce research results in the form of publications, experimental results, prototypes and written reports for the Lightkone European project deliverables.

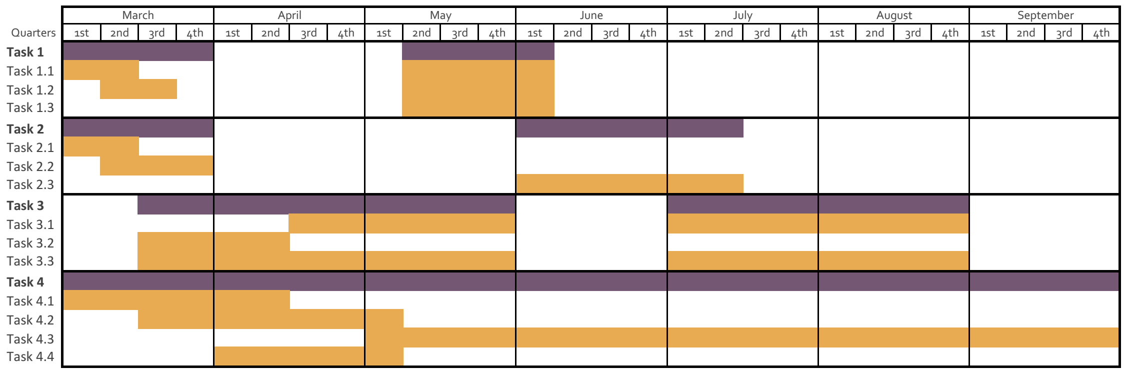

Figure 3.2: Work Plan

The conferences selected for submission of results associated with the work conducted in this thesis was based on the topic of the conference and the contribution. As well as the timing of the submission period, being compatible with the planning of the thesis. Target conferences might need to be adjusted in the course of the work.

Bibliography

[1] I. Aad and C. Castelluccia. "Differentiation mechanisms for IEEE 802.11." In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 1. Apr. 2001, 209–218 vol.1. doi: 10.1109/INFCOM.2001.916703.

[2] M. Akter, A. Islam, and A. Rahman. "Fault tolerant optimized broadcast for wireless Ad-Hoc networks." In: *2016 International Conference on Networking Systems and Security (NSysS)*. Jan. 2016, pp. 1–9. doi: 10.1109/NSysS.2016.7400690.

[3] I. F. Akyildiz and X. Wang. "A survey on wireless mesh networks." In: *IEEE Communications Magazine* 43.9 (Sept. 2005), S23–S30. issn: 0163-6804. doi: 10.1109/MCOM.2005.1509968.

[4] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "A survey on sensor networks." In: *IEEE Communications Magazine* 40.8 (Aug. 2002), pp. 102–114. issn: 0163-6804. doi: 10.1109/MCOM.2002.1024422.

[5] I. F. Akyildiz, D. Pompili, and T. Melodia. "Challenges for Efficient Communication in Underwater Acoustic Sensor Networks." In: *SIGBED Rev.* 1.2 (July 2004), pp. 3–8. issn: 1551-3688. doi: 10.1145/1121776.1121779. url: http://doi.acm.org/10.1145/1121776.1121779.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications." In: *IEEE Communications Surveys Tutorials* 17.4 (Oct. 2015), pp. 2347–2376. issn: 1553-877X. doi: 10.1109/COMST.2015.2444095.

[7] C. Ameixieira, A. Cardote, F. Neves, R. Meireles, S. Sargento, L. Coelho, J. Afonso, B. Areias, E. Mota, R. Costa, R. Matos, and J. Barros. "Harbornet: a real-world testbed for vehicular networks." In: *IEEE Communications Magazine* 52.9 (Sept. 2014), pp. 108–114. issn: 0163-6804. doi: 10.1109/MCOM.2014.6894460.

