

CONNECTIVITY SHARING FOR WIRELESS MESH NETWORKS

Khulan Batbayar

*Thesis submitted in partial fulfillment of the requirements for
the Degree of Doctor in Applied Sciences*

March 2022

Distributed Systems Group (DSG)
Departament d'Arquitectura de Computadors (DAC)
Universitat Politècnica de Catalunya (UPC)
Barcelona
Spain

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis advisors:

Dr. Roc Meseguer **Supervisor**, Universitat Politècnica de Catalunya, Spain
Pr. Ramin Sadre **Co-supervisor**, UCLouvain/ICTEAM, Belgium

Connectivity Sharing for Wireless Mesh Networks

by Khulan Batbayar

© Khulan Batbayar 2021
Distributed Systems Group
Universitat Politecnica de Catalunya
Jordi Girona, 1-6
08033, Barcelona
Spain

Acknowledgments

This thesis has been a challenging journey that has been possible, thanks to the guidance and support of many people I am very grateful for.

First and foremost, I would like to thank my advisors, Roc Meseguer and Ramin Sadre, for guiding me through my doctoral journey and being patient with me. I want to sincerely thank Roc and Ramin and give them credit for their time, effort, and encouragement to my research, for embracing me as their student. My thesis has benefitted substantially from their insightful recommendations.

Thank you, Roc, for the professional guidance and consistent encouragement from the first day we met. I am grateful for his helpful comments, suggestions, and constructive criticism throughout this entire research. Above all, his positive attitude towards everything, saying, "It is no problem, It is easy!" always kept me going (and Ph.D. Comics pictures). For my second advisor, Ramin, thank you for the patience and confidence you placed in me and for letting me pursue my ideas freely. I am grateful for the countless hours dedicated to my research, last-minute submission reviews, many meetings during my internship, most importantly, for my academic writing.

I am grateful to my colleagues at Distributed Systems Group at UPC, Menan Selimi, Emmanouil Dimogerontakis, Joao Neto, Nuno Apolonio, Jawad Manzoor, Roger Pueyo, for all the ping pong games, late Spanish university lunches at Unity, coffee at the FIB bar. Many thanks go to the friends I made during the journey, Sana Imtiaz, Zainab Abbas, Dur Zahra, Atika Zulfiqar, Rosana Veroneze, Juhee Bae, Igor Zavalysyn, Paolo Laffranchini, Olivier Goletti, Francois Aubry, Michael Saint-Guillain, John Aoga, Mathieu Jadin, Gorby Kabasele Ndonga, Nicolas Laurent.

I thank the Mongolian community in Barcelona for taking care of Ivel and me while adjusting to a new life in Spain. Especially, Battulga, Suvd-Erdene Byambajav, Erdenezaya Myagmar, Nomin Batzolboo for their emotional support, Saturday basketball games, overnight daaluu, Mongolian food.

I cannot begin to express my gratitude to my family for all of the love, support, encouragement, and prayers they have sent my way along this journey. You are the first people I share all of my news with, the good and the bad. To my parents, thank you for supporting me from the beginning; your unconditional love means the world to me. My father, **Batbayar Tumen**, thank you for always setting higher goals for me and doing everything in

your power to help me achieve them. Without your vision and belief, I would not imagine myself going to study abroad, pursuing a Ph.D. degree in the first place. My mother, **Gereltuya Nergui**, thank you for her unconditional love and support. I genuinely believe that your daily prayers helped me during the journey. My sister, **Javkhlan**, thank you for always coming to the rescue, for believing in me, being on my side through thick and thin. We struggle together, but we grow together during this journey. Your support has meant more to me than you could possibly realize.

My son, **Iveel**, you are my motivation, and source of energy. Being a child of PhD student has not been easy for you, studying 3 different schools in 3 different languages from the age of 4. While I am struggling with my PhD, you grow up as an excellent, intelligent boy everyone adores. I apologize for all the times we spent together on UPC Lab C6-E208, for all the times I could not spend with you. I thank you for being healthy, growing up beyond your years, being obedient, flexible, and understanding.

My boyfriend, **Raziel Carvajal**, my life is changed since the day I met you, in a good way. You brought so much joy, energy, and I laugh every day because of you. Thank you for being a supportive and loving partner.

Finally, I am grateful for Erasmus Mundus Joint Doctorate Program for giving me an opportunity to conduct research in two prestigious European universities. This experience has afforded me the opportunity to work with some of the best and brightest, and the resources for me to achieve great success. Thank you for this opportunity.

Abstract

Internet access is still unavailable to one-third of the world population due to the lack of infrastructure, high cost, and the digital divide. Many access-limited communities opt for shared Internet access where they build common network infrastructures to mitigate the cost. Internet connectivity in such infrastructures is typically provided by several limited, sometimes non-dedicated, gateways. Client nodes, i.e., the end-user hosts, use one gateway and switch to another when the first fails. In this scheme, the gateway configuration is done manually on the end-user side. This form of Internet connectivity is widespread and has the advantage that no central control is required, but it is also unreliable and inefficient due to several factors, such as unbalanced traffic load across the gateways. We believe that the network would benefit from a gateway selection mechanism that can provide good connectivity to the client nodes as well as balanced load distribution and a dynamic adaptation to the current network state. However, providing such a dynamic gateway selection is complicated: since the performance of the gateways as perceived by the client nodes changes frequently and might depend on the location of the latter in the network, and optimal selection would require the continuous monitoring of the gateway performance by the client node. The cost of such network-wide performance monitoring is high in large-scale networks and would outweigh the benefits of the dynamic gateway selection.

The thesis' goal is to design a low-cost, distributed mechanism that provides an efficient and dynamic gateway selection while considering the overall balanced gateway selection distribution. To this end, we have split the problem of gateway selection into different sub-problems. First, we focus on reducing the cost of gateway performance monitoring. We propose an approach to reduce the number of monitoring requests generated by each node and analyze its effect on the gateway selection. Then, we present a collaborative monitoring method that allows neighbor nodes to share the load of the gateway monitoring. We show that every node can carry out the necessary tasks, that is performance monitoring, collaboration with its neighbors, and fault tolerance measures, with little computation and communication overhead. Second, to improve the gateway selection, we focus on making a selection decision that fulfills the individual performance requirements of the client nodes as well as global load balancing requirements. The solutions developed by us for the different sub-problems are embedded into a general and

extensible, layered framework for gateway selection that we have called the *Sense-Share-Select* framework.

Experimental validation and comparison with existing methods show that our framework provides accurate collaborative performance monitoring, improves the Quality of Experience (QoE) for the nodes, and distributes the client nodes over the gateways in a balanced manner. The simplicity and flexibility of the framework make it adaptable to other network domains such as Internet of Things (IoT) networks and other scenarios where resource monitoring and load balancing are required.

Contents

Acknowledgments	i
Abstract	iii
Table of Contents	v
1 Introduction	1
1.1 Problem statement	2
1.2 Main objectives	6
1.3 Contributions	6
1.4 Publications	8
1.5 Thesis structure	9
2 Background and related work	11
2.1 Background	11
2.2 Discussion of related work	15
3 The Sense-Share-Select framework	21
3.1 Introduction	21
3.2 Objectives	22
3.3 Overview of the framework	24
4 Sensing layer	27
4.1 Introduction	27
4.2 Design requirements	27
4.3 Design overview	28
4.4 Experimental algorithm and results	31
4.5 Conclusion	35
5 Collaborative layer	37
5.1 Introduction	37
5.2 Design requirements	38
5.3 Design overview	39
5.4 Gateway selection	46
5.5 Evaluation	47
5.6 Conclusion	58

6	Selection layer	61
6.1	Introduction	61
6.2	Design requirements	62
6.3	Design overview	64
6.4	Evaluation	68
6.5	Conclusion	76
7	Production network integration	77
7.1	Community network	77
7.2	Implementation of Sense-Share-Select in guifi.net environment	79
7.3	Conclusion	93
8	Conclusion	95
8.1	Main conclusions	95
8.2	Future work	96
8.3	Discussion	96
8.4	Financial support	97

List of Figures

1.1	Internet gateway in the network	2
1.2	Hourly requests passed through gateways	3
1.3	Guifi.net gateway latency variance	4
1.4	Herd behavior in selection example	5
2.1	Example: Cluster-based collaborative monitoring	12
2.2	CoPing example	17
2.3	Vivaldi based algorithm example	18
3.1	The layered architecture of Sense-Share-Select framework	22
3.2	Example of Sense-Share-Select collaboration	24
3.3	Sense-Share-Select framework structure	25
4.1	Experiment with different gateway performance measurements	29
4.2	Total gateway measurements	34
4.3	Client perception of the selected gateway's performance.	35
5.1	Example scenario for forming a group of collaborating nodes	38
5.2	Layered structure of the collaborative algorithm	40
5.3	Periodic sensing phase	41
5.4	Performance table obtained through collaborative sensing	43
5.5	Layered sensing framework	44
5.6	Network topology for experiments	50
5.7	Similarity results for different collaboration schemes	50
5.8	Gateway performance change (Collaborating with all)	52
5.9	Gateway performance change (Collaborating with <i>TopK</i>)	53
5.10	Collaborating node change	53
5.11	Faulty nodes effect (Collaborating with all)	54
5.12	Faulty nodes effect (Collaborating with <i>TopK</i>)	55
5.13	Behavior of the faulty nodes	55
5.14	Sensitivity towards the number of gateway nodes	56
5.15	Results of nodes leaving/joining	57
5.16	Gateway selection result	58
6.1	Selection algorithm design	65

6.2	Estimated gateway capacity classification	66
6.3	Gateway latency variance	67
6.4	Experiment network topology.	69
6.5	Wireless - Performance prediction result.	70
6.6	Client node distribution over the gateway nodes.	71
6.7	Wireless gateway distribution	72
6.8	Number of gateway selection changes in 100 rounds.	73
6.9	Download time through the gateway nodes.	73
6.10	Wireless - Client node distribution with gateway delay.	74
6.11	Download times with different gateway delays	75
6.12	Wireless - Download size sensitivity.	75
6.13	Wireless - Gateway distribution.	76
7.1	Number of new nodes (Data from www.guifi.net)	78
7.2	Experimental nodes topology	81
7.3	Collaborative sensing similarity	81
7.4	Vivaldi accuracy	82
7.5	Collaborative message exchange	83
7.6	Download time	84
7.7	Gateway distribution per round	85
7.8	Number of gateway changes	85
7.9	Sensitivity analysis of RTT threshold value and size of close neighbors cluster	87
7.10	Download time	88
7.11	Node unavailability test	90
7.12	In the presence of faulty nodes	90
7.13	Wireless experiment distribution	91
7.14	Full collaboration vs full isolation	93
7.15	Experiment with different bandwidth	94

List of Tables

3.1	Example of gateway performance table at each node	25
4.1	Gateway performance categories	30
4.2	Message overhead per selection round.	33
5.1	Messagess cost & gateway visibility at individual nodes per measurement round, g : number of gateways, n : number of close neighbors, r : number of repetitions of the message ex- change in one measurement round.	48
5.2	Average message exchange at individual node per measure- ment round	51
5.3	Sensitivity towards the RTT threshold vs increasing the num- ber of samples	57
6.1	Partial gateway performance table, every round,	64
6.2	Gateway performance categories	65
6.3	Example gateway performance table after Step 1	66

Introduction

1

Internet connectivity is an essential part of our daily life; teleworking, online learning, accessing public services, purchasing everyday essentials, ordering food, and so forth. However, many studies [ITUStat; Aff20] show that the worldwide Internet penetration rate is only 61%, and that there is a clear gap in the coverage between urban areas (72%) and rural areas (51%). The Internet Affordability Drivers Index (ADI) has increased in many countries, according to the Alliance for Affordable Internet (A4AI) [Aff20] report. ADI scores are calculated based on the internet infrastructure, broadband adoption, and equitable access policies [Aff20]. The price of Internet subscription is still high relative to the income in many developing countries.

There are many initiatives to connect previously unconnected communities to the Internet, such as Google's Project Loon [Loon], Facebook Connectivity initiatives [FBCon] through fiber optics, Internet hotspots, or mobile broadband service. However, most of them are technologically difficult to attain, financially infeasible, and not self-sustainable. All these initiatives are still not able to satisfy the global demand for affordable Internet access. As a cheaper alternative, establishing wireless networks in the areas lacking connectivity has been the most popular Internet connectivity approach. In the last years, Wireless Mesh Network (WMN) infrastructures have been widely adopted as a cost-effective technology to provide large-scale Internet access networks.

In WMN (see Figure 1.1), the mesh nodes are not all directly connected to the Internet, but specific nodes act as Internet gateways to provide Internet access to the others. Good examples of such networks are Wireless Community Network (WCN) or Do-It-Yourself access networks which are gaining more popularity for self-provision of Internet access in underserved regions [Rey+13]. WCN allows community members to build their own local network infrastructure by utilizing off-the-shelf equipment with minimal technical expertise. It provides various free, neutral, secure communication services, including shared Internet access. There are many successful Community Network (CN)s using Internet gateways such as the guifi.net community network [guifi] (>36000 nodes, 450 Internet proxies/gateways), Roofnet [Bic+05] (36 mesh nodes), or Freifunk community network [HDS17]. Such large-scale networks are built from heterogeneous devices using various

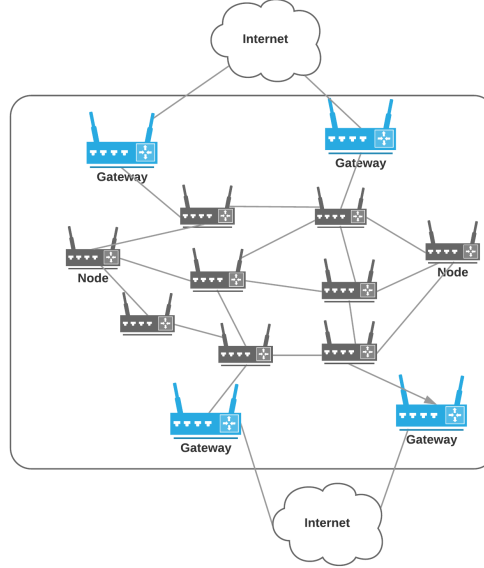


Figure 1.1: Internet gateway in the network

operating systems and protocols. The Internet gateways act as a translator between the local networks and the Internet.

All traffic from the client nodes to the Internet and back passes through the Internet gateway nodes. Although there are multiple gateways to choose from in large-scale deployments, their number usually is much smaller than the number of client nodes. Choosing the “right” gateway can improve the Quality of Service perceived at the client node. However, doing so can be challenging in practice: Gateways might have different capacities to handle client traffic. Furthermore, the performance of a gateway will depend on the number of clients that send their traffic through that gateway. Measurements have shown that the performance of a gateway will vary dynamically depending on the number of active clients at a time, rush hours, etc. [DMN17; Bat+19a]. As the network grows, the traffic load at the gateways increases and they become susceptible to congestion and failure. As a result, the Internet connectivity perceived by the client nodes can become excruciatingly slow.

1.1 Problem statement

Selecting the best Internet gateway is key to provide good Internet connectivity for the end-user. State-of-the-art algorithms can provide the optimal

selection for an individual client, but there is a gap in the state of the art concerning the challenges of low cost and fair distribution. In the following, we explain the problems we are addressing in this thesis.

1.1.1 Static gateway selection

A common practice in wireless networks is to more or less statically configure the gateway, which is often identical to one of default gateways given by the ISP. For example, in guifi.net, the nodes choose the gateway depending on the locality (zone), popularity, previous experience, and word-of-mouth knowledge. A gateway distribution imbalance is inevitable since the gateways are configured on the client side without prior knowledge of their current or future performance. Often, gateways have limited capacity to handle all client requests and the performance is likely to degrade.

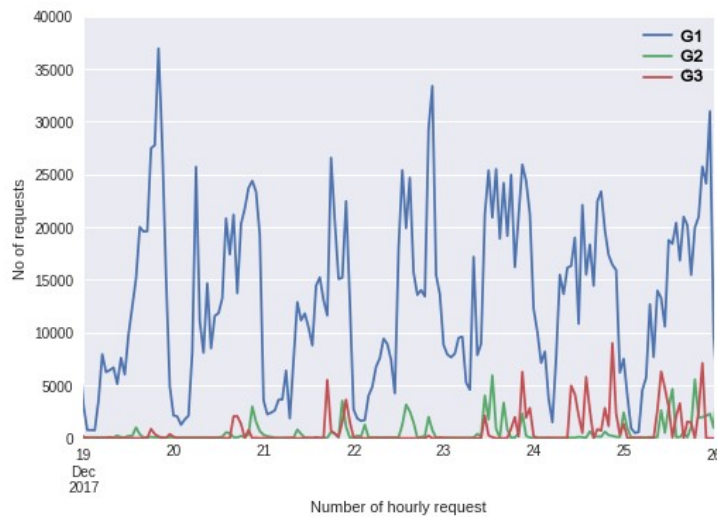


Figure 1.2: Hourly requests passed through gateways

Figure 1.2 shows the total number of hourly HTTP requests that pass through three different gateways of a zone in the guifi.net CN during one month. The log has been collected anonymously at the gateway nodes¹. We can see that the blue gateway receives most of the requests, while the green and the red gateways remain barely used. Obviously, there is an imbalance between the available gateways in terms of selection. The unpopular gateways should be selected by the client nodes, too, to decrease the load on the popular gateways.

¹<http://logs.confine-project.eu/proxy/>

1.1.2 Gateway performance fluctuations

Gateway performance changes frequently depending on the nature of the traffic, the number of users, and the time of the day. In Figure 1.3, we have measured the latency at five different guifi.net gateways to download a 0.1MB file from a dedicated server on the Internet every 2 minutes. We observe that the performance of the green and red gateways is inconsistent. The orange gateway is a user or limited capacity gateway that becomes unresponsive when the number of requests increases. The best gateway selection candidate changes several times during the observation period.

To provide good connectivity to all client nodes under such conditions, periodic performance monitoring of all gateways is essential. On the other hand, the monitoring (and the distribution of the monitoring results) has to be done carefully in order to not add additional load to the network.

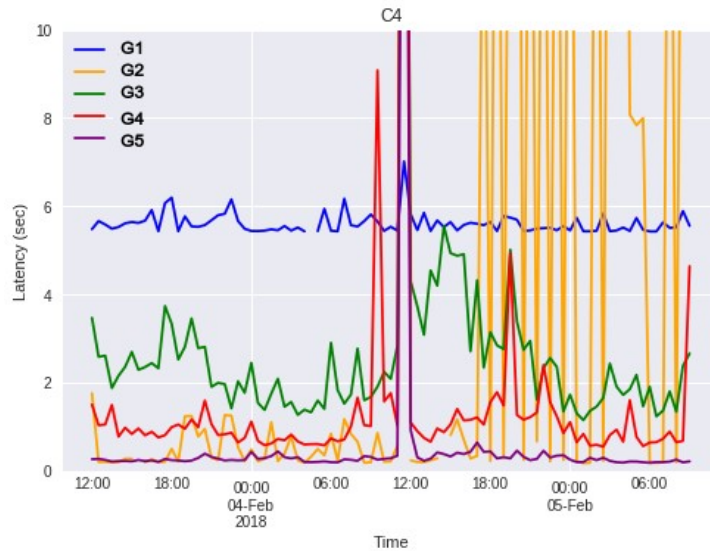


Figure 1.3: Guifi.net gateway latency variance

1.1.3 Unfair distribution of gateway usage

Simple gateway selection algorithms assign client nodes to the currently best-performing gateway. This approach leads to a situation called *herd behavior* where most nodes select the same gateway, quickly degrade its performance, and then move to the next best gateway and so on. Therefore, state-of-the-art algorithms propose evenly distributing the client nodes over the available gateways. However, even such load-balancing gateway selection algorithms

have a preference for the least-loaded gateways, which circles back to the previous problem.

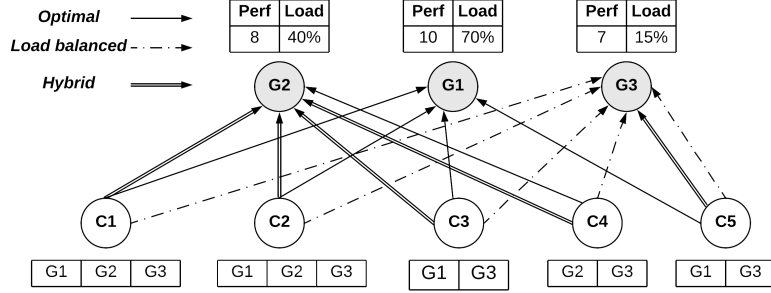


Figure 1.4: Herd behavior in selection example

Consider the selection scenario shown in Figure 1.4. Let us assume three gateways of the network, $G1, G2, G3$, are given with their current performance (here expressed as latency) and load information. There are four client nodes $C1, C2, C3, C4$ in the network. Note that, as each node is in a different part of the network, it will observe a different gateway performance. Furthermore, not every client node will necessarily have performance information on *all* gateway nodes. For example, $C1$ node has measurements of $G1, G2, G3$ gateways. Three different optimal selection examples are shown based on the local measurements of the nodes. In the (locally) optimal strategy, each node selects the best performing gateway; in this case, the majority of the nodes select $G1$. With a load-balancing strategy, the nodes try to select the least loaded gateway. As a result, all clients select $G3$ even though its performance is the lowest. Let us also consider a hybrid selection strategy that gives equal weight to both metrics (Equation 1.1). Similarly, the majority of the client nodes select $G3$. Since each node is unaware of the other nodes' gateway selection at their time of choosing, they end up with the same choice.

$$perf_gateway = 0.5 * load_metric + 0.5 * performance_metric \quad (1.1)$$

The above behavior is the consequence of the fact that in the type of networks we are interested in, such as the community networks, there is no central or distributed network-wide coordination of all nodes' behavior. As a client node has no control over the selection behaviour of the other nodes, a greedy strategy might look promising, but will lead to the above effects. The globally optimal Internet gateway selection solution would require a network-wide gateway load monitoring and distribution algorithm, which is not suitable for a dynamic large-scale network due to the high cost of monitoring and coordination.

1.2 Main objectives

The main objective of this thesis is to propose a general-purpose Internet gateway selection framework for large-scale, heterogeneous networks. It aims to provide best-effort Internet connectivity to all the network nodes in a shared Internet access network, meanwhile achieving an overall fair (i.e., balanced) distribution of the nodes over the gateways. It should be easily implementable in the client node without disrupting network performance and require minimal technical knowledge by the users.

The framework should have a small network footprint, meaning it should reduce the cost of performance monitoring and communication. Last but not least, it should provide accurate measurements, which is a prerequisite for the nodes to perform informed gateway selections.

1.3 Contributions

The main contributions of our thesis can be summarized as follows:

- C1 Client-side, lightweight gateway selection framework.** We facilitate each node involved in their gateway selection process to provide customized gateway selection. The nodes are responsible for gateway monitoring, collaboration, and selection. We aim to use simple calculations that can be done by a wide range of devices in the network, such as devices with limited capacity, battery, bandwidth, etc. The collaborative monitoring algorithm reduces inter-network communication and reuses gateway measurements for the local selection decision. Our framework is scalable and suitable for heterogeneous network environments.

In Chapter 3, we formulate the Sense-Share-Select framework to achieve this contribution. Detailed descriptions of the framework are given in the following chapters.

- C2 Accurate, low-cost performance monitoring algorithm** A middle layer of our framework uses a collaborative performance monitoring algorithm. Our proposed collaborative performance monitoring algorithm collaborates with local, closer neighbors (in terms of RTT) selected by each node. This reduces the inter-node communication while allowing nodes to benefit from data sharing.

In a collaborative monitoring algorithm, there is a chance that other nodes might send malicious/wrong measurements to lure nodes to other gateways. In our monitoring algorithm, nodes use their collaboration

history with other nodes to judge the behavior of neighbors. We demonstrate that close neighbor collaborative monitoring provides >90% accurate performance measurements on average for each node.

The main results related to this contribution are presented in Chapter 5 and were originally reported in [Bat+19a; Bat+19b].

- C3 Balance between better performance and fair distribution.** Manual Internet gateway selection is popular in decentralized, bottom-up network initiatives because it requires minimal network management. In the Community Network setting, the gateway nodes are manually selected at each node side based on their popularity, previous experience with the gateway, or locality. However, as explained, this can lead to over-popularization of single gateways. In a previous study [Dim+17b], researchers analyzed the log files of the guifi.net gateways and found that clients do not select certain gateways because they are not well known or a bit further away from the user. Our selection algorithm gives gateways that are performing well equal opportunity to be selected. Results show that our selection algorithm gives a better Internet latency compared to selecting the best performing gateway at a time.

- C4 Maintaining a long-term, stable selection.** With manual gateway selection, a client does not change its selected gateway unless the gateway becomes unresponsive. In this happens, the node moves to the next gateway in its list of known gateways. State-of-the-art gateway selection algorithms do not take it into consideration future performance. Our framework categorizes gateways based on their performance capacity and avoids selecting gateways showing varying performance. This feature allows nodes to have stable, good Internet connectivity and avoid changing the gateways frequently.

The main results related to C3 and C4 are presented in Chapter 6 and were originally reported in [Bat+20]

- C5 Production network implementation and evaluation.** We implement our gateway selection framework and evaluate its performance in the guifi.net CN environment with testbed network nodes and production network gateways with real background traffic. We design and run different scenarios in the testbed to study our framework's adaptability and resilience. The experiment results are consistent with results from emulations with Mininet.

The main results related to this contribution are presented in Chapter 7, and are under IEEE Access journal submission.

1.4 Publications

In this section, we summarize our list of publications.

Accepted

- P1 **"The RIMO gateway selection approach for mesh networks: Towards a global Internet access for all"**[Bat+18], The 12th International Conference on Ubiquitous Computing and Ambient Intelligence, 2018, first author. This paper covers the client-side, non-collaborative gateway selection based on randomized performance sensing. The algorithm focuses on the drawbacks of the manual gateway selection in guifi.net and offers an algorithm, RIMO, a periodic randomized sampling of the gateway performance to select the best gateway. The RIMO algorithm defines the base layer of the Sense-Share-Select gateway selection framework.
- P2 **"Collaborative informed gateway selection in large-scale and heterogeneous networks"**[Bat+19a], IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, first author. The first prototype of the low-cost collaborative performance monitoring algorithm. The algorithm builds on top of RIMO [Bat+18]'s performance sensing and adds close neighbor collaborative sensing. The experiments are performed in the community-lab ² guifi.net's testbed with production network gateways. (CORE2018 RANK A)
- P3 **"Sense-Share: A Framework for Resilient Collaborative Service Performance Monitoring"** [Bat+19b], 15th International Conference on Network and Service Management (CNSM), 2019, first author. Improved collaborative service performance monitoring algorithm that includes fault tolerance and implementation of performance-based sensing. The experiments are performed using the Mininet emulation environment. (CORE2019 RANK B)
- P4 **"GateSelect: A novel Internet gateway selection algorithm for client nodes"** [Bat+20], 16th International Conference on Network and Service Management (CNSM) 2020, first author. Final informed gateway selection algorithm to provide long-term, balanced gateway selection for the client nodes. The algorithm is tested in emulated wired, wireless, and wireless mobile network environments to evaluate the compatibility with different types of networks. (CORE2020 RANK B)

²<https://community-lab.net/>

Pending review

- P5 **"Sense-Share-Select: Sharing the access to the Internet within Community Network"**, Under submission, first author. Integration of the different algorithms into a single Internet gateway selection framework. The experiments are conducted using experimental nodes in the guifi.net and production network gateways.

Workshops

- W1 **Connectivity sharing for wireless mesh networks**, The 6th Networking Networking Women Workshop (N2Women), 2017.
- W2 **Sense-Share: Resilient collaborative service performance monitoring framework**, 8th Networking Networking Women Workshop (N2Women), 2019.

1.5 Thesis structure

The rest of the thesis is structured as follows:

Chapter 3 presents the design requirements of the gateway selection algorithm and introduces the general structure of our proposed framework.

