UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Connectivity sharing for wireless mesh network

Khulan Batbayar

Connectivity Sharing for Wireless Mesh Network

Khulan Batbayar

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

September 2021

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 Politecnica de Catalunya, Spain
Pr. Ramin Sadre **Co-supervisor**, UCLouvain/ICTEAM, Belgium

Connectivity Sharing for Wireless Mesh Network

by Khulan Batbayar

Acknowledgments

This thesis has been a challenging and enriching 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. Their understanding and expertise in my area of research significantly improved the contents of this thesis,

I would like to thank my first advisor, 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.

I would like to thank my second advisor, Ramin, for the patience and confidence that you placed in me. He let me pursue my ideas freely and provide consistent guidance. Thank you for the countless hours dedicated to my research. My thesis has benefitted substantially from his insightful recommendations.

I am grateful to my colleagues at Distributed Systems Group at UPC, Mennan 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. I am thankful for the friends I made during the journey, Sana Imtiaz, Zainab Abbas, Dur Zahra, Atika Zulfiqar, Rosana Veroneze, Juhee Bae, Francois Aubry, Michael Saint-Guillain, Igor Zavalyshyn, Paolo Laffranchini.

I thank the Mongolian community in Barcelona for taking care of Iveel 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 would like to thank my family for their support, love, and sacrifice. For my son, Iveel, thank you for being my motivation and source of energy. I apologize for all the times we spent together on UPC Lab C6-E208, for all the times I could not spend with you while I was working, went away for conference and research exchange. And for my sister, Javkhlan, thank you for having my back, coming to the rescue, believing in me without any doubt. Your support has meant more to me than you could possibly realize. For my parents, Batbayar and Gereltuya, I am forever indebted to you for making me who I am today. For my boyfriend, Raziel Carvajal, thank you for your emotional support, professional feedback, understanding, and being my best friend.

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. There is no doubt that the network would benefit from a gateway selection mechanism that can provide good connectivity to the client node 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 perceived performance of the gateways changes frequently and might depend on the location of the client node 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 can outweigh the benefits of the dynamic gateway selection.

The thesis's 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: 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 5
1.3 Contributions 6
1.4 Publications 7
1.5 Thesis structure 9

2 Background and Related work **11**

2.1 Background 11
2.2 Related work 13

3 General framework: Sense-Share-Select framework **19**

3.1 Introduction 19
3.2 Objectives 21
3.3 Overview of the framework 22

4 Sensing layer **25**

4.1 Introduction 25
4.2 Design requirements 25
4.3 Design overview 26
4.4 Results 30
4.5 Conclusion 34

5 Collaborative layer **35**

5.1 Introduction 35
5.2 Design requirements 36
5.3 Design overview 37
5.4 Results 44
5.5 Conclusion 57

6 Selection layer 59
6.1 Introduction . . . 59
6.2 Design requirements . . . 60
6.3 Design overview . . . 62
6.4 Results . . . 67
6.5 Conclusion . . . 75

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 1
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 Coping algorithm example 16
2.2 Vivaldi based algorithm example 17

3.1 The layered architecture of Sense-Share-Select framework . . 20
3.2 Example of Sense-Share-Select collaboration 22
3.3 Sense-Share-Select framework structure 23

4.1 Experiment with different gateway performance measurements 27
4.2 Total gateway measurements 32
4.3 Sensitivity analysis of RTT threshold value and size of close neighbors cluster 33
4.4 Client perception of the selected gateway's performance. . . . 34

5.1 Example scenario for forming a group of collaborating nodes . 36
5.2 Layered structure of the collaborative algorithm 38
5.3 Bootstrapping and periodic sensing phase 39
5.4 Performance table obtained through collaborative sensing . . 41
5.5 Layered sensing framework 42
5.6 Network topology for experiments 48
5.7 Similarity results for different collaboration schemes 49
5.8 Sensitivity analysis (nodes leaving/joining) 50
5.9 Gateway performance change (Collaborating with all) 51
5.10 Gateway performance change (Collaborating with *TopK*) . . . 52
5.11 Collaborating node change 52
5.12 Faulty nodes effect (Collaborating with all) 53
5.13 Faulty nodes effect (Collaborating with *TopK*) 54
5.14 Behavior of the faulty nodes 54
5.15 Sensitivity towards the number of service nodes 55
5.16 Gateway selection result 57

6.1 Selection algorithm design 63
6.2 Estimated gateway capacity classification 64
6.3 Gateway latency variance. 66
6.4 Experiment network topology. 67
6.5 Wireless - Performance prediction result. 69
6.6 Client node distribution over the gateway nodes. 70
6.7 Wireless gateway distribution 70
6.8 Number of gateway selection changes in 100 rounds. 71
6.9 Download time through the gateway nodes. 71
6.10 Wireless - Client node distribution with gateway delay. 72
6.11 Download times with different gateway delays 73
6.12 Wireless - Download size sensitivity. 74
6.13 Wireless - Gateway distribution. 74

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 84
7.8 Number of gateway changes 85
7.9 Download time 87
7.10 Node unavailable test 88
7.11 In the presence of faulty nodes 89
7.12 Collaborative sensing accuracy 90
7.13 Download latency 91
7.14 Wireless experiment distribution 91
7.15 Full collaboration vs full isolation 93
7.16 Experiment with different bandwidth 94

List of Tables

3.1 Example of gateway performance table at each node 23

4.1 Gateway performance categories 28
4.2 Message overhead per selection round. 31

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. 46
5.2 Average message exchange at individual node per measurement round 50
5.3 Sensitivity towards the RTT threshold vs increasing the number of samples 56

6.1 Partial gateway performance table, every round 62
6.2 Gateway performance categories 64
6.3 Example gateway performance table after Step 1 64

Introduction 1

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

There are many initiatives to connect the unconnected communities to the Internet through fiber optics, Internet hotspots, mobile broadband service, Google's Project Loon [Loon], Facebook Connectivity initiatives [FB-Con]. However, most of them are technologically unattainable, financially infeasible, and not self-sustainable. All these initiatives are still not enough for the global demand for affordable Internet access. Establishing wireless networks in the areas lacking connectivity is the most popular Internet connectivity approach. In the last years, Wireless Mesh Network (WMN) infrastructure has been widely adopted as a cost-effective technology to provide large-scale Internet access networks.

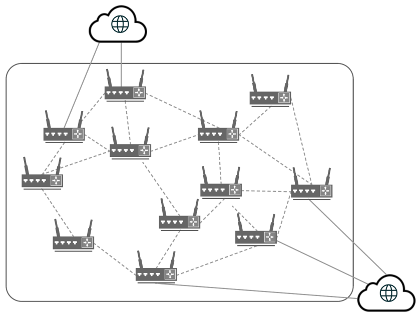


Figure 1.1: Internet gateway in the network

The Internet access in large-scale WMN is arranged through Internet gateways (see Figure 1.1). The Internet gateway is a node in the network that connects the local network nodes to the Internet. In WMN, the mesh nodes are not all connected to the Internet, but specific nodes connected to the Internet act as an Internet gateway to provide access to the others. A good example of such networks is Wireless Community Network (WCN) or Do-It-Yourself access networks are gaining more popularity for self-provision of Internet access in underserved regions [Rey+13]. WCN allows community members to build their local network infrastructure by utilizing over-the-counter equipment with minimal technical expertise and provides various free, neutral, secure communication services (including shared Internet access). There are many successful Community Network (CN)s uses Internet gateways such as guifi.net community network [guifi] (>36000 nodes, 450 Internet proxies/gateways), Roofnet [Bic+05](36 mesh nodes), Freifunk community network [HDS17], and many more. Such large-scale networks are built with heterogeneous devices, and the Internet gateway acts as a translator between infrastructure-less local networks to the Internet. We focus on the large-scale Wireless Mesh Network with mesh nodes sharing their spare Internet connectivity through Internet gateways.

All the traffic to/from the Internet passes through the Internet gateway nodes; often, the number of gateways is comparatively smaller than the number of client nodes. As the network grows, the traffic load at the Internet gateways increases immensely, and gateways become susceptible to congestion, failure, and overcrowding. As the demand for the gateway node increases, the Internet connectivity perceived at the client nodes can become excruciatingly slow. 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. 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]. Internet gateways are configured automatically by Internet Service Provider companies, common in the home Internet access. There are multiple gateways available in the large-scale network, and choosing the “right” gateway can improve the Quality of Service perceived at the client node.

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 provide the optimal selection, but there is a gap to achieve good Internet gateway selection with low cost and fair distribution. Here are the following problems we are addressing in this thesis:

1.1.1 Static gateway selection

The common practice in the network is to configure the Internet gateway is to associate the default gateway given by the ISP companies or, in the CN's example, the default gateway in the region. The manual gateway selection, more likely all the nodes in the same locality select the default gateway even though there are equally good performing gateways that exist. In guifi.net, the nodes choose the gateway depending on the locality (zone), popularity, previous experience, and word-of-mouth knowledge. The gateway distribution imbalance is inevitable since the gateways are configured on the client-side without prior knowledge of their performance. Often gateways have limited capacity to handle client requests as the demand for the proxy increases, and the performance is likely to degrade.

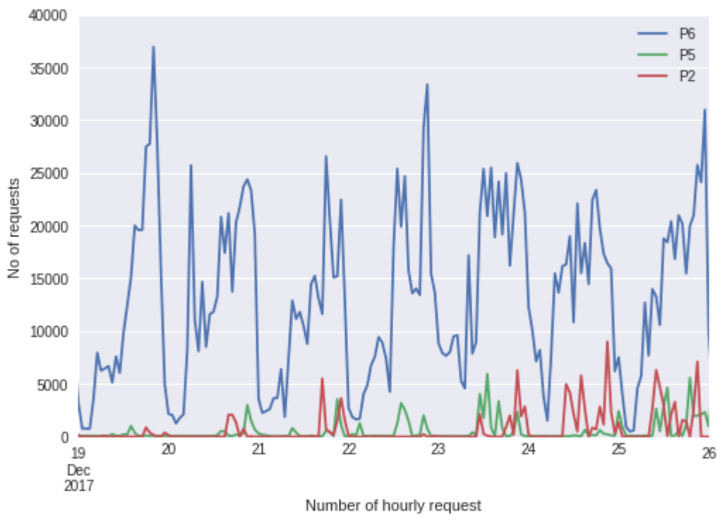


Figure 1.2: Hourly requests passed through gateways

In Figure 1.2, we show the total number of hourly requests that pass through three different gateways in the guifi.net for a one-month duration. The log is collected anonymously at the gateway node. From the result, the blue gateway is more popular as the number of requests per hour is noticeably high, while the green and the red gateways remain barely used. Therefore, there is a clear imbalance between the available gateways in terms of selection.

1.1.2 Frequent gateway performance changes

Gateway performances change frequently depending on the request size, the number of users, and the time of the day. In Figure 1.3, we measured five different guifi.net gateway latency to download 0.1MB file from the dedicated server on the Internet every 2 minutes. From the result, we see that the performance of the green and red gateways is inconsistent. Also, the best gateway selection is changing over time. Therefore, to select a good-performing gateway, the nodes should keep up with the latest gateway performances. The

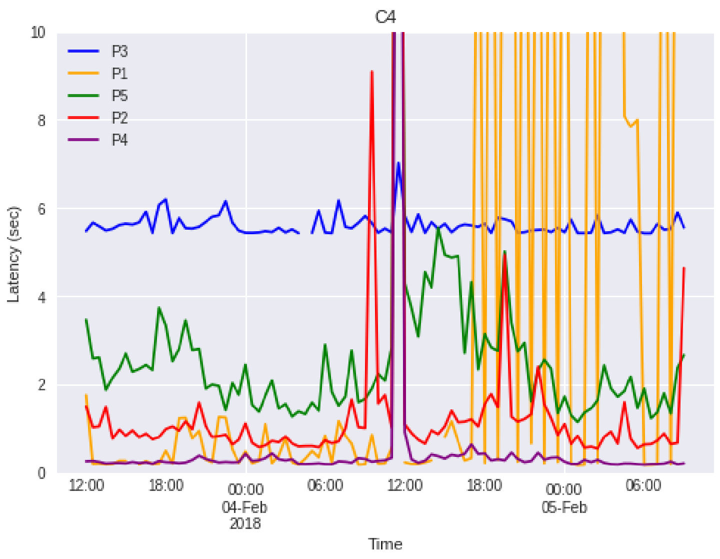


Figure 1.3: Guifi.net gateway latency variance

best-performing gateway should be selected based on periodic performance monitoring. Active performance monitoring becomes costlier as the network size grows. Every node's gateway performance perception is different; therefore, advertising gateway performance regularly is unsuitable. On the other hand, monitoring the performance of all gateways by the client nodes is costly and can become an essential source of network traffic congestion as the network size grows.

1.1.3 Unfair distribution of gateway usage

The gateway selection algorithms often choose the best-performing gateway to achieve a good result. The best selection leads to the situation called "herd behavior," where most nodes select the same gateway and quickly degrade its performance and then move to the next best option. Doing so creates an imbalance in the gateway distribution and short-term performance degradation. Even in the load-balanced selection algorithms, the nodes select the least loaded gateway.

Consider the selection scenario shown in Figure 1.4, $G1, G2, G3$ gateways are given with its current performance and the load. There are $C1, C2, C3, C4$

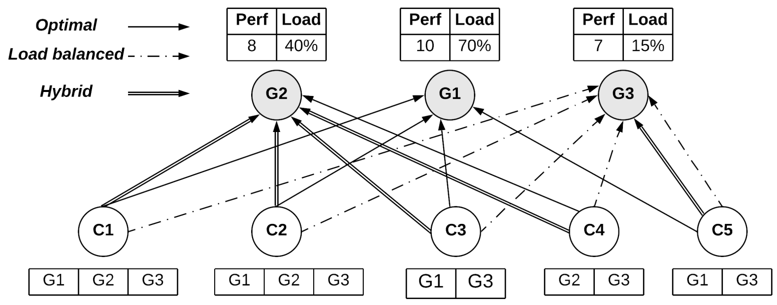


Figure 1.4: Herd behavior in selection example

nodes in the network; each has a different gateway measurement list. With the performance-oriented selection, all nodes select *G*1, whereas the nearly equally well-performing *G*2 is selected only by *C*4. Any client node does not select *G*3 although it is less loaded. With the load-oriented selection, the client nodes try to select the least loaded gateway node. In this case, the clients select *G*3 even though its performance is the lowest. Let us consider a hybrid gateway selection that gives equal weight to both metrics. In this case, the majority of the client nodes select *G*3. The main problem in choosing the locally optimal gateway at the time of selection is that other nodes choose the same gateway node. In every selection example, the algorithm favors some gateways over others. The greedy selection is not an ideal selection scenario for the end-users as the frequent gateway selection change creates short-term performance jitters. Load balancing focused selection algorithms [HLT04; Nan+06] propose algorithms to calculate the load of the gateway nodes, but in the end, select the least loaded gateway node and suffer from herd behavior.