[8] D. P. Anderson. "BOINC: A System for Public-Resource Computing and Storage." In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. GRID '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10. isbn: 0-7695-2256-4. doi: 10.1109/GRID.2004.14. url: http://dx.doi.org/10.1109/GRID.2004.14.

[9] N. Anwar and H. Deng. “Ant Colony Optimization based multicast routing algorithm for mobile ad hoc networks.” In: *2015 Advances in Wireless and Optical Communications (RTUWO)*. Nov. 2015, pp. 62–67. DOI: 10.1109/RTUWO.2015.7365721.

[10] F. Araujo, L. Rodrigues, J. Kaiser, C. Liu, and C. Mitidieri. “CHR: a distributed hash table for wireless ad hoc networks.” In: *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*. IEEE. 2005, pp. 407–413.

[11] Y. Azar, A. Z. Broder, A. R. Karlin, N. Linial, and S. Phillips. “Biased Random Walks.” In: *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*. STOC ’92. Victoria, British Columbia, Canada: ACM, 1992, pp. 1–9. ISBN: 0-89791-511-9. DOI: 10.1145/129712.129713. URL: http://doi.acm.org/10.1145/129712.129713.

[12] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus. “Extrema Propagation: Fast Distributed Estimation of Sums and Network Sizes.” In: *IEEE Transactions on Parallel and Distributed Systems* 23.4 (Apr. 2012), pp. 668–675. ISSN: 1045-9219. DOI: 10.1109/TPDS.2011.209.

[13] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. *Estimating Aggregates on a Peer-to-Peer Network*. Technical Report 2003-24. Stanford InfoLab, Apr. 2003. URL: http://ilpubs.stanford.edu:8090/586/.

[14] K. Birman and R. Cooper. “The ISIS Project: Real Experience with a Fault Tolerant Programming System.” In: *Proceedings of the 4th Workshop on ACM SIGOPS European Workshop*. EW 4. Bologna, Italy: ACM, 1990, pp. 1–5. DOI: 10.1145/504136.504153. URL: http://doi.acm.org/10.1145/504136.504153.

[15] K. P. Birman. “Replication and Fault-tolerance in the ISIS System.” In: *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*. SOSP ’85. Orcas Island, Washington, USA: ACM, 1985, pp. 79–86. ISBN: 0-89791-174-1. DOI: 10.1145/323647.323636. URL: http://doi.acm.org/10.1145/323647.323636.

[16] K. P. Birman. “Replication and Fault-tolerance in the ISIS System.” In: *SIGOPS Oper. Syst. Rev.* 19.5 (Dec. 1985), pp. 79–86. ISSN: 0163-5980. DOI: 10.1145/323627.323636. URL: http://doi.acm.org/10.1145/323627.323636.

[17] B. H. Bloom. “Space/Time Trade-offs in Hash Coding with Allowable Errors.” In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: http://doi.acm.org/10.1145/362686.362692.

[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. “Fog Computing and Its Role in the Internet of Things.” In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC ’12. Helsinki, Finland: ACM, 2012, pp. 13–16. ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. URL: http://doi.acm.org/10.1145/2342509.2342513.

[19] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. "Design and Implementation of a Routing Control Platform." In: *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*. NSDI'05. Berkeley, CA, USA: USENIX Association, 2005, pp. 15–28. URL: http://dl.acm.org/citation.cfm?id=1251203.1251205.

[20] J.-Y. Chen, G. Pandurangan, and D. Xu. "Robust Computation of Aggregates in Wireless Sensor Networks: Distributed Randomized Algorithms and Analysis." In: *IEEE Transactions on Parallel and Distributed Systems* 17.9 (Sept. 2006), pp. 987–1000. ISSN: 1045-9219. DOI: 10.1109/TPDS.2006.128.

[21] Cisco. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update*. https://tinyurl.com/zzo6766. 2016.

[22] L. Consortium. *D5.1: Infrastructure Support for Aggregation in Edge Computing*. https://github.com/LightKone/wp5-public/blob/master/deliverables/D5.1.pdf. URL: \url{https://github.com/LightKone/wp5-public/blob/master/deliverables/D5.1.pdf} (visited on 02/09/2018).

