Investigating Optimal Internet Data Collection in Low Resource Networks

By
Taveesh Sharma

A Dissertation submitted in partial satisfaction of the
requirements for the degree of
Master of Science (MSc)
in
Computer Science
at the
University of Cape Town

2022

Plagiarism Declaration

I, **Taveesh Sharma** hereby declare that this thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software). Plagiarism is to use another's work and pretend that it is one's own and I know that plagiarism is wrong. I confirm that I have discussed and resolved any concerns emanating from the Turnitin report with my supervisor.

7 May, 2022

Signed by candidate

Taveesh Sharma

Date

This page has been intentionally left blank.

Acknowledgments

I would like to thank my supervisor, Dr. Josiah Chavula whose guidance and advice made this project a reality. I would also like to thank the members of my research proposal committee for providing their valueable feedback during the initial phase of my studies.

Further, I would like to thank the members of my research group for their encouragement and feedback towards this project.

Lastly, I would like to thank my family, especially my father, Brij Bhushan Sharma, for their undconditional support and love.

Publications

Some of the ideas presented in this dissertation have previously appeared in the following publications:

- **Taveesh Sharma** and Josiah Chavula (2021) Investigating Measurement Scheduling Strategies in Low Resource Networks (Poster). In ACM SIGCAS Conference on Computing and Sustainable Societies (COMPASS) (COMPASS '21), June 28-July 2, 2021, Virtual Event, Australia. ACM, New York, NY, USA 4 Pages.
  Available: https://doi.org/10.1145/3460112.3472310

- **Taveesh Sharma** and Josiah Chavula (2021) Topology-Aware Measurement Scheduling Strategies in Low Resource Networks. Proceedings of Southern African Telecommunications Networks and Applications Conference, 21-24 November 2021, Central Drakensberg, Kwazulu Natal, South Africa.
  Available: https://pubs.cs.uct.ac.za/id/eprint/1502/

- Enock Samuel Mbewe and **Taveesh Sharma** and Josiah Chavula (2022) PowerQoPE: A Personal Quality of Internet Protection and Experience Configurator, Proceedings of IFIP International Symposium on Human Aspects of Information Security & Assurance (HAISA 2022), 6-7 July 2022, Mytilene, (Greece), Springer.
  Available: https://pubs.cs.uct.ac.za/id/eprint/1528/

- Dominique Oosthuizen, **Taveesh Sharma**, Josiah Chavula and Melissa Densmore (2022). Investigating the Usability and Quality of Experience of Mobile Video-Conferencing Apps Among Bandwidth-Constrained Users in South Africa. In Proceedings of 43rd Conference of the South African Insti (Vol. 85, pp. 243-256).
  Available: https://login.easychair.org/publications/download/bchj7

Abstract

Community networks have been proposed by many networking experts and researchers as a way to bridge the connectivity gaps in rural and remote areas of the world. Many community networks are built with low-capacity computing devices and low-capacity links. Such community networks are examples of low resource networks. The design and implementation of computer networks using limited hardware and software resources has been studied extensively in the past, but scheduling strategies for conducting measurements on these networks remains an important area to be explored. In this study, the design of a Quality of Service monitoring system is proposed, focusing on performance of scheduling of network measurement jobs in different topologies of a low-resource network. We also propose a virtual network testbed and perform evaluations of the system under varying measurement specifications. Our results show that the system is capable of completing almost 100% of the measurements that are launched by users. Additionally, we found that the error due to contention for network resources among measurements stays constant at approximately 34% with increasing number of measurement nodes.

Contents

1

Introduction

1.1 Motivation

Rosson *et al.*[1] define community networks as follows:

"A network community is a group of people whose communication and collaboration over networks strengthens and facilitates their shared identity and goals[..] A community network is a special case of a network community in which a physical community coextends with the network community."

According to this definition, a community exists not only due to collaboration through the network but also due to the contributions by people with their own resources. These networks are often run by non-profit organizations in partnership with citizens [2, 3, 4, 5]. Some of the services provided by community networks include local networking, content sharing, voice connections, and most importantly, internet access [6, 3]. Since multimedia content is usually relayed through these networks, it is important to measure and evaluate the users' perceived Quality of Service (QoS) and Quality of Experience (QoE) [7]. On a basic level, QoE reflects the actual experience of the end user while QoS entails the measure-

ment of quantitative aspects like throughput, error rate, delay, jitter, frames per second, and bits per pixel. Although QoE is a function of different QoS parameters, there is an indirect relation between QoS parameters and QoE. Therefore, it is important to understand which kind of degradation in QoS parameters lowers the user experience [7].