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 the best-effort Internet connectivity in a shared Internet access network for all the network nodes, meanwhile achieving an overall fair (i.e., balanced) distribution of the nodes on 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. Finally, as one of the most concerning features in collaborative performance monitoring, it should provide accurate measurements.

1.3 Contributions

Main contribution of our thesis can be summarized as follows:

C1 **Client-side, lightweight gateway selection framework.** We facilitate each node to be involved in their gateway selection process to provide customized gateway selection. The nodes are responsible for gateway monitoring, collaboration, and selection. The framework uses simple calculations to make the framework fit for heterogeneous network devices. The collaborative monitoring algorithm focuses on reducing inter-network communication and reusing the 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 collaborative monitoring algorithm** A middle layer of our framework uses the collaborative performance sensing algorithm. We proposed a collaborative performance monitoring algorithm with our twist. Instead of collaborating with a wider, fixed number of nodes, each node selects its collaborator nodes. Doing so reduces the range of collaboration of each node, a.k.a, reducing the inter-node communication while utilizing other measurements for their benefit. In a collaborative monitoring algorithm, there is a chance that other nodes might send malicious/wrong measurements to lure nodes to other gateways. However, in our monitoring algorithm, a node collaborates periodically with others. Therefore, it has a collaboration history to judge the behavior of trusted neighbors. We demonstrate that close neighbor collaborative monitoring provides >90% accurate performance measurement 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.** Building upon the accurate measurements from Chapter 5, the Internet gateway selection focused on the best-effort, balanced gateway selection. We studied that optimal gateway selection algorithms often lead to over-popularization and equally degrade all selected nodes' Internet performance. Often in decentralized, bottom-up networks such as community networks, the gateways are selected manually based on previous experience, popularity, or word-of-mouth knowledge of the gateways. By examining the anonymous log files of the guifi.net gateways as well as shown in [Dim+17b], many gateways are not selected by the nodes because they are not well known or a bit further away from the user node. Our selection algorithm gives an equal opportunity for the gateways to perform well but not the best. Results show that our selection algorithm result shows our selection algorithm gives a better Internet latency compared to selecting the best performing gateway at a time.

C4 **Maintain a long-term, stable selection.** With the manual gateway selection, the selected gateway does not change unless the gateway becomes unresponsive. In this case, the node moves on to the next gateway on the web browser list. State-of-the-art gateway selection algorithms do not guarantee future performance. Therefore, we implemented the feature in the framework to categorize the gateways based on their performance capacity and avoid selecting the unstable gateways. This feature allows nodes to have stable, good Internet connectivity and avoid changing the gateways frequently.

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

C5 **Production network implementation.** We implemented the gateway selection framework in the guifi.net CN environment with testbed network nodes and production network gateways with real background traffic. The implementation result is consistent with the Mininet emulation network result. We designed different scenarios in the testbed user nodes to test our framework's adaptability and resilience.

The main results related to this contribution are presented in Chapter 7, and the implementation and the results 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 the RIMO [Bat+18]'s performance sensing and adds close neighbors' collaborative sensing. The experiments are designed in the community-lab [1] 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 covers the fault tolerance and implementation of the performance-based sensing. The experiments were designed 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 the wired, wireless, and wireless mobile network environment to prove the compatibility. (CORE2020 RANK B)

Pending review

P5 **"Sense-Share-Select: Sharing the access to the Internet within Community Network"**, IEEE Access, 2021, 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

[1]https://community-lab.net/

W2 **Sense-Share: Resilient collaborative service performance monitoring framework**, 8th Networking Networking Women Workshop (N2Women), 2019

1.5 Thesis structure

The thesis is structured as follows:

Chapter 3 introduces the general structure of the framework and design requirements for the whole architecture.

Chapter 4 presents the *Sensing layer* algorithm, focusing on the individual node's gateway performance sensing, how to reduce the number of gateway performance monitoring requests. The algorithm is compared with the distributed performance sensing, the Brute-Force performance sensing algorithm. The effectiveness of the algorithm is shown in the Experimental result section. The collaboration algorithm further reduces the collaboration within more similar collaborators, which in turn increases the accuracy of the collaborative measurements and reduces the in-network communication (message exchange).

Chapter 5 proposes the *Collaborative layer* algorithm, which manages the sending/receiving measurements, finding the collaborator nodes, and building the gateway performance table. The trust-based collaborator ranking and performance smoothing components are integrated with the proposal.

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

Chapter 7 is the final chapter where it covers the integration of the Sense-Share-Select framework in the guifi.net environment with production gateway nodes. The framework is tested under different network uncertainties 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

2.1.1 Internet gateway selection

The large-scale Internet network has multiple Internet gateways to distribute the connectivity, and network nodes should select one of the gateways to connect to the Internet. 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 network which contains 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 the word-of-mouth knowledge. The advantage of the 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 Internet gateway selection algorithms focus on changing the gateway selection based on the network dynamics such as gateway performance, load balance, and node mobility, etc. Dynamic gateway selection algorithms are divided further into centralized gateway selection and distributed gateway selection. **Centralized gateway selection algorithm** is where the central entity recommends gateway to the nodes [GRS08; HMW09]. The algorithm proposed in [GRS08] uses central 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 do not provide 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, collaborative performance monitoring algorithms to divide the monitoring tasks among the nodes. Distributed gateway selection algorithms provide the best gateway selection tailored for network nodes. However, the selection algorithm has more cooperation/coordination between nodes (a.k.a, more network management), more communication cost, and is not suitable for scaling.

2.1.2 Collaborative performance monitoring

The main goal of the Internet 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. The large-scale network gateways are susceptible to frequent performance fluctuations depending on the amount of traffic that passes through them, rush hours, and the locality of the gateway.

The periodic, active performance monitoring should be done at each node that 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 distributing the performance monitoring tasks and sharing their measurements among other nodes. Instead of increasing the gateway monitoring traffic, the collaborative monitoring algorithms utilize the communication between the nodes to share measurements. The collaborative monitoring algorithm's performance is measured by the accuracy of the measurements collected at each node. There are two different collaborative monitoring algorithms, centralized and the distributed collaborative monitoring.

Centralized collaborative monitoring algorithms [HCB00; HCB02; YF04; Lia+16; SRS12; FM02; ZC08] often involve a clustering mechanism where the group of nodes selects a representative node (cluster head) to measure the performance of the gateways and distribute it to the others. Each node adjusts the measurement based on the relationship between the cluster head. Centralized algorithms reduce the monitoring cost, but the accuracy of the measurements is often not high. Meaning the performance measurements received at each node end are not tailored to them. 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 and over-popularize it.

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

2.1.3 Gateway discovery

A few conventional ways gateway discovery has been done and mostly done in the network with mobile nodes. **Proactive gateway discovery** [TSC03; Jon+00; SBP02] where 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 gateway list or dismiss it. This way is expensive, generating periodic mass gossiping where some nodes might not be interested. **Reactive gateway discovery** [BMJ99] is a reverse procedure of the proactive discovery when the node initiates the discovery process. The node generates a Gateway Solicitation (GWSOL) message, and gateways that hear this message replies 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** [RK03; Lee+03; Bin+05] algorithms proposed in the literature that combines both proactive and reactive gateway discovery. A 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.

Our target network is the large-scale wireless network such as a community network, where the only mobility is gateway is becoming online/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. The nodes choose their gateway based on their network zone, locality. Therefore, in our framework, we did not cover the gateway discovery component, and this information is the input to the sensing algorithm.

2.2 Related work

2.2.1 Gateway performance sensing

To our understanding, most of these 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. In [Wan+17] the authors present flexible and not only network-based criteria (e.g., economical features or user preferences) for gateway selection. However, it cannot be easily implemented and deployed on a heterogeneous and distributed community network because it is a conceptual proposal based on game theory. 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] proposes 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 with the large-scale network, especially, Brute-Force performance sensing algorithms [Sal+18; Aou+06] where all nodes measure the performances individually. Other algorithms [Ko+13; Abu+15; AAJ09b] uses representative nodes to measure the performance, and other nodes request the measurement from the node, 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 common 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 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 distribute amongst each other. The works represented in [Bat+19a; Dim+17b] have similar features to our proposed framework and in the collaborative Internet gateway selection domain, and we explained each algorithm in detail later in this section. Collaborative algorithms are susceptible to the 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 the Sense-Share-Select framework to encourage the nodes to select their trusted neighbors while staying away from the different nodes.

2.2.3 Gateway selection

In [Wan+17], 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]. However, 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 a 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]. 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 used for the load balancing algorithm such as Load Balancing Index in [HLT04] using total traffic and the gateway capacity, in [Nan+06] using queue length and hybrid load measurements [AAJ09b; Yan+13] combining gateway load, path quality, number of hops, so on. 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 and not suitable for the large-scale network. Cluster-based resource allocation algorithms [SRS12; Lia+16; YG19] reduces the complexity of the load balancing, resource selection and often used for the resource-constrained network scenario. In [SRS12] proposes the energy-efficient resource allocation considering the cluster node's residual energy. In [Lia+16] uses two-step resource allocation consists of resource block allocation and game-based power allocation for the sensor nodes. Besides the clustering complexity, cluster-based algorithms propose a top-down approach where cluster head nodes decide the resource allocation.

2.2.4 Compared algorithms

Following algorithms are most related to our thesis and we refer these algorithms to compare our results.

2.2.4.1 Coping algorithm

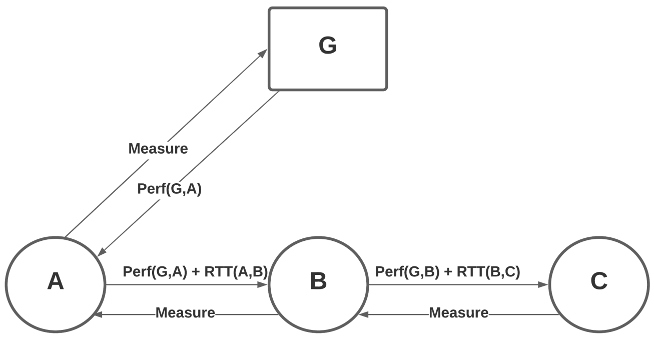


Figure 2.1: Coping algorithm example

CoPing algorithm [Ko+13] stands for C̈ooperative Pingänd designed for the 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 they obtain gateway measurements from their upstream node as shown in Figure 2.1. For this example scenario, node C wants to know the 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 measurement from the upstream node, each node update 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 node and the gateway. The measurement frequency in the experiment is every 5 seconds, and the gateway performance is measured by the ping latency.

The downside of the CoPing algorithm is if any node becomes offline or any other topology change requires a reorganization between the nodes. The accuracy of the collaborative sensing reduces as the number of intermediary nodes increases; therefore, not suitable for a multi-hop mesh network like the guifi.net CN environment.

2.2.4.2 Synthetic coordinate based gateway selection algorithm

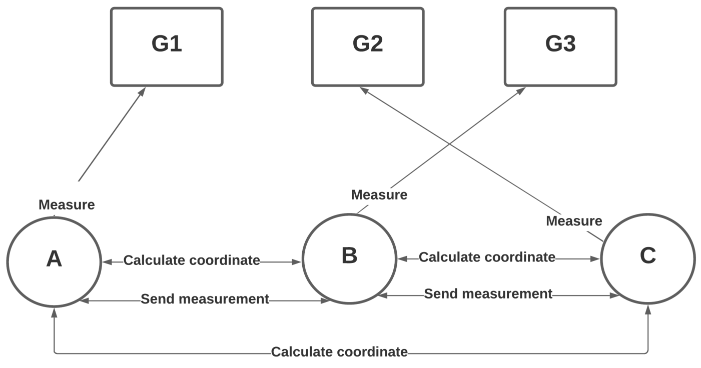


Figure 2.2: Vivaldi based algorithm example

Vivaldi algorithm-based collaborative Internet gateway selection algorithm is proposed in [Dim+17b]. Vivaldi algorithm [Dab+04] creates a network-wide coordinate system by assuming the distance between 2 nodes' coordinates will accurately measure their latency between them. By doing so, every node can determine the distance (latency) between any node. The algorithm incorporates proactive random node probing and adjusting the synthetic coordinates of the network nodes slowly. From the Vivaldi paper [Dab+04], the accuracy of the synthetic coordinate system stabilizes from the 100th rounds, thus a slow start solution. In Vivaldi-based Internet gateway selection algorithm [Dim+17b] utilize Vivaldi algorithm and combine gateway nodes and all the other nodes in the network to create a whole synthetic coordinate system. After creating the synthetic coordinate system, every node knows the performance of the gateway without measuring it, which is the main advantage of using the Vivaldi algorithm in [Dim+17b] (shown in Figure 2.2).

However, the gateway performances change frequently; thus, continuous adjustment of the synthetic coordinates is necessary. The result in [Dim+17b] shows that gateway estimation error is low and over 80% of the experiment estimation has lower than 2.5ms median error.

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

General framework: Sense-Share-Select framework

3

3.1 Introduction

The gateway is a node in the network (router) that provides access to different networks. The Internet gateway is a specific gateway node where all the traffic to the Internet passes. The Internet gateways regulate the network traffics directed towards the Internet and act as a translator between Internet Service Providers (infra-structured network) and the heterogeneous network nodes (infrastructure-less network). In the home network setting, the internet gateway is assigned by default by the Internet Service Provider. Nevertheless, this is not the case in the large-scale network 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 sections (we call it layers) in the Sense-Share-Select framework, shown in Figure 3.1 that seamlessly support the gateway selection. The main idea of the Sense-Share-Select is to enable nodes to make an informed, individual Internet gateway selection with the help of collaborative performance monitoring. The large-scale network is heterogeneous, having different topology, devices, localized network planning. Therefore, the framework is designed to be infrastructure-agnostic and easily integrated with the existing network. For example, guifi.net decentralized community network in Spain (explained in Chapter 7) already has 36000 nodes utilizing approximately 400 internet gateways, and the existing users are often reluctant to new changes, making the implementation process slow. Therefore, we designed our framework to work independently, but our algorithm shines

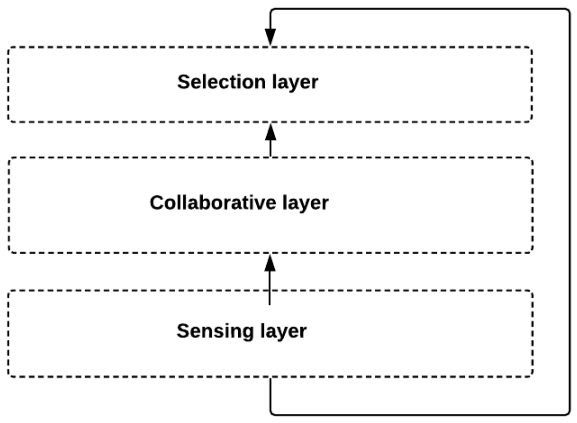


Figure 3.1: The layered architecture of Sense-Share-Select framework

when collaborating with the other nodes. The framework runs periodically in rounds to measure, share and select the gateway for each node. The bottom layer is the *Sensing layer*, responsible for providing an individual gateway performance monitoring and manages 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, receiving measurements. The upper layer is the *Selection layer* to finalize and provide the gateway selection to provide the best-effort gateway node while maintaining the global fair distribution of gateways.