[23] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[24] I. Demirkol, C. Ersoy, and F. Alagoz. "MAC protocols for wireless sensor networks: a survey." In: *IEEE Communications Magazine* 44.4 (2006), pp. 115–121. ISSN: 0163-6804. DOI: 10.1109/MCOM.2006.1632658.

[25] T. Dillon, C. Wu, and E. Chang. "Cloud Computing: Issues and Challenges." In: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. Apr. 2010, pp. 27–33. DOI: 10.1109/AINA.2010.187.

[26] S. C. Ergen and P. Varaiya. "TDMA scheduling algorithms for wireless sensor networks." In: *Wireless Networks* 16.4 (May 2010), pp. 985–997. ISSN: 1572-8196. DOI: 10.1007/s11276-009-0183-0. URL: https://doi.org/10.1007/s11276-009-0183-0.

[27] S. C. Ergen and P. Varaiya. "TDMA scheduling algorithms for wireless sensor networks." In: *Wireless Networks* 16.4 (2010), pp. 985–997. ISSN: 1572-8196. DOI: 10.1007/s11276-009-0183-0. URL: https://doi.org/10.1007/s11276-009-0183-0.

[28] E. C. Eze, S. Zhang, and E. Liu. "Vehicular ad hoc networks (VANETs): Current state, challenges, potentials and way forward." In: *2014 20th International Conference on Automation and Computing*. Sept. 2014, pp. 176–181. DOI: 10.1109/IConAC.2014.6935482.

[29] B. Ferreira. “Privacy-preserving efficient searchable encryption.” Doctoral dissertation. Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2016.

[30] T. K. Forde, L. E. Doyle, and D. O’Mahony. “Ad hoc innovation: distributed decision making in ad hoc networks.” In: *IEEE Communications Magazine* 44.4 (2006), pp. 131–137. ISSN: 0163-6804. DOI: 10.1109/MCOM.2006.1632660.

[31] C. Gomez, J. Oller, and J. Paradells. “Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology.” In: *Sensors* 12.9 (2012), pp. 11734–11753.

[32] G. T. C. Gunaratna, P. V.N. M. Jayarathna, S. S. P. Sandamini, and D. S. D. Silva. “Implementing wireless Adhoc networks for disaster relief communication.” In: *2015 8th International Conference on Ubi-Media Computing (UMEDIA)*. Aug. 2015, pp. 66–71. DOI: 10.1109/UMEDIA.2015.7297430.

[33] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Van Renesse. “Kelips: Building an efficient and stable P2P DHT through increased memory and background overhead.” In: *International Workshop on Peer-to-Peer Systems*. Springer. 2003, pp. 160–169.

[34] Z. J. Haas, J. Y. Halpern, and L. Li. “Gossip-based ad hoc routing.” In: *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*. Vol. 3. 2002, 1707–1716 vol.3. DOI: 10.1109/INFCOM.2002.1019424.

[35] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. “Optimized link state routing protocol for ad hoc networks.” In: *Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International*. IEEE. 2001, pp. 62–68.

[36] P. Jesus, C. Baquero, and P. S. Almeida. “A Survey of Distributed Data Aggregation Algorithms.” In: *IEEE Communications Surveys Tutorials* 17.1 (Jan. 2015), pp. 381–404. ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2354398.

[37] P. Jesus, C. Baquero, and P. S. Almeida. “Fault-Tolerant Aggregation by Flow Updating.” In: *Distributed Applications and Interoperable Systems*. Ed. by T. Senivongse and R. Oliveira. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 73–86. ISBN: 978-3-642-02164-0.

[38] D. Johnson, N. Ntlatlapa, and C. Aichele. “Simple pragmatic approach to mesh routing using BATMAN.” In: *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries (WCITD’2008)*. IFIP. 2008.