Chapter 4 presents the *Sensing layer* algorithm, focusing on the individual nodes' gateway performance sensing and lowering the number of gateway performance monitoring requests. The proposed algorithm is compared with distributed performance sensing, a brute-force performance sensing algorithm. The effectiveness of the algorithm is shown in the Experimental result section.

In the following chapters, we further propose a more efficient way of reducing the monitoring requests with the help of the collaborative performance monitoring algorithm.

Chapter 5 proposes the *Collaborative layer* algorithm, which incorporates a simple idea of collaborating with their close-range neighbors to improve the availability of the up-to-date gateway performance measurements at each node. In other words, each node collects its performance measurement with the help of the others. However, the neighbor nodes could send wrong measurements to lure the nodes to other gateways. Therefore, the *Collaborative layer* algorithm includes a trust-based ranking mechanism where nodes continuously filter faulty neighbors.

Chapter 6 covers the *Selection layer* algorithm, GateSelect, where the final gateway selection is made. The algorithm categorizes the gateway nodes and creates a gateway selection candidate list by removing undesired candidates.

GateSelect offers better Internet latency as compared to the greedy selection and manual selection algorithms.

Chapter 7 covers evaluating the Sense-Share-Select framework in the guifi.net environment with production gateway nodes. The framework is tested under different network conditions and real network loads.

In Chapter 8, the main contributions of this thesis are summarized, and potential directions for future research are suggested.

Background and related work

2

2.1 Background

The usage of shared wireless access networks is not limited to the use case of community networks described in Chapter 1. With the advent of the IoT, whole new types of heterogeneous devices are introduced to the Internet ecosystem, including smart home devices, sensor networks, and wearable devices. The IoTAnalytics research [Lue+18] published in 2018 estimated that the number of active Internet-connected devices, including IoT devices, will reach over 20 billion in 2020. Each connected device tries to send or receive information through the Internet. Providing network connectivity to them becomes a challenging task. Especially, the presence of mobile devices that might join or leave the network at any time makes network planning difficult. To cope with this, connectivity through large-scale wireless networks has been proposed. Such networks typically contain a larger number of client nodes and one or more gateway nodes [DMN17]. All Internet traffic, including large content (audio, video), passes through the gateways.

Since Internet gateway nodes are the main bridge of connectivity to all connected network devices, they are susceptible to congestion, failure, and overcrowding. In large networks, these problems are further aggravated by the fact that the gateways might be heterogeneous with different capacities to handle client traffic. Furthermore, the performance of a gateway will depend on the number of clients that send their traffic through that gateway.

2.1.1 Gateway performance monitoring

Gateway performance monitoring is the continuous analysis of the Internet gateways to detect and avoid any performance-related issues to provide "best-effort" QoS to the end-users. The process involves collecting statistics to determine the expected Internet connection quality from the gateways.

The main goal of a gateway selection algorithm is to provide the best gateway for each node in the network. For that, every node needs to know the latest performance of all gateways to keep up with the dynamics of the Internet gateway performance. The large-scale network gateways are susceptible to frequent performance fluctuations depending on the amount of

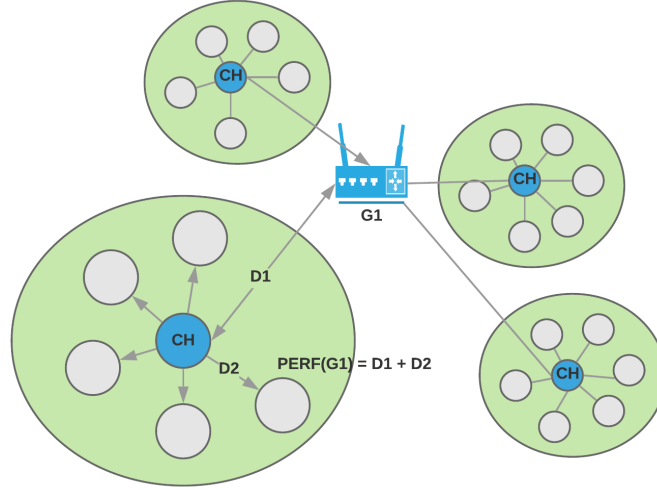


Figure 2.1: Example: Cluster-based collaborative monitoring

traffic that passes through them, rush hours, and the location of the gateway. As we mentioned in the previous chapter, that continuous monitoring mechanism is unsuitable for a large-scale network since the cost of measurement can outweigh the benefits of gateway selection. Even though the overhead of a single measurement request is small, in a resource-constrained, large-scale network, the overall traffic generated by measurement requests from every node becomes a significant contributor to network congestion.

Many algorithms propose collaborative performance monitoring to reduce the cost of measurement. In such algorithms, nodes share measurement results. One of the difficulties in such an approach is that the performance a gateway can offer to a node can depend on the position of the node in the network. Therefore, two nodes at different locations in the network will not necessarily experience the same performance for the same gateway, sometimes for the simple reason that one node is further away from that gateway than another.

There are two different types of collaborative monitoring algorithms: centralized and distributed collaborative monitoring.

Centralized collaborative monitoring algorithms [HCB00; HCB02; YF04; Lia+16; SRS12; FM02; ZC08] often involve a clustering mechanism where a group of nodes selects a representative node (cluster head) to measure the performance of the gateways on behalf of the other group members. Each node adjusts the so obtained measurement based on the relationship between the cluster head, as shown in Figure 2.1. Centralized algorithms reduce the mon-

itoring cost, but the accuracy of the measurements is often not high because the performance of the gateway as measured by the cluster head is not representative for what the other nodes will experience when using that gateway. Consequently, there is a chance the end gateway selection is not the best one. Additionally, every node in the same cluster might end up selecting the same gateway.

Distributed collaborative monitoring algorithms [Abu+15; Dim+17b; Ko+13] require multiple management operations to achieve accurate performance monitoring, such as finding collaborative nodes, scheduling the gateway monitoring (who measures which gateway) among the collaborating nodes, sharing the measurement results between the nodes, and finally adjusting the measurements sent by others. This results in an increased in-network traffic between the network nodes, which increases network congestion. Additionally, not all the nodes in the network are capable of carrying out complex tasks as they have different capacities and resource constraints.

2.1.2 Gateway selection

The selection of the Internet gateway is directly related to the quality of the Internet connectivity provided to the end-user. As the load on the gateway node increases, the Internet connectivity perceived at the client nodes using that gateway can become excruciatingly slow. Measurements have shown that the performance of a gateway will vary dynamically depending on the number of active clients at a time, rush hours, etc. [DMN17; Bat+19a].

There are two main ways for Internet gateway selection: static gateway selection and dynamic gateway selection. The static Internet gateway selection is used in networks such as wireless mesh network, home access network, and community networks, which contain mostly static nodes. The gateway is selected based on the locality of the nodes, the number of hops to the gateway, the previous selection experience, the popularity of the nodes, or word-of-mouth knowledge. The advantage of static gateway selection is zero management, plus no need for performance monitoring, load balancing, and resource allocation. The disadvantages are over-popularization, unbalanced load distribution, and uneven QoS at the client nodes.

Dynamic algorithms focus on changing the gateway selection based on the network dynamics, for example depending on gateway performance, load balance, node mobility, etc. Dynamic gateway selection algorithms are divided further into centralized gateway selection and distributed gateway selection. In **centralized gateway selection algorithms** a central entity recommends gateways to the nodes. The algorithm proposed in [GRS08] uses centralized monitoring, which monitors all gateway traffics and provides load balancing on the flow level by redirecting the nodes. The algorithm proposed

in [HMW09] used Mixed Integer Linear Programming to calculate the minimum node utilization to avoid selecting the overloaded or likely to be congested gateways. The advantage of the centralized algorithm is the low-cost monitoring and dedicated gateway performance monitoring available for all nodes. The disadvantages of the centralized gateway selection algorithms include a single point of failure and no tailored gateway selection for each node. **Distributed gateway selection** algorithms involve network nodes for the gateway performance monitoring, which provides a more tailored gateway selection. Distributed algorithms use clustering algorithms and collaborative performance monitoring algorithms to divide the monitoring tasks among the nodes. Their drawback is that they need more cooperation/coordination work between nodes and cause more communication cost, which is not suitable for scaling.

2.1.3 Gateway discovery

The network nodes have a gateway discovery mechanism where they find a suitable gateway to connect within their coverage area. Gateway discovery is essential in Mobile Ad-hoc Network where mobile nodes require to find an Internet gateway node to connect to the Internet.

Proactive gateway discovery [TSC03; Jon+00; SBP02] follows an approach where the gateway itself periodically announces its performance through Gateway Advertisement (GWADV) messages flooding through the network. The nodes receive the messages and compare them with their previous GWADV message to update the detected gateway list or dismiss it. This way is expensive and generates periodic mass gossiping in which some nodes might not be interested. *Reactive gateway discovery* [BMJ99] is the reverse procedure of the proactive discovery. Here, the client node initiates the discovery process. The node generates a Gateway Solicitation (GWSOL) message, and gateways that hear this message reply with a direct, unicast message. This is an inexpensive approach, but there is a waiting period where the node has to wait for any gateway to reply. There are many *hybrid gateway discovery* algorithms [RK03; Lee+03; Bin+05] proposed in the literature that combine both proactive and reactive gateway discovery. There is also distributed gateway discovery [Jav+08; RG04; AR08] where nodes extract gateway information from the IP header and save the gateway information for the other intermediary nodes to pass on.

For the networks that we target, i.e., community networks and similar networks, we assume that the only "mobility" is that a gateway is becoming online or offline. In other words, the gateways are planned, and the nodes are aware of the available gateways. For example, guifi.net uses a service node registry for advertising all available Internet gateways and the nodes choose

their gateway based on their network zone, locality, etc. Therefore, in this thesis, we do not cover the gateway discovery component and assume that its information is available to the other framework components.

2.2 Discussion of related work

2.2.1 Gateway performance sensing

To our understanding, most of the published works have some important limitations. Proposals like [BH11; AJB10] fail in heterogeneous environments since they are based on solutions that operate on the mesh routing layer and require modifications to the infrastructure routers. A similar work presented in [AAJ09a] also operates on the mesh routing layer and requires additional software on the gateways' side. The proposal in [Sal+18] fails on the low monitoring overhead requirement since it uses a kind of brute-force approach periodically probing all the available gateways.

There are many different metrics used to make an efficient gateway selection. In [Kim+07], gateway selection is based on the load of the gateway while the authors of [AAJ09a] use a combination of gateway load, route interference, and expected link quality metrics to estimate the gateway performance. Recent work in [Wan+17] presents a flexible gateway selection based on not only network-based criteria (e.g., economical features or user preferences) for gateway selection. It is a conceptual proposal based on game theory with a thorough information processing at the client node, therefore difficult to implement and deploy on a large-scale network. The PAWS [Abu+15] algorithm redirects node requests over gateways based on the flow demand and residual capacity of the gateway. In [HXA08], the authors propose two graph-based gateway selection algorithms, a degree-based greedy dominating tree set where the gateway is selected based on the connectivity degree and a weight-based greedy dominating tree set that assigns weights to the gateways based on their coverage and number of hops.

2.2.2 Gateway performance monitoring

There are many ways to monitor the performance of the gateways. However, not all of them are suitable for large-scale networks, especially brute-force performance sensing algorithms [Sal+18; Aou+06] where all nodes measure the performances individually. Other algorithms [Ko+13; Abu+15; AAJ09b] use representative nodes to measure the performance, and other nodes request the measurement from those nodes, but the collaborative measurements are not accurate.

Collaborative sensing algorithms aim to reduce the cost of monitoring and assist nodes in making an informed selection decision. The most com-

mon collaborative algorithms are cluster-based collaboration, where nodes are grouped into different sections of nodes. Similarity-based clustering algorithms were first proposed in [JP73] and improved over time in [MK04; HCB00; YF04; XV07; OV17; LZ10]. The selected cluster head node performs the performance monitoring and is responsible for the performance measurement distribution over the cluster nodes. The clustering algorithms impose additional processing overhead such as frequent cluster head election, recalculation of the cluster members, adjustment of the measurements, and are often not scalable.

Decentralized collaborative sensing algorithms are suitable for large-scale, heterogeneous networks such as guifi.net. The main idea is that every node in the network is responsible for the performance monitoring, and they find a common feature to distribute the performance measurement task and the obtained results among each other.

Collaborative algorithms are susceptible to faulty collaborator nodes [Lin+17; ZSL13]. In [Lin+17; ZSL13], the nodes create a reputation mechanism to rank the collaborators to improve the quality of the collaboration. Other algorithms focus on anomaly detection [Zha+18; Sun+13; Lyu+16] to stop collaborating with the faulty nodes. We address both of the concerns in our Sense-Share-Select framework to encourage the nodes to select their trusted neighbors while staying away from the other nodes.

2.2.3 Gateway selection

We already mentioned [Wan+17] where the authors present a flexible gateway selection based on more than just network-based criteria (e.g., economic features or user preferences) for gateway selection. The gateway selection is based on *Greedy selection* where the node selects the best performing gateway without considering the other nodes' selection [Dim+17b; Ko+13; Liv+10].

In the race of choosing the best Internet gateway, algorithms often eliminate the other equally good performing gateways out of the selection. Doing so creates the situation called *herd behavior* where the majority of the nodes choose the same gateway and quickly degrades its performance and moves to the next best option. This behavior enforces overall network load imbalance, congestion, and short/long term performance degradation [RT02; SK03; AJO07; Zam+16]. Therefore, in a distributed gateway selection algorithm, the individual nodes have to consider the other nodes' selection impact.

Many algorithms introduce a load balancing mechanism to cope with the herd behavior [HLT04; Nan+06; AAJ09b; Yan+13; KBM12; NMS15; ZC08]. Different metrics have been proposed for load balancing, such as Load Balancing Index in [HLT04] using total traffic and the gateway capacity, or the work in [Nan+06] using queue length, and hybrid load measurements [AAJ09b;

Yan+13] combining gateway load, path quality, number of hops, etc. Learning Automata-based Load Balancing (LALB) [KBM12] uses an algorithm similar to reinforcement learning to find the perfect combination of performance and load balancing. Load balancing algorithms need load monitoring which is a resource-intensive task. Cluster-based resource allocation algorithms [SRS12; Lia+16; YG19] reduce the complexity of the load balancing. The authors of [SRS12] propose energy-efficient resource allocation considering the cluster node's residual energy. In [Lia+16], the two-step resource allocation consists of resource block allocation and game-based power allocation for the sensor nodes. The drawback is the complexity of the clustering mechanism.

2.2.4 Compared algorithms

In the following, we present two algorithms that we consider as closely related to the goals of our thesis. In the next chapters, we will compare our results to them.

2.2.4.1 The CoPing algorithm

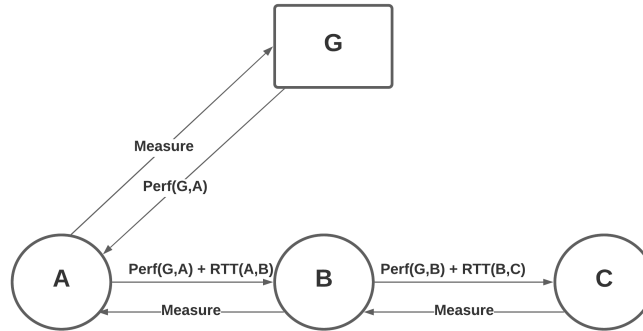


Figure 2.2: CoPing example

CoPing [Ko+13] stands for "Cooperative Ping", an algorithm designed for hybrid wireless network. The primary motivation is to help mobile ad-hoc nodes to find a gateway node based on up-to-date network performance metrics without incurring an additional cost in the network. Thus, CoPing introduces cooperation between mobile nodes where nodes obtain gateway measurements from their upstream node as shown in Figure 2.2. For this example scenario, node C wants to know gateway G's performance. Node B and node A are on the path to reach gateway G. Therefore, the closest upstream node to the gateway G, in this case, node A measures the gateway performance and

distribute the measurement to B, and B sends the measurement to its downstream node C. Upon receiving the measurement from the upstream node, each node updates the measurement by adding their ping latency between the two nodes. The algorithm performs better with lower node mobility and fewer intermediary nodes between the nodes and the gateway. The measurement frequency is every 5 seconds, and the gateway performance is measured by the ping latency.

The downside of CoPing is that if any node becomes offline or any other topology change happens a reorganization between the nodes is required. Furthermore, the accuracy of the collaborative sensing reduces as the number of intermediary nodes increases. Therefore, it is less suitable for a multi-hop mesh network like the guifi.net CN environment.

2.2.4.2 Synthetic coordinate based gateway selection algorithm

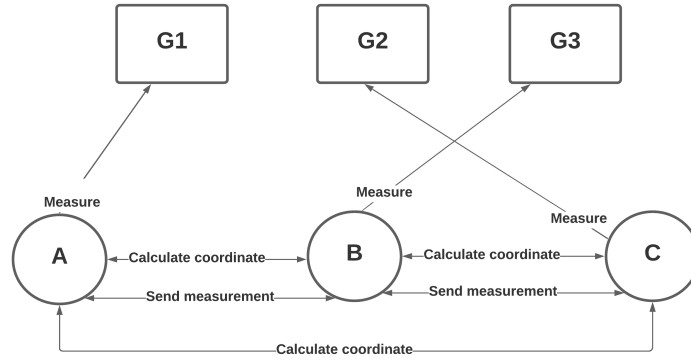


Figure 2.3: Vivaldi based algorithm example

The Vivaldi algorithm [Dab+04] creates a network-wide coordinate system by assuming the distance between 2 nodes' coordinates will accurately measure latency between them. By doing so, every node can determine the distance (latency) between any node. The algorithm incorporates proactive random node probing and slowly adjusting the synthetic coordinates of the network nodes. In the original paper [Dab+04], the accuracy of the synthetic coordinate system stabilizes from the 100th rounds. The Vivaldi-based Internet gateway selection algorithm [Dim+17b] (Figure 2.3) uses the Vivaldi algorithm and combines gateway nodes and all the other nodes in the network to create a whole synthetic coordinate system. Thanks to this, every node knows the performance of the gateway without measuring it, which is the main advantage of the approach. The result in [Dim+17b] shows that

the distance estimation error is low and over 80% of the experiment estimations have lower than 2.5ms median error. However, since the gateway performances can change frequently in practice, continuous adjustment of the synthetic coordinates is necessary.

The main disadvantage is that the algorithm has a long bootstrapping phase of the collaborative measurements to adjust to an acceptable (>80%) accuracy. The collaboration between the nodes is global; therefore, lots of inter-node communication is done. As the size of the network grows, the cost of communication grows exponentially as well; therefore, we do not consider it suitable for large CNs like the guifi.net network.

The Sense-Share-Select framework

3

3.1 Introduction

The Internet gateways regulate the network traffics directed towards the Internet and act as a translator between Internet Service Providers (infrastructured network) and the heterogeneous network nodes (infrastructure-less network). In a home network setting, the internet gateway is assigned by default by the Internet Service Provider. This is not the case in large-scale networks where Internet connectivity is shared with many others. Sharing access is an option for areas where Internet connectivity is scarce. Instead of a single default Internet gateway with ample capacity, many local access networks rely on multiple but limited connections shared across all clients, with a high ratio of nodes per gateway.

We propose a collaborative, informed Internet gateway selection framework, *Sense-Share-Select*, with a lightweight and accurate gateway performance monitoring coupled with a reliable, balanced gateway selection. There are three main layers in the Sense-Share-Select framework, shown in Figure 3.1, that support the gateway selection. The main idea of Sense-Share-Select is to enable nodes to make an informed, individual Internet gateway selection with the help of collaborative performance monitoring. We target networks that are large, heterogeneous, with different topology, devices, and localized network planning. Therefore, the framework is designed to be infrastructure-agnostic and easily integrated with existing networks. For example, guifi.net's decentralized community network in Spain (see Chapter 7) already has 36000 nodes using approximately 400 internet gateways, and the existing users are often reluctant to new changes, making the implementation process slow. Therefore, we have designed our framework to work independently from the underlying network technology.

The bottom layer is the *Sensing layer*, responsible for providing an individual gateway performance monitoring, managing the orchestration of the randomized sampling and finding the close neighbors to assist the *Collaborative layer*. The middle layer is the *Collaborative layer* where the business logic of sharing measurements, ranking neighbors, and receiving measure-

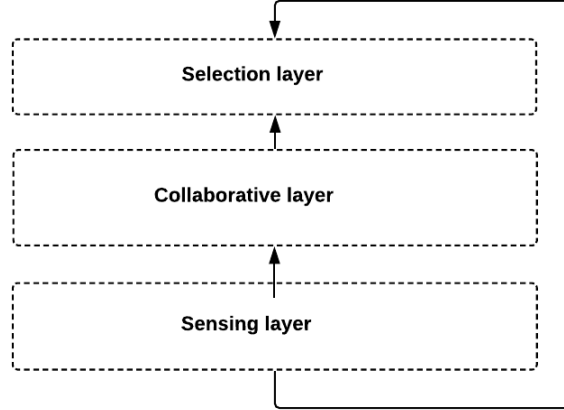


Figure 3.1: The layered architecture of Sense-Share-Select framework

ments resides. The upper layer is the *Selection layer*. It performs the gateway selection while maintaining the global fair distribution of nodes over the gateways.

However, in the framework, we do not cover the following features:

1. **Gateway discovery** - As explained in the previous chapter, we assume that gateway nodes are set up prior to the gateway selection and that the network under consideration is mostly non-mobile. For example, in some community networks, the gateway nodes are known to the users through a registry service. We refer to the existing literature on gateway discovery through broadcasting and node registry services.
2. **Node discovery** - Similarly to the gateway discovery, neighbor node discovery is not part of the framework. The list of available nodes is given as input to the framework. For example, guifi.net and Freifunk networks have a node registry service that provides information on nodes based on their zone, area, and locality.

3.2 Objectives

The main objective behind the framework is to equip each node with the latest gateway performance measurements to select a stable, good-performing gateway for an extended period. We describe the requirements to our framework in the following.

3.2.1 Low-cost performance monitoring

Selecting the best gateway implies that every node should be fully informed about the latest gateway performance, which requires periodic performance monitoring. Moreover, if the gateway performance is not stable, the frequency of the measurement period should be high. The objective to keep track of the performance fluctuations generates a considerable amount of monitoring requests at the gateways; the demand increases exponentially with the size of the network.

In this thesis, *Low-cost performance monitoring* refers to reducing the amount of performance monitoring requests as well as the number of messages exchanged between the network nodes. We propose an individual node performance monitoring algorithm using randomized sampling, client-side performance prediction, and gateway capacity-based performance sensing.

3.2.2 Accurate collaboration

With the low-cost performance monitoring, each node is able to produce its performance measurement. Usually, in the large-scale network setting, the density of nodes is high, and geographically or topologically close nodes often share the same path to the gateways. Therefore, their perception of the performance of the gateways is often similar. We use this common feature to our advantage to share the measurements among close neighbors. Doing so improves each node's available gateway performance measurements and allows them to better choose an appropriate gateway. Many collaborative monitoring algorithms have to choose between accuracy and management overhead. We propose a simple, local collaboration with low overhead that achieves accurate monitoring.

3.2.3 A good gateway selection

Letting a node choose selfishly the best gateway is not ideal, as it can result in overloaded gateways (see Chapter 1). Our main idea for the final selection is to create a list of viable gateway candidates and choose one randomly. Each node creates its own gateway selection candidates as their performance perception of the gateways might be different from the others. Taking away the power of selecting the best gateway from the nodes gives the other less popular gateways an opportunity to be selected and creates a balance in the overall network gateway distribution.

3.2.4 Fair gateway distribution

The Internet gateway distribution in a large-scale network is not efficient, leaving some gateways unused. Some gateways are good performing and able to handle multiple requests but are not the best performing gateway. The nodes should give a chance for those equally good performing gateways to be selected instead of overloading the default local gateway.

3.3 Overview of the framework

In WCN, physically close nodes will often share a similar path towards the gateway nodes, which means their performance perception of the gateway nodes is similar. In our framework, close nodes can use each other's measurements without modification. An example scenario is shown in Figure 3.2, where node A and node B are close neighbors and node C and node D are close neighbors, and they exchange measurements with each other. This is the main foundation of the Sense-Share-Select framework.

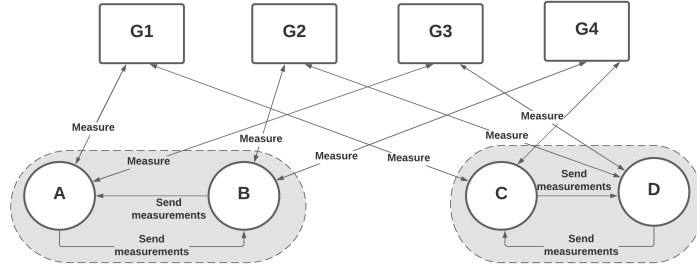


Figure 3.2: Example of Sense-Share-Select collaboration

Figure 3.3 shows the detailed structure of the Sense-Share-Select framework. The framework runs periodically in rounds. Every round, the client nodes sense gateway performances, share measurements with the collaborator nodes, receive measurements from the others and select their gateway node. A node stores measurement results in a local *gateway performance table*, similar to the one shown in Table 3.1. Every node works for its own benefit to receive good Quality of Service by choosing a "good" Internet gateway and avoiding the bad ones. This thesis refers to a *good* Internet gateway for a specific client node as the gateway node, which provides good performance (end-user experience) with minimal performance deviation over an extended period.

Measurements can be performed in different ways. Algorithms proposed in the literature use network-level metrics such as Round Trip Time, bandwidth, signal-to-noise ratio, packet delivery rate, etc., but they only cover

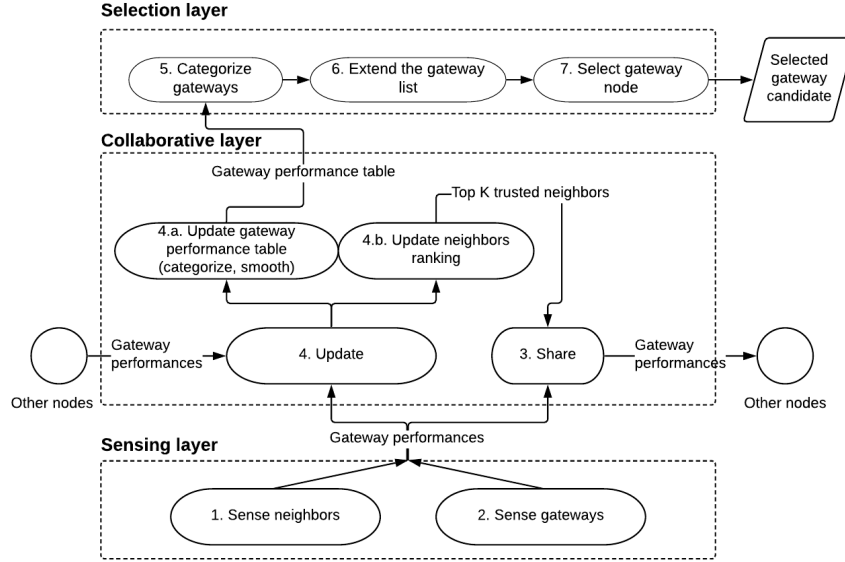


Figure 3.3: Sense-Share-Select framework structure

Table 3.1: Example of gateway performance table at each node

Gateway	Latency	Timestamp
Gateway 1	0.6ms	2018-07-01 14:39:14
Gateway 2	0.4ms	2018-07-01 14:37:27

the network path condition. In this thesis, we define that a gateway performance measurement consists of measuring the time (called "latency" in the following) needed to download a 0.1MB¹ resource via HTTP from a file server located on the Internet. End users care about their Internet experience, and the "Internet latency" is an end-to-end metric to measure the gateway performance.

¹The resource size of 0.1MB was chosen as a compromise between measurement overhead and accuracy. Depending on the network, smaller sizes could be used.

Sensing layer

| 4

4.1 Introduction

The *Sensing layer* is the fundamental layer of the framework that provides the periodic gateway performance measurements.