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

1. **Gateway discovery** - The gateway nodes in the large-scale network have prior planning, and the network under consideration is mostly static (non-mobile). The gateway nodes are often known to the public through the registry service. However, there are plenty of algorithms in the literature that focus on gateway discovery through broadcasting, node registry services for the other type of networks.

2. **Node discovery** - Similarly with the gateway discovery, the neighbor node discovery is not part of the framework. The available nodes are given as input to the framework. For example, guifi.net and Freifunk networks have a node registry service that provides 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 a more extended period. To achieve this objective, we have the following mini objectives.

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 short. 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. The general objective of each node is to utilize its capacity for improving the visibility of the gateway performances and not to strive to achieve the complete performance measurement. At the performance monitoring level, each node should think about their performance monitoring without relying on the others while not creating too much congestion at the gateways.

3.2.2 Accurate collaboration

With the low-cost performance monitoring, each node is equipped with producing its performance measurement. Usually, in the large-scale network setting, the density of nodes is high, and the nodes in the same locality 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 performances at a time and allows them to choose the appropriate gateway. Many collaborative monitoring algorithms have to choose between accuracy and management overhead. We propose a simple, local collaboration with low overhead; less scheduling yet achieves high accurate monitoring.

3.2.3 A good gateway selection

The best gateway selection is not an ideal solution (explained previously), as they overload specific gateways. 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 gateway 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 not the best performing gateway. The manual gateway selection leads to the uneven distribution of the gateways. 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, most of the physically close nodes will share a similar path towards the gateway nodes, which means their performance perception of the gateway nodes is similar. If the node can fine-tune their similar neighbors, the selected close neighbors will have very similar measurements. Therefore, nodes can use each other's measurements as if their own without modification, simplifying the reorganization process at the node. This is the main foundation of the Sense-Share-Select framework. The 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.

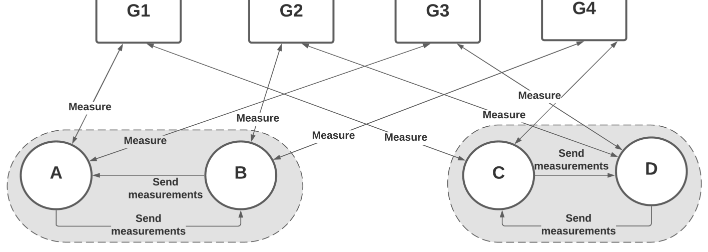


Figure 3.2: Example of Sense-Share-Select collaboration

Figure 3.3 shows the detailed structure of the Sense-Share-Select framework. Every round, the node sense gateway performances, share measurements with the collaborator nodes, receive measurements from the others and select the gateway node. Every node works for their own benefit to receive the best Quality of Service from their selected gateway. The framework allows nodes to work independently on the client node side by choosing the "Good" Internet gateway and avoiding the bad ones. This thesis refers to a *Good* Internet gateway for the client node as the Internet gateway node, which provides good performance (end-user experience) with minimal performance deviation over the extended period.

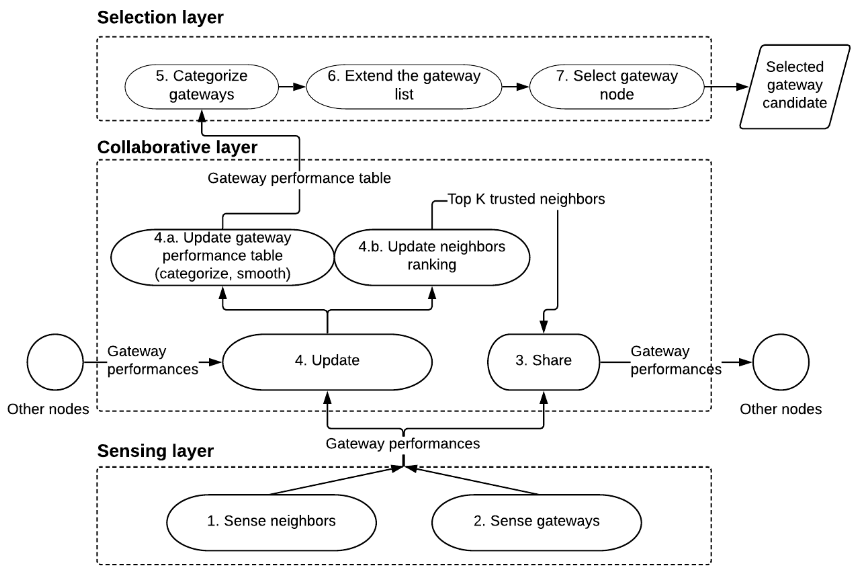


Figure 3.3: Sense-Share-Select framework structure

The framework runs periodically in rounds, measuring the gateway and stores in the *gateway performance table*, similarly shown in Table 3.1.

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

This gateway performance measurement consists of measuring the time (called "latency" in the following) needed to download a 0.1MB resource via HTTP from a file server located on the Internet. The end-users (a.k.a nodes) care about the end Internet quality, and the "Internet latency" is an end-to-end metric to measure the gateway performance accurately. Many state of the art algorithms use network-level metrics such as Round Trip Time, bandwidth, signal-to-noise ratio, packet delivery rate, so on, but they only cover the network path condition. The download file size (0.1MB) is an example; it could be smaller in the production network.

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 where every node measures all of the gateway performances (Brute-Force). However, the cost of the Brute Force measurement is high, especially in large-scale networks. In the end, the node chooses only one gateway as their default Internet gateway and discards other measurements. Thus, plenty of measurements are left unused to provide one gateway[SSK97]. The other extreme is manual gateway selection, where the node does not measure any gateway performance and select the gateway based on the locality, word-of-mouth knowledge, or previous experience. Manual selection leads to load imbalance over different gateway nodes by over-popularizing some gateways and provide 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. Therefore, it is unavoidable to reduce the measurement costs generated by each node to reduce the overall performance measurement cost.

In the *Sensing layer*, we tackle reducing the measurement overhead generated by the network nodes to select a suitable gateway 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 serve Internet requests. 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, requiring a great deal of scheduling and organizing among the nodes.

- **Reuse the previous measurements to better 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. For example, some gateways are too far away and should not even be considered for specific nodes as a selection candidate and, therefore, not to waste resources on it.

In the sensing layer, we aim to find a good balance of the right number of measurements to be done from the client node.

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 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 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 PoTC [Mit01]. In the PoTC algorithm, every entity randomly selects two resources, measures its performances, and selects the best out of two options for the resource allocation and has been used for many resource load balancing algorithms [Gho+15; Zha+17b; Ber+17] and job scheduling [Ous+13] algorithms for balanced resource distribution with minimal complexity.

The *Sensing layer* algorithm is designed where the node monitors gateway performance independently and efficiently (with as few overhead as possible) a gateway node to connect while considering the global network-wide load balancing. Every round, the node selects two random gateway performances and store them in the gateway performance table. The new measurements will be distributed to the close neighbors. As a result, the number of individual performance measurements to the gateways reduces from *n* down to 2 per measurement period, being *n* 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.

The algorithm's objective is to promote the individual node's gateway performance monitoring with low complexity and low cost. *Randomization* is a technique often used for removing the complexity of scheduling tasks in distributed algorithms and is scalable without incurring too much traffic. We employ each node measuring random gateway performance to distribute the overall measurement requests to the gateways per round.

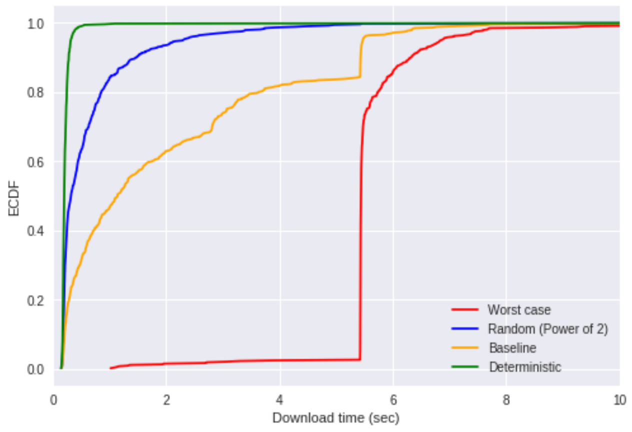


Figure 4.1: Experiment with different gateway performance measurements

We conducted a small experiment with four client nodes in the university lab selecting from 5 different Internet gateways from the production network environment. We ran the Internet gateway selection algorithms for one day. Performance monitoring requests to the five gateways are made every 3 minutes, and for each algorithm, clients request 0.1 MB content from the Internet through the selected gateway every 1 minute. Experiment consists of 4 different gateway selection:

- 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, distributes the loads over the gateways.

- 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.

In the results depicted in Fig. 4.1 shows that the randomized proxy selection algorithm (Power of Two) achieves better results than the worst case scenario and random gateway selection (baseline), and provides download results closer to the Best case scenario.

4.3.2 Capacity based service monitoring

Nodes learn the behavior of gateways from the past measurements and measurements collected from the others. With collected measurements, nodes can remove unreliable (i.e., bad performing or inconsistently performing) service node candidates to lower the monitoring cost. We consider two features to characterize a service node's capacity to provide satisfying performance:

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

The reason behind the categorization is to save the node from monitoring the unnecessary gateway performance. For example, as nodes are 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, or the client is downloading a large file then performs well after some time. In this case, the node cannot eliminate measuring the "bad" gateways. The node assigns

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

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 average service performance value and the average performance deviation, as shaped after every round based on the measurements received from the trusted $TopK$ collaborators, define the threshold value of the *High, Low* category of the service node and performance deviation.

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 service nodes are added into the sensing list of gateways every other measurement round. The *Prototype* applies constraints on the *Bad* and *Inconsistent* service 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

Sense neighbors component provides the list of neighbors to collaborate. 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 neighbor nodes is given as an input for our algorithm; in other words, every node knows the network nodes. However, there are different possible ways for a node to identify its neighbors. 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 registry service. In addition, 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 which is relatively stable in WCN. 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 main challenge in this approach is the fine-tuning of the RTT threshold value. As shown later in the experiments, 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.

4.3.4 Randomized Informed Minimized Overhead algorithm

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.

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 $\mathbf{\Gamma}$ and obtain a subset of gateways $\mathbf{\Gamma_{filtered}}$. The **impact to the network** (HOPStoGateways procedure, line 9) 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: $MaxHOPS + 1 - hops$ (line 12), 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 $\mathbf{\Gamma_{weighted}}$. We use network hops as a quasi-static client-gateway distance, so the result of the algorithm is also quasi-static.

4.4 Results

4.4.1 Experimental setup

We performed experiments in Mininet [LHM10] emulation framework with ten gateways and 50 client nodes. To simulate the performance variations, we used Linux Network Emulator (tc netem) [tc-] by adding 10ms delay for the "Bad" gateways ($g8, g9, g10$), 3ms delay with 1-3ms delay distribution for the "Inconsistent" gateways($g4, g5, g6, g7$) and 1ms delay for the "Good" gateways ($g1, g2, g3$). The gateway performance measurement is done every 2 minutes for 100 rounds.