[39] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. "Energy-efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet." In: *SIGARCH Comput. Archit. News* 30.5 (Oct. 2002), pp. 96–107. ISSN: 0163-5964. DOI: 10.1145/635506.605408. URL: http://doi.acm.org/10.1145/635506.605408.

[40] J. Kangasharju, J. Roberts, and K. W. Ross. "Object replication strategies in content distribution networks." In: *Computer Communications* 25.4 (2002), pp. 376 –383. ISSN: 0140-3664. DOI: https://doi.org/10.1016/S0140-3664(01)00409-1. URL: http://www.sciencedirect.com/science/article/pii/S0140366401004091.

[41] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web." In: *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. STOC '97. El Paso, Texas, USA: ACM, 1997, pp. 654–663. ISBN: 0-89791-888-6. DOI: 10.1145/258533.258660. URL: http://doi.acm.org/10.1145/258533.258660.

[42] B. Karp and H. T. Kung. "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks." In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*. MobiCom '00. Boston, Massachusetts, USA: ACM, 2000, pp. 243–254. ISBN: 1-58113-197-6. DOI: 10.1145/345910.345953. URL: http://doi.acm.org/10.1145/345910.345953.

[43] D. Kempe, A. Dobra, and J. Gehrke. "Gossip-based computation of aggregate information." In: *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.* Oct. 2003, pp. 482–491. DOI: 10.1109/SFCS.2003.1238221.

[44] H.-S. Kim, M.-S. Lee, Y.-J. Choi, J. Ko, and S. Bahk. "Reliable and Energy-Efficient Downward Packet Delivery in Asymmetric Transmission Power-Based Networks." In: *ACM Trans. Sen. Netw.* 12.4 (Sept. 2016), 34:1–34:25. ISSN: 1550-4859. DOI: 10.1145/2983532. URL: http://doi.acm.org/10.1145/2983532.

[45] J. Leitão. "Gossip-Based Broadcast Protocols." Master's thesis. Faculdade de Ciências da Universidade de Lisboa, 2007.

[46] J. Leitão. "Topology Management for Unstructured Overlay Networks." Doctoral dissertation. Technical University of Lisbon, 2012.

[47] J. Leitão, J. P. Marques, J. Pereira, and L. Rodrigues. "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks." In: *IEEE TPDS* 23.11 (Nov. 2012).

[48] J. Leitão, J. Pereira, and L. Rodrigues. "Epidemic broadcast trees." In: *Proc. of SRDS'07*. IEEE. 2007.

[49] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A membership protocol for reliable gossip-based broadcast." In: *Proc. of DSN'07*. IEEE. 2007.

[50] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. "TinyOS: An Operating System for Sensor Networks." In: *Ambient Intelligence*. Ed. by W. Weber, J. M. Rabaey, and E. Aarts. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115–148. ISBN: 978-3-540-27139-0. DOI: 10.1007/3-540-27139-2_7. URL: https://doi.org/10.1007/3-540-27139-2_7.

[51] T. Liu and M. Martonosi. "Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems." In: *SIGPLAN Not.* 38.10 (June 2003), pp. 107–118. ISSN: 0362-1340. DOI: 10.1145/966049.781516. URL: http://doi.acm.org/10.1145/966049.781516.

[52] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. "Implementing Software on Resource-constrained Mobile Sensors: Experiences with Impala and ZebraNet." In: *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*. MobiSys '04. Boston, MA, USA: ACM, 2004, pp. 256–269. ISBN: 1-58113-793-1. DOI: 10.1145/990064.990095. URL: http://doi.acm.org/10.1145/990064.990095.

[53] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TAG: A Tiny AGgregation Service for Ad-hoc Sensor Networks." In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2002), pp. 131–146. ISSN: 0163-5980. DOI: 10.1145/844128.844142. URL: http://doi.acm.org/10.1145/844128.844142.