There are two extremes for the gateway selection algorithm related to the cost of performance monitoring. The first one is where every node measures the performance of all gateways (Brute Force). However, the cost of the Brute Force strategy is high, especially in large-scale networks. In the end, the node chooses only one gateway as their default Internet gateway and discards all the other measurements. Thus, plenty of measurements are left unused to select one gateway [SSK97]. The other extreme is a completely random selection or a manual gateway selection, where the node does not measure any gateway performance and selects the gateway based on the location, word-of-mouth knowledge, or previous experience. Manual selection leads to load imbalance over different gateway nodes by over-popularizing some gateways and bad or uneven performance for the selected nodes.

The ideal situation is to find a balance between a simple non-informed gateway selection method (random), which has zero communication overhead but can lead to poor performance, and a comprehensive informed gateway selection (brute-force), which has a significant communication overhead but can achieve the best overall performance. The goal of the *Sensing layer* is to reduce the measurement costs generated by each node.

4.2 Design requirements

There are two main design requirements for the *Sensing layer* algorithm.

- **Reduce the performance monitoring overhead** - The gateway's main purpose is to provide fast Internet access. Therefore, the active probing algorithm should avoid generating too many measurement requests per gateway. Ultimately, the sensing layer algorithm should focus on distributing the overall gateway probing requests generated by all nodes in a balanced manner.

- **Reuse the previous measurements to improve the monitoring process** - Active probing enables the node to collect measurements over time, and there is plenty of room to analyze the collected measurements to reduce the probing cost.

4.3 Design overview

The fundamental principle in gateway performance sensing we use is "random sampling". Instead of measuring the performance of all gateways, each node only samples a random subset. When coupled with the collaboration among close neighbors described in Chapter 5, good visibility of the performance of the gateway nodes can still be achieved. Existing work suggests that a small number of samples per node can be sufficient [Mit01; Zha+17a; Gho+17; Nas+15].

4.3.1 Gateway performance monitoring

The fact that some choice is good doesn't necessarily mean that more choice is better.

by Barry Schwartz in "Paradox of Choice".

We argue that a partial knowledge of the gateways' performance can help to select a "good-enough" gateway and avoid the worst choices. The idea of exploiting partial knowledge while providing some options to the client node led us to explore approaches based on the Power of Two Choices (PoTC) [Mit01]. In PoTC, every entity randomly selects two resources, measures their performance, and selects the best out of two options. This algorithm has been used for many resource load balancing algorithms [Gho+15; Zha+17b; Ber+17] and job scheduling algorithms [Ous+13] for balanced resource distribution with minimal complexity.

The *Sensing layer* is designed to let the client node monitor gateway performance independently and efficiently (with as few overhead as possible) while considering global network-wide load balancing. Every round, the node randomly selects two gateways from the list of gateways and measures their performance. The results are stored in the node's gateway performance table. As a result, the number of performance measurements per node reduces from n down to 2 per measurement period, with n being the number of available gateways. Two is the minimum number of measurements that an isolated node should make to allow an informed decision. Measuring two random gateways in each measurement period reduces the probability that two neighbor nodes will measure the same gateway. Every node has a small sample of its own measurements accumulated over time to better judge the gateway performance.

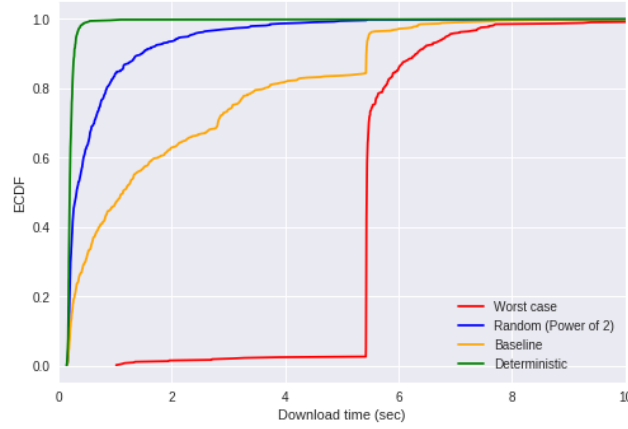


Figure 4.1: Experiment with different gateway performance measurements

To illustrate our idea, we conducted a small experiment with four client nodes in the university lab selecting from five different Internet gateways from the guifi.net production network. We ran different selection algorithms for one day. Performance monitoring requests to the five gateways are made every three minutes, and for each algorithm, clients request 0.1 MB content from the Internet through the selected gateway every one minute. The algorithms were:

- Static (Worst case scenario) - Static gateway selection where nodes do not have any gateway performance measurements. Zero communication would lead to zero overhead, but also to a non-informed proxy selection.
- Random (Baseline scenario) - Client selects one random gateway node every round. This achieves even load balancing.
- Random (Power of Two choices) - Measures 2 random gateway performances and selects the best performing gateway.
- Deterministic (Best case scenario) - Measuring all gateway performances and selects the best performing gateway.

The results depicted in Fig. 4.1 show that the randomized proxy selection algorithm (Power of Two) achieves better results than the worst case scenario and random gateway selection (baseline), and slightly worse download times than the Best case scenario.

Performance	Deviation	Perf. category
$M_{tj} \leq \mu$	$stdev(M_j, k) \leq stdev(M)$	Good
$M_{tj} \leq \mu$	$stdev(M_j, k) > stdev(M)$	Inconsistent
$M_{tj} > \mu$	$stdev(M_j, k) \leq stdev(M)$	Inconsistent
$M_{tj} > \mu$	$stdev(M_j, k) > stdev(M)$	Bad

Table 4.1: Gateway performance categories

4.3.2 Capacity based monitoring

Nodes learn the behavior of gateways from past measurements. With these measurements, nodes can remove unreliable (i.e., bad performing or inconsistently performing) gateway node candidates from their list of monitored gateways to lower monitoring costs. We consider two features to characterize a gateway's capacity to provide satisfying performance:

1. Current performance M : The measurement result.
2. Performance deviation $stdev(M)$: The standard deviation of the performance over the previous measurement rounds. The higher the variance, the worse the reliability of the network service provided by the gateway.

For example, as a node is trying to find a "good" gateway, there is no reason to frequently measure the performance of the "bad" performing gateway. On the other hand, the gateway deemed "bad" might be experiencing slight congestion, then performs well after some time. In this case, the node should not eliminate the "bad" gateways.

The node assigns a *performance category* to gateway node G_j based on its current performance M_{tj} and the standard deviation $stdev(M_j, k)$ of its performance in the last k rounds ($k = 10$ by default), following the rules in Table 4.1. The thresholds μ and $stdev(M)$ are the average performance over all gateways at round t_i and the standard deviations of all gateway performances within the last k rounds, respectively. The classification is done every time a node receives a new measurement M_{ij} . For example, if the current performance of the gateway is lower than the threshold and the current gateway standard deviation is lower than the threshold standard deviation, the gateway performance is categorized as *Good*.

The *Good* category gateways have a high probability of being a selection candidate for the node. Therefore, the *Sensing layer* component senses the *Good* category gateways more frequently by adding them into the sensing list of gateways upon every round. The *Inconsistent* and *Bad* category gateway nodes are added into the sensing list of gateways every other measurement

round. The *Prototype* applies constraints on the *Bad* and *Inconsistent* gateway nodes, probing them with less frequency, therefore, making more difficult their transition to the *Good* category gateway. Depending on the number of *Inconsistent* and *Bad* gateways, the gateway selection list contains a smaller number of candidates resulting in fewer measurement requests.

4.3.3 Close neighbor sensing

In addition to performing gateway measurements, the Sensing layer contains a *Sense neighbors* component that computes the list of close neighbor nodes to collaborate with in the Collaboration layer (see Chapter 5). The main idea is that close neighbor nodes have similar performance perceptions of the gateways; therefore, they benefit from exchanging performance measurements.

The list of nodes in the network is given as an input to the *Sense neighbors* component, i.e., every node knows the network nodes. There are different possible ways for a node to find other nodes. A node can broadcast or multicast discovery messages, passively listen to wireless communication in its neighborhood in promiscuous mode, or use a node discovery service. For example, the guifi.net CN has a central node registry service. In our implementation of this component, we rely on the existing registry service.

In order to decide whether a neighbor node is a *close* neighbor, we propose to perform round-trip time (RTT) measurements. We define the RTT as the ping delay between two nodes. As explained, WCN can be geographically very extended with multiple hops between nodes, and the RTT can be used as a simple metric roughly correlated to the physical or logical distance between two nodes. A node regards neighbors with an RTT below a certain threshold as close neighbors.

The *Sense neighbors* component can be also called to verify whether a specific node is a close neighbor node. This feature will be used in Chapter 5 to verify the origin of unsolicited collaboration messages.

The main challenge in this approach is the fine-tuning of the RTT threshold value. As shown later, the smaller the threshold value, the more accurate the gateway performance perception will be since there is a higher probability that the identified close neighbors of a node share the same paths to the different gateways. On the other hand, a big threshold value leads to a large set of close neighbors and, therefore, more available measurement results, leading to a more informed gateway selection on a node.

4.4 Experimental algorithm and results

We designed the Sense-Share-Select framework to function individually as a separate algorithm. For the *Sensing layer's* case, we proposed an algorithm

called RIMO (Randomized Informed Minimized Overhead) where every node individually filters the gateway before the measurement and combines with the PoTC gateway performance monitoring component.

4.4.1 Gateway performance sensing (RIMO algorithm)

The idea of Algorithm 1 is to assign weight and filter all the available gateways to improve the result of the randomized selection by eliminating bad choices. The **user's preferences** (Filter procedure, line 1) are some user-based reasons (e.g., the economic cost and the trust) to filter the available gateway list Γ and obtain a subset of gateways Γ_{filtered} . The **impact to the network** (HOPStoGateways procedure, line 8) is minimized by a weighted mechanism that reduces the network usage. The weight of each gateway has an inverse linear relationship with the number of hops between the client and the gateway: $\text{MaxHOPS} + 1 - \text{hops}$ (line 9), which can be explained since communication with a close gateway uses fewer network resources (traffic on links) than communication with a far one. As a secondary impact of this weighting mechanism, we expect a slightly better network performance, as shown in [DMN17]. The output of the Algorithm 1 is the weighted gateway list Γ_{weighted} . We use network hops as a quasi-static client-gateway distance, so the result of the algorithm is also quasi-static.

Algorithm 1 Weighting gateways

Input: $\Gamma = \{G_1, G_2, \dots, G_j\}$ ▷ Available gateway list
Input: *preferences* ▷ User preferences

```

1: procedure FILTER( $\Gamma, \text{preferences}$ )
2:   for all gateway  $\in \Gamma$  do
3:     if gateway  $\in \text{preferences}$  then
4:        $\Gamma_{\text{filtered}} \leftarrow \text{ADD}(\text{gateway})$ 
5:   return  $\Gamma_{\text{filtered}}$ 

6: procedure HOPSTOGATEWAYS( $\Gamma_{\text{filtered}}$ )
7:   for all gateway  $\in \Gamma_{\text{filtered}}$  do
8:      $\text{hops} \leftarrow \text{HOPS}(\text{gateway})$ 
9:      $\Gamma_{\text{weighted}} \leftarrow \text{ADD}(< \text{gateway}, \text{MaxHOPS} + 1 - \text{hops} >)$ 
10:  return  $\Gamma_{\text{weighted}}$ 
```

Output: Γ_{weighted} ▷ Weighted gateway list

In the Sense-Share-Select framework, we incorporated PoTC principles from the RIMO algorithm and the general concept of filtering and ranking

Strategy	Analytical	Experimental
Brute force	$O(\text{size}(\Gamma) * \text{size}(\mathcal{C}))$	59.39
RIMO	$O(\text{size}(\mathcal{C}))$	9.95
Power of Two	$O(\text{size}(\mathcal{C}))$	9.94
Random	$O(0)$	0

Table 4.2: Message overhead per selection round.

the gateway nodes based on the gateway performance measurements. More details are given in Chapter 5 regarding the organization of the measurements at each node. In this section, we show the result of the randomized sampling (a.k.a RIMO algorithm) approach to reduce the cost of the monitoring and final Internet download latency through the selected gateways. The result of the close neighbor collaboration is shown together with the *Collaborative Layer's* experiment results in Chapter 5.

4.4.2 Experimental setup

We performed experiments in Mininet [LHM10] with ten gateways and 50 client nodes. To simulate the performance variations, we used the Linux Network Emulator (tc netem) [tc-] by adding 10ms delay for the "Bad" gateways (g_8, g_9, g_{10}), 3ms delay with 1-3ms delay distribution for the "Inconsistent" gateways (g_4, g_5, g_6, g_7) and 1ms delay for the "Good" gateways (g_1, g_2, g_3). The gateway performance measurement is done every 2 minutes for 100 rounds.

4.4.3 Experiments

4.4.3.1 Overhead

Each client performs a gateway selection process every round for validation purposes. The algorithm is compared to **Random**- Not informed selection, **Brute-force**- Best informed selection based on complete knowledge of the gateways' performance. Probing all gateways, **PoTC**- Randomized informed selection based on partial knowledge of the gateways' performance. Probing only two gateways, **Omniscient-based**- Post-experiment selection based on the best selection using the Omniscience metric. The Omniscience metric was defined, a synthetic and post-experiment metric, to show the real performance of the gateway. This metric is defined as the average HTTP content latency of a window of 5 min from the selection moment.

From the overhead perspective, Table 4.2 shows the network cost of the different selection approaches in terms of messages per selection round, where Γ is the set of gateways and $\mathcal{C}$ is the set of clients. Our algorithm is significantly better than the Brute-force approach, close to zero communication

overhead approaches (e.g., Random). It is important to note that the cost of the Brute-force approach depends linearly on the number of available gateways and the number of clients involved in the selection process at the current round.

4.4.3.2 Measurement distribution

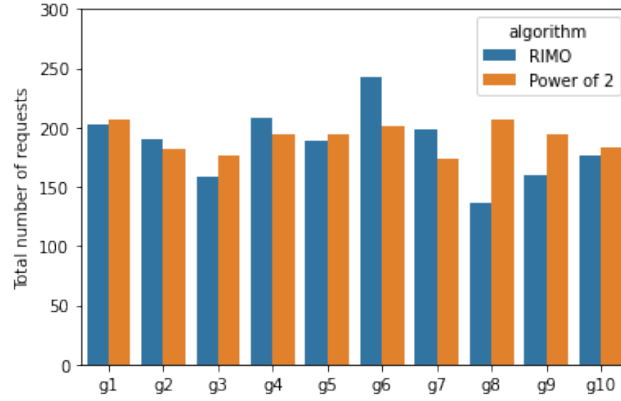


Figure 4.2: Total gateway measurements

Figure 4.2 shows the total gateway performance measurement requests per gateway and the experiment for 20 rounds. The result is compared with the PoTC, randomized sampling (sample size=2) algorithm. The advantage of using the PoTC algorithm for the gateway performance monitoring is evident from Figure 4.2. From the result, our *Sensing layer* algorithm sends fewer measurement requests to "Bad" gateways [g8, g9, g10] as compared to the other gateways. The PoTC algorithm (without performance bias) distributes the performance measurement requests equally over all gateways without performance bias. Thus, our performance categorization component works efficiently to reduce the number of measurements.

4.4.3.3 Download performance of RIMO algorithm

Figure 4.3 shows an empirical cumulative distribution function (ECDF) of the performance of the different selection gateway approaches. For the 80% of the requests (0.8 of the y axis), the download latency is lower than 0.03, 0.09, 0.18, 0.21, and 0.54sec for the Omniscient-based, Brute-force, RIMO, PoTC, and Random approaches, respectively. The RIMO algorithm result is better than the Random and the PoTC selection algorithms and very close to the Brute-force selection. Analyzing the results deeper, we observe that a user perceives an additional delay when the best gateway is not selected. This

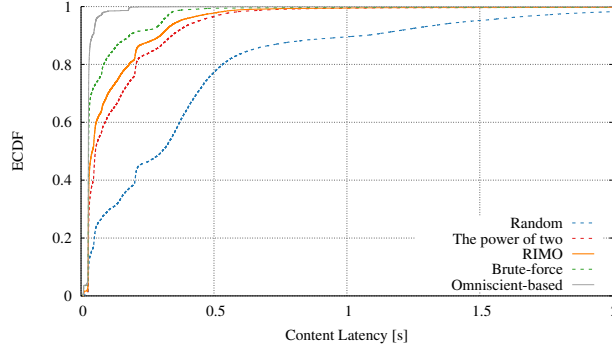


Figure 4.3: Client perception of the selected gateway's performance.

cost in time can be defined as the difference between the Content Latency of the approach and the Omniscient-based as a best one. For the 80% of the requests, the difference is lower than 0.06, 0.14, 0.18, and 0.5 sec for Brute-force, RIMO, PoTC, and Random approaches, respectively. These additional delays are small enough not to have any impact on the users QoE.

4.5 Conclusion

The *Sensing layer* algorithm puts importance on the node's performance measurements coupled with a preselection of candidates based on past performance. The algorithm reduces the traffic overhead to the gateways. The randomized gateway performance monitoring achieves good distribution of the requests to the gateways and reduces unnecessary performance monitoring requests.

The outcome of the *Sensing layer* is (a) the node's gateway performance table filled with the latest performance measurement results of the gateway nodes and (b) the list of the close neighbors to share measurements with.

The proposed individual gateway selection algorithm, RIMO, provides a good-enough gateway selection because the selection list is limited. The collaborative monitoring algorithm proposed in Chapter 5 is essential for improving the gateway selection candidates.

Collaborative layer

| 5

This chapter explores the design of a collaborative performance monitoring mechanism. We show that creating a customized close neighbors list to collaborate is a simple and effective way to assure high accurate performance monitoring. Large parts of the work presented in this chapter have been previously published by us in [Bat+19a; Bat+19b].

5.1 Introduction

The explicit measurement of all gateways by each node is not suitable for a large-scale network setting since the cost of measurement can outweigh the benefits of gateway selection. Even though the overhead of a single measurement request is small, in a resource-constrained, large-scale network, the overall traffic generated by measurement requests from every node becomes a major contributor to network congestion. Many performance monitoring algorithms [Dim+17b; Ko+13; BH11; AJB10] pursue providing full visibility of the gateways just to select one best suitable gateway for the node. Thus, in the end, a significant amount of measurements and message exchanges is left unused. As we have seen in Chapter 4, a partial view of gateways is also effective in terms of reducing in-network traffic, measurement overhead, and balancing the client nodes over the gateways.

In this chapter, we will explore a collaborative mechanism that allows nodes to share their gateway performance measurements to reduce the number of measurements needed to make an informed gateway selection. Unfortunately, in large networks such as a WCN or a Wireless Sensor Network, collaborative sensing between *all* nodes is not scalable. Instead, nodes are typically organized into smaller groups, and collaboration happens only inside those groups. Different solutions have been proposed to (self-)organize nodes into groups in order to achieve network-wide objectives such as load balancing, fault tolerance, or energy efficiency [HCB02; NWZ16; FM02; LQL13; ZC08].

Sharing measurements only makes sense among nodes that similarly perceive the gateway performance, but the grouping process is tricky. For example, a simple topological grouping applied to the network shown in Figure 5.1a might lead to the two groups shown in Figure 5.1b. Depending on the charac-

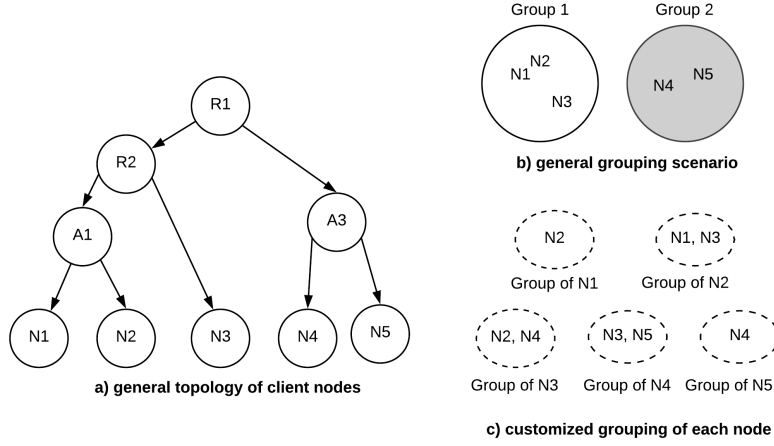


Figure 5.1: Example scenario for forming a group of collaborating nodes

teristics of the network links, the nodes N1, N2 and N3 might see or not see a similar performance of a gateway. Algorithms like the Synthetic coordinate-based algorithm and CoPing algorithm presented in Chapter 2.2.4.1 perform complex measurements and adjustments to organize the nodes and compensate for the nodes' differences.

We propose a more straightforward mechanism for collaborative sensing that avoids such adjustments. Instead, we allow each network node to choose close neighbors to collaborate with, resulting in individual groups such as the ones depicted in Figure 5.1c. The underlying assumption is that nodes that are close to each other (in terms of, e.g., round trip time) probably also share most of the paths to the gateway nodes, and therefore they see similar gateway performance.

5.2 Design requirements

Low collaborative monitoring overhead (R1). Collaborative performance monitoring introduces a different set of communication such as exchanging measurements, scheduling, adjusting measurements, continuous node checking, and more. Excessive inter-node communication can create congestion in the network, which also harms the QoE of the nodes. Therefore, the algorithm should avoid creating an extensive amount of traffic within the network.

Accuracy (R2). Ideally, the collaboratively collected measurements at each node should be as accurate as the node's own measurements. We express the accuracy of a collaborative sensing algorithm for a node by the cosine

similarity index defined as

$$\cos(\mathbf{M}, \mathbf{A}) = \frac{\mathbf{M} \cdot \mathbf{A}}{\|\mathbf{M}\| \cdot \|\mathbf{A}\|} \quad (5.1)$$

where $\mathbf{A} = [A_0, A_1, \dots, A_p]$ is the real performance metric of the p gateways that the node would measure without collaboration, and $\mathbf{M} = [M_0, M_1, \dots, M_p]$ is the estimation obtained by sharing measurement results with other nodes. As a result, an index of 1 (or 100%) would mean that the performance measurement results obtained through collaboration are identical to those that a node would obtain when performing its own (more expensive) measurements without sharing.

Scalability (R3). The gateway monitoring framework should be able to adapt to large networks with heterogeneous devices. The monitoring process should not influence other network traffic negatively, therefore the cost of monitoring should be kept minimal.

Resilience (R4). The collaborative monitoring algorithm is subject to faulty or malicious collaborators. We consider a node faulty or malicious if it sends wrong measurements consequently over a certain period of time. The framework should be also resilient to selfish (or lazy) nodes that accept measurements from other nodes but do not perform their own measurements or do not share them.

Elasticity (R5). There are many uncertainties associated with the dynamic behavior of networks and gateways caused by, amongst others, link failure, node downtimes, or congestion [DMN17]. Timely reaction to changes is, therefore, crucial. State-of-the-art algorithms react to such changes with increased computational complexity caused by, for example, cluster head re-elections.

5.3 Design overview

We propose a client-side collaborative performance monitoring algorithm where network nodes collaborate with closely located neighbor clients to sense the performance of the gateways. As explained in Chapter 4 each node keeps a gateway performance table, where it stores the results of its own gateway performance measurements for the different gateways. Now, in addition, it also stores the gateway measurement results obtained from its neighbor nodes. The main objective of collaborative monitoring is to reduce the in-network traffic and increase awareness about gateway nodes at each client node.

5.3.1 Sharing measurements

The overview of the *Collaborative layer* is shown in Figure 5.2. The bottom layer provides the performance monitoring covered in Chapter 4. The middle layer, *Collaborative layer*, is in charge of the actual collaboration between network nodes, i.e., the exchange of measurement results. The top layer selects one gateway from the table of measured gateways.

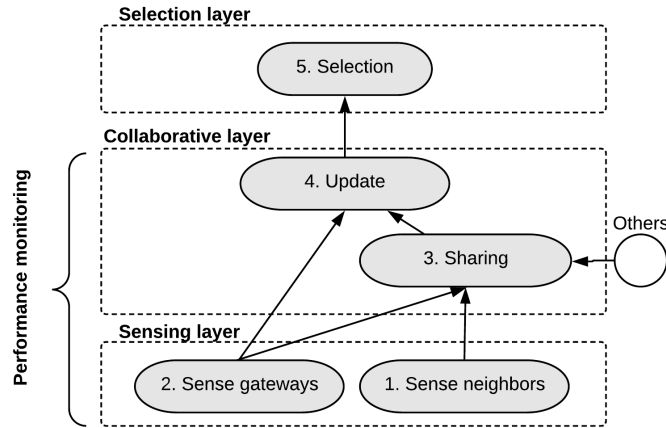


Figure 5.2: Layered structure of the collaborative algorithm

All components of the framework run on the client-side, i.e., on the network nodes, and their execution can be roughly divided into two phases, the *bootstrapping phase* and the *periodic sensing phase*. When a node is activated, it starts with an empty gateway performance table. Therefore, the goal of the bootstrapping phase is to identify the set of close neighbors and receive their measurement results to fill the node's table. With this initial version of the table, the node can make a first gateway selection. After the bootstrapping phase, the node enters the periodic sensing phase, where it performs its own measurements (sensing) and exchanges measurement results with neighbor nodes.

The interaction of the different layers during these two phases is depicted in Figure 5.3. We will now explain the different components in more detail.

Collaboration, i.e. the exchange of gateway measurement results, is limited to the close neighbors identified by the *Sensing layer*, assuming that the performance perception of the gateways between close neighbors is similar. The collaboration consists of three parts: (a) Sending own measurement information to close neighbors, (b) receiving information from those nodes, and (c) updating the node's gateway performance table.

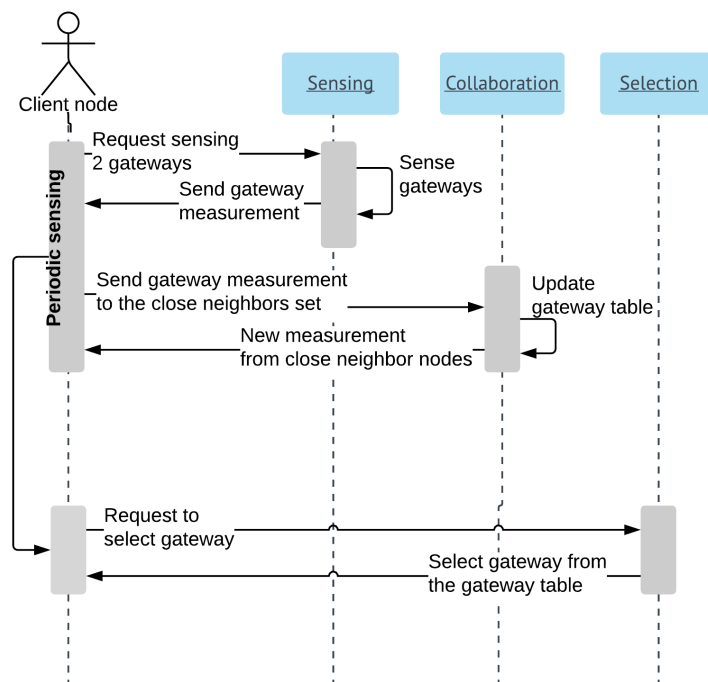


Figure 5.3: Periodic sensing phase

Algorithm 2 Collaborating with close neighbors

Input: $close_neighbors = \{C_1, C_2, \dots, C_k\}$
Input: $gateway_table$ ▷ Gateway measurement table item[]

```

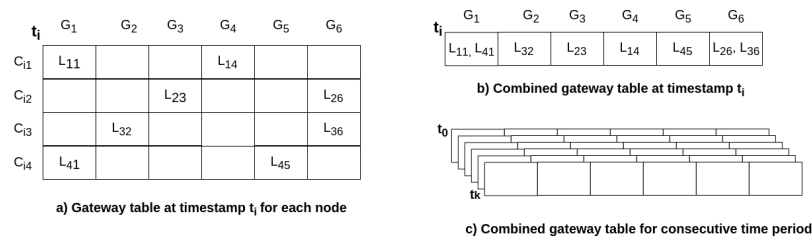
1: procedure RECEIVE(sender, message)
2:   if sender  $\in$  close_neighbors then
3:     if message == 'whole table request' then
4:       SEND(gateway_table)
5:     else
6:       for all info  $\in$  message do
7:         gw_address  $\leftarrow$  info.gw_address
8:         if info.timestamp > gateway_table[gw_address].timestamp
9:       then
10:         gateway_table[gw_address]  $\leftarrow$  info
11:     else
12:       isNeighbor  $\leftarrow$  SENSENEIGHBOR(sender)
13:       if isNeighbor then
14:         goto line 3
15: procedure SEND(message)
16:   for all neighbor  $\in$  close_neighbors do
17:     SEND_DIRECT(neighbor, message)

```

Sending: Whenever a node has performed its performance measurements in the gateway sensing component, the measurement results are directly sent to its close neighbors identified by the neighbor sensing component.