Algorithm 1 Weighting gateways

```
Input: $\Gamma = \{G_1, G_2, \ldots, G_j\}$                    ▹ Available gateway list
Input: $preferences$                                           ▹ User preferences

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

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

4.4.2 Experiments

4.4.2.1 Overhead

Strategy	Analytical	Experimental
Brute force	$O(size(\mathbf{\Gamma}) * size(\mathbf{C}))$	59.39
Our proposal	$O(size(\mathbf{C}))$	9.95
Random	$O(0)$	0

Table 4.2: Message overhead per selection round.

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.

From the overhead perspective, Table 4.2 shows the network cost of the different selection approaches in terms of messages per selection round, where $\mathbf{\Gamma}$ is the set of gateways and $\mathbf{C}$ 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.

4.4.2.2 Measurement distribution

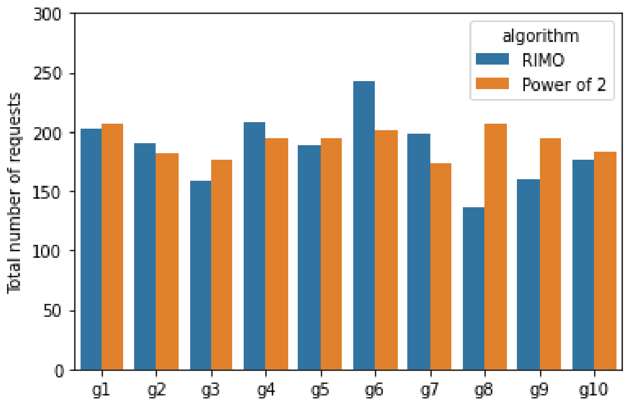
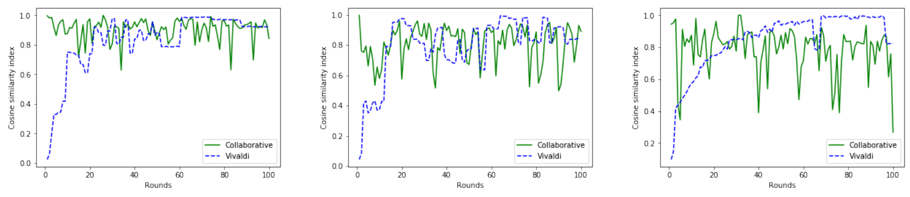


Figure 4.2: Total gateway measurements

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.

4.4.2.3 Accuracy of the RTT threshold

We studied 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 our 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 4.3 shows the precision of



(a) $RTT < 5ms, size = 10$ **(b)** $RTT < 5ms, cluster = 15$ **(c)** $RTT < 5ms, size = 20$

Figure 4.3: Sensitivity analysis of RTT threshold value and size of close neighbors cluster

the gateway performance estimation of our collaborative sensing algorithm and Vivaldi-based algorithm proposed in [Dim+17b]. From Figure 4.3a, 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 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 4.3a. When increasing the close neighbors RTT threshold value in Figure 4.3b and Figure 4.3c, 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 4.3c, the precision of estimation stays constantly above 0.8 duration of the experiment.

4.4.2.4 Download performance of RIMO algorithm

Figure 4.4 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

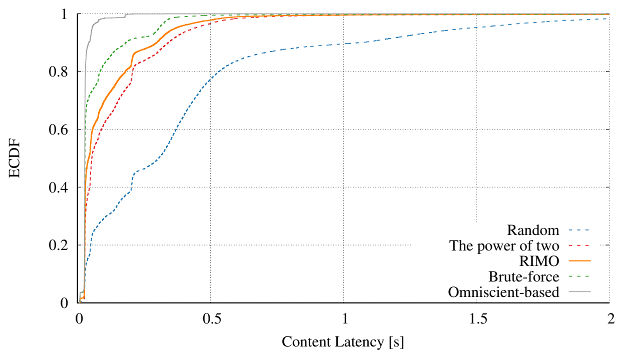


Figure 4.4: Client perception of the selected gateway's performance.

perceives an additional delay when the best gateway is not selected. This 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 own performance sensing coupled with performance-based sensing candidates. The algorithm improves network utilization, achieving low 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 the partially full gateway performance table with the latest performance of the gateway nodes and the list of the close neighbors to share measurements.

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 [**chap:collaborative**] is essential for improving the gateway selection candidates.

Collaborative layer | 5

This chapter explores the design and experiments of the collaborative performance monitoring problem. It is shown that creating a customized close neighbors list to collaborate is the simple 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 from Section 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.

More concretely, the problem of gateway selection in Wireless Community Network, we want nodes to share their gateway performance measurements to reduce the number of measurements needed to make an informed gateway selection. In collaborative monitoring, nodes share measurement results to build a common ground of information while keeping the measurement overhead low. 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 ex-

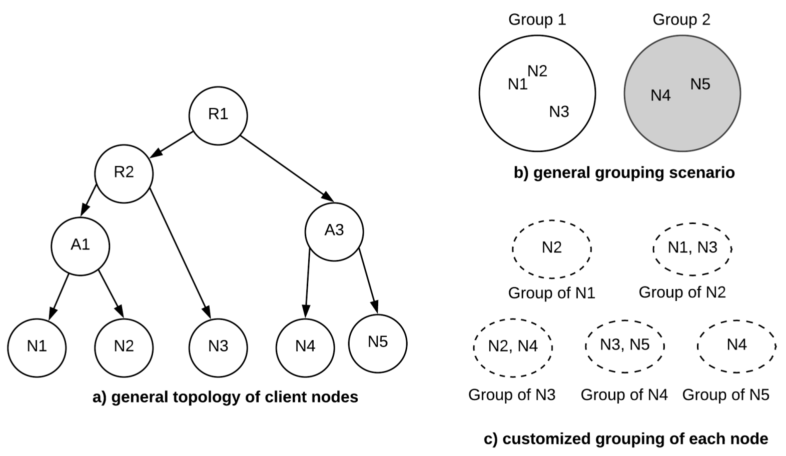


Figure 5.1: Example scenario for forming a group of collaborating nodes

ample, a simple topological grouping applied to the network shown in Figure 5.1a might lead to the two groups of nodes N1–N5 shown in Figure 5.1b. Depending on the characteristics of the network links, the nodes N1, N2 and N3 might see or not see a similar performance of gateway N5. Algorithms like the Synthetic coordinate-based algorithm and CoPing algorithm 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. Inter-node communication creates 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

$$cos(\boldsymbol{M}, \boldsymbol{A}) = \frac{\boldsymbol{M} \cdot \boldsymbol{A}}{||\boldsymbol{M}|| \cdot ||\boldsymbol{A}||} \tag{5.1}$$

where $\boldsymbol{A} = [A_0, A_1, \ldots, A_p]$ is the real performance metric of the p servers that the node would measure without collaboration, and $\boldsymbol{M} = [M_0, M_1, \ldots, M_p]$ 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 distributed collaborative monitoring algorithms generates a lot of traffic within the network, therefore, not scalable. 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 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 services caused by, amongst others, link failure, service 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 reelections.

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. Each node keeps a table, called *gateway performance table*, where it stores the results of its own gateway performance measurements for the different gateways as well as 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 where our main collaborative monitoring logic resides. The top layer selects one gateway from the table of measured gateways.

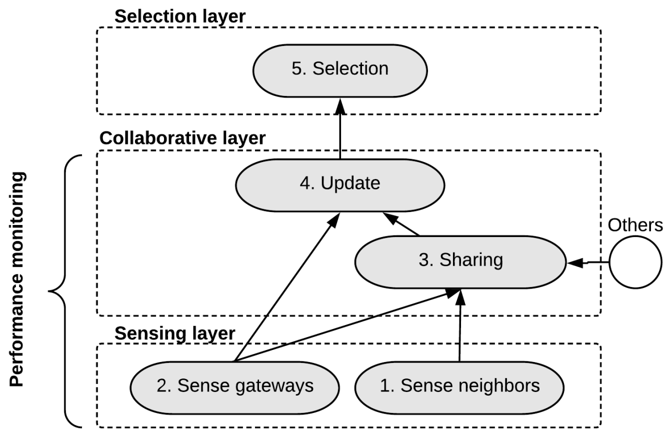


Figure 5.2: Layered structure of the collaborative algorithm

All components of the algorithm run on the client-side, i.e., on the network nodes, and their execution 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 close neighbors, 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

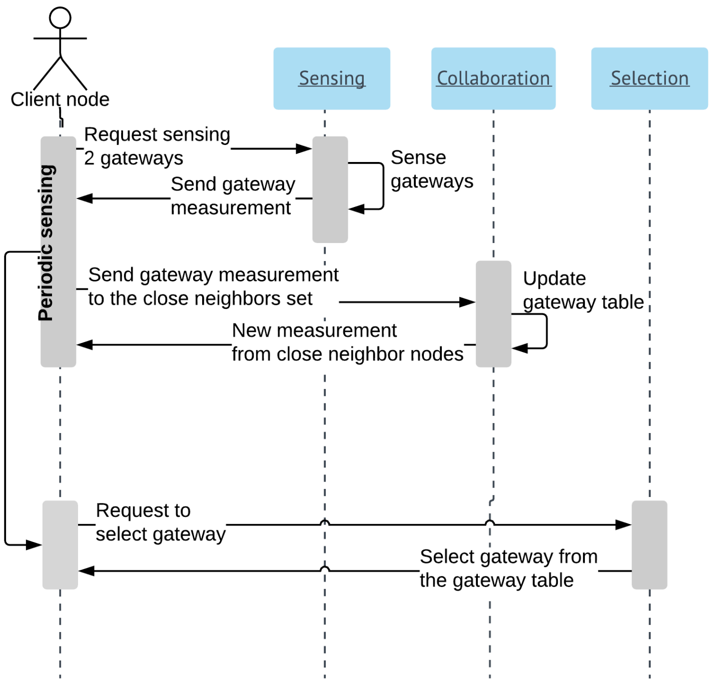

Client node
Sensing
Collaboration
Selection
Periodic sensing
Request sensing 2 gateways
Sense gateways
Send gateway measurement
Send gateway measurement to the close neighbors set
Update gateway table
New measurement from close neighbor nodes
Request to select gateway
Select gateway from the gateway table

Figure 5.3: Bootstrapping and periodic sensing phase

performance table.

Algorithm 2 Collaborating with close neighbors

```
Input: $close\_neighbors = \{C_1, C_2, \ldots, C_k\}$
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 $info.timestamp > gateway\_table[gw\_address].timestamp$
    then
 9:                     $gateway\_table[gw\_address] \leftarrow info$
10:     else
11:         $isNeighbor \leftarrow$ SENSENEIGHBOR($sender$)
12:         if $isNeighbor$ then
13:             goto line 3
14: procedure SEND($message$)
15:     for all $neighbor \in close\_neighbors$ do
16:         SEND_DIRECT($neighbor$, $message$)
```

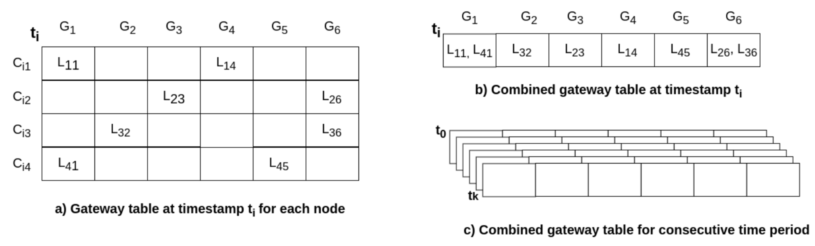
Sending: Whenever a node has performed its performance measurements in the gateway sensing component, the measurement results are directly sent to its close neighbors identified by the neighbor sensing component.

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 13). 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 is 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 6–11). The message is not simply discarded when the sender is unknown to the receiving node (lines 14-17). 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. 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 periodical measurements (Figure5.4c). As can be



t_i
G_1 G_2 G_3 G_4 G_5 G_6
C_{i1} L_{11} L_{14}
C_{i2} L_{23} L_{26}
C_{i3} L_{32} L_{36}
C_{i4} L_{41} L_{45}
a) Gateway table at timestamp t_i for each node
t_i
G_1 G_2 G_3 G_4 G_5 G_6
L_{11}, L_{41} L_{32} L_{23} L_{14} L_{45} L_{26}, L_{36}
b) Combined gateway table at timestamp t_i
t_0
t_k
c) Combined gateway table for consecutive time period

Figure 5.4: Performance table obtained through collaborative sensing

seen, by the above example, the overhead is not optimal: in the case that both the clients C_{i1} and C_{i4} have a similar perception of the performance of G_1 their measurement results L_{11} and L_{41} will be redundant, and there is no need to share them. When sharing the measurement, the node assumes that all collaborators are benevolent, i.e., no node distributes false information. This assumption can result incorrect 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 service instance.
- A node could also be unfair and try to maximize its access to a particular service instance by leading other nodes away from it.

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

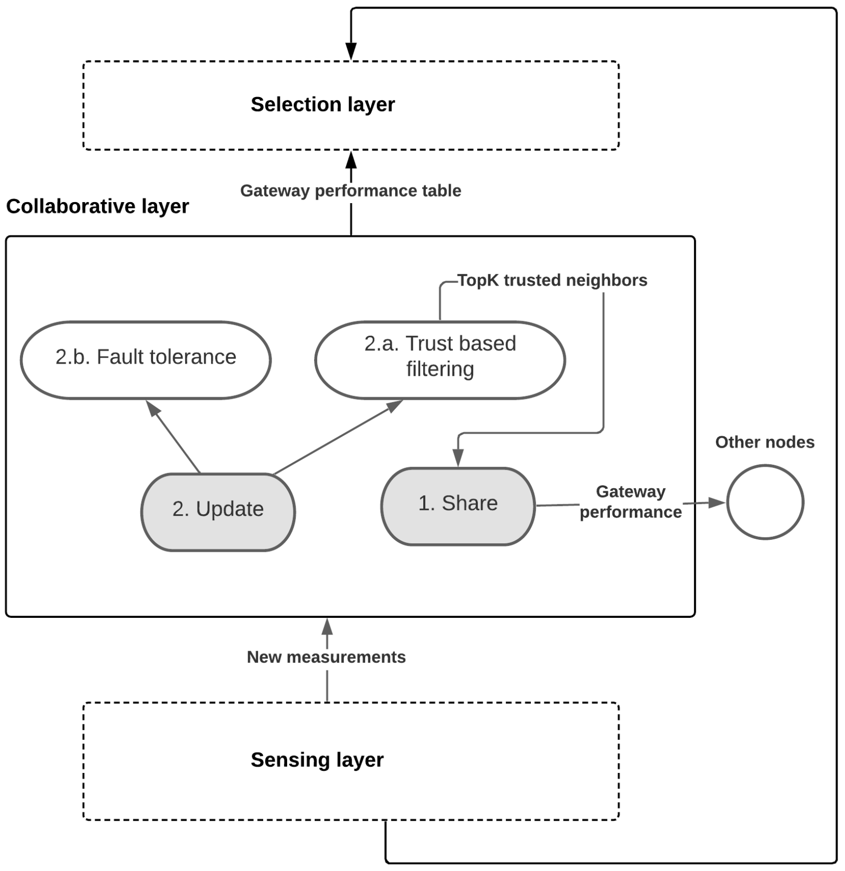


Figure 5.5: Layered sensing framework

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; the node considered as reliable for the 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 the Algorithm 2 when the node receives measurement from the 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 services and the average of the corresponding service 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 collaborating with the top K trusted ones. We define $K = min(n/2 + 1, k)$, where n is the number of services to measure, and k is the number of close neighbors.

Algorithm 3 Filter trusty collaborator nodes

```
Input: $trust\_score = \{N_1 : T_1, N_2 : T_2, \ldots, N_k : T_k\}$    ▷ Trust scores of neighbors
1: procedure UPDATE_TRUST($neighbor, measurements$)
2:     Score = 0
3:     for all $(service, measurement) \in measurements$ do
4:         $m$ = average of all measurements for $service$ 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 $\forall w_i <= w_{i+1}, w_i < 1$ and $w_i = \frac{i}{r*(r+1)/2}$:

$$gateway\_performance\_table[gateway] = \sum_{i=1}^{r} w_i \cdot M_i \tag{5.2}$$

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 service. 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.

5.3.3 Gateway selection

Based on the *Collaborative* layer outcome, we proposed two different gateway selection algorithms which rely on the measurements stored in the gateway performance table.

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.

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).

5.4 Results

We evaluated the feasibility, performance of the algorithm in this section and compared with the other collaborative and non-collaborative approaches.