[54] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. "TinyDB: an acquisitional query processing system for sensor networks." In: *ACM Transactions on database systems (TODS)* 30.1 (2005), pp. 122–173.

[55] R. Mahmud, R. Kotagiri, and R. Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*. Ed. by B. Di Martino, K.-C. Li, L. T. Yang, and A. Esposito. Singapore, 2018. DOI: 10.1007/978-981-10-5861-5_5. URL: https://doi.org/10.1007/978-981-10-5861-5_5.

[56] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh. "Peer Counting and Sampling in Overlay Networks: Random Walk Methods." In: *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*. PODC '06. Denver, Colorado, USA: ACM, 2006, pp. 123–132. ISBN: 1-59593-384-0. DOI: 10.1145/1146381.1146402. URL: http://doi.acm.org/10.1145/1146381.1146402.

[57] H. Miranda, A. Pinto, and L. Rodrigues. "Appia, a flexible protocol kernel supporting multiple coordinated channels." In: *Proceedings 21st International Conference on Distributed Computing Systems*. Apr. 2001, pp. 707–710. DOI: 10.1109/ICDSC.2001.919005.

[58] H. Miranda, S. Leggio, L. Rodrigues, and K. Raatikainen. "A Power-Aware Broadcasting Algorithm." In: *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*. Sept. 2006, pp. 1–5. DOI: 10.1109/PIMRC.2006.254191.

[59] G. Montenegro, C. Schumacher, and N. Kushalnagar. *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals*. RFC 4919. Aug. 2007. DOI: 10.17487/RFC4919. URL: https://rfc-editor.org/rfc/rfc4919.txt.

[60] S. Motegi, K. Yoshihara, and H. Horiuchi. "DAG based in-network aggregation for sensor network monitoring." In: *International Symposium on Applications and the Internet (SAINT'06)*. Jan. 2006, 8 pp.–299. DOI: 10.1109/SAINT.2006.20.

[61] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. "The Broadcast Storm Problem in a Mobile Ad Hoc Network." In: *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*. MobiCom '99. Seattle, Washington, USA: ACM, 1999, pp. 151–162. ISBN: 1-58113-142-9. DOI: 10.1145/313451.313525. URL: http://doi.acm.org/10.1145/313451.313525.

[62] C. E. Perkins and E. M. Royer. "Ad-hoc on-demand distance vector routing." In: *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*. Feb. 1999, pp. 90–100. DOI: 10.1109/MCSA.1999.749281.

[63] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran. "Study on ZigBee technology." In: *2011 3rd International Conference on Electronics Computer Technology*. Vol. 6. Apr. 2011, pp. 297–301. DOI: 10.1109/ICECTECH.2011.5942102.

[64] A. Raniwala and T. cker Chiueh. "Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network." In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.* Vol. 3. Mar. 2005, 2223–2234 vol. 3. DOI: 10.1109/INFCOM.2005.1498497.

[65] G. F. Riley and T. R. Henderson. "The ns-3 Network Simulator." In: *Modeling and Tools for Network Simulation*. Ed. by K. Wehrle, M. Güneş, and J. Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34. ISBN: 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_2. URL: https://doi.org/10.1007/978-3-642-12331-3_2.

[66] M. G. Rubinstein, I. M. Moraes, M. E. M. Campista, L. H.M. K. Costa, and O. C.M. B. Duarte. "A Survey on Wireless Ad Hoc Networks." In: *Mobile and Wireless Communication Networks*. Ed. by G. Pujolle. Boston, MA: Springer US, 2006, pp. 1–33. ISBN: 978-0-387-34736-3.

[67] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge Computing: Vision and Challenges." In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[68] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. “Medians and beyond: new aggregation techniques for sensor networks.” In: *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM. 2004, pp. 239–249.

[69] B. Sidhu, H. Singh, and A. Chhabra. “Emerging wireless standards-wifi, zigbee and wimax.” In: *World Academy of Science, Engineering and Technology* 25.2007 (2007), pp. 308–313.