Receiving and updating: Information reception from neighbors is handled by the Receive procedure in Algorithm 2. This procedure distinguishes between information received from nodes known to be close neighbors of the current node (line 2) and unknown senders (line 10). In the former case, two types of messages are supported: During the bootstrapping phase, a node can request the whole gateway performance table from its closest neighbors to quickly obtain an initial version of the table. Such a message is handled in lines 3–5. The second type of message includes messages sent during the periodic sensing phase. They contain performance measurement results from a neighbor node. If a contained measurement result is more recent than the information currently stored in the node's table, the corresponding entry in the table is overwritten (lines 6–9). The message is not simply discarded when the sender is unknown to the receiving node (lines 10–13). Instead, the node requests the sensing layer to check whether the sender is a close neighbor. If this is the case, the message is processed.

It should be noted that collaboration between nodes is not transitive, in



the sense that a node will not forward a copy of the information received from a neighbor node to its other neighbors. This choice enforces the precision of the estimation within a neighborhood and reduces unnecessary network traffic generated by gossiping.

5.3.2 Trust-based collaborative monitoring

Periodic collaborative monitoring algorithm builds historical gateway performance measurements. When collaborating with multiple close neighbors, the probability of measurement overlaps in the gateway performance table increases. As an example, let's assume that four clients C_{i1} through C_{i4} have performed performance measurements on the six gateways G_1 through G_6 . Table 5.4 shows the results L of the individual probing actions at time t_i . After sharing the results, each client can construct a combined measurement table (Figure 5.4b) which is updated through periodic measurements (Figure 5.4c). In Figure 5.4b, the client node has two measurements of G_1 and G_6 , which is a measurement overlap. In Figure 5.4b, the client node has two measurements of G_1 and G_6 gateways received from the collaborator nodes. Overlapped measurements are redundant because all the selected collaborative nodes C_1, C_2, C_3, C_4 are close neighbors with similar measurements. When collecting measurements over the n consecutive periods, the overhead of the overlapped measurements could be wasteful. The *Collaborative layer* addresses this issue through a ranking mechanism to curtail the number of collaborative neighbors.

When receiving measurements from neighbors, the node assumes that all collaborators are benevolent, i.e., no node distributes false information. This assumption can result in an incorrect gateway performance table for various reasons:

- The node might be simply faulty.
- A node could be malicious and try to perform a DoS attack by directing other nodes toward the same gateway.

- In the *Collaborative layer*, we add the components shown in Figure 5.5 to ensure the trusted and accurate collaborative sensing.

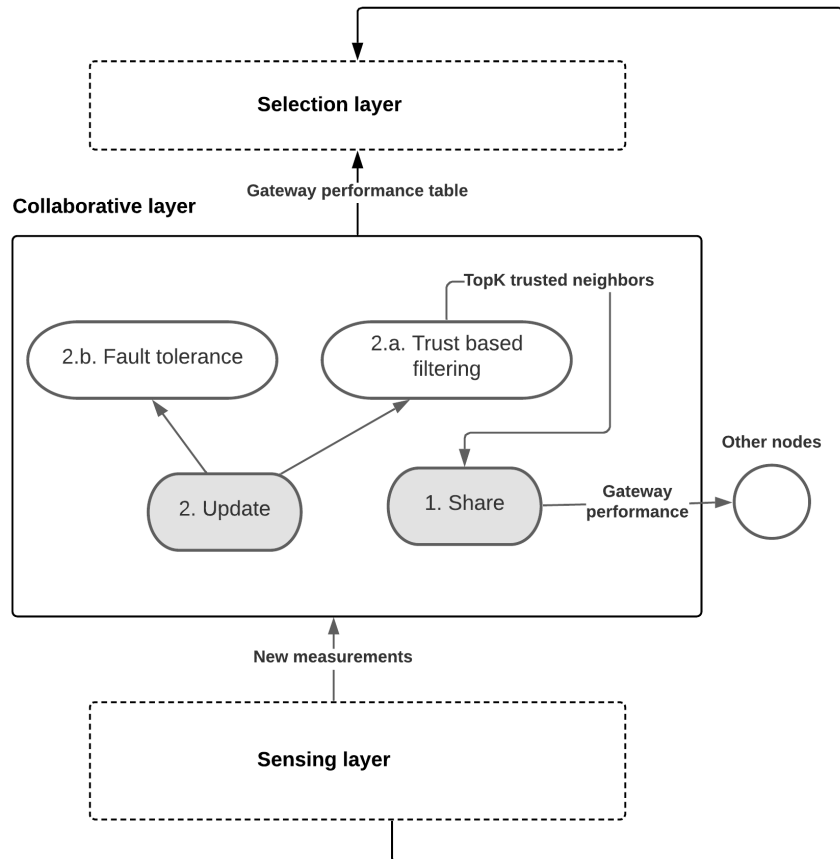


Figure 5.5: Layered sensing framework

5.3.2.1 Trust based filtering

The main idea of the filtering component is to avoid collaborating with less similar nodes. Each node calculates a trust score for its neighbors that indicates confidence towards their collaboration. In the Sense-Share-Select framework, collaboration is limited within the close neighbor nodes; therefore, trust computation can be done locally at each node. The trust is non-transitive; a node considered as reliable for one node might not be considered the same

by another. Moreover, considering that the trust values fluctuate in a dynamic environment and degrade over time [CSC09; Che+14], the trust score is updated every time the node receives a measurement from a collaborator.

Algorithm 3 is added to Algorithm 2 when the node receives a measurement from a neighbor. It calculates the trust score for a neighbor *neighbor* from which the node has received *measurements*. The trust score is calculated as the mean absolute difference between the measurements received from that node for the different gateways and the average of the corresponding gateway measurements received from other trusted nodes. Trust scores are used for further reducing the number of close collaborating nodes by ranking the close neighbors and only collaborating with the top K trusted ones. We define $K = \min(n/2+1, k)$, where n is the number of gateways to measure, and k is the number of close neighbors.

Algorithm 3 Filter trusty collaborator nodes

Input: $trust_score = \{N_1 : T_1, N_2 : T_2, \dots, N_k : T_k\}$ ▷ Trust scores of neighbors

- 1: **procedure** UPDATE_TRUST(*neighbor*, *measurements*)
- 2: Score = 0
- 3: **for all** (*gateway*, *measurement*) $\in$ *measurements* **do**
- 4: m = average of all measurements for *gateway* by other neighbors
- 5: Score = Score + $|measurement - m|$
- 6: Score = Score / size(*measurements*)
- 7: $trust_score[neighbor] = Score$

5.3.2.2 Fault tolerance

The objective of the fault-tolerance component is to reduce the impact of false (relatively higher or lower than the real measured value) measurements sent by the trusted nodes. Instead of storing the measurement result for a gateway directly into the gateway performance table, we use a weighted moving average of the results $M^{(i)}$ of the last r measurement rounds (with default $r = 2$), with $\forall w_i \leq w_{i+1}$, $w_i < 1$ and $w_i = \frac{i}{r*(r+1)/2}$:

$$gateway_performance_table[gateway] = \sum_{i=1}^r w_i \cdot M_i \quad (5.2)$$

In case of sudden performance changes, the moving average catches up within a few measurement rounds, depending on the frequency of received measurements from the trusted neighbors for the specific gateway. On the

other hand, intentional false measurements are smoothed out by the moving average until the *Trust based filtering* component detects the deviating behavior of the sending node.

5.4 Gateway selection

For the evaluation of the collaboration mechanism presented in this chapter, we propose two light-weight gateway selection algorithms which rely on the measurements stored in the gateway performance table. The Sense-Share-Select framework's gateway selection is different, more thorough than these selection algorithms, which we will discuss in detail in Chapter 6.

The first algorithm is *collaborative-best*. It selects the best gateway, i.e., the gateway with the lowest latency measure, from the gateway performance table within the last measurement period. The algorithm is simple and selfish since it does not consider overall load balancing: Since all nodes in a close neighborhood choose similar gateway performance tables, they will likely select the same gateway with this algorithm. As a possible result, the gateway might get overloaded, and performance may degrade for all.

Algorithm 4 Collaborative-Fair

Input: *gateway_table*

Input: *latency_thr* ▷ Threshold for gateway category

Input: *current_gateway*

Input: *measurement_period*

```

1: procedure SELECTGATEWAY(latency_thr)
2:   good_gateways = []
3:   for all gw ∈ gateway_table do
4:     if (NOW − gw.timestamp) < measurement_period    &
       gw.latency < latency_limit then
5:       good_gateways ← ADD(gw)
6:   if SIZE(good_gateways) < 2 then
7:     return SELECTGATEWAY(2 * latency_thr)
8:   if current_gateway ∈ good_gateways then
9:     return current_gateway
10:  else
11:    return RANDOM(good_gateways)

```

The second selection algorithm is *collaborative-fair* where a node will randomly choose a gateway among the best performing gateway nodes according to some latency threshold, as shown in Algorithm 4. The algorithm builds a

list of all gateways for which recent measurements are available and whose measured latency is below the specified threshold (line 4). The procedure is restarted with a doubled threshold if the search does not return at least two gateway candidates (lines 6–8).

We have designed *collaborative-fair* to be conservative in the choice of the gateway. Frequent changes of the gateway could result in sudden short-term performance variations at the gateways, perturbing the performance measurements of the other nodes. Therefore, if the previously selected gateway still belongs to the list of good gateway candidates, the current gateway is not changed (lines 8–9).

5.5 Evaluation

We evaluate the feasibility and performance of the algorithm and compared with other collaborative and non-collaborative approaches. First, we explain the analytical comparison of the *Collaborative layer* algorithm to other algorithms in terms of message exchange and computation complexity. Then we conduct general and parameter-specific experiments to test the performance of the proposal.

5.5.1 Analytic comparison

The idea behind collaborative sensing is that a node can make an informed gateway selection thanks to gateway performance measurements performed by it and by other nodes in the network. This reduces the number of measurements the individual nodes have to make. On the other hand, it also introduces an overhead due to the messages exchanged between the nodes. This overhead is absent in non-collaborative approaches.

The simplest non-collaborative algorithm is *brute-force* selection: In order to obtain the maximum visibility of the gateway performances, a node has to measure all gateways periodically. Alternatively, the node can decide to only measure *two* gateways, which gives us the *PoTC* [Dab+04] algorithm. Its advantage is the reduction of the number of measurements, and if the two gateways are chosen randomly, a distribution of the measurement load over all available gateways. However, the approach only gives partial visibility of the gateway performance to the node. Finally, a node that could not make any measurements at all would simply select randomly one gateway.

For the collaborative algorithms, we consider our approach, sensing based on the synthetic coordinates [Dim+17b] (referred to as Vivaldi-based algorithm), and the CoPing algorithm [Ko+13] as they are both informed and collaborative gateway selection algorithms in the heterogeneous wireless network. Due to the random selection and the small number (two) of measure-

ments, our collaborative algorithm cannot guarantee full visibility of the performance of all gateways. On the other hand, its measurement and collaboration overhead is the smallest among the compared collaborative algorithms discussed here.

In the Vivaldi-based algorithm, each node measures a different set of (more than two) gateway nodes, effectively leading to full visibility of the performance of all gateways. The measurement results are distributed within a set of close neighbors and additional random neighbors. This distribution is done frequently (r times) within one measurement period to improve the estimation of the so-called synthetic coordinates between nodes.

In CoPing, ancestor nodes are identified as responsible for measuring the performance of all gateways, thus leading to full visibility. Those ancestor nodes then distribute the performance information down to the descendant nodes. The cost of the collaborative layer of our proposal and the CoPing algorithm is similar. However, the child nodes in the CoPing algorithm need to adjust the information received from the ancestor node according to their RTT between them. The same applies to the Vivaldi-based algorithm, requiring adjustments of the measurement information depending on the sender nodes' synthetic coordinates. As compared to other collaborative algorithms, our collaborative sensing algorithm eliminates this last step, significantly reducing the complexity of our proposal.

	Message exchange			Visibility
	Sensing	Collaboration	Adjustment	
Our proposal	2	n	No	Partial
Vivaldi based	> 2	$r \cdot (n + \#random\ nodes)$	Yes	Full
CoPing	g	$\#descendant\ nodes$	Yes	Full
Brute-Force	g	0	N/A	Full
PoTC	2	0	N/A	2
Single random	0	0	N/A	1

Table 5.1: Messages cost & gateway visibility at individual nodes per measurement round, g : number of gateways, n : number of close neighbors, r : number of repetitions of the message exchange in one measurement round.

Table 5.1 summarizes the characteristics of the different approaches.

For our *Collaborative layer algorithm*, the framework runs periodically every m minute, a value that can be adjusted depending on the dynamics of the network. Total communication overhead (gateway performance measurement, inter-node communication) of each node is

$$overhead_{comm} = r \cdot (overhead_{monitor} + overhead_{gossip}) \quad (5.3)$$

where r is the total number of rounds, $overhead_{monitor}$ is equal to 2 gateway performance measurements and $overhead_{gossip}$ is the cost of collaboration. The gateway performance measurement overhead adds load to the network as it consists in measuring the time needed to download a 0.1MB file from the Internet through the gateway nodes. Thus, every node creates 0.2MB upstream requests to the gateways in every round. The collaboration overhead is

$$overhead_{gossip} = message_{sent} + message_{receive} \quad (5.4)$$

where $message_{receive}$ is the total number of measurements received from all close neighbors, and $message_{sent}$ is the total number of measurements sent to the trusted neighbors. Obviously, these numbers depend on the network and the behavior of the other nodes. However, since the collaboration is kept within the close circle of neighbors, our algorithm is scalable and the communication overhead is relatively static.

Besides collaboration, the node does the following operations in the *Sensing layer* and the *Collaborative layer*: construct a gateway performance table, update the trust score of the neighbors, and update the capacity categorization of gateways. This process is done every time node receives measurements from the other neighbors. The calculations are simple and can be done even on resource-constrained devices. Space complexity for the gateway measurement log at each node is $O(k * m * 2)$ where k is the number of latest measurements retained at the node (for the capacity based monitoring; by default $k = 10$), and m is the number of neighbors sending measurements.

5.5.2 Experimental setup

We emulate a network with 100 client nodes and ten gateway nodes in Mininet v2.3 [LHM10]. We consider a balanced tree topology where each branch has ten client nodes and a custom tree topology where each branch has 5-15 randomly assigned nodes, as shown in Figures 5.6a and 5.6b. The RTT between client nodes is uniformly distributed between 5 and 20 ms, and the RTT between the gateway nodes and the client nodes ranges from 10 to 20 ms. The RTT between client nodes and the gateway nodes is determined from anonymous daily live gateway log files collected over a period of 1 year in the guifi.net community network [DMN17; Veg+15]. The gateways are randomly assigned with 0.3, 0.8, and 1.0 CPU capacity to create performance variations throughout all the presented experiments. The duration of a measurement round is set to two minutes, and the RTT-threshold to identify close neighbors in the *Sensing layer* is set to 10 ms. The resulting average number of close neighbors for each client node is 8 in the balanced tree topology and 10 in the custom tree topology. Each node chooses to collaborate with $TopK = 6$

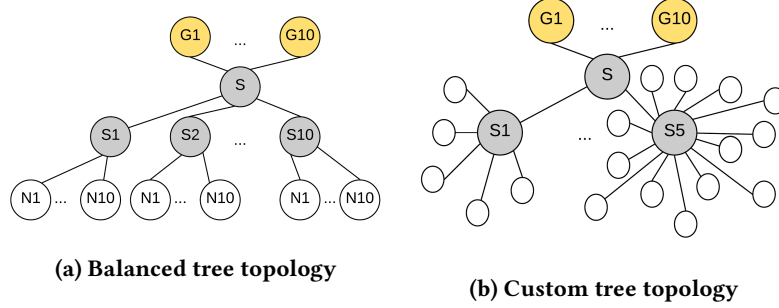


Figure 5.6: Network topology for experiments

trusted neighbor nodes to achieve full performance measurements of all gateway nodes.

We evaluate our framework by studying the effect of faulty nodes, resiliency, gateway performance change, network node failure, and sensitivity towards the framework parameters. Each experiment runs for 100 measurement rounds (i.e., 200 minutes), and experiments are repeated ten times for each topology.

5.5.2.1 Full collaboration vs top- K collaboration

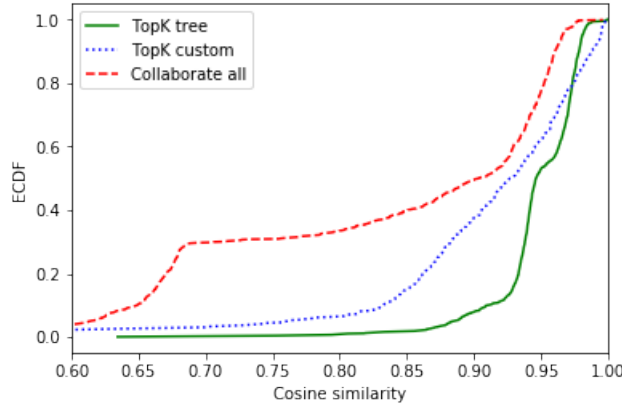


Figure 5.7: Similarity results for different collaboration schemes

We first compare the accuracy of the *TopK* neighbors' collaborative performance monitoring with collaborating with *all* the neighbors. Figure 5.7 shows the benefit of collaborating with more similar nodes using our trust-based approach instead of collaborating with a fixed number of close neighbors. Collaborating with all close neighbors (red, dashed line) gives an av-

erage 83% cosine similarity index of the constructed gateway measurement table. The top trusted collaborators for the balanced tree topology (green, straight line) result in an average similarity index of 94%. Under the custom tree topology (blue, dotted line), we observe that the cosine similarity index has an average value of 90%, showing a difference of 4% compared to the balanced tree topology. Based on these results, we argue that the Sense-Share framework provides the same quality of collaborative measurements in the two different network topologies with consistently accurate measurements.

5.5.2.2 Number of messages exchanged

In Table 5.2, we explore the average number of messages exchanged between *Sensing layer* and *Collaborative layer* at individual node per measurement round in the testbed of Figure 7.9b, with the close neighbors set containing an average of 15 close neighbors. For our algorithm, each round, the node senses the performance of 2 gateway and sends and receives two new gateway measurements from close neighbors. Meanwhile, the Vivaldi-based algorithm senses the majority of the gateways and all the neighbor nodes, then sends whole gateway table entries to the neighbor nodes and receives information from others. Table 5.2, as compared to Vivaldi based algorithm, shows that our algorithm reduces sensing overhead by 10x, message exchange between network nodes by 5x on average.

	Sensing layer	Collaborative layer	
		Send	Receive
Our algorithm	2	30	30
Vivaldi based algorithm	22	110	156

Table 5.2: Average message exchange at individual node per measurement round

5.5.2.3 Self-adaptation of the collaborative sensing

As mentioned earlier, the gateways in the large-scale network are susceptible to performance changes, including the scenario of a short-termed valley in the gateway performance and the effects of congestion. In Figure 5.8, we introduce a 2 seconds delay to one selected gateway after 100 rounds and remove the delay after 150 rounds (1 round = 2 minutes). We plot the estimated gateway performance perceived at our proposed solution (solid line) with the actual measurement of the node (dashed line) and Vivaldi-based algorithm estimation (dotted-dash line). The Vivaldi-based algorithm takes 30 rounds to adjust to the actual gateway performance change (note, Vivaldi

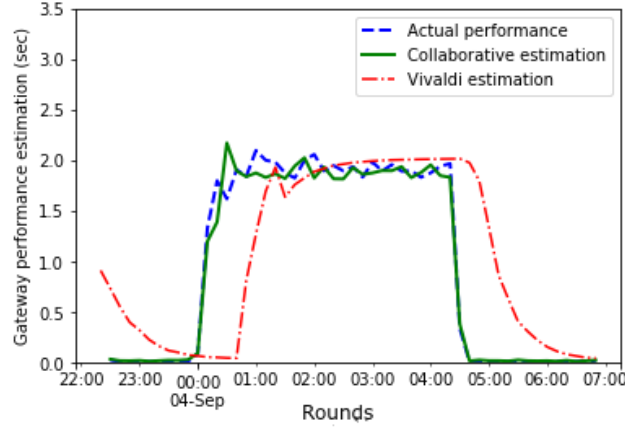


Figure 5.8: Gateway performance change (Collaborating with all)

parameters could be adjusted to become more reactive); comparatively, our proposed algorithm can sense the gateway performance increase after two rounds upon receiving the measurement from the nearest neighbor node and adjust its gateway table. When eliminating the delay, the Vivaldi-based algorithm takes 20 rounds to adjust to the normal performance change while our algorithm senses the performance change at the same round.

In our proposed algorithm, the adaptation period to any performance change depends on the probability of receiving the specific gateway performance information from the other nodes, resulting in faster convergence within a few rounds. On the other hand, the Vivaldi-based algorithm is slower to adapt to changes, moving towards the direction of the change, therefore not able to timely react to the gateway performance change.

Figure 5.9 plots the result of the gateway performance change adaptation when collaborating with trusted *TopK* neighbors versus actual latency. The introduced performance change on the gateway node is captured within 2-3 rounds. During the added delay period, the mean estimated latency follows the shape of the actual latency and stays very close to the latency change curve. When removing the *netem* delay at round 80, the nodes can adjust to the performance change within the same round. The performance change adaptation is consistent with collaborating with all neighbors where changes are captured in 0-3 rounds. In both Figure 5.8 and Figure 5.9, the collaborative gateway performance estimation is close to the actual gateway performance. Therefore, reducing the range of the collaboration holds the same high accurate performance estimation.

We simulate sudden performance changes on the node side, causing collaborator nodes to leave (link down) and join the network again (link up). As

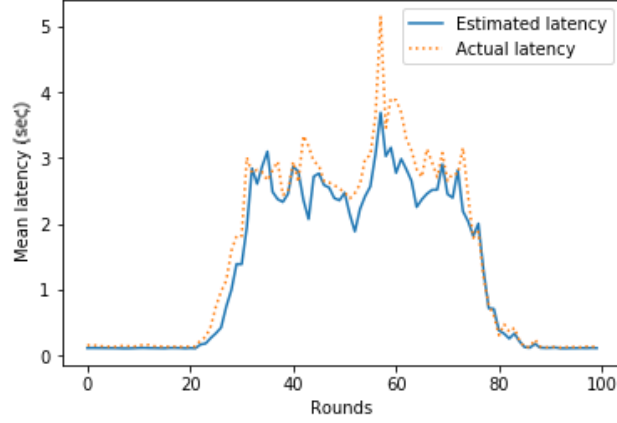


Figure 5.9: Gateway performance change (Collaborating with *TopK*)

depicted in Figure 5.10, at the 15th round, two nodes become offline. The *Collaborative layer* algorithm responds to the change at the 18th round, where the number of collaborators is decreased to 8. As a result of this change, the

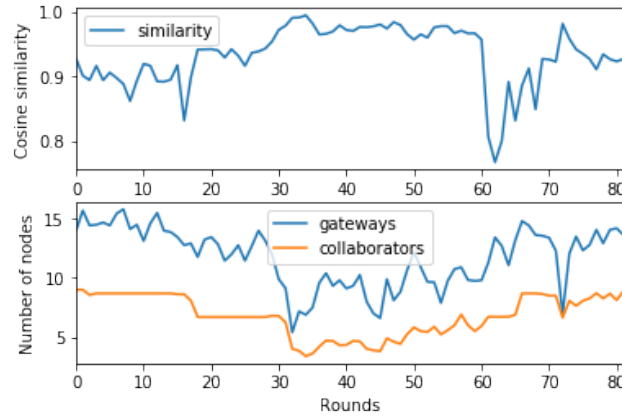


Figure 5.10: Collaborating node change

number of available gateways is decreased from 15 to 13, but the cosine similarity index of the gateway table is stable. During round 30, two other nodes become offline, and the total numbers of collaborating nodes and available gateways decrease to 6 and nine correspondingly. At the 45th round, two nodes become online again, and we can see corresponding changes in the number of gateways. Overall, we observe that the cosine similarity index has not been compromised by nodes leaving or joining the network. As soon as the node's link is down, the *Collaborative layer* sense the absence of the node

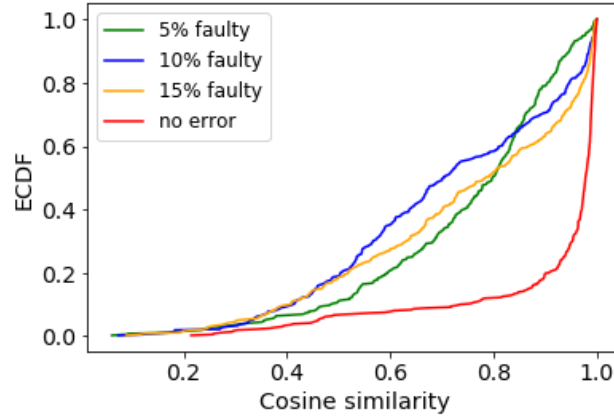


Figure 5.11: Faulty nodes effect (Collaborating with all)

within 1-2 rounds, while when the link is up again within the same round, as soon as the nodes send measurements.

5.5.2.4 Fault tolerance

We perform two different experiments to explore the fault tolerance of our proposal. We introduce faulty nodes (10%, 20%, and 30% faulty network nodes) that are programmed to send false measurement values (original value multiplied by 5 or 10 randomly) to their collaborators in random periods.

Figure 5.11 exemplifies the impact of faulty nodes on the collaborative sensing algorithm when collaborating with all close neighbors. For different ratios of faulty nodes in the network, the ECDF of the accuracy of the sensing results. As can be seen, the accuracy decreases significantly even if the number of faulty nodes is small.

Figure 5.12 depicts the ECDF of the collaboratively achieved gateway performance table's cosine similarity result. As observed, our trust-based filtering component rules out the faulty nodes and continues collaborating with the other similar (trusted) nodes in the neighbor list. As depicted in Figure 5.12, the average cosine similarity index is reduced from 95% on average to 93%, 92%, and 90% as the number of faulty collaborators increases. This slight decrease of the similarity index results from our approach where each node chooses to collaborate with less similar nodes.