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 $gw \in gateway\_table$ do
 4:         if (NOW − $gw.timestamp$) < $measurement\_period$ & $gw.latency < latency_{limit}$ then
 5:             $good\_gateways \leftarrow$ ADD($gw$)
 6:     if Size(good_gateways)<2 then
 7:         return SelectGateway($2 * latency_{thr}$)
 8:     if $current\_gateway \in good\_gateways$ then
 9:         return $current\_gateway$
10:     else
11:         return Random($good\_gateways$)
```

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

5.4.1 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 approach 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] approach. 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 approaches, 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 measurements, our approach 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 approaches 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 (*n* 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.

Table 5.1 summarizes the characteristics of the different approaches.

	Message exchange			
	Sensing	Collaboration	Adjustment	Visibility
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.

For our *Collaborative layer algorithm*, the nodes can function independently with little collaboration with the other nodes. The framework runs periodically in every m minute and could be adjusted depending on the dynamics of the network. Total communication overhead (gateway performance measurement, inter-node communication) of each node stands for Equation 5.3, r is the total number of rounds, $overheard_{monitor}$ is equal to 2 gateway performance measurement and $overhead_{gossip}$ is the cost of collaboration.

$$overhead_{comm} = r * (overhead_{monitor} + overhead_{gossip}) \tag{5.3}$$

The gateway performance measurement overhead adds a bit of challenge in the network as it is measured by time to finish downloading 0.1MB file from the Internet through the gateway nodes. Every node creates 0.2MB upstream requests to the gateway in every round.

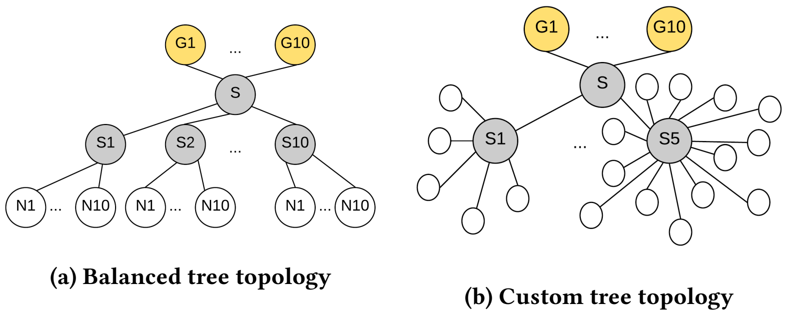
$$overhead_{gossip} = message_{sent} + message_{receive} \tag{5.4}$$

The collaboration overhead is also minimal due to the close neighbor collaboration. $message_{receive}$ is a total number of measurements received from all close neighbors, and $message_{sent}$ is the total number of measurements sent to the trusted neighbors. $message_{sent} = n_{gateways}/2 + 1$ and smaller than the detected number of close neighbors. The number of measurements received at the node varies for each node as other node's trusted neighbors are different. Even though the number of nodes increases, the amount of collaboration is kept within the close circles, therefore, our algorithm is scalable, and communication overhead is static.

Besides collaboration, the node does the following operations during 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 use simple mathematical formulas and could be done with every heterogeneous device. Space complexity for the gateway measurement log at each node is $O(k * m * 2)$ where k is the latest number of measurements retained at the node (by default $k = 10$), m is the number of neighbors sending measurements.

5.4.2 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 $TopK = 6$ trusted neighbor nodes to achieve full performance measurements of all gateway nodes.



(a) Balanced tree topology **(b) Custom tree topology**

Figure 5.6: Network topology for experiments

We evaluate our framework by studying the effect of faulty nodes, resiliency, service 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.

5.4.3 Experiments

The experiments in this section focus on the precise estimation of the proposed collaborative algorithm as it estimates the performance of the gateway nodes according to their selected close neighbors. We conducted different sensitivity experiments of the algorithm, sensitivity towards RTT threshold, close neighbor nodes size, and performance change.

5.4.3.1 Accuracy

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. Col-

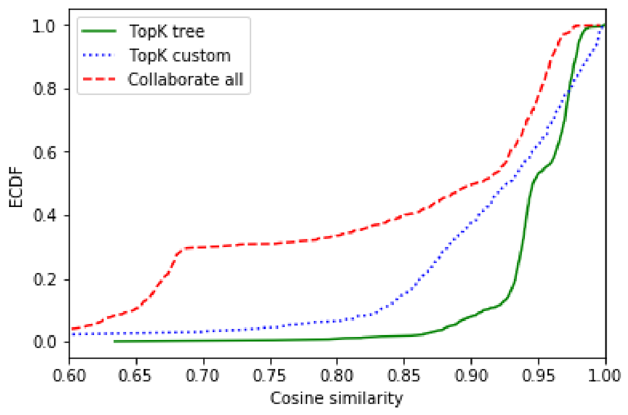


Figure 5.7: Similarity results for different collaboration schemes

laborating with all close neighbors (red, dashed line) gives an average 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 their results, we argue that the Sense-Share framework provides the same quality of collaborative measurements in the different network topology with consistent high accurate measurements.

5.4.3.2 Sensitivity of nodes offline/online

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.

Figure 5.8 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 removed two client nodes from the experiment. However, the size of the measured gateways and precision of estimation is not affected. We further removed two clients at the 32nd round, then the number of measured gateways decreased, but the precision of estimation remains the same. We ob-

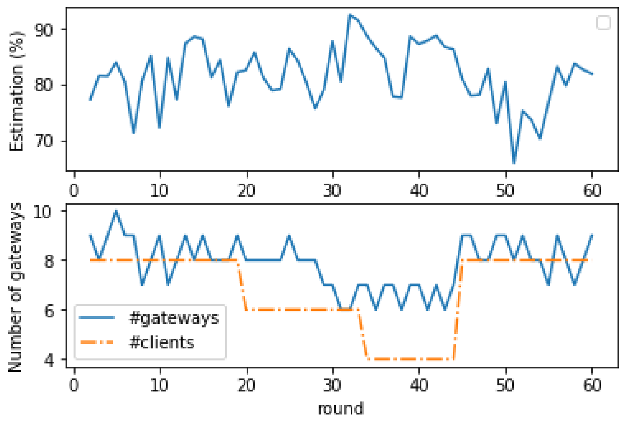


Figure 5.8: Sensitivity analysis (nodes leaving/joining)

served that the ideal number of the nodes in the close neighbors set should be around $(available\_gateways)/2$ to provide enough gateway measurements.

5.4.3.3 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 4.3b, 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.

		Collaborative layer	
	Sensing 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

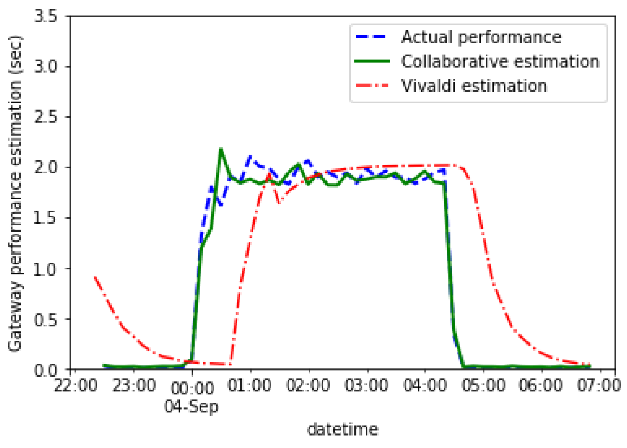


Figure 5.9: Gateway performance change (Collaborating with all)

5.4.3.4 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 on gateway performance and the effect of congestion. In Figure 5.9, we introduced a 200ms delay to the selected gateway after 100 rounds and removed the delay after 150 rounds. We plotted 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 took 30 rounds to adjust to the actual gateway performance change; 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 took 20 rounds to adjust to the normal performance change while our algorithm sensed 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 5.10 plots the result of the gateway performance change adaptation when collaborating with trusted *TopK* neighbors. 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 re-

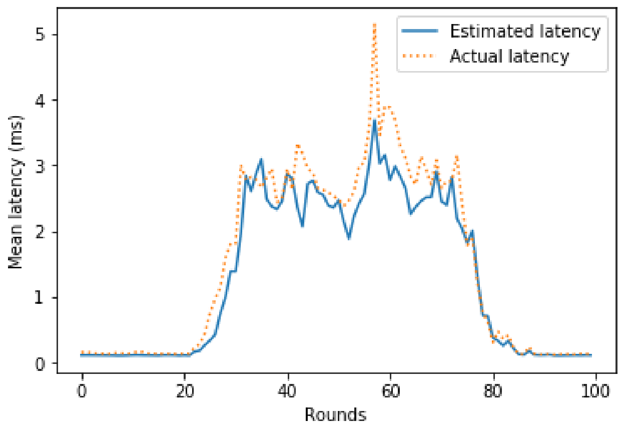


Figure 5.10: Gateway performance change (Collaborating with *TopK*)

moving 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.

We simulate sudden performance changes on the node side, causing collaborator nodes to leave (link down) and join the network again (link up). As depicted in Figure 5.11, 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

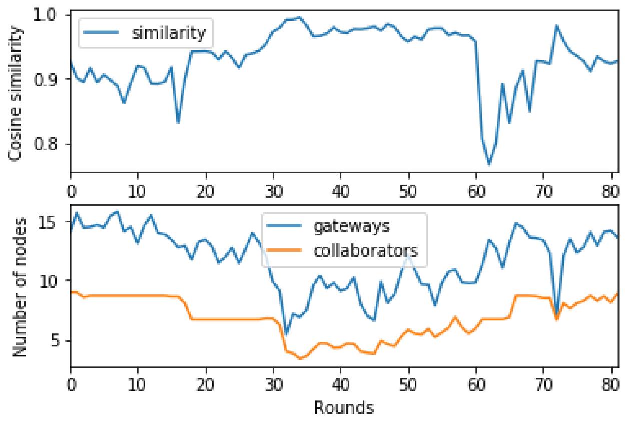


Figure 5.11: Collaborating node change

number of available gateways is decreased from 15 to 13, but the cosine similarity index of the service 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 within 1-2 rounds, while when the link is up again within the same round, as soon as the nodes send measurements.

5.4.3.5 Fault tolerance

We performed 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.

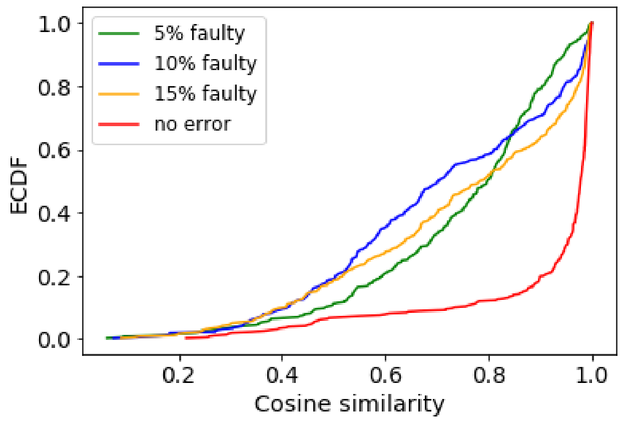


Figure 5.12: Faulty nodes effect (Collaborating with all)

Figure 5.12 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 5.13 depicts the ECDF of the collaboratively achieved service 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 5.13, 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.

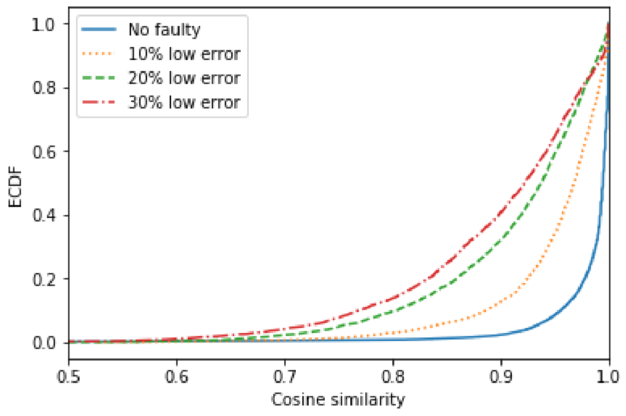


Figure 5.13: Faulty nodes effect (Collaborating with *TopK*)

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 constructed service performance table. In the results of the experiment presented in Figure 5.14, 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 presented

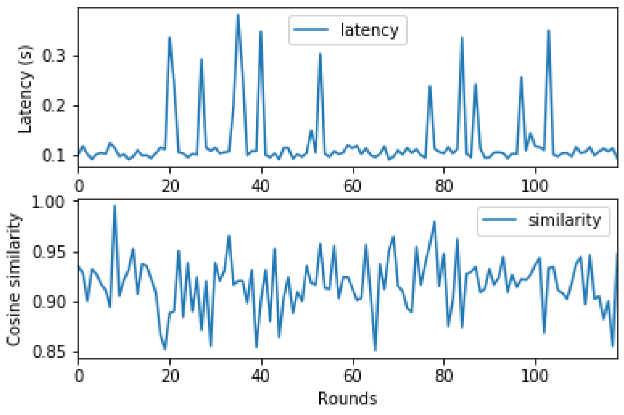


Figure 5.14: Behavior of the faulty nodes

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

5.4.3.6 Sensitivity analysis

In this section, we present our study concerning the sensitivity of the proposed framework in terms of the number of available services, the close neighbor sensing threshold, and the number of randomized sampling, which are the parameters that affect the accuracy of the proposal.

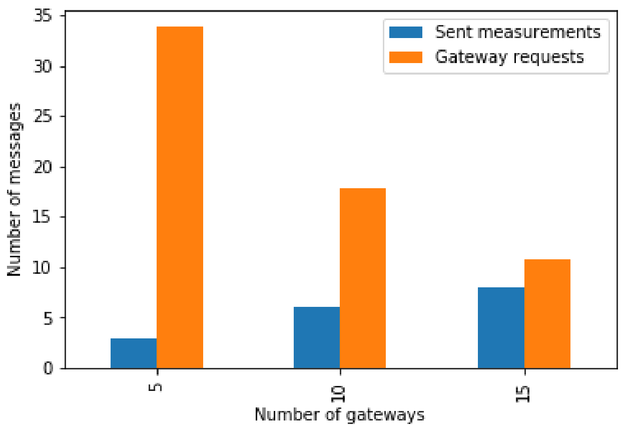


Figure 5.15: Sensitivity towards the number of service nodes

Figure 5.15 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 gateway node. The number of service 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 5.15 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

	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

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 similarity 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.

5.4.3.7 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.

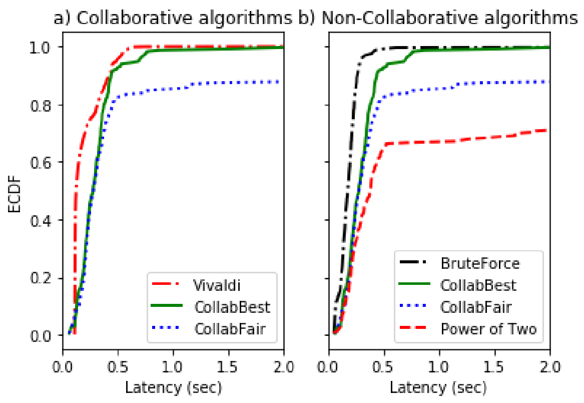


Figure 5.16: Gateway selection result

5.5 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 service 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

Due to the *Collaborative layer* in the previous chapter, the nodes have partially-full gateway performance measurements readily available. Therefore, the final gateway selection verdict is easy. In *Sensing layer* and *Collaborative layer*, we proposed an optimal gateway selection algorithm and showed the download latency of the proposal. Each download latency result shows close to the Brute-Force (optimal gateway selection) selection result, but not better. We present an optimized selection algorithm, GateSelect, the gateway selection algorithm based on balanced distribution and capacity estimation of the gateways. For different network scenarios, the performance of the proposed *Selection layer* is compared to the existing distributed gateway selection algorithms presented in this chapter.

6.1 Introduction

The process of choosing a gateway for a client node among the available gateways in a network is called *gateway selection.* 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.

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.

We introduce *GateSelect* (Selection layer), a client-side distributed gateway selection method to tailor the gateway selection for each client node by utilizing the measurements collected locally at their side. 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

In the type of networks we consider in this paper, client nodes connect to the Internet through gateway nodes. 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. This situation is called herd behavior. 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 based on the measurement results [Dim+17b; Ko+13; Abu+15; GRS08]. Such periodic reevaluations are called *rounds* in the following. 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 The objective of the algorithm

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 gate-

ways. 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.

6.3 Design overview

From the *Sensing layer* and *Collaborative layer*, the node is equipped with a partially full gateway performance table, shown 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.

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

Table 6.1: Partial gateway performance table, every round

The aim of *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 *sta-*

ble 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.

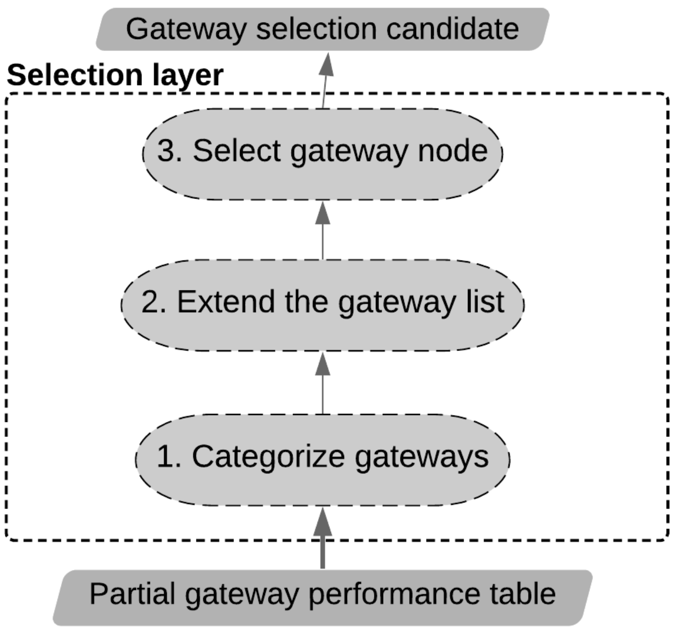

Gateway selection candidate
Selection layer
3. Select gateway node
2. Extend the gateway list
1. Categorize gateways
Partial gateway performance table

Figure 6.1: Selection algorithm design

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 in the *Sensing layer*. The gateways are categorized into *Good*, *Inconsistent* and *Bad* performing gateways.

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

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 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.

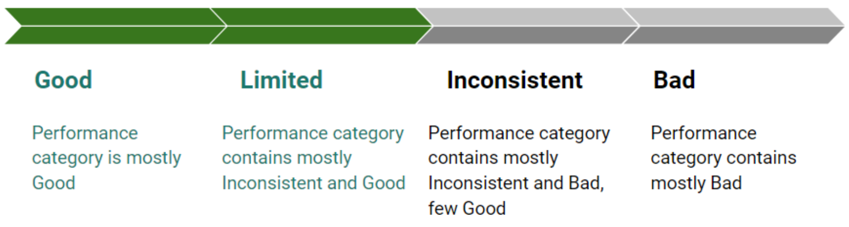
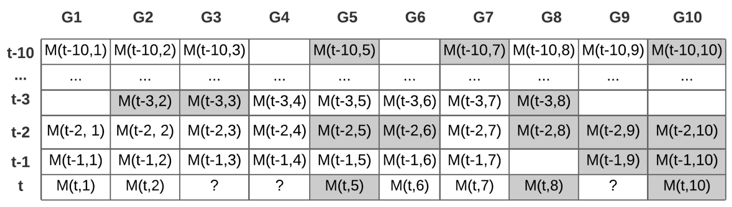

Good
Performance category is mostly Good
Limited
Performance category contains mostly Inconsistent and Good
Inconsistent
Performance category contains mostly Inconsistent and Bad, few Good
Bad
Performance category contains mostly Bad

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

Table 6.3 shows a possible outcome of the classification step. 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 $G4$ 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.

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. 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

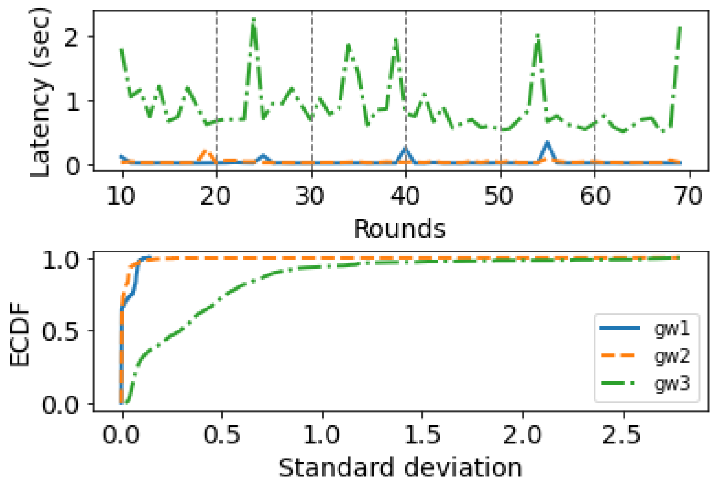


Figure 6.3: Gateway latency variance.

changes are avoided unless the current gateway's performance deteriorates such that it is eliminated 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:     $good\_gateways = \{G \in gateways\_candidates \mid G$ 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 $current\_gateway$
```