[70] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. “Sensor Network-based Countersniper System.” In: *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*. SenSys ’04. Baltimore, MD, USA: ACM, 2004, pp. 1–12. ISBN: 1-58113-879-2. DOI: 10.1145/1031495.1031497. URL: http://doi.acm.org/10.1145/1031495.1031497.

[71] M. P. Spertus, S. Kritov, D. M. Kienzle, H. F. Van Rietschote, A. T. Orling, and W. E. Sobel. *Efficient backups using dynamically shared storage pools in peer-to-peer networks*. US Patent 7,529,785. May 2009.

[72] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. “Chord: A scalable peer-to-peer lookup service for internet applications.” In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001).

[73] A. Z. Tomsic, T. Crain, and M. Shapiro. “Scaling Geo-replicated Databases to the MEC Environment.” In: *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*. Sept. 2015, pp. 74–79. DOI: 10.1109/SRDSW.2015.13.

[74] M. Torrent-Moreno, S. Corroy, F. Schmidt-Eisenlohr, and H. Hartenstein. “IEEE 802.11-based One-hop Broadcast Communications: Understanding Transmission Success and Failure Under Different Radio Propagation Environments.” In: *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*. MSWiM ’06. Terromolinos, Spain: ACM, 2006, pp. 68–77. ISBN: 1-59593-477-4. DOI: 10.1145/1164717.1164731. URL: http://doi.acm.org/10.1145/1164717.1164731.

[75] R. Van Renesse, T. M. Hickey, and K. P. Birman. *Design and performance of Horus: A lightweight group communications system*. Tech. rep. Cornell University, 1994.

[76] R. Van Renesse, K. P. Birman, and W. Vogels. “Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining.” In: *ACM transactions on computer systems (TOCS)* 21.2 (2003), pp. 164–206.

[77] P. Verissimo and L. Rodrigues. *Distributed Systems for System Architects*. Norwell, MA, USA: Kluwer Academic Publishers, 2001. ISBN: 0792372662.

[78] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. “Deploying a wireless sensor network on an active volcano.” In: *IEEE Internet Computing* 10.2 (Mar. 2006), pp. 18–25. ISSN: 1089-7801. DOI: 10.1109/MIC.2006.26.

[79] B. Wong and S. Guha. "Quasar: A Probabilistic Publish-subscribe System for Social Networks." In: *Proceedings of the 7th International Conference on Peer-to-peer Systems*. IPTPS'08. Tampa Bay, Florida: USENIX Association, 2008, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=1855641.1855643.

[80] J. Yeo, H. Lee, and S. Kim. "An efficient broadcast scheduling algorithm for TDMA ad-hoc networks." In: *Computers & Operations Research* 29.13 (2002), pp. 1793 –1806. ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(01)00057-0. URL: http://www.sciencedirect.com/science/article/pii/S0305054801000570.

[81] S. Yi, C. Li, and Q. Li. "A Survey of Fog Computing: Concepts, Applications and Issues." In: *Proceedings of the 2015 Workshop on Mobile Big Data*. Mobidata '15. Hangzhou, China: ACM, 2015, pp. 37–42. ISBN: 978-1-4503-3524-9. DOI: 10.1145/2757384.2757397. URL: http://doi.acm.org/10.1145/2757384.2757397.

[82] J. Yick, B. Mukherjee, and D. Ghosal. "Wireless sensor network survey." In: *Computer Networks* 52.12 (2008), pp. 2292 –2330. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2008.04.002. URL: http://www.sciencedirect.com/science/article/pii/S1389128608001254.

[83] E. Ziouva and T. Antonakopoulos. "CSMA/CA performance under high traffic conditions: throughput and delay analysis." In: *Computer Communications* 25.3 (2002), pp. 313 –321. ISSN: 0140-3664. DOI: https://doi.org/10.1016/S0140-3664(01)00369-3. URL: http://www.sciencedirect.com/science/article/pii/S0140366401003693.