In the second experiment, we study the effect of smoothing non-faulty peaks and valleys applying a moving average. During the experiment, a node receives from the random trusted neighbor short-burst of high measurement where the original measurement values multiplied by a factor of 5 to a randomly chosen neighbor, measuring the effect on cosine similarity of the con-

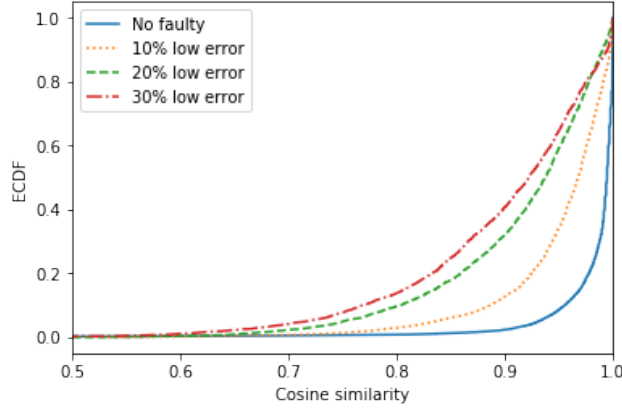


Figure 5.12: Faulty nodes effect (Collaborating with $TopK$)

structured gateway performance table. In the results of the experiment presented in Figure 5.13, we observe approximately 10 points in time during the experiment where the node receives high values. During that time, the similarity of the measurements stays relatively stable, maintaining 92% of the cosine similarity index on average. The results of the two experiments pre-

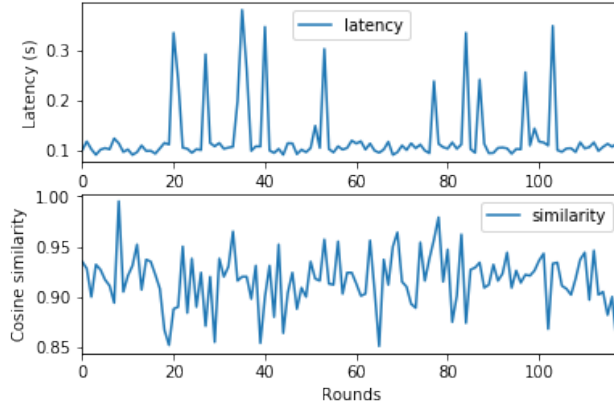


Figure 5.13: Behavior of the faulty nodes

sented above show that our proposal accomplishes the requirement **R5** of providing resilience towards faulty behavior.

5.5.2.5 Sensitivity analysis

In this section, we present our study concerning the sensitivity of the proposed framework in terms of the number of available gateways, the close

neighbor sensing threshold, and the number of randomized sampling, which are the parameters that affect the accuracy of the proposal.

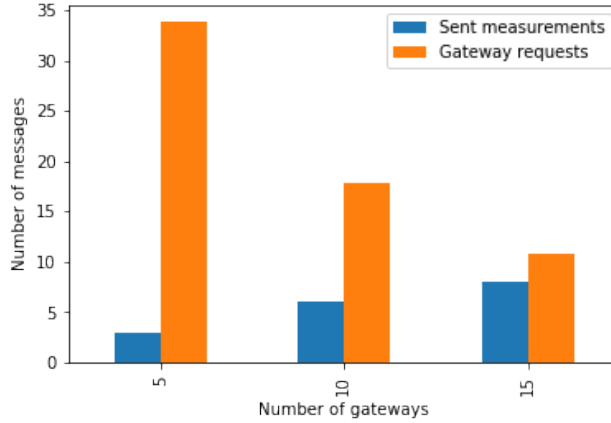


Figure 5.14: Sensitivity towards the number of gateway nodes

Figure 5.14 shows the number of measurements sent from each client node and the average number of the measurement requests received at ten gateways. The average number of close neighbors for each client node is 8, while we test 5, 10, and 15 available gateways accessible by each client node. For five gateways available at each node, the average number of collaborating neighbors is 3, and the number of measurement requests at each gateway node is 33 on average. When increasing the number of available gateways to 10, the number of collaborating neighbors becomes average 6, and the number of measurement requests is reduced to 18. When the number of gateways increased to 15, the number of collaborating neighbors increases to 8, and the number of measurement requests are 11 on average at each node. The number of gateway performance measurement requests at each gateway, on average, decreases as the number of gateways increases since performance measurement requests are distributed over the client nodes. The requirement **R2** is achieved through the result presented in Figure 5.14 that our framework scales the number of collaborations with the demand of the increased number of gateway nodes.

On the other hand, when there are not enough neighbors to provide full visibility of the performance, the algorithm could either 1) increase the close neighbor sensing RTT threshold value to create a wider range of collaboration or 2) increase the number of samples from each client node to sense. Increasing the close neighbor sensing threshold results in receiving less accurate collaborative measurements at each node with the expense of reducing the cost of measurements shown in Table 5.3. As the RTT threshold increases, the sim-

	No of samples	Cosine similarity	Standard deviation
RTT<5ms	3	0.95	0.01
RTT<15ms	2	0.91	0.02
RTT<20ms	2	0.875	0.035

Table 5.3: Sensitivity towards the RTT threshold vs increasing the number of samples

ilarity of the performance measurements received from the other nodes decreased to 91% and 87.5% shown in Table 5.3. The performance measurements received from the other nodes contain more variations increasing, therefore, the standard deviation value.

5.5.2.6 The reaction towards nodes joining/leaving collaborative monitoring

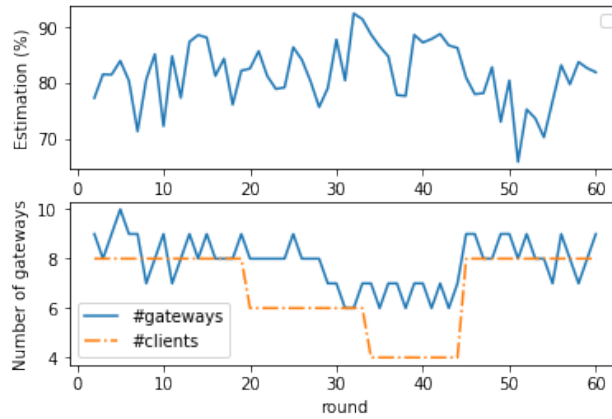


Figure 5.15: Results of nodes leaving/joining

Figure 5.15 shows the effect of the number of nodes in the close neighbors set leaving (decrease) and joining (increase). The experiment runs with the average close neighbors set the size of 8 (RTT<5ms) and ten gateways. Each measurement round, the number of measured gateways are different depending on what gateways the nodes in the close neighbors set are sampling. In round 20, we remove two client nodes from the experiment. However, the size of the measured gateways and precision of estimation is not affected. We further remove two clients at the 32nd round, then the number of measured gateways decreased, but the precision of estimation remains the same. We observe that the ideal number of the nodes in the close neighbors set should be around $(available_gateways)/2$ to provide enough gateway measurements.

5.5.2.7 Download latency

We compared the collaborative and non-collaborative algorithms with our proposed Collaborative-Best and Collaborative-Fair algorithms. Figure 5.16 shows the Empirical Cumulative Distribution Function (ECDF) of downloading 0.1 MB file from the selected gateway for each of the studied algorithms. In the result, the PoTC algorithm sets the lower bound, where the gateway performance table consists of each node's own two measurements. On the other hand, the Brute-Force gateway selection algorithm sets the upper bound, where the node selects the best gateway at the time of selection out of all gateway nodes' performance. As shown by non-collaborative algorithms in Figure 5.16, both our proposed gateway selection algorithms outperform the worst-case scenario 60% of the time. Moreover, the Collaborative-Best gateway selection performs slightly slower (0.1ms slower) than the Brute-Force gateway selection. Among the collaborative algorithms in Figure 5.16, the Vivaldi-based algorithm performs better than our selection algorithms. Brute-Force and Vivaldi-based algorithms perform better than our proposed selection algorithm because they offer full visibility of the gateway nodes at the cost of more management overhead (more messages and computation). In comparison, our algorithm achieves partial visibility with less management overhead.

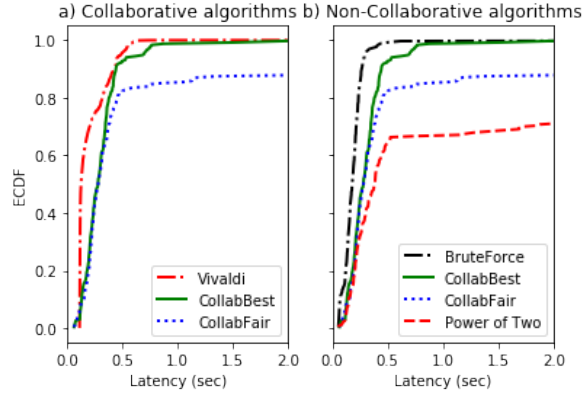


Figure 5.16: Gateway selection result

5.6 Conclusion

Collaborative layer is a gateway monitoring algorithm based on the close neighbor's collaboration, a simple collaborative monitoring algorithm tailored for each client node. The similarity-based close neighbors collaborative sensing reduces the gateway performance monitoring overhead and the

inter-node communication dramatically. Experiments show that the accuracy of collaborative monitoring is very high (94% on average) despite the network topology. The proposed algorithm is resilient towards the faulty collaborators by limiting the collaboration with more trusted nodes and adapts to any performance changes within the first 1-3 rounds of the sensing without compromising the quality of the collaborative measurement.

At the end of the chapter, we also proposed two selection algorithms, Collaborative-Best and Collaborative-Fair. Both algorithms provide good gateway selection, but the average download latency experiment result is not better than the Brute-Force algorithm. In the *Selection layer*, we tackle the challenge of achieving as good download latency as the Brute-Force algorithm while also achieving good distribution and stable gateway selection.

Selection layer

| 6

6.1 Introduction

The Internet gateway selection problem is becoming very important as the number of Internet-connected devices increases and stresses the limited number of Internet gateway nodes. The gateway nodes often experience frequent performance fluctuations, and the best gateway selection candidate frequently changes with growing network dynamics. Consequently, choosing the “right” gateway can improve the Quality of Service perceived at the client node.

As explained, state-of-the-art algorithms for gateway selection are often based on monitoring the performance or state of the available gateways to help client nodes make an informed decision. Many monitoring algorithms use an active monitoring approach where nodes periodically probe the available gateways to keep up with the network dynamics. The algorithms aim to provide the clients with an accurate, up-to-date view on the gateway performance to enable them to select the best performing [Bat+19a; Dim+17b; Ko+13; Liv+10], the least-loaded [HLT04; AAJ09b; Yan+13; Del+12] or the nearest gateway [XJJ15; Xu+19]. The idea behind the above algorithms is that client nodes should aim to select the gateway node that is the best for them, be it in terms of performance, load, or distance. However, from a network point of view, this is not efficient. Selecting the best gateway means that similar or “nearly as good” gateways are ignored, potentially resulting in load imbalances and frequent changes in the network.

In the following, we introduce *GateSelect*, a client-side distributed gateway selection method for the *Selection layer* of our framework. The algorithm categorizes available gateway nodes down to the potential good candidates based on their ability to provide good, stable performance for the future and then selects the gateway from the candidate list with minimal selection bias. We compare the proposed method with a greedy gateway selection, random (load-balanced) gateway selection, and static gateway selection in different network scenarios. The experimental results show that our method results in long-term stable Internet connections and balances the overall gateway allocation.

6.2 Design requirements

For cost or security reasons, the number of gateway nodes is typically much smaller than the number of client nodes. Use cases for such networks are large IoT installations or open community-driven networks like `guifi.net` [guifi]. In such scenarios, it is not desired to have a fixed assignment of client nodes to specific gateways. Instead, a client node should be dynamically assigned to a gateway depending on network conditions and gateway loads. The goal of the gateway selection is to map the client nodes to those gateway nodes that provide long-term Internet connectivity with good performance. In this context, *good* performance means that the gateway provides better long-term performance than the average at a given time. Otherwise, its performance is called *bad*. Depending on the users' goals or the network operator, the performance can be defined in different ways, such as latency, achievable throughput, and more. The instantaneous performance of a gateway as perceived by a client node can vary over time; it not only depends on the capacity and load of the gateway itself but also on external and internal factors such as the quality of the connections of the gateway to the Internet or the network condition inside the client network. Every node's gateway selection is to choose a good gateway, but the following problems occur when each client selects its locally optimal gateway.

6.2.1 Herd behavior

Selection algorithms where a client selects its locally optimal gateway are considered selfish since they ignore the impact of the selection on the other nodes. Performance-oriented selection algorithms select the gateway with the currently best performance [Ko+13; Dim+17b; Bat+19a]. Load-oriented algorithms choose the gateway with the lowest load [HLT04; Nan+06; KBM12]. Between those two extremes, the hybrid selection algorithms [Abu+15; AAJ09a] combine both metrics. The main problem in choosing the locally optimal gateway at the time of selection is that other nodes choose the same gateway node. We have called this situation *herd behavior* in Chapter 1. The result is that the selected gateway becomes overloaded, and its performance degrades.

6.2.2 Frequent gateway selection changes

Typically, the decision of a gateway selection algorithm is not static. Many algorithms proposed in the literature are based on continuous active monitoring of the gateways, where the selection is periodically reevaluated (rounds) based on the measurement results [Dim+17b; Ko+13; Abu+15; GRS08]. In large-scale networks, the gateway performance as perceived by the clients will vary over time due to changes in network conditions, the appearance

or disappearance of client nodes, or simply the activity of the client nodes. For a selection algorithm aiming at finding the locally optimal gateway, this would mean changing the selected gateway in every round. Not only would this create an imbalance in the gateway allocation, but it would also force each client node to find a new route to its selected gateway repeatedly. As a result, locally, each node experiences short-term performance degradation while waiting for the new route, and globally, within the client network, additional management traffic is generated due to frequent rerouting and mass migration.

6.2.3 Instability of gateway performance

Here, stability means that the gateway selected by a client node should exhibit good performance over time. Heterogeneous networks contain gateways with different capacities, some capable of handling many client nodes, others only a few. Gateways with limited capacity might show good performance at a time, but a slight increase in demand, e.g., new clients joining or existing clients increasing their network activity, would cause a performance drop. The current performance and load of a gateway, therefore, do not necessarily indicate its future performance.

6.2.4 Requirements

The goal of gateway selection is to select the gateway that provides a good Internet connection. We focus on the following requirements:

1. **Individual selection decision** - Nodes perform their selection without coordinating the selection decision with the other nodes or gateways. Adding coordination means extra communication, scheduling among the nodes for the load balancing, which could be costly in terms of the number of messages and additional processing.
2. **Stable gateway selection** - Unless the performance of the selected gateway deteriorates too much, a change of the selection should be avoided. Client nodes should give gateways with similarly good performance a chance to get selected.
3. **Load balancing** - A node's decision should not negatively influence the overall stability of the gateway distribution as well as the performance perceived by the other client nodes.
4. **Minimum knowledge** - Client nodes do not know the network topology or the specifications of the gateways. In particular, the latter means that the clients do not know how many clients a specific gateway can handle, nor can they directly query its current load.

	G_1	G_2	G_3	G_4	...	G_{M-1}	G_M
t_1		0.8	0.6		...	0.9	
t_2	0.7			1.3	...	0.8	0.4
...	...	...	...	...	...	...	
t_K	0.4	0.8		0.2	...		0.5

Table 6.1: Partial gateway performance table, every round,

6.3 Design overview

From the *Sensing layer* and *Collaborative layer*, the node is equipped with a gateway performance table, as illustrated in Table 6.1. It is important to note that, as visible in Table 6.1, measurements are not complete, i.e., a client node does not measure every gateway in every round. Doing so would result in unacceptable overhead in the network. Instead, different strategies can reduce the number of measurements, such as random sampling or the usage of dedicated measurement nodes that perform measurements and distribute the results to the other nodes. In previous chapters, we proposed a collaborative scheme where each client node would measure the performance of a randomly selected subset of gateways and share the results with its neighbor nodes, assuming that close neighbors have a similar perception of the gateway performance. As a result, the list of gateways to choose from varies in each round.

The aim of the *Selection layer* is to find a gateway for each client node that provides good performance for a long time. We call such a gateway a *stable gateway*. The proposed algorithm's structure is shown in Figure 6.1. Its main advantage is that the selection is performed locally by each node, without specific knowledge of the network topology or the decisions of the other nodes. The algorithm runs periodically and consists of 3 steps:

1. Build an initial list of candidate gateways by classifying them based on their performance;
2. Increase the number of candidates by filling the gaps in the partial performance table;
3. Select a gateway to obtain (a) a good Internet connection for the client and (b) a balanced gateway distribution, i.e., a nearly equal number of nodes per good performing gateway.

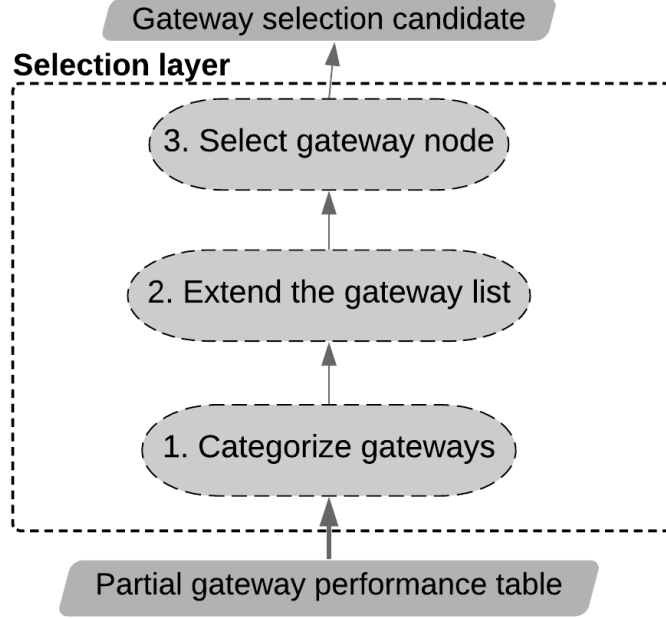


Figure 6.1: Selection algorithm design

Performance	Deviation	Perf. category
$M_{tj} \leq \mu$	$stdev(M_j, k) \leq stdev(M)$	Good
$M_{tj} \leq \mu$	$stdev(M_j, k) > stdev(M)$	Inconsistent
$M_{tj} > \mu$	$stdev(M_j, k) \leq stdev(M)$	Inconsistent
$M_{tj} > \mu$	$stdev(M_j, k) > stdev(M)$	Bad

Table 6.2: Gateway performance categories

6.3.1 Gateway capacity estimation - Classification

This step's goal is to identify gateways that consistently provide good performance. For this, we extend our previous *Sensing layer's* performance-based categorization. The node already assigned a *performance category* to the gateways following the rules in Table 6.2 (copied for the convenience of the reader from Table 4.1). The gateways are categorized into *Good*, *Inconsistent* and *Bad* performing gateways.

We propose to further classify the gateways according to the evolution of their performance category over time. Again, we use a time window of the last k rounds. The possible classes are shown in Figure 6.2. If a gateway's performance is categorized as *Good* in the majority of the last k rounds, it is

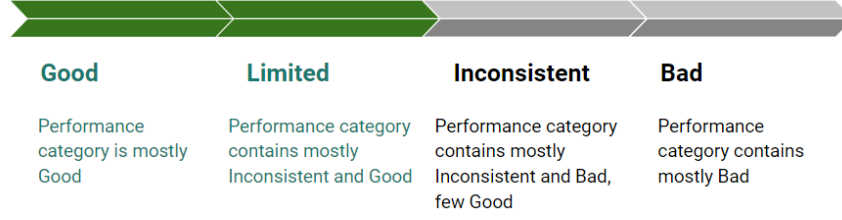


Figure 6.2: Estimated gateway capacity classification

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
t-10	M(t-10,1)	M(t-10,2)	M(t-10,3)		M(t-10,5)		M(t-10,7)	M(t-10,8)	M(t-10,9)	M(t-10,10)
...	...	...	...	...	...	...	...	...	...	...
t-3		M(t-3,2)	M(t-3,3)	M(t-3,4)	M(t-3,5)	M(t-3,6)	M(t-3,7)	M(t-3,8)		
t-2	M(t-2,1)	M(t-2,2)	M(t-2,3)	M(t-2,4)	M(t-2,5)	M(t-2,6)	M(t-2,7)	M(t-2,8)	M(t-2,9)	M(t-2,10)
t-1	M(t-1,1)	M(t-1,2)	M(t-1,3)	M(t-1,4)	M(t-1,5)	M(t-1,6)	M(t-1,7)		M(t-1,9)	M(t-1,10)
t	M(t,1)	M(t,2)	?	?	M(t,5)	M(t,6)	M(t,7)	M(t,8)	?	M(t,10)

Table 6.3: Example gateway performance table after Step 1

classified as a *Good* gateway and assumed to provide stable and good performance in the future. If it is categorized mostly as *Inconsistent* with a few cases of *Good*, then the algorithm classifies it as a gateway of *Limited* capacity. The classification is motivated by the idea that gateways with high capacity are less sensitive to performance fluctuations. Similar reasoning is used to classify gateways as *Inconsistent* and *Bad*. *Inconsistent* and *Bad* gateways are not considered as suitable candidates for the gateway selection.

Table 6.3 shows a possible outcome of the classification step (example data). In the table, the entries with *Inconsistent* and *Bad* classification are shown with a grey background. In the latest round t , the gateways $G3$, $G4$, and $G9$ are not classified due to missing measurements.

6.3.2 Extend the candidate list

The first step reduces the number of viable selection candidates (gateways) to a small list of *Good* and *Limited* category gateways. However, in this step, we can use previously collected performance measurements to fill the gap. To extend the list of candidates, we propose a performance prediction step to fill the missing entries of the current round in the performance table. The prediction uses the last k measurements. Since we are only interested in *Good* and *Limited* gateways, we only perform the prediction for gateways with a missing measurement in the current round that was classified as *Good* or *Limited* in the previous round. We then repeat step 1 for those gateways. In our example (Table 6.3), this is done for $G3$ and $G4$ since they were classified *Good*

gateways in previous rounds and increases the number of candidates from 4 to 6.

Figure 6.3 shows empirical measurements for three gateways from the real large-scale community network guifi.net. Measurements are performed every 2 minutes. The top plot shows the timeseries of the latency of downloading a file of 0.1MB from the Internet with the selected gateway. The bottom plot shows the ECDF of the standard deviation of the latency for $k = 10$. Since we only perform this step on gateways classified as relatively stable in the previous round, we can expect that the prediction provides reasonably good estimations of the gateways' current performance. Gateways *gw1* and *gw2* are stable, and we consider *gw3* an unstable gateway as its performance varies significantly over time.

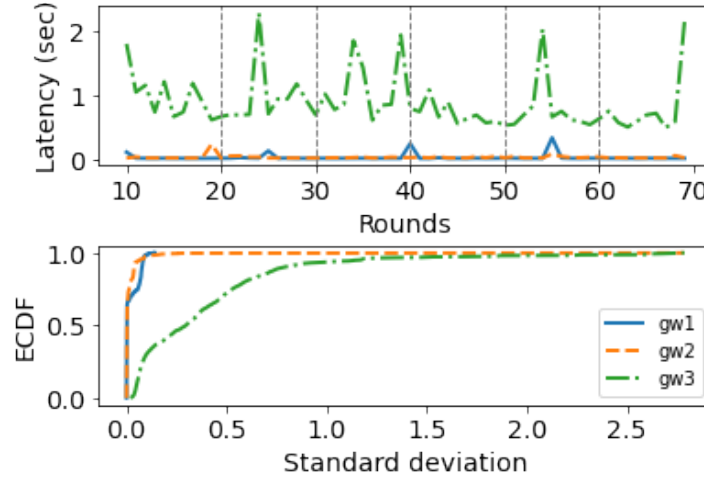


Figure 6.3: Gateway latency variance

We use Linear Regression (LR) with $y = \beta_0 + \beta_1 X + \epsilon$ to predict the performance y of a gateway from its past k measurements X . β_0 is the intercept term, β_1 is the slope of the line, and ϵ is the error.

6.3.3 Gateway node selection

Every node wants to achieve the best Internet performance through the gateway nodes, so selecting the best performing gateway is natural. As explained earlier, this is not always desirable. To limit this selfish behavior, we propose restricting the node's ability to select the best gateway. After Steps 1 and 2, the gateway selection candidates narrowed down to *Good* and *Limited* nodes. Therefore, any of these gateways provide good performance for the node.

Algorithm 5 shows how the gateway is selected from the list of candidates.

Algorithm 5 Gateway selection**Input:** *gateway_candidates***Input:** *current_gateway***Input:** $D = 2$

```

1: procedure SELECTGATEWAY
2:   if current_gateway  $\in$  gateway_candidates then
3:     return current_gateway
4:   good_gateways = { $G \in \text{gateways\_candidates} \mid G \text{ is Good}$ }
5:   if SIZE(good_gateways) <  $D$  then
6:     good_gateways = gateway_candidates
7:   candidates = RANDOM(good_gateways,  $D$ )
8:   current_gateway = BEST(candidates)
9:   return current_gateway

```

The algorithm does not change the gateway if the currently selected gateway of the client node is among the candidates (lines 2-3). In this way, frequent changes are avoided unless the current gateway's performance deteriorates such that it is eliminated from the list of candidates.

As nodes are oblivious to the decisions of their peers, it is not easy to ensure a fair distribution of the clients over the good gateways. We incorporate a randomized load balancing to reduce the selection bias. The algorithm randomly samples D candidates from the list (line 7) and selects the best-performing one (line 8). By default, the random sample size is $D = 2$ by using the Power of Two Choices (PoTC) algorithm [Mit01]. Many network load balancing [Zha+17b; Gho+17; Ber+17] and job scheduling [Ous+13] algorithms use PoTC for balanced resource distribution. The *Good* gateways are given priority for the selection (line 4). However, if the number of good gateways is too small (line 5), the algorithm also considers the *Limited* gateways (line 6).

6.4 Evaluation

6.4.1 Experimental setup

We test the *Selection layer* algorithm in a simulated network with 40 client nodes and ten gateway nodes. We test three different client network configurations: wired, wireless, and wireless mobile networks. Gateway performance is measured every round by recording the time clients need to download a 0.1MB file through the gateways from an HTTP Server connected to the gateways through wired links. The client nodes also download a 1MB file

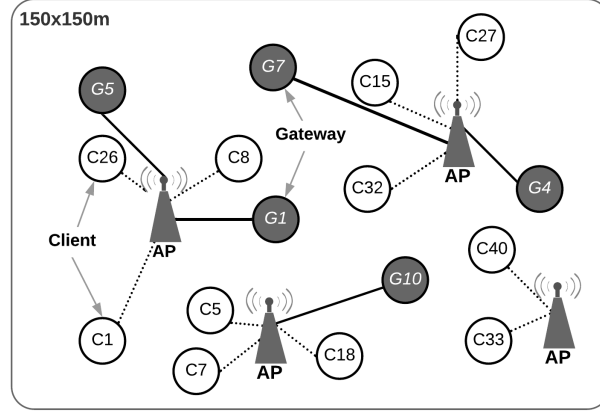


Figure 6.4: Experiment network topology.

through their selected gateways every minute to mimic client activities.

In the *wired* configuration, we use MiniNet [LHM10] to simulate a wired client network where every ten nodes connect to one switch. Random link delays between 1ms and 3ms (chosen at the beginning of each simulation run) are set for all links between nodes and switches, and a fixed 5ms delay is set for the links between two switches. In addition, gateways add a random delay between 1ms and 3ms to each response.

For the *wireless* configuration, we use Mininet-Wifi [Fon+15] to simulate a wireless IEEE 802.11 network in a 150m×150m area with the network topology shown in Figure 6.4. We place the client nodes at random positions and four access points (AP) at fixed locations. The gateways are connected to the APs by a wired link with a delay between 1 - 3ms (chosen at the beginning of the simulation run). Again, a random delay of 1ms to 3ms is added dynamically when a client downloads a file. The wireless interference model is provided by `mac80211_hwsim` in Mininet-Wifi. The transmission range is 250m.

Finally, we also simulate a *wireless mobile* configuration that is identical to the above wireless configuration, but client nodes move according to a Random Walk mobility model with 1m/s and connect to the nearest access point.

We compare four different selection algorithms:

1. GateSelect: Our algorithm;
2. Greedy (Brute-Force) selection: Clients always select the best gateway every round;

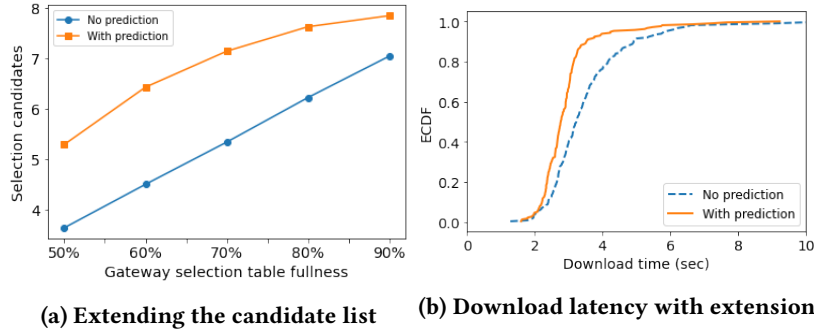


Figure 6.5: Wireless - Performance prediction result.