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 Results

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 through their selected gateways every minute to mimic client activities.

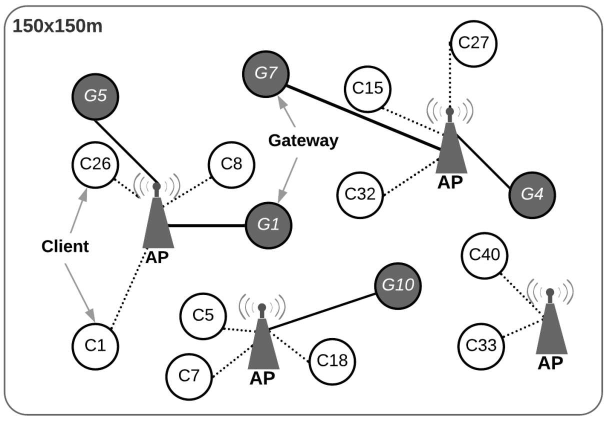


Figure 6.4: Experiment network topology.

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:

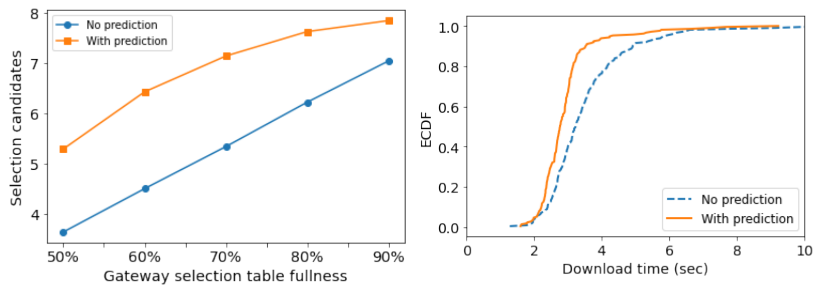
1. GateSelect: Our algorithm;
2. Greedy (Brute-Force) selection: Clients always select the best gateway every round;
3. Random selection (Power of 2 choices): Clients choose two random gateways and select the best every round;
4. Static selection: Clients select the best gateway at the beginning of the 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 Prediction of missing values

In our first experiment, we test the impact of Step 2 of the *Selection layer* 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



(a) Prediction selection candidates (b) Prediction download latency

Figure 6.5: Wireless - Performance prediction result.

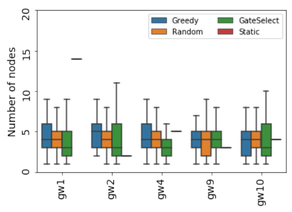
run the algorithm with and without the prediction step (Step 2) for the same network settings. We observe that the download time improves with prediction because the prediction step increases the number of selection candidates for each node. As a result, the traffic load is distributed more evenly over the gateways.

6.4.2.2 Gateway selection distribution

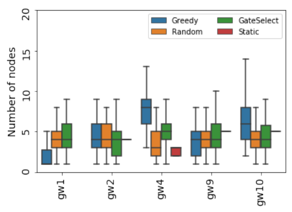
We compare the distribution of the client node over the gateways for the different selection algorithms. To this end, we calculate the average distribution of the clients after a single simulation (100 rounds). For all three network configurations, identical start positions were chosen for the nodes. 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.

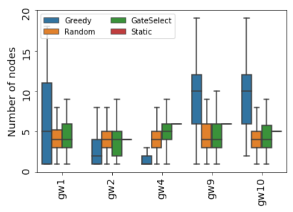
Figure 6.7 shows the overall distribution of client nodes per gateway per round in the wireless and wireless mobile network. The simulation runs are repeated five times with random positions of nodes. The results show that



(a) Wired network



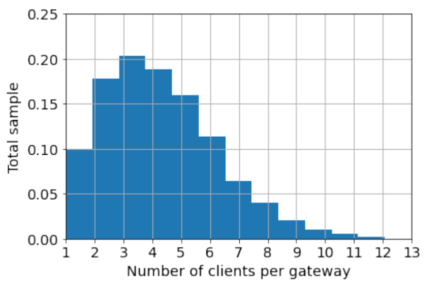
(b) Wireless network



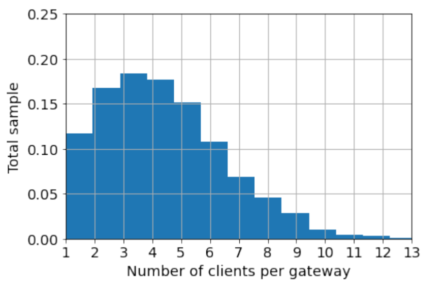
(c) Wireless mobile

Figure 6.6: Client node distribution over the gateway nodes.

both networks behave similarly. In most rounds, the 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 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.



(a) GateSelect - Wireless



(b) GateSelect - Mobile

Figure 6.7: Wireless gateway distribution

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

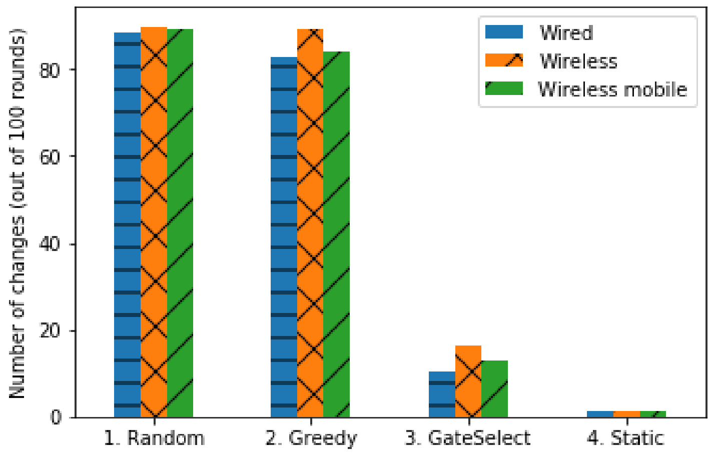
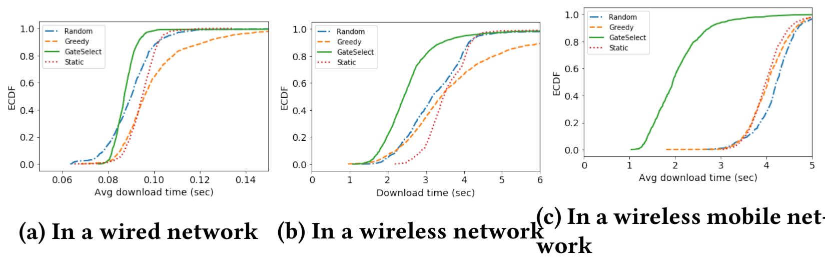


Figure 6.8: Number of gateway selection changes in 100 rounds.

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.



(a) In a wired network **(b) In a wireless network** **(c) In a wireless mobile network**

Figure 6.9: Download time through the gateway nodes.

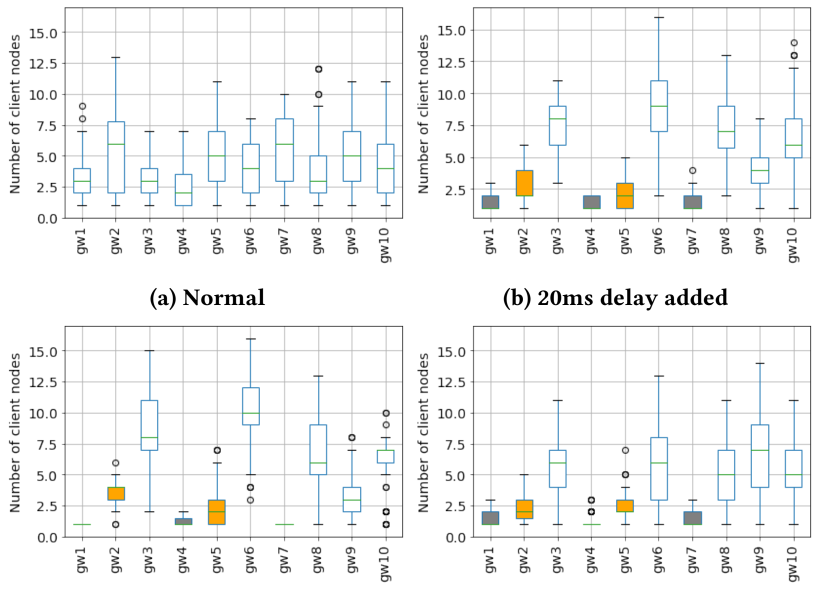
In the wired network (Figure 6.9a), the download time of GateSelect is less than 0.09 seconds on average and is better than that of the other algorithms for 90% of the downloads. Other compared algorithms' results show performance close to our proposal due to the stability of the links in the wired network. 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.

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



(a) Normal **(b) 20ms delay added**

(c) 40ms delay added **(d) 40ms delay + 2x download**

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

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

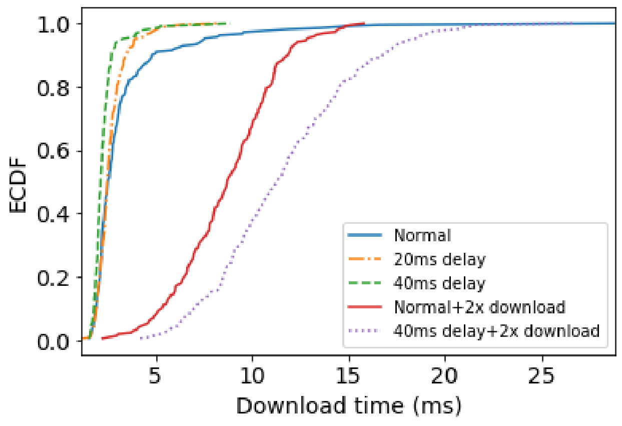
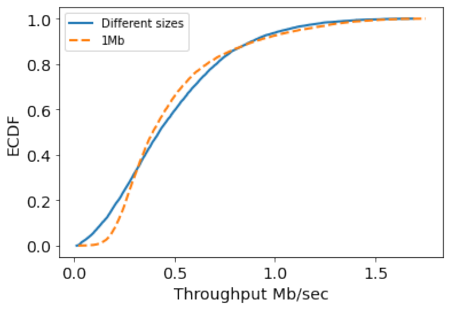


Figure 6.11: Download times with different gateway delays

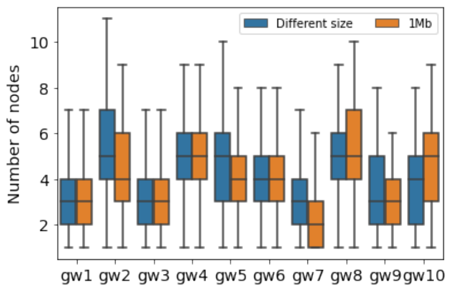
Figure 6.11 shows the download times with the different link delays. As the link delay increases, the download time increases because every client's gateway selection pool is narrowed down to a few good gateway nodes. When increasing the download frequency, the download time increases due to queueing 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 experiments. Figure 6.12a shows that the achieved throughput per download is slightly lower than with the 1MB file. The gateway distribution (Figure 6.12b)



(a) Throughput



(b) Gateway distribution

Figure 6.12: Wireless - Download size sensitivity.