3. Random selection (Power of 2 choices): Clients choose two random gateways and select the best every round;
4. Static selection: Clients select the best gateway at the beginning of the experiment and never change the selection unless the gateway becomes unresponsive.

Experiments are repeated five times with different random positions and delays, and each experiment runs for 100 selection rounds, where a selection round has a duration of two minutes.

6.4.2 Results

6.4.2.1 Extend the candidate list (Prediction of missing values)

In our first experiment, we test the impact of Step 2 of the GateSelect algorithm on the size of the candidate list. To this end, we run "Step 1" of the *Selection layer* with gateway selection tables filled to 50%, 60%, 70%, 80%, and 90% with gateway measurements at any selection round. Figure 6.5a shows the average resulting number of candidates at a node without (i.e., after Step 1) and with (i.e., after Step 2) predictions for the wireless network. Without prediction, the number of candidates shrinks down to 3-4 candidates on average for gateway performance tables filled to 50%. The number of candidates increases with the degree of filling. When the prediction step is added, the number of candidates increases by 10 to 20%, depending on the degree of filling.

We quantify the impact of the prediction on the overall system performance by measuring how much time the clients need to download the 1MB file through their selected gateways. Figure 6.5b shows the results when we run the algorithm with and without the prediction step (Step 2) for the same

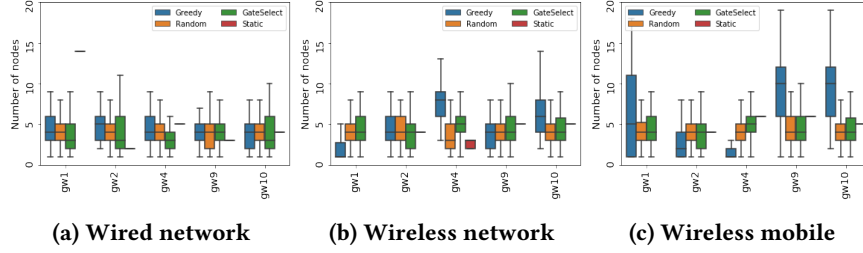


Figure 6.6: Client node distribution over the gateway nodes.

network settings. We observe that the download time improves with prediction because the prediction step increases the number of selection candidates for each node. As a result, the traffic load is distributed more evenly over the gateways.

6.4.2.2 Gateway selection distribution

We compare the distribution of the client node over the gateways for the different selection algorithms. To this end, we calculate the average distribution of the clients after a single simulation (100 rounds). For all three network configurations, identical start positions were chosen for the nodes. Figure 6.6a shows the obtained distribution for the wired network. The gateway distribution is similar for GateSelect and the Greedy and Random selection algorithms. The 40 client nodes are more or less evenly distributed over the ten gateways, i.e., four clients per gateway. This is because, in the wired network, the impact of the variances of the link delays is very small. Therefore all gateways have an equal chance of being selected. In the static selection, we see an unequal distribution due to random effects when the clients chose their gateway at the beginning of the simulation.

For the wireless network without mobility (Figure 6.6b), Greedy selection shows a more uneven distribution with higher variances for several gateways than the other algorithms due to the herd behavior. These variances are increased in the wireless network with mobility (Figure 6.6c) where the movements of the client nodes cause the Greedy algorithm to change the selection frequently. In general, GateSelect behaves like the Random selection, which is expected since both algorithms use PoTC to achieve load balancing.

Figure 6.7 shows the overall average distribution of client nodes per gateway per round in the wireless and wireless mobile network. The simulation runs are repeated five times with random positions of nodes. The results show that both networks behave similarly. In most rounds, the average number of clients per gateway is between 2 and 5, which corresponds to a relatively balanced distribution of the clients over the gateways. At the beginning of each

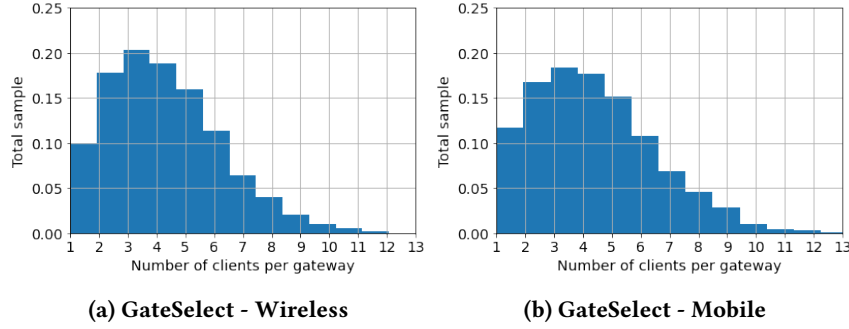


Figure 6.7: Wireless gateway distribution

simulation run, network nodes only have a limited number of gateway performance measurements. Therefore, there is a small fraction of rounds with a higher number of clients for some gateways.

6.4.2.3 Gateway selection changes

Figure 6.8 shows the average number of gateway selection changes over 100 rounds for the different selection algorithms. The simulations were run five times for each network and selection combination with different random positions and link delays. Greedy and Random selection algorithms change the default gateway significantly more frequently than our approach in all three network configurations since they always try to find the best gateway among all gateways (Greedy) or two randomly chosen gateways (Random). GateSelect reduces the number of changes by a factor of four. The static selection performs, by design, the smallest number of selections. It only changes the selected gateway if it becomes unresponsive.

In the wired network, gateway changes are slightly less frequent because the links are stable, and the wireless network is noisy; thus, more selection changes occur. The wireless mobile network shows fewer changes because the gateways are within the access range of every node, and depending on the mobility direction, the performance of the selected gateway stays stable for some nodes.

6.4.2.4 Gateway selection performance

We measure the effectiveness of the gateway selection by how long it takes a client node on average to load content (1MB file) through the selected gateway. The results after five simulation runs are shown in Figure 6.9.

In the wired network (Figure 6.9a), the download time of GateSelect is less than 0.09 seconds on average and is better than that of the Greedy, Static se-

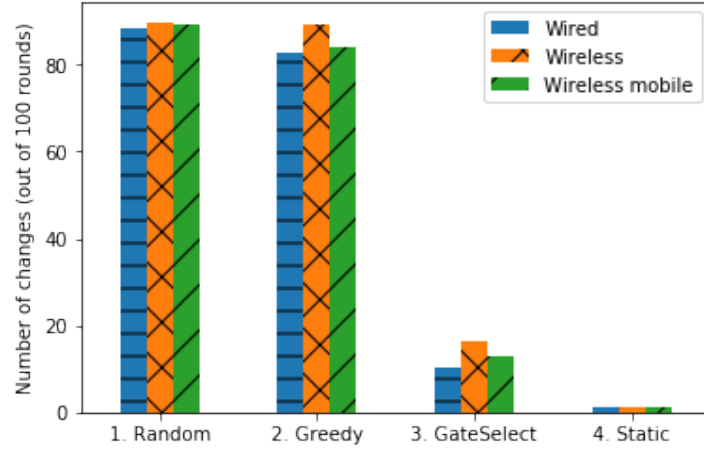


Figure 6.8: Number of gateway selection changes in 100 rounds.

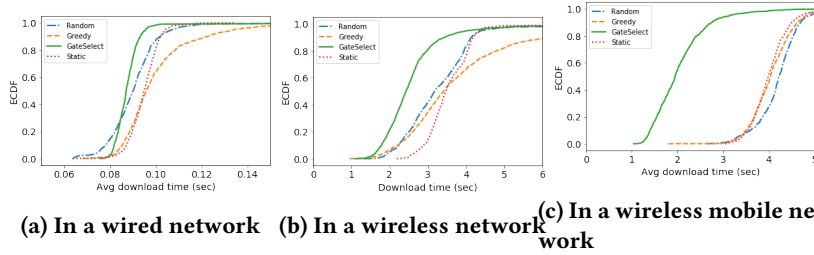


Figure 6.9: Download time through the gateway nodes.

lection algorithms for 90% of the downloads. GateSelect provides 70% better download time than Random selection algorithm, because there are chances where Random selection proposes the best selection candidate. Other compared algorithms' results show performance close to our proposal due to the stability of the links in the wired network. In wireless networks without mobility (Figure 6.9b), GateSelect offers a significantly better download time for the majority of the downloads compared to the other algorithms. The Greedy algorithm's download time is the worst due to herd behavior, resulting in overloaded gateways. The advantage of our approach is even clearer in wireless networks with mobility (Figure 6.9c). The download time stays below 2 seconds in 80% of the downloads because fewer selection changes and mobile nodes do not have to find a new route to the new gateway. The other algorithms show download times between 4 to 5 seconds.

Overall, GateSelect shows better download times than the others by combining a conservative approach to selection changes and balancing the load.

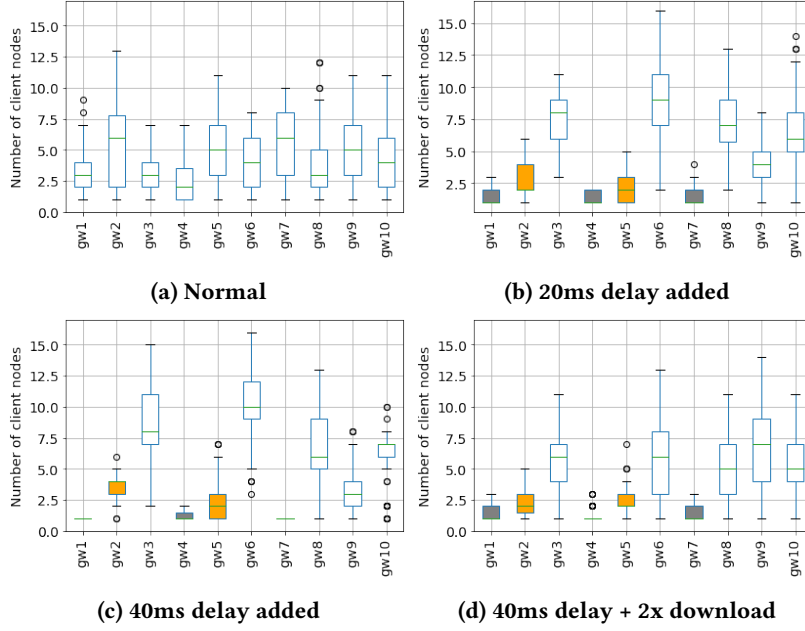


Figure 6.10: Wireless - Client node distribution with gateway delay.

6.4.3 Sensitivity analysis

In the following experiments, we study the sensitivity of the GateSelect algorithm to its parameters and the simulated scenario. We use the wireless network configuration with client nodes equally distributed over the 150x150m area at fixed positions to better show the impact of the different parameters. We show the results of single experiment runs.

6.4.3.1 Link performance changes

This experiment aims to see if GateSelect can correctly identify the capacity of the gateways and distribute the nodes accordingly. The normal scenario is shown in Figure 6.10a. On average, 4-5 nodes are assigned per gateway per round.

Next, we assign a fixed 20ms delay to gateways *gw1*, *gw4*, and *gw7* and a random 5ms to 10ms delay to *gw2* and *gw5*. As a result, the client nodes tend to avoid selecting those gateways (Figure 6.10b). If we further increase the latency for *gw1*, *gw4*, and *gw7* to 40ms, this effect is intensified (Figure 6.10c). Consequently, the other gateways have to serve more client nodes. However, if we increase the download frequency by a factor of two while keeping the 40ms latency, the distribution becomes again more even, as shown in Figure 6.10d. As download frequency increases, the performance of the normal

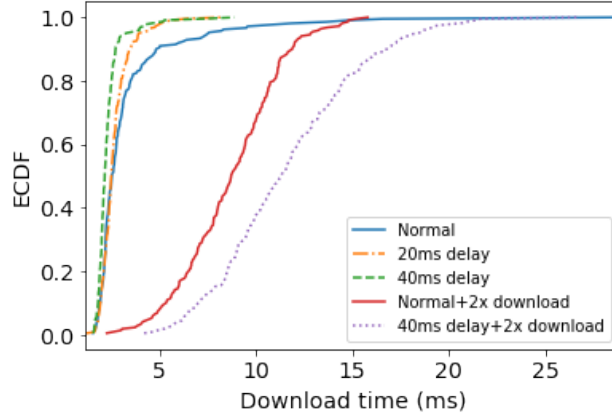


Figure 6.11: Download times with different gateway delays

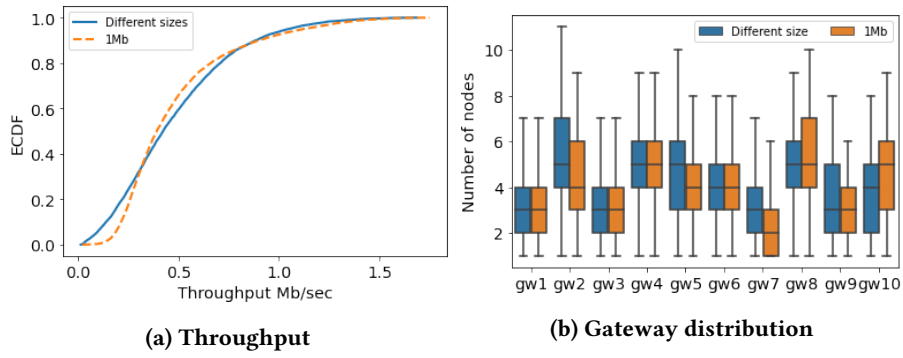


Figure 6.12: Wireless - Download size sensitivity.

gateways degrades as well. As a result, the high-latency gateways become more attractive again and are selected by some nodes.

Figure 6.11 shows the download times with the different link delays. As the link delay increases, the download time increases because every client's gateway selection pool is narrowed down to a few good gateway nodes. When increasing the download frequency, the download time increases due to queuing delays and network congestion. We conclude that GateSelect can correctly detect the capacity of the gateways based on the measurements collected over the selection rounds and allocates the nodes evenly to the good gateways.

6.4.3.2 Varying download file size

In this experiment, the nodes download files of different random sizes (0.25MB, 0.5MB, 1MB, 3MB, 5MB) instead of the 1MB file used in the other experi-

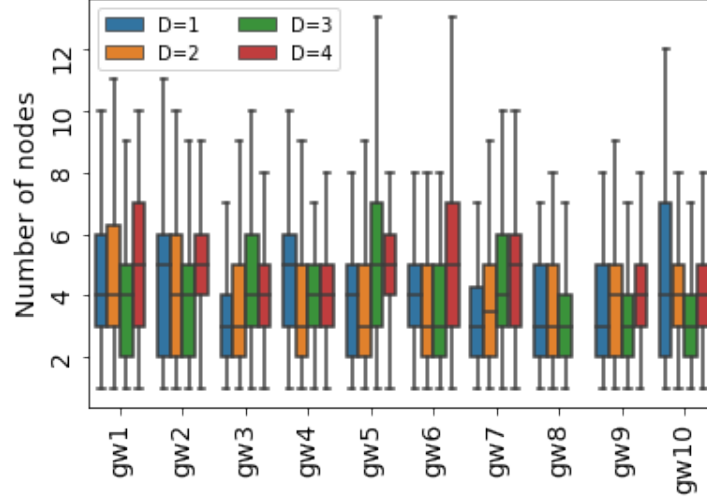


Figure 6.13: Wireless - Gateway distribution.

ments. Figure 6.12a shows that the achieved throughput per download is slightly lower than with the 1MB file. The gateway distribution (Figure 6.12b) is nearly identical for both scenarios. We conclude that the gateway selection is not affected by the traffic load.

6.4.3.3 Power of D choices for the selection decision

By default, GateSelect selects $D = 2$ gateways randomly from the selection candidate list, but this number is configurable. We run experiments with $D = [1, 2, 3, 4]$. As shown in Figure 6.13, when D is lower, the gateways are distributed equally over the nodes. However, the greater D , the higher the probability that many clients will select the same gateway. For $D = 3$, this is the case for *gw3* and *gw5*, and for $D = 4$, *gw1*, *gw5*, and *gw6* are in high demand. We conclude that $D = 2$ is a good choice.

6.5 Conclusion

We have presented our *Selection layer* algorithm called GateSelect, a distributed gateway selection algorithm tailored to fulfill each client node's Internet gateway selection using performance-based classification and random sample selection. GateSelect is simple yet effective, infrastructure- and technology-agnostic, and supports incremental implementation. We have demonstrated GateSelect's adaptability in different network scenarios and compared the performance with a greedy, random, and static selection to show improved Internet gateway performance while maintaining gateway selection balance.

Production network integration

7

In the previous chapters we described the individual layers of the Sense-Share-Select framework and presented experiments in the emulated network environment. The Internet gateways did not have background traffic in the emulated network experiments; thus, the behavior of the gateways is stable, predictable. However, the production network gateways experience different traffic patterns, loads, and requests regularly. Therefore, this chapter aims to test the Sense-Share-Select framework in the real large-scale network environment. We implemented the framework in the guifi.net community network environment with guifi.net gateways with real traffic load. The implementation aims to whether our proposed framework holds the same result in the real large-scale network environment.

7.1 Community network

Community networks happen when people come together to build and maintain the necessary infrastructure for Internet connection. Internet by the people, for the people. (source: Internet society)

A Wireless Community Network (WCN) is an affordable, alternative network infrastructure built by the local community members where they share the price of the Internet connectivity instead of opting for the ISP (top-down approach). Network devices have become affordable in recent years, and the number of local community networks is gaining popularity in developing countries to provide last-mile connectivity. Examples of successful CNs are guifi.net from Spain, Detroit Community network, NYC Mesh from the USA, Coopesur from Argentina, Freifunk from Germany, Zenzelini networks from South Africa, and many more.

Technically, a CN is a decentralized wireless mesh network built with interconnected rooftop routers [Mac+; Dim+17a]. The general network management and organization are nonexistent, and everybody is free to join/leave the CN. The size of the network grows organically. WCN is built with over-the-counter, inexpensive devices with different hardware, software, and ca-

capacity, therefore, highly heterogeneous. There are two main functionalities WCN provides. First, it provides local communication services (VoIP, file server, local chat, so on) with more neutral, secure communication. The members of the CN could communicate directly through the secure medium instead of the Internet servers. It is one of the main advantages of CN in the era of data privacy. Second, it provides Internet access to the community members and closes the gap of the Digital Divide. Internet connectivity in CN is typically built, instead of a single dedicated default gateway, from the combination of several limited, sometimes non-dedicated gateways. Clients use one of these gateways and switch to another when the first fails. Internet connectivity in this form is widespread but uncertain and inefficient due to several factors, including unbalanced traffic load across the gateway nodes.

7.1.1 guifi.net community network

One of the largest CN is called guifi.net, located in Spain, consists of over 36000 nodes where community members built up the network using simple network devices. The guifi.net was established in 2004, and the main concept of the network is to provide a free, open, and neutral network focused on providing local network services within the community. The technical overview of guifi.net is studied in [Veg+15]. Guifi.net nodes are wireless with few fiber-optic links. People are free to join the network and access information about the available nodes and services through the node registry service. The number of nodes in guifi.net grows organically, about 30 nodes are added weekly, and many are in the planning and deploying stage, see Figure 7.1.

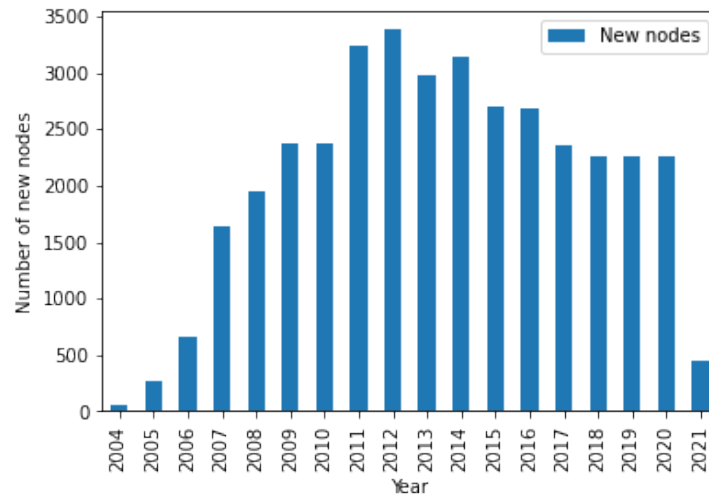


Figure 7.1: Number of new nodes (Data from www.guifi.net)

The Internet gateway selection in guifi.net has been studied in detail in [Veg+15; Dim+17b; DMN17]. Clients connect to the supernodes through Wifi, and then the supernodes are connected through dedicated links [Veg+15]. The community members share their Internet access with the other members with the help of the Internet gateway nodes (Internet proxies), the most commonly used local service in guifi.net. "Sharing Internet access" is when the primary user turns their network router into an Internet proxy node, registers the gateway in the network node registry, and allows other members to access the Internet. There are currently over 450 active Internet gateways available which serve approximately 12500 registered users (in 2016) [DMN17]. Most of the registered Internet gateways in guifi.net are municipal offices or public libraries donating their Internet access to the community members. All federated Internet gateways use the same authentication service, and users have freedom of movement within available proxies. The web proxy information is available in guifi.net's node registry system. The gateways serve multiple nodes simultaneously, and the number of nodes increases 2x, 3x higher during rush hours and holidays. Therefore, end-users receive uneven Quality of Service from the selected gateway.

The proxies are configured in the user browser, and the node has a list where it changes the primary gateway selection if the first one fails. Nodes often choose the default Internet gateway within the community. Default gateways thus become overloaded/over-popularized while there are other equally good gateways left unused. Manual selection leads to the gateway distribution imbalance in the network, which is connected to slow Internet connectivity at the end-users (bad Quality of Experience).

7.2 Implementation of Sense-Share-Select in guifi.net environment

Large-scale, heterogeneous networks like guifi.net have many issues related to the gateway selection, such as manual selection, imbalanced distribution, uneven QoS, and gateways with different capacities [Veg+15; Veg+12; DMN17]. Providing the best QoS for every node in the network is next to impossible. In the previous studies [Dim+17b; Dim+17a], the dynamic Internet gateway selection mechanism improves the quality of the Internet connection at the end-users. Guifi.net is the perfect network infrastructure to implement the Sense-Share-Select framework due to the following reasons.

- The guifi.net network is densely populated, and there are plenty of nodes to collaborate. In urban areas, guifi.net has more nodes; for example, the Osona zone has more than 8512 nodes and 36 internet proxies.

- Our framework is lightweight and implementable on the client node side, suitable for a heterogeneous network like guifi.net built with off-the-shelf, cheap devices with different capacities.
- The guifi.net has a node and Internet proxy(gateway) registry service required for our framework.

7.2.1 Experiment setup

We assess the performance of our proposal by deploying experimental nodes in guifi.net with the real production gateways. Our experiments are designed to test the following aspects of the Sense-Share-Select framework.

1. Accuracy of the collaborative performance monitoring
2. Number of messages created within the nodes
3. Distribution of the gateway nodes
4. QoE received at nodes through selected gateway
5. Handling network uncertainties, link failure, sudden performance change;

In total, we deployed 50 virtual nodes inside guifi.net's supernode at Universitat Politecnica de Catalunya ¹. Experimental nodes are distributed in three different servers in Universitat Politecnica Catalunya's supercomputing center. For all the experiments, we choose ten production gateways from the guifi.net CN. To add latency variance between the nodes shown in Figure 7.2 using Linux's network emulator tool (netem) consists of 4 groups (cluster) of nodes with no delay added, 3ms delay, 5ms delay, and 7 ms delay added in all incoming/outgoing requests. We compare our framework with the Vivaldi-based Internet gateway selection algorithm [Dim+17b], CoPing [Ko+13] algorithm. We created small-scale experiments with 20 nodes and increased the size of the network up to 50 nodes. We measured the accuracy, cost of collaboration and the gateway selection distribution, and download time. In the CoPing experiment setup, the outermost cluster nodes perform the measurements and distribute the measurements to the next-hop (next cluster) nodes, so on. Experiments run 100 rounds for each compared algorithm, 1 round = 2 minutes.

7.2.2 General experiments

7.2.2.1 Collaborative sensing accuracy

In the Sense-Share-Select framework, we strive to provide each node a gateway close to its performance sensing result (Brute-Force sensing) through

¹<https://guifi.net/en/UPCC6lab104>

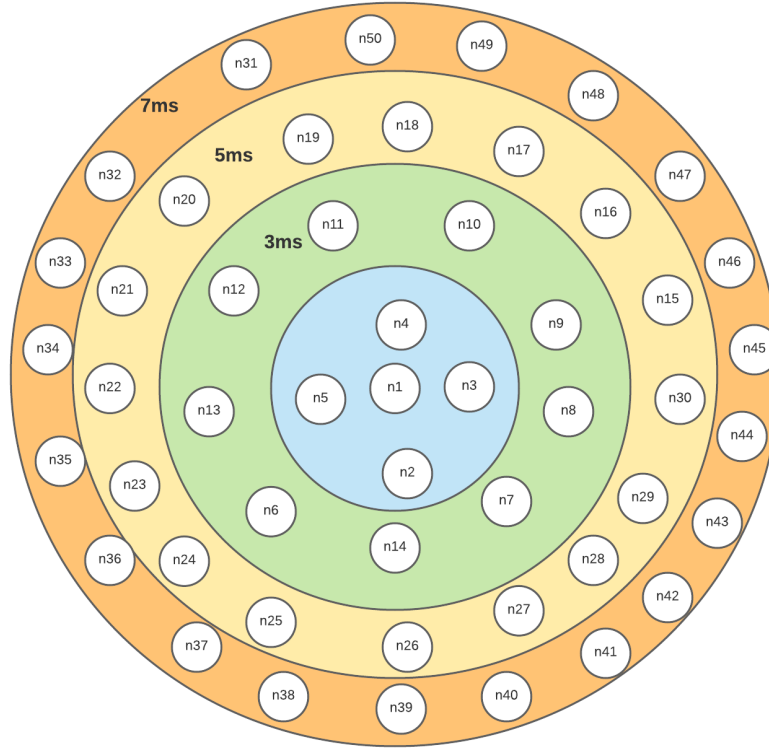


Figure 7.2: Experimental nodes topology

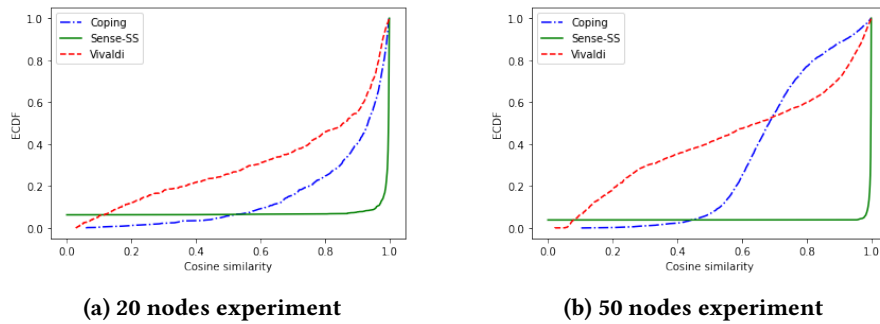


Figure 7.3: Collaborative sensing similarity

our *Collaborative layer*. This section shows the accuracy of the collaborative sensing of each node compared with the Brute-Force performance sensing. Figure 7.3 shows the accuracy of the collaborative performance sensing.