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.

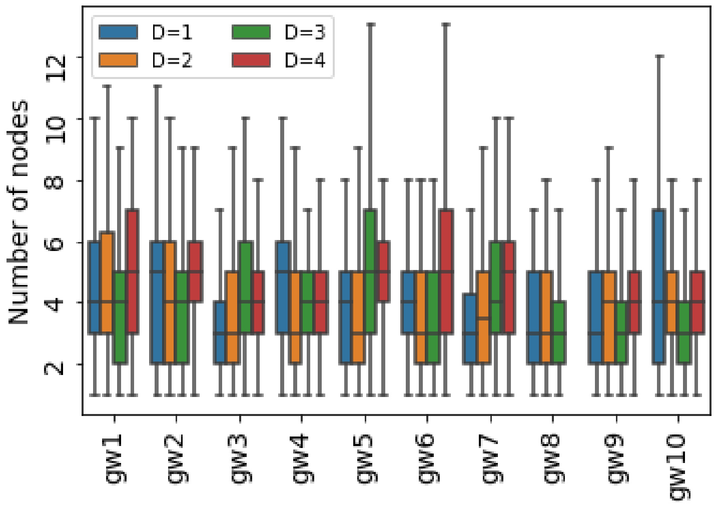


Figure 6.13: Wireless - Gateway distribution.

6.5 Conclusion

We present our *Selection layer* algorithm is called GateSelect, a distributed gateway selection algorithm tailored to fulfill each client node's Internet gateway selection using the following techniques: performance-based classification and random sample selection. GateSelect is simple yet effective, infrastructure- and technology-agnostic, and supports incremental implementation. We demonstrate 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

Previous chapters described the individual layers of the Sense-Share-Select framework and presented the 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. 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 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.

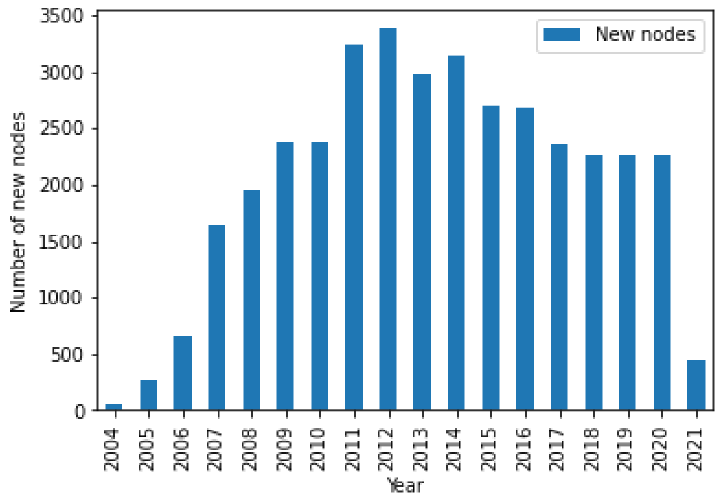


Figure 7.1: Number of new nodes (Data from www.guifi.net)

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 aspect 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 with close to its performance sensing result (Brute-Force sensing) through our *Col-*

[1] https://guifi.net/en/UPCC6lab104

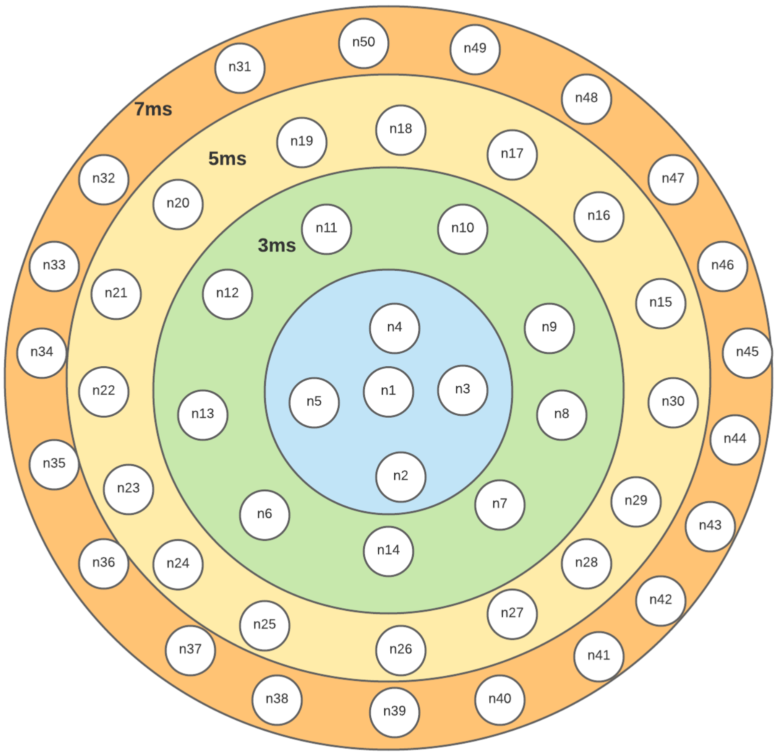
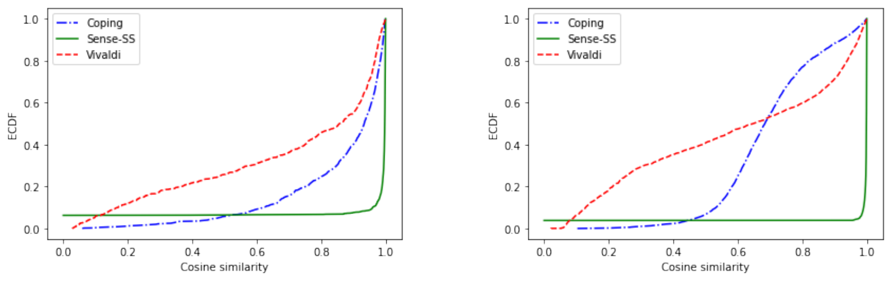


Figure 7.2: Experimental nodes topology



(a) 20 nodes experiment **(b) 50 nodes experiment**

Figure 7.3: Collaborative sensing similarity

laborative 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.

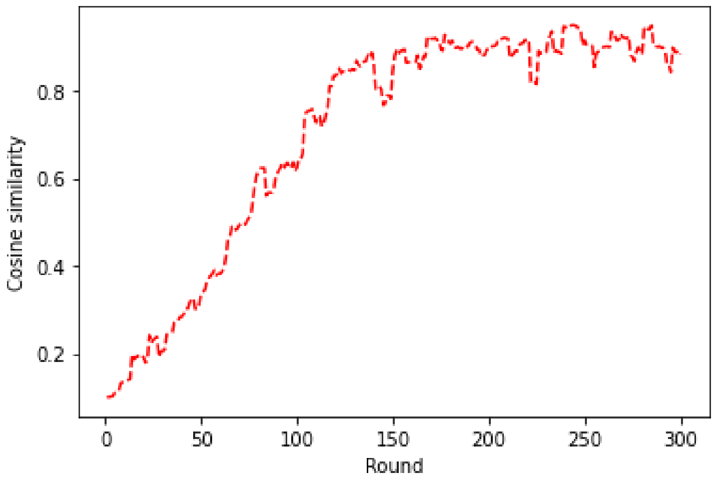
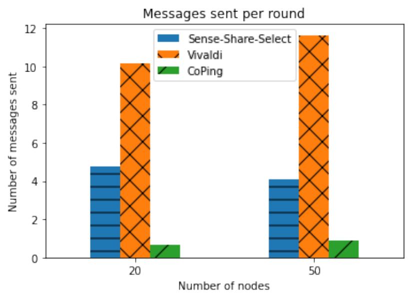


Figure 7.4: Vivaldi accuracy

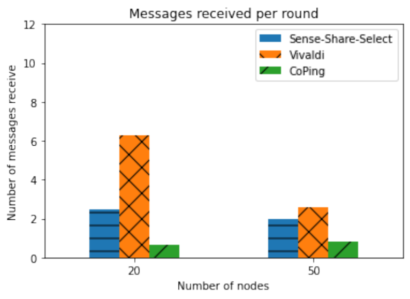
Both experiments in Figure 7.3 show that the accuracy of the collaborative monitoring is consistently higher for the Sense-Share-Select result than the others. In Figure 7.3a, the Coping algorithm's accuracy is on average >80% because the nodes have a maximum of 2 hops. In the 50 nodes experiment in 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 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 70-80 rounds 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



(a) Messages sent per round



(b) Messages received per round

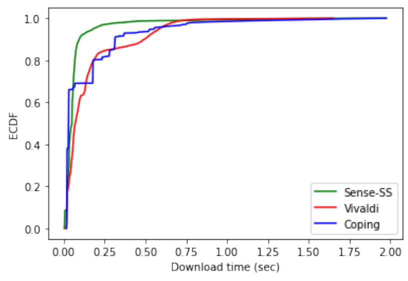
Figure 7.5: Collaborative message exchange

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 bin) 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 the Coping is the lowest. In Vivaldi based algorithm result, 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.

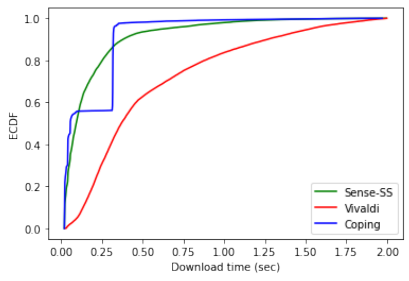
The Coping algorithm produces comparatively fewer gossip messages, and so does our framework. However, the Vivaldi-based algorithm is the most expensive one, which requires constant, mass gossiping within network nodes to keep the balance. Additionally, the Sense-Share-Select achieves a high accuracy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

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 in every 1 minute for 100 rounds (1 round = 2 minutes). The results are shown



(a) 20 nodes experiment



(b) 50 nodes experiment

Figure 7.6: Download time

in Figure 7.6, and the Sense-Share-Select framework provides better download time as compared to the others in both. 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 they 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-

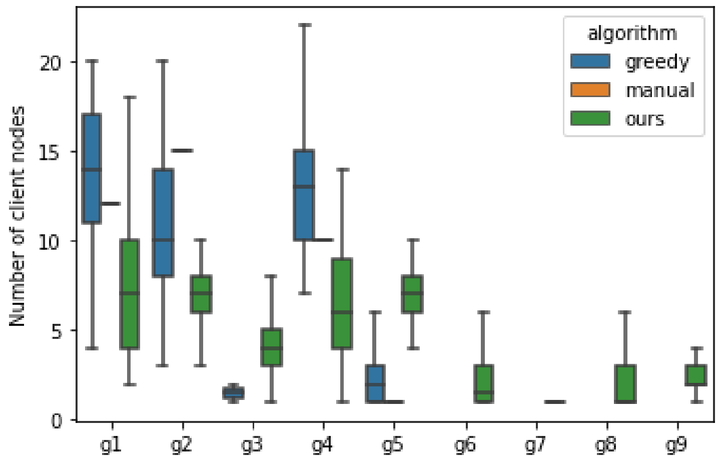


Figure 7.7: Gateway distribution per round

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 the good-performing gateways and varied performing gateways and distributes a few nodes to the bad performing gateways.

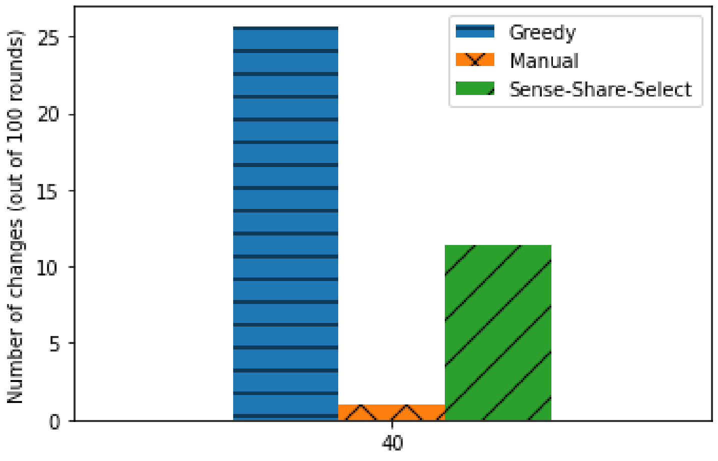


Figure 7.8: Number of gateway changes

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 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, adaptability of the Sense-Share-Select framework.

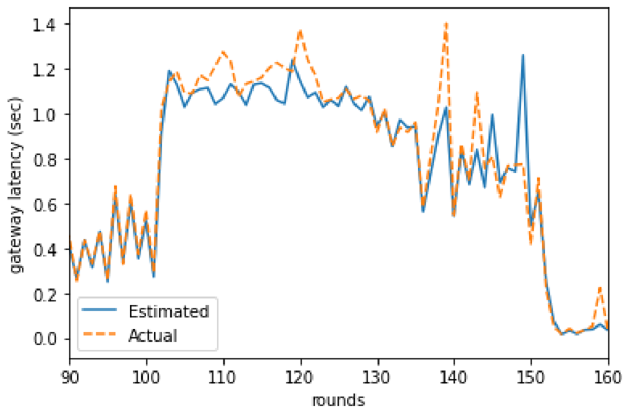
7.2.3.1 Introducing delay at the gateway

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 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 7.9. The orange dotted line represents the actual gateway performance, and the straight blue line is the estimated performance through the collaborative algorithm.

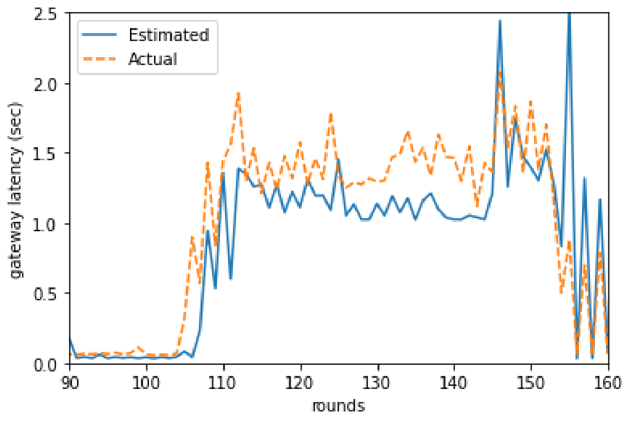
From the result of the Sense-Share-Select framework is in Figure 7.9a, the gateway performance change caught immediately and stayed in line with the actual latency curve. When removing the delay, the change is caught in the same round. On few occasions, the estimated performance is slightly different 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 7.9b) 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.