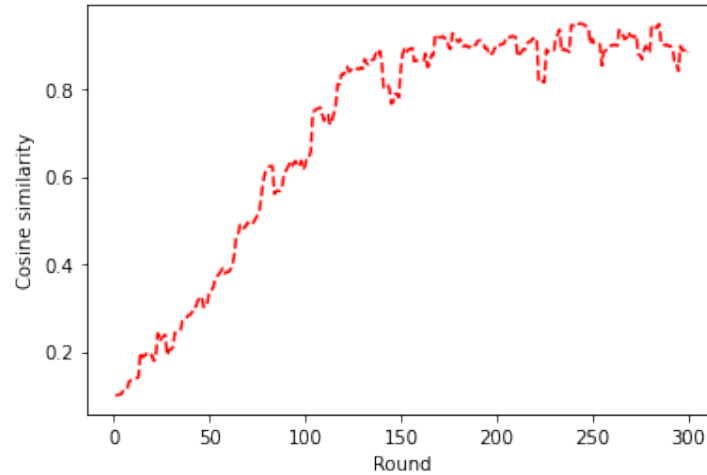


Figure 7.4: Vivaldi accuracy

Both experiments in Figure 7.3 show that the accuracy of the collaborative monitoring is consistently higher for the Sense-Share-Select result than the others. In Figure 7.3a, the CoPing algorithm's accuracy is on average >80% because the nodes have a maximum of 2 hops. In the 50 nodes experiment in Figure 7.3b, the average accuracy drops to >60% because the number of hops increased to 3 hops. The CoPing algorithm's accuracy drops as the number of hops increases which is not suitable for the multi-hop guifi.net network. In the Vivaldi-based algorithm experiment, the accuracy increases as the number of nodes increase in Figure 7.3 results. The nodes benefit from the higher collaboration because the synthetic coordinates adjust faster and become more accurate. However, high accuracy comes with the high cost of inter-node communication. The average accuracy of the Vivaldi-based algorithm is >80%, but the algorithm requires around 50-80 rounds (depending on the Vivaldi parameter setting) to stabilize the synthetic coordinates. Our experiment stopped at the 100th round, which is not enough to provide accurate monitoring. However, after the bootstrap period, the average estimation accuracy is kept above 90% (see Figure 7.4).

Compared to the Vivaldi-based algorithm, the Sense-Share-Select framework and the CoPing algorithm provide higher collaborative monitoring accuracy without the bootstrap phase. The collaborative monitoring accuracy is the best in the Sense-Share-Select framework, consistently kept higher and not affected by the number of nodes, the number of hops, nor the number of

rounds.

7.2.2.2 Number of messages

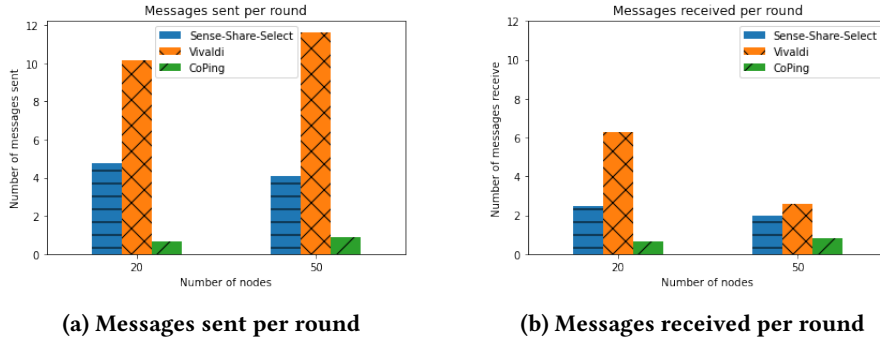


Figure 7.5: Collaborative message exchange

We counted the average number of performance measurement messages per round exchanged between the nodes in this experiment. Overall, from both figures, the number of messages sent is higher than the received messages. Because the collaboration is not mutual, neighbor nodes are not obliged to send their measurements back. In the CoPing algorithm, the root node performs the measurement and disseminates the measurements to the descendent neighbor nodes. In our experimental setup of the CoPing algorithm, every node transmits to one node. From Figure 7.5, the Sense-Share-Select framework (blue bar) sends measurement on average to 4 close neighbors and receive from 2 different neighbors in both 20-node and 50-node experiment. From the result shown in Figure 7.5, the Vivaldi-based algorithm exchanges the highest number of messages, and CoPing is the lowest. In the Vivaldi-based algorithm, the number of sent messages is higher than others, but there are no significant differences in the 20-node experiment and the 50-node experiment. However, in received messages, the 50-node experiment result shows a lower number of messages received than the 20-node experiment result because some nodes are left out from the collaboration.

The CoPing algorithm produces comparatively fewer gossip messages, and so does our framework. Clearly, the Vivaldi-based algorithm is the most expensive one, which requires constant, mass gossiping within network nodes to keep the balance. Additionally, Sense-Share-Select achieves a higher accuracy of monitoring than the others.

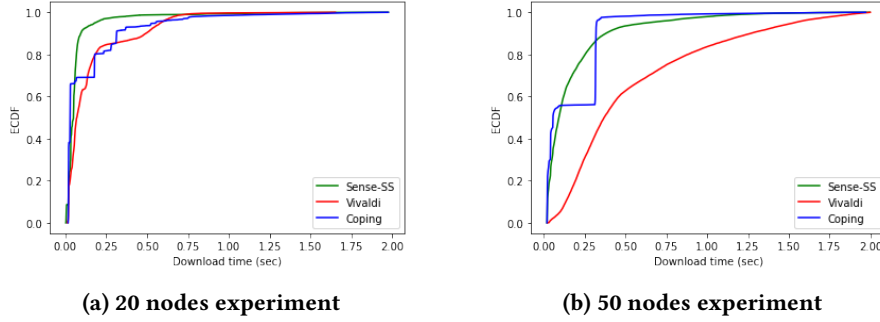


Figure 7.6: Download time

7.2.2.3 Download time

The QoE at the nodes through the selected gateway is assessed by downloading a 0.1MB file from the Internet. The download frequency is one in every 1 minute for 100 rounds (1 round = 2 minutes). The results are shown in Figure 7.6, and the Sense-Share-Select framework provides better download time as compared to the others for both network sizes. The CoPing algorithm and the Vivaldi-based algorithm provide similar download time in the 20-node experiment (see Figure 7.6a). However, in the 50-node experiment in Figure 7.6b, the CoPing algorithm provides better download time. The download latency is linked to the "herd behavior" as nodes select the best gateway in both compared algorithms. But the Vivaldi-based algorithm's high latency is linked to the slow adjustment to the network dynamics.

7.2.2.4 Gateway selection

We compare the node distribution over gateways for three different gateway selection algorithms, greedy selection, manual selection, and Sense-Share-Select framework. The gateway distribution experiment is done in the 40-node network scenario with nine production gateways. $[g1, g2, g4]$ gateways are good, $[g7, g8, g9]$ are bad, and $[g3, g5]$ are unstable performing. The result of the distribution is shown in Figure 7.7. With the greedy selection algorithm (blue bar), the nodes select the good-performing gateways and avoid unstable gateways. Thus, $[g1, g2, g4]$ have a higher number of nodes per gateway. Similar selection behavior is repeated in the manual selection algorithm (orange bar), but there is no deviation in the selection since it is static. The Sense-Share-Select framework (green bar) equally distributes nodes to the good-performing gateways and varied performing gateways and distributes a few nodes to the bad performing gateways.

The gateway selection stability is shown in Figure 7.8. Out of 50 rounds of experiment, the greedy selection changes the selection every two rounds

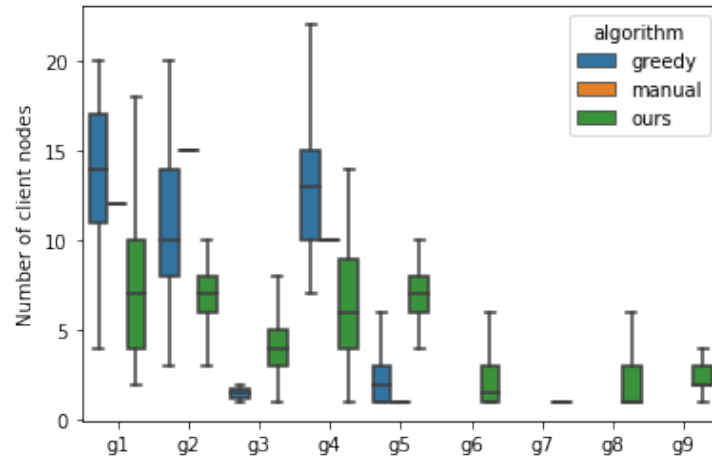


Figure 7.7: Gateway distribution per round

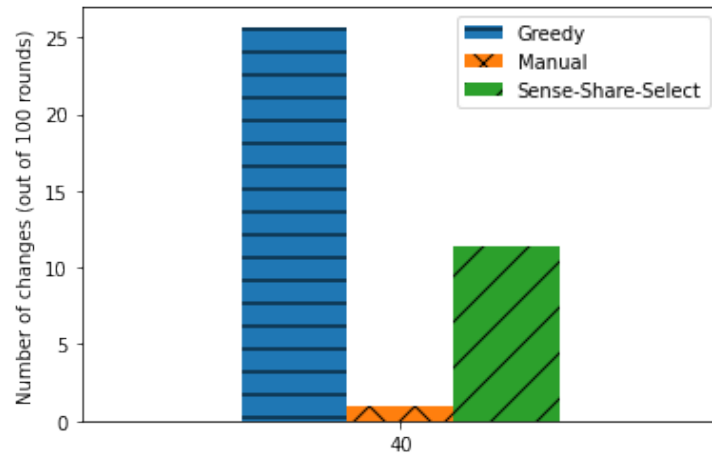


Figure 7.8: Number of gateway changes

on average. Our gateway selection algorithm reduces the gateway selection changes by 60%, changing the gateway selection every five rounds. The manual selection is close to 0 since the algorithm does not change the gateway unless the current gateway is unavailable.

The Sense-Share-Select framework keeps the balanced distribution over the good performing and limited capacity gateways and allocates some nodes to the bad performing gateways. At the same time, the framework maintains a stable gateway selection by not allowing frequent selection changes.

7.2.2.5 General experiment conclusion

All experiment results in Section 7.2.2 show that our proposed framework provides the best collaboration accuracy with the fastest download latency. The framework effectively reduces inter-node communication even the network size increases. The Vivaldi-based algorithm results are unfavorable as it is expensive, provides the worst download latency, and has a long bootstrapping period. Both compared algorithms show insufficient accuracy, unstable gateway selection, and do not scale.

7.2.3 Parameter specific experiments

This section focuses on testing the features of the Sense-Share-Select framework under different network conditions such as adding delay on the gateway, nodes sending false measurements, nodes becoming offline, etc. This section's purpose is to test the resilience and adaptability of the Sense-Share-Select framework.

7.2.3.1 Sensitivity to RTT threshold and number of close nodes

We study the two main factors that influence the precise estimation of the gateway performance, 1) the RTT threshold value used for creating a close neighbors cluster and 2) the number of nodes in the same close neighbors set. We run our experiment for 100 rounds with three different RTT threshold values to form a close neighbor set, 5ms, 10ms, and 20ms, having an average of 10, 15, 20 close nodes respectively at each node to see the effect of the number of nodes and the threshold values. The latency between the nodes are added based on the experiments carried in guifi.net in [DMN17] where average latency between network nodes in the same zone was less than 15ms in 80% nodes. In the evaluation, we used the cosine similarity function (0 to 1, 1 being higher similarity), which is the similarity between the received measurement against the node's perception of the gateway performance when receiving information. This experiment conveys how precise estimations our collaborative sensing approach can achieve. Figure 7.9 shows the precision of

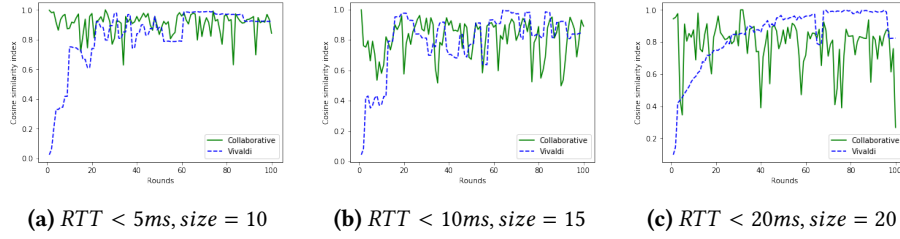


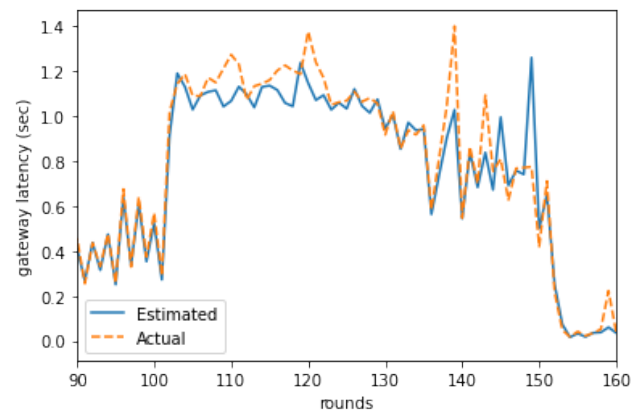
Figure 7.9: Sensitivity analysis of RTT threshold value and size of close neighbors cluster

the gateway performance estimation of our collaborative sensing algorithm and the Vivaldi-based algorithm proposed in [Dim+17b]. From Figure 7.9a, both of the collaborative approaches result in the near-optimal estimation of the performance of gateway nodes, which is important for the selection of the best suitable gateway for a client node. With the Vivaldi-based algorithm, the average gateway similarity estimation starts below 0.05 to reach almost 0.98 after 60 rounds, showing a progressive increase of the precision of the estimation as time passes. Comparatively, our collaborative sensing algorithm provides more than 0.8 performance monitoring accuracy throughout the experiment, as shown in Figure 7.9a. When increasing the close neighbors RTT threshold value in Figure 7.9b and Figure 7.9c, the precision of the gateway performance estimation drops down to average 0.6-0.8. As the RTT threshold value increases, the similarity between nodes in the close neighbors set decreases. The peak values result from measurement received from the closer nodes in the close neighbors set, and similarly, lower values result from performance measurement received from the distant node in the close neighbors set. On the other hand, the Vivaldi-based algorithm outperforms our algorithm in the 20-25th round, and in Figure 7.9c, the precision of estimation stays constantly above 0.8 during the experiment.

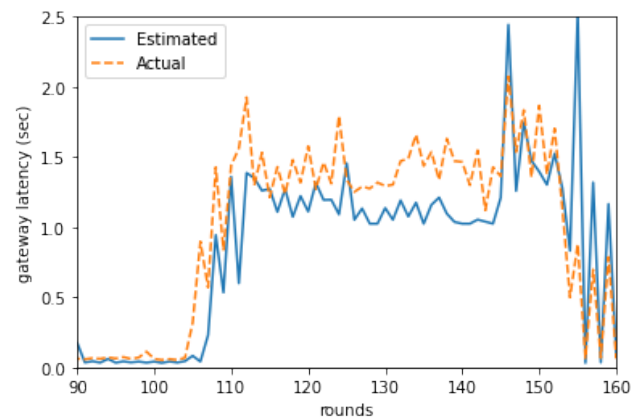
Our collaborative sensing algorithm's precision of estimation is shown to reach faster precise performance sensing during the initial stage of the algorithm and maintained stable performance throughout the experiment. The Vivaldi-based algorithm is stable after 15-20 rounds, where it requires thorough (reactive) information exchange between network nodes and gateway nodes and information processing at every node.

7.2.3.2 Introducing delay at the gateway

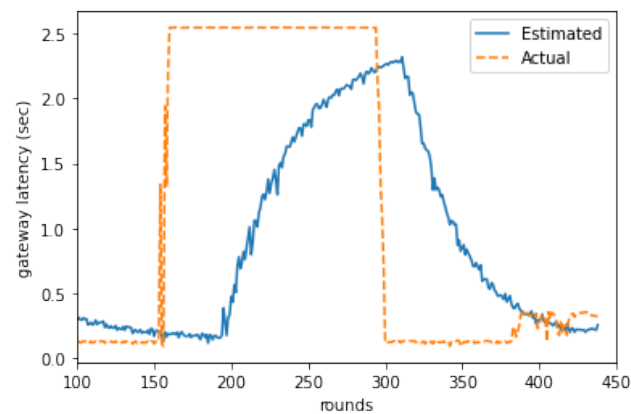
In the collaborative sensing algorithm, the collaboration accuracy should not change when the performance changes suddenly. As guifi.net gateways are unstable, the framework should catch the performance fluctuations as soon as possible. In the experiment, we added a 200ms delay at one gateway around



(a) Sense-Share-Select



(b) CoPing algorithm



(c) Vivaldi based algorithm

Figure 7.10: Download time

the 100-150th round (based on the stabilization period) and removed it after the 50th round to see the reaction of our collaborative sensing algorithm's accuracy. The results of the experiment in three different algorithms are in Figure 7.10. The orange dotted line represents the actual gateway performance, and the straight blue line is the estimated performance through the collaborative algorithm.

From the result of the Sense-Share-Select framework is in Figure 7.10a, the gateway performance change caught immediately and stayed in line with the actual latency curve. When removing the delay, the change is caught in the same round. On few occasions, the estimated performance is slightly different from the actual due to the smoothing factor of the trust-based collaboration component.

The actual gateway performance change is caught within 1-5 rounds later in the CoPing algorithm (see Figure 7.10b) because nodes wait for the upstream node to send the measurements. The estimated delay and the actual gateway delay have a 0.5 seconds estimation gap because the accuracy of collaboration degrades in the downstream nodes (see Figure 7.3). Overall, the CoPing algorithm responds to the gateway performance change quickly within less than five rounds.

The Vivaldi-based algorithm does not cope well with the sudden gateway performance change, shown in Figure 7.10c since it is not reactive. Any change in the synthetic coordinate system reflects all nodes in the network, and the nodes update the coordinates slowly using small steps towards the direction of the change. Therefore, the changes are caught after 150 rounds approximately. The same pattern repeated when removing the delay the algorithm is caught with the change after 50 rounds.

7.2.3.3 Removing nodes

The technological survey conducted in guifi.net [Veg+15] shows that half of the nodes in guifi.net have less than 10% reachability. The Barcelona area nodes have more than 60% of reachability. Therefore, the nodes becoming unreachable/offline is common in guifi.net. We removed 10%, 20%, and 30% of the nodes from the experiment at the three stages of rounds noted in black dots in Figure 7.11. We skipped the CoPing implementation because the experiment requires active node discovery in the mobile ad-hoc network (out of the scope). Removing nodes has no impact on the accuracy of both algorithms in the result obtained in Figure 7.11. For our framework, the offline nodes are replaced by the others from the neighbor list ranked higher, and the collaboration accuracy is still high. For the Vivaldi-based algorithm, every node is a collaborator node sending measurements, and once the algorithm's synthetic coordinate system stabilizes, losing the collaborator node does not affect the

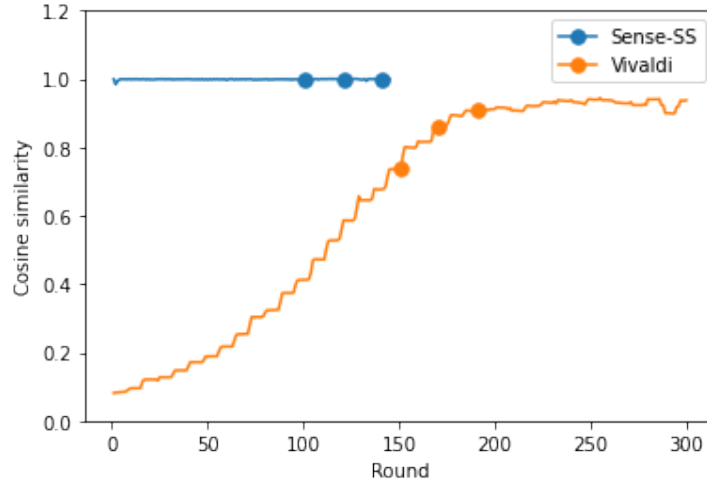


Figure 7.11: Node unavailability test

accuracy.

7.2.3.4 Faulty collaborators

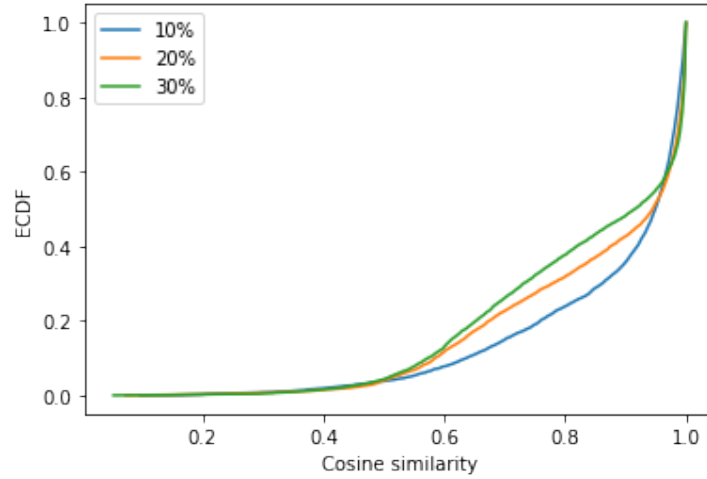


Figure 7.12: In the presence of faulty nodes

We rank the nodes based on their sending behavior and collaborate with the nodes with a higher similarity of measurements in the Sense-Share-Select framework. We tested the fault tolerance of our proposal in Figure 7.12 and introduced 10%, 20%, 30% of the faulty nodes in the network where they inten-

tionally send 10x higher measurements than the others. In the Sense-Share-Select framework, as we increase the faulty neighbors, we see that the average similarity of the collaboration drops from >95% accuracy to >94% and >91% on average. Still, the average similarity of collaborative sensing is more than 90%, and the accuracy drop is due to shifting the collaboration with less similar nodes. In the Vivaldi-based algorithm, once the synthetic coordinates stabilize, the availability of the nodes has almost no effect on the accuracy of the estimation.

7.2.3.5 Isolated nodes

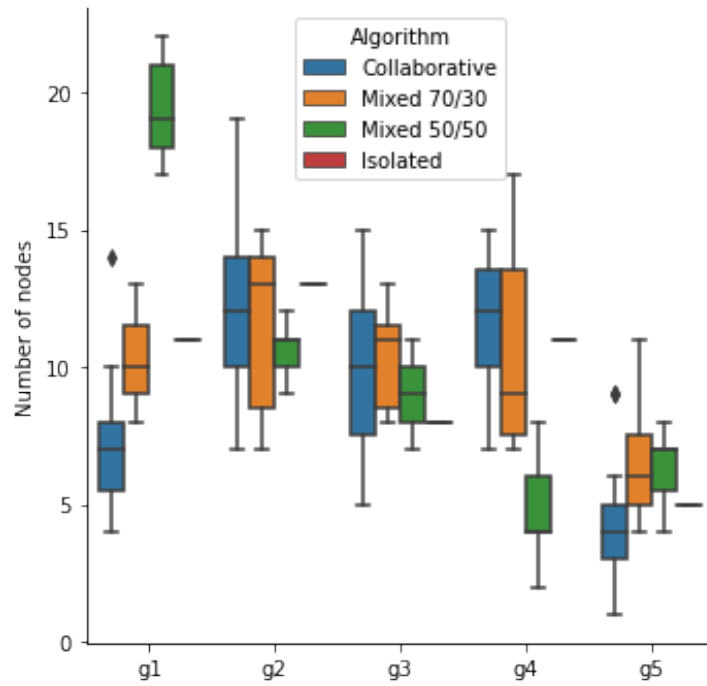


Figure 7.13: Wireless experiment distribution

The Sense-Share-Select algorithm works best when nodes voluntarily collaborate. However, in the production network, the adoption of the new framework can be slow since people are reluctant to change. In that case, the nodes might not find close neighbors to collaborate and end up isolated. This experiment aims to cover different isolation effects on the gateway selection distribution. We experimented with the following settings:

1. 100% of the nodes are collaborating with our framework,

2. 70% of the nodes are collaborating with our framework, 30% use manual selection,
3. 50% nodes are collaborating with our framework, 50% use manual selection
4. 100% of the nodes use manual selection

The experiment consists of 50 nodes and five gateways, distributed uniformly in a 150x150m area. $[g1, g2]$ are "Good" performing gateway, $g3, g4$ are "Limited-Capacity" gateway, and $g5$ is "Bad" performing gateway. The experiment results shown in Figure 7.13.

In the 100% isolated nodes result in Figure 7.13, the nodes avoid the bad performing gateways and select "good enough" gateway. As the number of the manual nodes increases, the popularity of the "good" gateway increases (see Figure 7.13. In the "50/50" experiment, on average, 30 nodes are allocated to the "Good" gateways, while in the "30/70" and "Full collaborated" experiment, it is approximately 20 nodes. In the "Fully collaborative" experiment, the nodes avoid selecting the "Bad" gateways, and four nodes on average selected $g5$ gateway per round.

Another problem with the implementation is that some nodes might be far from the others' RTT threshold. In such circumstances, the node has to rely on its periodic performance measurements. We run our algorithm with the following conditions.

1. Full collaboration of nodes using our framework
2. Full isolation where all nodes out of each others' RTT range and using our framework

In the isolation experiment, the node selects from the two gateway performances and previous measurements collected (no collaborating nodes). Therefore in Figure 7.14, all gateways were selected equal, including $g5$, because of the limited measurements. In the collaborative experiment result, the nodes avoid selecting $g5$ gateway and are distributed to the "Good" and "Limited capacity" gateways in a balanced manner.

7.2.3.6 Gateways with different bandwidth

Figure 7.15 shows an experiment with different bandwidths and delays on the gateways. In the blue bar experiment, $[g1, g2]$ gateways have 50mbps bandwidth, $[g3, g4]$ have 30mbps bandwidth, and $g5$ has 10mbps bandwidth. In the orange bar experiment, on top of the bandwidth, $g1, g2$ are added 1ms delay, $g3, g4$ are added 3ms delay, and $g5$ is added 5ms delay. The distinction between the gateways is not clear with different bandwidth experiment

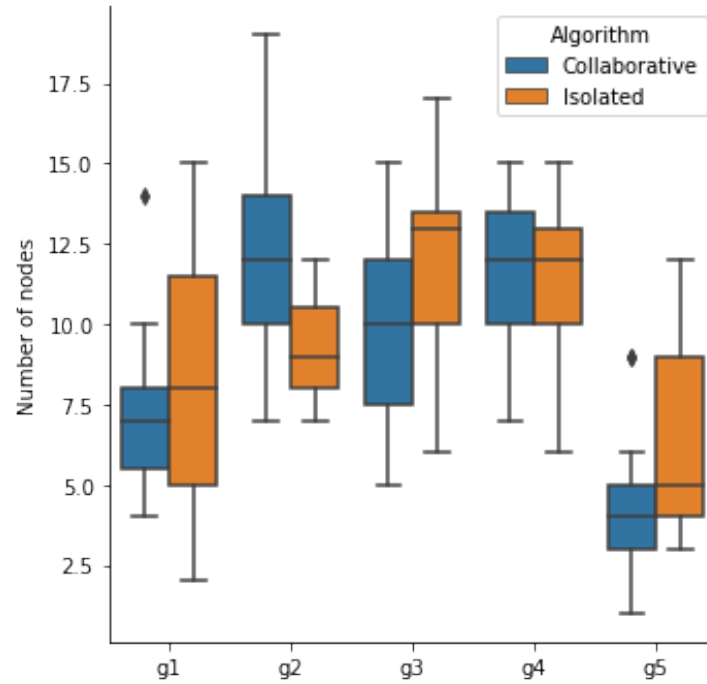


Figure 7.14: Full collaboration vs full isolation

results in Figure 7.15 because the nodes do not exhaust gateway capacities. However, the result in the orange bar shows a clear difference where nodes avoided from the *g5* gateway.

7.3 Conclusion

We implemented the Sense-Share-Select framework in the guifi.net community network environment with production gateways to demonstrate the efficiency of the proposed framework. The Sense-Share-Select framework improves the current manual gateway selection of the guifi.net by improving the QoE of all client nodes and performs better than the state-of-the-art algorithms. We tested the framework with different network scenarios such as node failure, gateway service degradation, faulty collaborators, and different network topologies. The results obtained from the implementation are consistent with the emulated network experiments, proving the efficiency of the Sense-Share-Select framework.