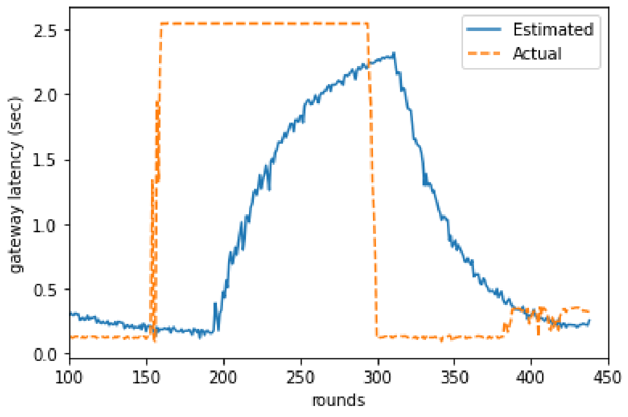
The Vivaldi-based algorithm does not cope well with the sudden gateway performance change, shown in Figure 7.9c since it is not reactive. Any change in the synthetic coordinate system reflects all nodes in the network, and the nodes update the coordinates slowly using small steps towards the direction



(a) Sense-Share-Select



(b) CoPing algorithm



(c) Vivaldi based algorithm

Figure 7.9: Download time

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.

7.2.3.2 Removing nodes

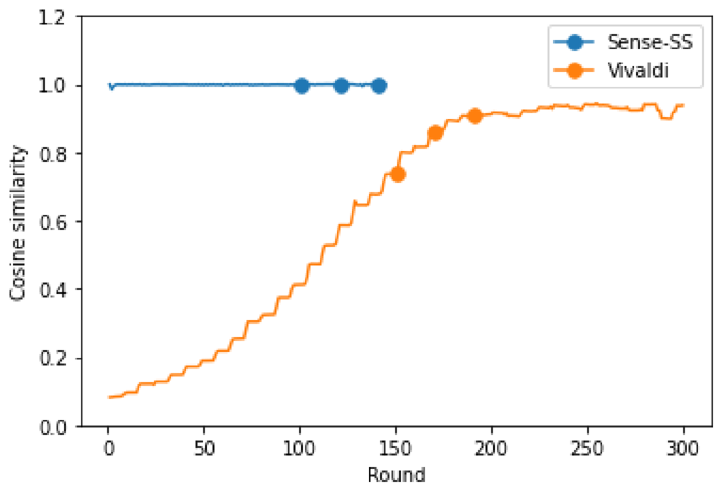


Figure 7.10: Node unavailable test

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 7.10. 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 7.10. 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 accuracy.

7.2.3.3 Faulty collaborators

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

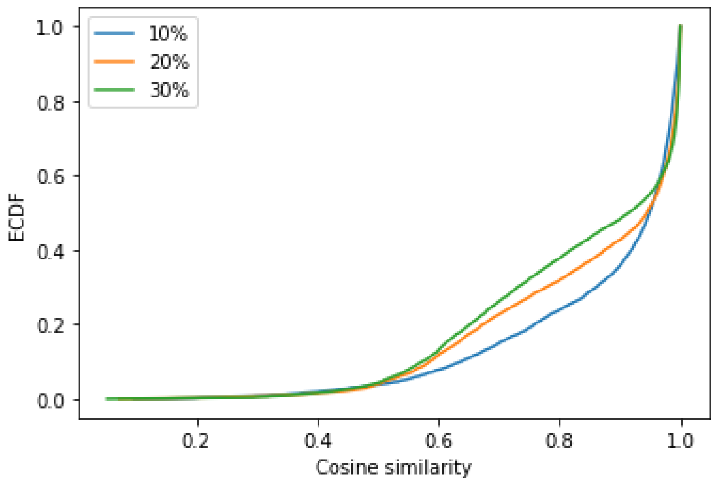


Figure 7.11: In the presence of faulty nodes

framework. We tested the fault tolerance of our proposal in Figure 7.11 and introduced 10%, 20%, 30% of the faulty nodes in the network where they intentionally 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.

7.2.3.4 Different network topology

The Sense-Share-Select framework is designed for a large-scale, heterogeneous network, and adaptability is important in the implementation. We designed experiments with different topologies such as star topology, tree topology in the wired and wireless mesh topology using Mininet and Mininet-Wifi framework. The emulation environment consists of 20 nodes and 10 experimental gateways. In a wireless network experiment, nodes were placed in a 100x100m area with two access points. In a wireless mobile network experiment, all nodes are within their access range since our algorithm is not designed for mobile network scenarios.

In the experiment in Figure 7.12 the result of the star topology shows more than 90% accuracy on average, while the tree topology experiment shows >74% accurate measurements. In tree topology, any link congestion in middle

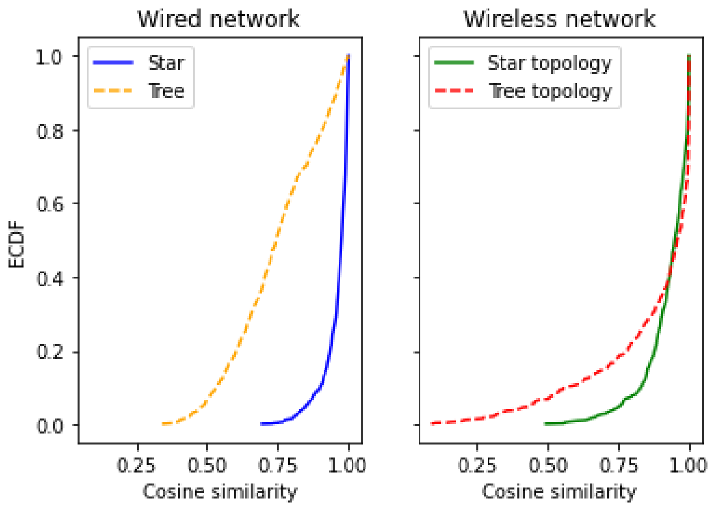


Figure 7.12: Collaborative sensing accuracy

links affects the descendant nodes', and they will have higher latency even though the RTT value is small. Therefore, in Figure 7.12 the accuracy of the collaboration is lower than the other topologies.

Figure 7.13 shows that wired network experiments show similar download latency about 90% of results as the network links are more stable. In the wireless network, download latency is 10x higher than the wired network result due to the interference between nodes.

Overall, the Sense-Share-Select framework works well in different network scenarios without losing the high collaborative monitoring accuracy.

7.2.3.5 Isolated nodes

The Sense-Share-Select algorithm works best when nodes voluntarily collaborate. However, in the production network, the adoption of the new framework is slow, reluctant to change. In that case, the nodes might not find close neighbors to collaborate and ends 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% are with manual selection,
3. 50% nodes are collaborating with our framework, 50% are with manual selection

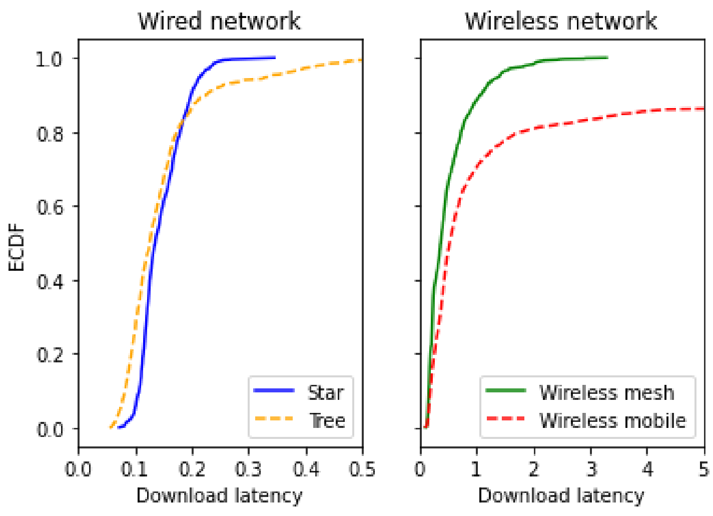


Figure 7.13: Download latency

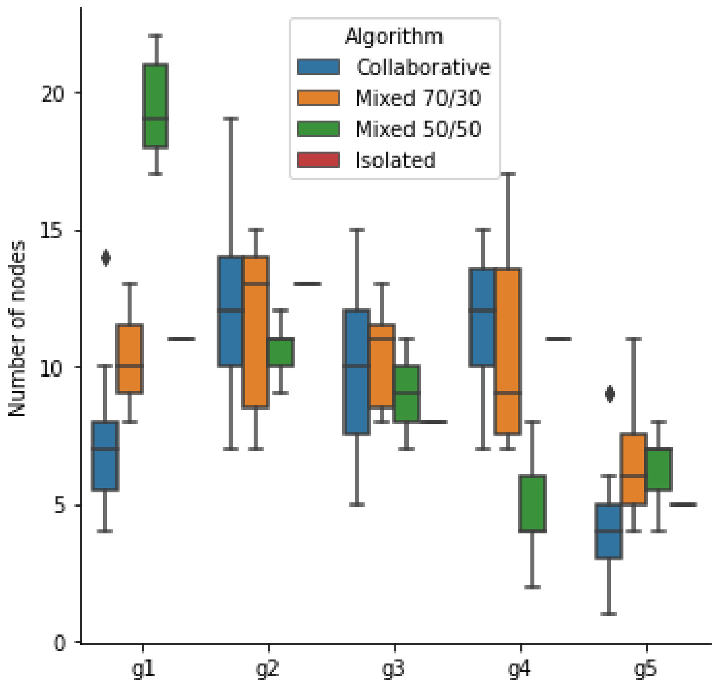


Figure 7.14: Wireless experiment distribution

4. 100% of the nodes are with 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 7.14.

In the 100% isolated nodes result in Figure 7.14, 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 7.14. 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 7.15, 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 7.16 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 results in Figure 7.16 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.

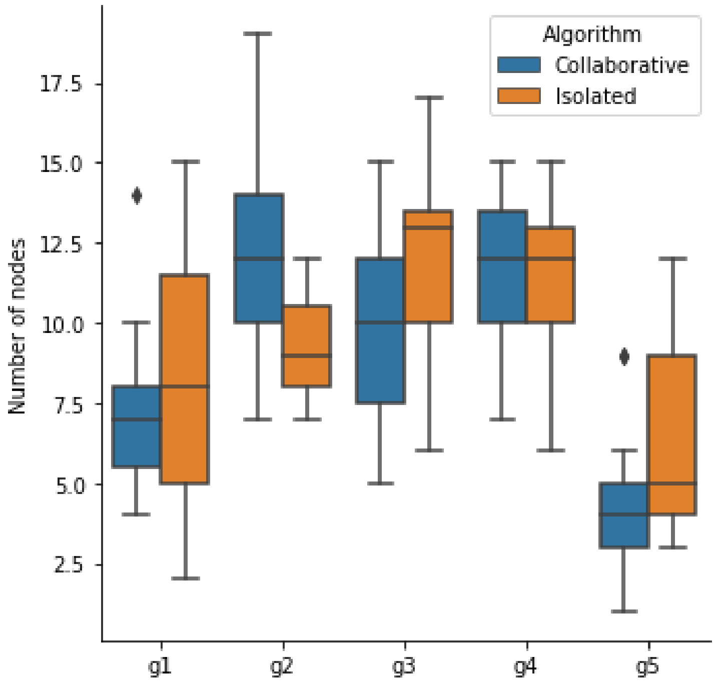


Figure 7.15: Full collaboration vs full isolation

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.

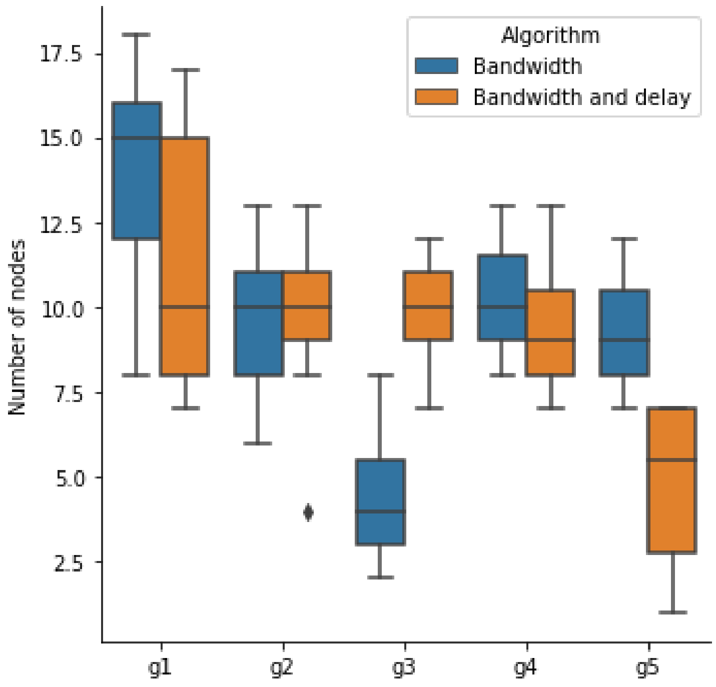


Figure 7.16: Experiment with different bandwidth

Conclusion 8

8.1 Main conclusions

By nature, the shared Internet gateway network consists of a high number of heterogeneous nodes, and manual gateway selection doesn't do any justice for bettering the Internet QoE. The majority of the nodes are built with over-the-counter network devices with limited capacity, the gateway selection mechanism requires to be simple and effective. The Sense-Share-Select framework specifically designed for the large-scale network with low complexity, fair resource distribution as the main goals.

Concerning the cost reduction of the gateway monitoring, we presented 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 the faulty collaborators. Every round, the nodes monitor different gateway performance which reduces the overall monitoring traffic to the gateway and eliminates the scheduling, reorganization within the collaborators.

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

Overall, we believe that our framework significantly improves the QoE of the end-users without complex technical knowledge such as gateway load, routing protocol, etc. We look at the problem from the client node perspective and improve the gateway selection. The Sense-Share-Select framework is designed for the Internet gateway selection domain but the real application is much more versatile. The main logic is simple enough for IoT network gateway selection for inter-device communication. The user could configure the framework based on their needs. For example, in mobile network, 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 future directions that we would like to investigate in the future.

1. **Adaptive performance monitoring** - We show that nodes collects significant measurements through periodic collaboration in *Collaborative layer*. The majority of the measurements are stable and have performance patterns peak/off-peak period. Further flexible performance monitoring period adjustment is possible. Timeseries based performance prediction coupled with close neighbors collaborative performance monitoring can be a low-cost solution suitable in client-side framework solutions.

2. **Towards secure, anonymous collaboration** - Sense-Share-Select is not an anonymous collaborative framework and is focused on bottom-up initiatives, local networks. The framework is not 100% secure since the foundation of the collaboration is based on the similarity of the measurements. Security is one of the biggest concerns in the Sense-Share-Select framework since nodes are aware of the collaborating nodes' physical addresses. 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 the nodes in large-scale, bottom-up wireless mesh networks to improve each node's individual Internet gateway experience. The implementation in the real large-scale network is challenging to achieve due to technical and social aspects. From the technical point of view, the algorithms implemented in the routing protocol might be more manageable than the client-side solution. However, client-side implementation is more suitable for the heterogeneous network with different routing protocols, capacities. From the social point of view, existing users are often reluctant to change to new features, 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 are designed for the local Wireless Mesh network with non-mobile network. For the mobile network environment, node discovery component is crucial to identify the collaborating nodes and gateway nodes within the range. The frequency of the gateway performance monitoring and collaboration should be increased in mobile network to ensure 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 (EMJDDC) 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 from 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. Sathiaseelan. “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 Semiannual 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.

[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. Magaudda, 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. Sathiaseelan, 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. Vlajic. “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.