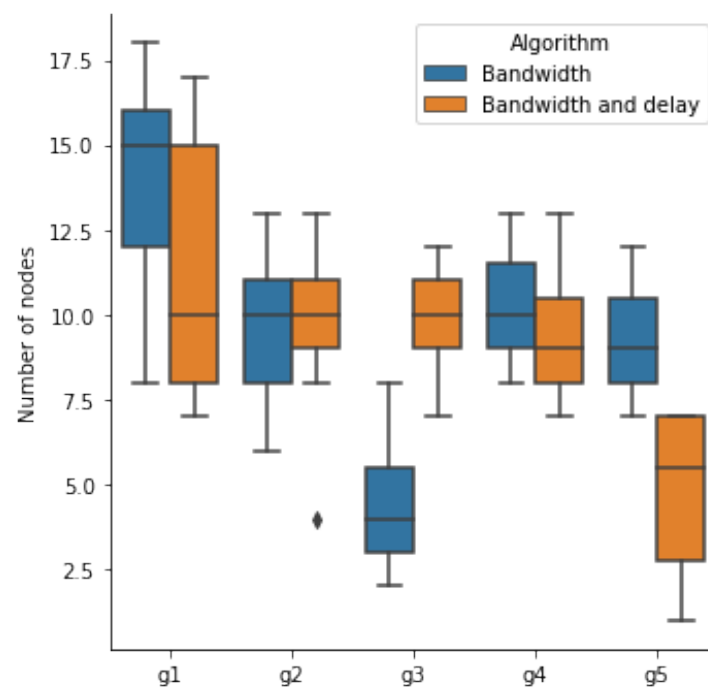


Figure 7.15: Experiment with different bandwidth

8.1 Main conclusions

By nature, a shared Internet access network consists of a high number of heterogeneous nodes. Since the majority of the nodes are built with over-the-counter network devices with limited capacity and resources, the gateway selection mechanism requires to be simple and effective. Our Sense-Share-Select framework has been specifically designed for large-scale network, with low complexity and fair resource distribution as the main goals.

To achieve low cost gateway monitoring, we introduced close neighbors collaboration and randomized performance sampling. Allowing the node to select the close neighbors to collaborate improves the accuracy of the collaborative monitoring and reduces the impact of faulty collaborators. Every round, the nodes monitor the performance of different gateways which reduces the monitoring traffic to the gateways and eliminates the need for scheduling and reorganization within the collaborators.

Concerning the balanced, fair distribution of the nodes over the gateways, our selection mechanism evaluates gateway candidates and filters good candidates based on their previous performance history. The selection algorithm doesn't force nodes to select the best performing gateway and let them settle with a good performing gateway. Doing so significantly reduces the over popularization of single gateways and provides long-term, sustainable gateway selection.

Overall, we believe that our framework significantly improves the QoE of the end-users without requiring complex technical knowledge such as gateway load, routing protocol, etc. The Sense-Share-Select framework as presented in this thesis has been designed for the Internet gateway selection domain but its application is much more versatile. The main logic is simple enough to be applied to, for example, network gateway selection for IoT device communication. The user can configure the framework based on their needs. For example, in mobile networks, the neighbor discovery component should be done frequently and the duration of the performance sensing should be increased to match the network dynamics. In sensor network, the frequency of the measurement should be adjusted differently for each node based on the battery life.

8.2 Future work

Along with the further development and analysis of the designed prototypes, there are several directions that other researchers might find interesting to investigate in the future.

1. **Adaptive performance monitoring** - We show that nodes collect significant numbers of measurements through periodic collaboration in the *Collaborative layer*. The majority of the measurements are stable and have patterns with peak/off-peak periods. Further flexible performance monitoring period adjustment is possible. Timeseries based performance prediction coupled with close neighbors collaborative performance monitoring can be a way to further reduce the number of performed measurements.
2. **Towards secure, anonymous collaboration** - Sense-Share-Select is not an anonymous collaborative framework and is focused on bottom-up initiatives. Since nodes are aware of the collaborating nodes' physical addresses, user tracking and reidentification is possible. In order to be suitable to open networks, security and anonymity must be addressed. Blockchain-based secure collaboration, or similar to the ToR network, is open for further investigation.

8.3 Discussion

The Sense-Share-Select framework is designed for nodes in large-scale, bottom-up wireless mesh networks with the goal to improve each node's individual Internet gateway experience. The implementation in real large-scale networks is challenging to achieve due to technical and social aspects. From the technical point of view, algorithms implemented in the routing protocol might be more manageable than client-side solutions such as our framework. However, client-side implementations are more suitable for heterogeneous networks with different routing protocols and capacities. From the social point of view, existing users are often reluctant to change to new features, as shown in a feasibility study conducted in [Sat+14]. According to the study, local network users are accustomed to the existing network functionality and often do not have technical/network experience to implement the framework in their nodes.

The experiments were designed for the local Wireless Mesh network with non-mobile network. For mobile network environments, the node discovery is crucial to identify the collaborating nodes and gateway nodes within the range. The frequency of the gateway performance monitoring and collabo-

ration should be increased in mobile networks to ensure the accuracy of the collaborative monitoring.

8.4 Financial support

This work was funded by European Commission (EACEA) through the Erasmus Mundus doctoral fellowship, via Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) programme. This work was supported by the EU Horizon 2020 Framework Program project netCommons (H2020-688768), by the Spanish Government under contract TIN2016-77836-C2-2-R, and by the Generalitat de Catalunya as Consolidated Research Group 2014-SGR-881. The final year of the doctoral program was funded by the European Union's Horizon 2020 research and Next Generation Internet (NGI) Explorers innovation programme under the Grant Agreement No 825183.

Bibliography

- [AAJ09a] U. Ashraf, S. Abdellatif, and G. Juanole. “Gateway selection in backbone wireless mesh networks”. In: *Wireless Communications and Networking Conference, (WCNC)*. IEEE, 2009. ISBN: 9781424429486. DOI: 10.1109/WCNC.2009.4917735.
- [AAJ09b] U. Ashraf, S. Abdellatif, and G. Juanole. “Gateway selection in backbone wireless mesh networks”. In: *Wireless Communications and Networking Conference (WCNC)*. IEEE, 2009, pp. 1–6.
- [Abu+15] A. Abujoda, D. Dietrich, P. Papadimitriou, and A. Sathiaselan. “Software-defined wireless mesh networks for internet access sharing”. In: *Computer Networks* 93 (2015), pp. 359–372. DOI: 10.1016/j.comnet.2015.09.008.
- [Aff20] A. for Affordable Internet. *2020 Affordability Report*. 2020. URL: <https://a4ai.org/affordability-report/report/2020/>.
- [AJB10] P. A. Acharya, D. L. Johnson, and E. M. Belding. “Gateway-aware routing for wireless mesh networks”. In: *Mobile Adhoc and Sensor Systems (MASS)*. IEEE. 2010, pp. 564–569.
- [AJO07] D. Acemoglu, R. Johari, and A. Ozdaglar. “Partially optimal routing”. In: *IEEE Journal on selected areas in communications* 25.6 (2007), pp. 1148–1160.
- [Aou+06] B. Aoun, R. Boutaba, Y. Iraqi, and G. Kenward. “Gateway placement optimization in wireless mesh networks with QoS constraints”. In: *IEEE Journal on Selected Areas in Communications* 24.11 (2006), pp. 2127–2136.
- [AR08] V. Ayyadurai and R. Ramasamy. “Internet Connectivity for Mobile Ad Hoc Networks Using Hybrid Adaptive Mobile Agent Protocol.” In: *International Arab Journal of Information Technology (IAJIT)* 5.1 (2008).
- [Bat+18] K. Batbayar, E. Dimogerontakis, R. Meseguer, L. Navarro, E. Medina, and R. M. Santos. “The RIMO gateway selection approach for mesh networks: Towards a global Internet access for all”. In: *Multidisciplinary Digital Publishing Institute Proceedings*. Vol. 2. 19. 2018, p. 1258.

- [Bat+19a] K. Batbayar, E. Dimogerontakis, R. Meseguer, L. Navarro, and R. Sadre. “Collaborative informed gateway selection in large-scale and heterogeneous networks”. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2019, pp. 337–345.
- [Bat+19b] K. Batbayar, E. Dimogerontakis, R. Meseguer, L. Navarro, and R. Sadre. “Sense-Share: A Framework for Resilient Collaborative Service Performance Monitoring”. In: *15th International Conference on Network and Service Management(CNSM)*. IEEE, 2019.
- [Bat+20] K. Batbayar, R. Meseguer, R. Sadre, and S. Subramaniam. “Gate-Select: A novel Internet gateway selection algorithm for client nodes”. In: *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE. 2020, pp. 1–9.
- [Ber+17] P. Berenbrink, P. Kling, C. Liaw, and A. Mehrabian. “Tight load balancing via randomized local search”. In: *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2017, pp. 192–201.
- [BH11] M. Boushaba and A. Hafid. “Best path to best gateway scheme for multichannel multi-interface wireless mesh networks”. In: *Wireless Communications and Networking Conference (WCNC)*. 2011, pp. 689–694. ISBN: 978-1-61284-255-4.
- [Bic+05] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. “Architecture and evaluation of an unplanned 802.11 b mesh network”. In: *Proceedings of the 11th annual international conference on Mobile computing and networking*. 2005, pp. 31–42.
- [Bin+05] S. Bin, S. Bingxin, L. Bo, H. Zhonggong, and Z. Li. “Adaptive gateway discovery scheme for connecting mobile ad hoc networks to the internet”. In: *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005*. Vol. 2. IEEE. 2005, pp. 795–799.
- [BMJ99] J. Broch, D. Maltz, and D. Johnson. “Supporting hierarchy and heterogeneous interfaces in multi-hop wireless ad hoc networks”. In: *Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN’99)*. 1999, pp. 370–375. DOI: 10.1109/ISPAN.1999.778966.
- [Che+14] I. Chen, F. Bao, M. Chang, and J. Cho. “Dynamic Trust Management for Delay Tolerant Networks and Its Application to Secure Routing”. In: *IEEE Transactions on Parallel and Distributed Systems* 25.5 (May 2014), pp. 1200–1210. ISSN: 1045-9219. DOI: 10.1109/TPDS.2013.116.

- [CSC09] J. Cho, A. Swami, and I. Chen. “Modeling and Analysis of Trust Management for Cognitive Mission-Driven Group Communication Systems in Mobile Ad Hoc Networks”. In: *2009 International Conference on Computational Science and Engineering*. Vol. 2. Aug. 2009, pp. 641–650.
- [Dab+04] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. “Vivaldi: A decentralized network coordinate system”. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 15–26.
- [Del+12] A. Y. Delgado, M. Gadeo-Martos, J. Fernandez-Prieto, and J. Canada-Bago. “A fuzzy balancing load system to improve hybrid ad hoc networks”. In: *2012 IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE. 2012, pp. 66–69.
- [Dim+17a] E. Dimogerontakis, R. Meseguer, L. Navarro, S. Ochoa, and L. Veiga. “Design trade-offs of crowdsourced web access in community networks”. In: *2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. 2017, pp. 24–29. DOI: 10.1109/CSCWD.2017.8066665.
- [Dim+17b] E. Dimogerontakis, J. Neto, R. Meseguer, L. Navarro, and L. Veiga. “Client-side routing-agnostic gateway selection for heterogeneous Wireless Mesh Networks”. In: *Integrated Network and Service Management (IM)*. 2017, pp. 377–385. DOI: 10.23919/INM.2017.7987301.
- [DMN17] E. Dimogerontakis, R. Meseguer, and L. Navarro. “Internet Access for All: Assessing a Crowdsourced Web Proxy Service in a Community Network”. In: *Passive and Active Measurement (PAM)*. 2017, pp. 72–84. DOI: 10.1007/978-3-319-54328-4_6.
- [FBCon] Facebook. *Facebook connectivity: Bringing more people online to a faster internet*. URL: <https://connectivity.fb.com>.
- [FM02] Y. Fernandess and D. Malkhi. “K-clustering in Wireless Ad Hoc Networks”. In: *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing*. POMC ’02. Toulouse, France: ACM, 2002, pp. 31–37. ISBN: 1-58113-511-4. DOI: 10.1145/584490.584497.
- [Fon+15] R. R. Fontes, S. Afzal, S. H. Brito, M. A. Santos, and C. E. Rothenberg. “Mininet-WiFi: Emulating software-defined wireless networks”. In: *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE. 2015, pp. 384–389.

- [Gho+15] S. Ghorbani, B. Godfrey, Y. Ganjali, and A. Firoozshahian. "Micro Load Balancing in Data Centers with DRILL". In: *ACM Workshop on Hot Topics in Networks*. Philadelphia, PA, USA: ACM, 2015, 17:1–17:7. ISBN: 978-1-4503-4047-2. DOI: 10 . 1145 / 2834050 . 2834107.
- [Gho+17] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian. "Drill: Micro load balancing for low-latency data center networks". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 225–238.
- [GRS08] J. J. Galvez, P. M. Ruiz, and A. F. Skarmeta. "A distributed algorithm for gateway load-balancing in Wireless Mesh Networks". In: *Wireless Days*. 2008, pp. 1–5. DOI: 10 . 1109 / WD . 2008 . 4812861.
- [guifi] *Open, Free and Neutral Network Internet for everybody.*
- [HCB00] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. "Energy-efficient communication protocol for wireless microsensor networks". In: *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. Jan. 2000, p. 10. DOI: 10 . 1109 / HICSS . 2000 . 926982.
- [HCB02] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. "An Application-specific Protocol Architecture for Wireless Microsensor Networks". In: *Trans. Wireless. Comm.* 1.4 (Oct. 2002), pp. 660–670. ISSN: 1536-1276. DOI: 10 . 1109 / TWC . 2002 . 804190.
- [HDS17] T. Hardes, F. Dressler, and C. Sommer. "Simulating a city-scale community network: From models to first improvements for Freifunk". In: *2017 International Conference on Networked Systems (NetSys)*. IEEE. 2017, pp. 1–7.
- [HLT04] C.-F. Huang, H.-W. Lee, and Y.-C. Tseng. "A two-tier heterogeneous mobile ad hoc network architecture and its load-balance routing problem". In: *Mobile networks and applications* 9.4 (2004), pp. 379–391.
- [HMW09] F. Hoffmann, D. Medina, and A. Wolisz. "Two-step delay based Internet gateway selection scheme for aeronautical ad hoc networks". In: *2009 IEEE 20th International Symposium on Personal, Indoor and Mobile Radio Communications*. 2009, pp. 2638–2642. DOI: 10 . 1109 / PIMRC . 2009 . 5449871.

- [HXA08] B. He, B. Xie, and D. P. Agrawal. "Optimizing deployment of internet gateway in wireless mesh networks". In: *Computer Communications* 31.7 (2008), pp. 1259–1275.
- [ITUStat] ITU. *ITU: Statistics*. URL: <https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx>.
- [Jav+08] U. Javaid, T. Rasheed, D.-E. Meddour, and T. Ahmed. "Adaptive distributed gateway discovery in hybrid wireless networks". In: *2008 IEEE Wireless Communications and Networking Conference*. IEEE. 2008, pp. 2735–2740.
- [Jon+00] U. Jonsson, F. Alriksson, T. Larsson, P. Johansson, and G. Q. Maguire. "MIPMANET-mobile IP for mobile ad hoc networks". In: *2000 First Annual Workshop on Mobile and Ad Hoc Networking and Computing. MobiHOC (Cat. No. 00EX444)*. IEEE. 2000, pp. 75–85.
- [JP73] R. Jarvis and E. Patrick. "Clustering Using a Similarity Measure Based on Shared Near Neighbors". In: *IEEE Transactions on Computers* 22.11 (Nov. 1973), pp. 1025–1034. ISSN: 0018-9340. DOI: 10.1109/T-C.1973.223640.
- [KBM12] M. Kashanaki, Z. Beheshti, and M. R. Meybodi. "A distributed learning automata based gateway load balancing algorithm in wireless mesh networks". In: *2012 international symposium on instrumentation & measurement, sensor network and automation (IMSNA)*. Vol. 1. IEEE. 2012, pp. 90–94.
- [Kim+07] Y. Kim, Y. Jeong, M. Seo, and J. Ma. "Load-balanced Mesh Portal Selection in Wireless Mesh Network". In: *MILCOM 2007 - IEEE Military Communications Conference*. Oct. 2007, pp. 1–6. DOI: 10.1109/MILCOM.2007.4454919.
- [Ko+13] B. J. Ko, S. Liu, M. Zafer, H. Y. S. Wong, and K.-W. Lee. "Gateway selection in hybrid wireless networks through cooperative probing". In: *Integrated Network Management (IM)*. IEEE, 2013, pp. 352–360.
- [Lee+03] J. Lee, D. Kim, J. Garcia-Luna-Aceves, Y. Choi, J. Choi, and S. Nam. "Hybrid gateway advertisement scheme for connecting mobile ad hoc networks to the internet". In: *The 57th IEEE Semi-annual Vehicular Technology Conference, 2003. VTC 2003-Spring*. Vol. 1. IEEE. 2003, pp. 191–195.

- [LHM10] B. Lantz, B. Heller, and N. McKeown. “A Network in a Laptop: Rapid Prototyping for Software-defined Networks”. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 19:1–19:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868466.
- [Lia+16] L. Liang, W. Wang, Y. Jia, and S. Fu. “A cluster-based energy-efficient resource management scheme for ultra-dense networks”. In: *IEEE Access* 4 (2016), pp. 6823–6832.
- [Lin+17] H. Lin, J. Hu, J. Ma, L. Xu, and Z. Yu. “A Secure Collaborative Spectrum Sensing Strategy in Cyber-Physical Systems”. In: *IEEE Access* 5 (2017), pp. 27679–27690. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2767701.
- [Liv+10] H. Livingstone, H. Nakayama, T. Matsuda, X. Shen, and N. Kato. “Gateway selection in multi-hop wireless networks using route and link optimization”. In: *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*. IEEE. 2010, pp. 1–5.
- [Loon] Google. *Loon: Connect people everywhere*. URL: <https://loon.com/>.
- [LQL13] Y. Liao, H. Qi, and W. Li. “Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks”. In: *IEEE Sensors Journal* 13.5 (May 2013), pp. 1498–1506. ISSN: 1530-437X. DOI: 10.1109/JSEN.2012.2227704.
- [Lue+18] K. L. Lueth et al. “State of the IoT 2018: Number of IoT devices now at 7B–Market accelerating”. In: *IoT Analytics* 8 (2018).
- [Lyu+16] L. Lyu, Y. W. Law, S. M. Erfani, C. Leckie, and M. Palaniswami. “An improved scheme for privacy-preserving collaborative anomaly detection”. In: *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. Mar. 2016, pp. 1–6. DOI: 10.1109/PERCOMW.2016.7457159.
- [LZ10] C.-H. Lung and C. Zhou. “Using hierarchical agglomerative clustering in wireless sensor networks: An energy-efficient and flexible approach”. In: *Ad Hoc Networks* 8.3 (2010), pp. 328–344. ISSN: 1570-8705. DOI: <https://doi.org/10.1016/j.adhoc.2009.09.004>.
- [Mac+] L. Maccari, P. Magaudo, S. Crabu, and F. Giovanella. “Hackivism, Infrastructures and Legal Frameworks in Community Networks: the Italian Case of Ninux. org”. In: ().

- [Mit01] M. Mitzenmacher. “The power of two choices in randomized load balancing”. In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (Oct. 2001), pp. 1094–1104.
- [MK04] Miin-Shen Yang and Kuo-Lung Wu. “A similarity-based robust clustering method”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.4 (Apr. 2004), pp. 434–448. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2004.1265860.
- [Nan+06] D. Nandiraju, L. Santhanam, N. Nandiraju, and D. P. Agrawal. “Achieving load balancing in wireless mesh networks through multiple gateways”. In: *2006 IEEE international conference on mobile Ad Hoc and sensor systems*. IEEE, 2006, pp. 807–812.
- [Nas+15] M. A. U. Nasir, G. D. F. Morales, N. Kourtellis, and M. Serafini. “When Two Choices Are not Enough: Balancing at Scale in Distributed Stream Processing”. In: *CoRR abs/1510.05714* (2015). arXiv: 1510.05714.
- [NMS15] S. Nasre, P. Malathi, and M. Sharma. “Wireless mesh network deployed in disaster area using gateway selection”. In: *International Journal of Research in Computer and Communication Technology (IJRCCT)* 4.6 (2015), pp. 405–409.
- [NWZ16] C. Nie, H. Wu, and W. Zheng. “Latency and Lifetime-Aware Clustering and Routing in Wireless Sensor Networks”. In: *2016 IEEE 41st Conference on Local Computer Networks (LCN)*. Nov. 2016, pp. 164–167. DOI: 10.1109/LCN.2016.33.
- [Ous+13] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. “Sparrow: distributed, low latency scheduling”. In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. 2013, pp. 69–84.
- [OV17] I. Okeke and F. Verdicchio. “Shape-based clustering in wireless sensor networks”. In: *2017 IEEE SENSORS*. Oct. 2017, pp. 1–3. DOI: 10.1109/ICSENS.2017.8233989.
- [Rey+13] C. Rey-Moreno, W. Tucker, N. Bidwell, and. “Experiences, Challenges and Lessons from Rolling Out a Rural WiFi Mesh Network”. In: (Jan. 2013).
- [RG04] P. M. Ruiz and A. F. Gomez-Skarmeta. “Maximal source coverage adaptive gateway discovery for hybrid ad hoc networks”. In: *International conference on ad-hoc networks and wireless*. Springer, 2004, pp. 28–41.
- [RK03] P. Ratanchandani and R. Kravets. *A hybrid approach to internet connectivity for mobile ad hoc networks*. Vol. 3. IEEE, 2003.

- [RT02] T. Roughgarden and É. Tardos. “How bad is selfish routing?” In: *Journal of the ACM (JACM)* 49.2 (2002), pp. 236–259.
- [Sal+18] S. Salsano, F. Patriarca, F. Lo Presti, P. L. Ventre, and V. Gentile. “Accurate and efficient measurements of IP level performance to drive interface selection in heterogeneous wireless networks”. In: *IEEE Transactions on Mobile Computing* (2018), pp. 1–1.
- [Sat+14] A. Sathiaselalan, R. Mortier, M. Goulden, C. Greiffenhagen, M. Radenkovic, J. Crowcroft, and D. McAuley. “A feasibility study of an in-the-wild experimental public access wifi network”. In: *Proceedings of the Fifth ACM Symposium on Computing for Development*. 2014, pp. 33–42.
- [SBP02] Y. Sun, E. M. Belding-Royer, and C. E. Perkins. “Internet connectivity for ad hoc mobile networks”. In: *International Journal of Wireless Information Networks* 9.2 (2002), pp. 75–88.
- [SK03] M. Seshadri and R. Katz. *Dynamics of simultaneous overlay network routing*, UC Berkeley. Tech. rep. CSD-03-1291, 2003.
- [SRS12] Sonam Palden Barfunga, P. Rai, and H. K. D. Sarma. “Energy efficient cluster based routing protocol for Wireless Sensor Networks”. In: *2012 International Conference on Computer and Communication Engineering (ICCCE)*. 2012, pp. 603–607.
- [SSK97] S. Seshan, M. Stemm, and R. H. Katz. “SPAND: Shared passive network performance discovery”. In: *USENIX Symposium on Internet Technologies and Systems*. 1997, pp. 1–13.
- [Sun+13] B. Sun, X. Shan, K. Wu, and Y. Xiao. “Anomaly Detection Based Secure In-Network Aggregation for Wireless Sensor Networks”. In: *IEEE Systems Journal* 7.1 (Mar. 2013), pp. 13–25. ISSN: 1932-8184. DOI: 10.1109/JSYST.2012.2223531.
- [tc-] tc-netem. *Traffic controller NetEm, Linux*.
- [TSC03] Y.-C. Tseng, C.-C. Shen, and W.-T. Chen. “Integrating Mobile IP with Ad Hoc Networks”. In: *Computer* 36.5 (May 2003), pp. 48–55. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1198236.
- [Veg+12] D. Vega, L. Cerdà-Alabern, L. Navarro, and R. Meseguer. “On the topology characterization of Guifi.net”. In: *International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2012, pp. 389–396. DOI: 10.1109/WiMOB.2012.6379103.
- [Veg+15] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, and L. Navarro. “A technological overview of the guifi.net community network”. In: *Computer Networks* 93 (2015), pp. 260–278.

- [Wan+17] X. Wang, D. Qu, K. Li, H. Cheng, S. K. Das, M. Huang, R. Wang, and S. Chen. "A flexible and generalized framework for access network selection in heterogeneous wireless networks". In: *Pervasive and Mobile Computing* 40 (2017), pp. 556–576.
- [XJJ15] H. Xu, L. Ju, and Z. Jia. "Enhance internet access ability for ad hoc network with on-demand gateway broadcast strategy". In: *International Journal of Wireless Information Networks* 22.4 (2015), pp. 415–427.
- [Xu+19] H. Xu, Y. Zhao, L. Zhang, and J. Wang. "A bio-inspired gateway selection scheme for hybrid mobile ad hoc networks". In: *IEEE Access* 7 (2019), pp. 61997–62010.
- [XV07] D. Xia and N. Vljajic. "Near-Optimal Node Clustering in Wireless sensor Networks for Environment Monitoring". In: *21st International Conference on Advanced Information Networking and Applications (AINA '07)*. May 2007, pp. 632–641. DOI: 10.1109/AINA.2007.97.
- [Yan+13] Y. Yan, L. Ci, Z. Wang, and W. He. "QoS-based gateway selection in MANET with Internet connectivity". In: *2013 15th International Conference on Advanced Communications Technology (ICACT)*. IEEE. 2013, pp. 195–199.
- [YF04] O. Younis and S. Fahmy. "HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks". In: *IEEE Transactions on Mobile Computing* 3.4 (Oct. 2004), pp. 366–379. ISSN: 1536-1233. DOI: 10.1109/TMC.2004.41.
- [YG19] M. Yemini and A. J. Goldsmith. "Virtual Cell Clustering with Optimal Resource Allocation to Maximize Cellular System Capacity". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2019, pp. 1–7.
- [Zam+16] R. U. Zaman, M. Waseem, A. Farokhi, A. V. Reddy, et al. "Traffic priority based gateway selection in Integrated Internet-MANET". In: *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*. IEEE. 2016, pp. 18–21.
- [ZC08] F. Zeng and Z. Chen. "Load Balancing Placement of Gateways in Wireless Mesh Networks with QoS Constraints". In: *2008 The 9th International Conference for Young Computer Scientists*. 2008, pp. 445–450. DOI: 10.1109/ICYCS.2008.15.

- [Zha+17a] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. “Resilient Datacenter Load Balancing in the Wild”. In: *Special Interest Group on Data Communication (SIGCOMM)*. 2017, pp. 253–266.
- [Zha+17b] H. Zhang, J. Zhang, W. Bai, K. Chen, and M. Chowdhury. “Resilient Datacenter Load Balancing in the Wild”. In: *Special Interest Group on Data Communication (SIGCOMM)*. 2017, pp. 253–266. DOI: 10.1145/3098822.3098841.
- [Zha+18] J. Zhao, Q. Liu, X. Wang, and S. Mao. “Scheduling of Collaborative Sequential Compressed Sensing Over Wide Spectrum Band”. In: *IEEE/ACM Transactions on Networking* 26.1 (Feb. 2018), pp. 492–505. ISSN: 1063-6692. DOI: 10.1109/TNET.2017.2787647.
- [ZSL13] T. Zhang, R. Safavi-Naini, and Z. Li. “ReDiSen: Reputation-based secure cooperative sensing in distributed cognitive radio networks”. In: *2013 IEEE International Conference on Communications (ICC)*. June 2013, pp. 2601–2605. DOI: 10.1109/ICC.2013.6654927.