Typically, community networks are marked by a heterogeneous structure [3], i.e, they combine a variety of wired and wireless architectures at the physical layer. As communities expand in size, the amount of mobile traffic is also expected to grow. Therefore, users' demand for high quality communication services will increase significantly. In order to address these demands, community networks are expected to increase their heterogeneity even further. The demand for efficient content delivery will become essential not only for wired networks but also wireless networks. Software defined networking (SDN) [8] has the potential to manage the network services and applications with greater efficiency [9]. SDN technologies like the OpenFlow protocol [10] were initially best suited for infrastructure-based networks [9]. More recently, a lot of research [11, 12, 13] has been conducted to modify these technologies for wireless mesh networks. Therefore, it is reasonable to assume that community networks in the future are expected to be managed and implemented as software defined networks.

Most community networks lack dedicated network measurement and monitoring platforms. The networks are implemented, maintained and extended by volunteers, which

results in the absence of a management and administrative authority to take decisions on hardware and software to be used in the nodes. The extension of some community networks occurs in a rather ad-hoc manner, without any implementation studies or cost-benefit analyses [14]. Further, the node owners in these networks are people with different levels of software and network expertise. Thus, any software or network issues in these networks are resolved by the users cooperatively, without the external help of an expert.

Community networks use a combination of links at the physical layer involving a higher proportion of wireless links as compared to wired links. As a consequence, the most popular way of accessing the internet is through smartphones. The usage of smartphones as a computing and networking device presents an opportunity in terms of the construction of mobile crowdsourcing applications [15]. Crowdsourcing is an attractive approach to conduct network monitoring as it allows network measurement activities to be executed on end users' hardware, without the economical and practical burden of managing a dedicated system [15]. Therefore, mobile crowdsourcing is an area worth exploring in order to design a monitoring system for community networks.

Architectures of systems based on crowdsourcing are generally more complex as compared to systems where design, implementation and execution are centralized [15]. Other challenges arise due to adopting smartphones as an execution unit. It is difficult to precisely capture internet data through smartphones because the conditions, like location and

bandwidth, are always changing over time, and as the subject moves with the device. Smartphones, unlike server-class or desktop computing systems, are powered from batteries that are limited in both size and capacity. For ensuring good user experience, this forces developers to make important architectural and implementation choices when designing applications. Thus managing energy is important in these devices [16, 17]. Further, monitoring internet connectivity in smartphones can be expensive due to communication costs incurred. In passive monitoring techniques, limited bandwidth and high communication costs do not pose a challenge, since the monitoring is done without sending/receiving packets. But in active techniques the generated high traffic raises a concern, especially when network monitoring is performed by leveraging crowdsourcing techniques [15]. When collecting information from smartphones, it is paramount to ensure that the identity of the subject cannot be associated with the data [17]. Thus, privacy protection is another challenge when it comes to internet measurement using smartphones. Lastly, smartphones are generally not well suited for conducting network monitoring measurements. The operating system in these devices is designed for the prospective applications that run on them, and not for low-level networking mechanisms [15], thus making compatibility a major research challenge.

Although the load of executing measurements is shifted to smartphones in a crowdsourced architecture, a centralized server should be able to allocate the measurements intel-

ligently, given the availability of resources like network bandwidth and execution capacity of smartphones. If a large number of measurements are carried out using a limited number of vantage points, the obtained results could suffer from the observer effect [15], i.e a bias in the measurements due to the measurement infrastructure itself. Measurement processes that are executed in different common points and links could contend for shared network resources. This contention for resources is also called measurement conflict problem [18]. Thus, scheduling and synchronization of measurements among the smartphones is important to ensure proper resource utilization and accuracy of results.

1.2 Problem statement

Low resource networks, such as those found in many underserved communities, present an interesting case of research. This is because measurement and monitoring techniques that apply to resource intensive networks may not be applicable to these networks as they operate using routers with legacy hardware, limited buffer sizes and sometimes outdated software. Mobile crowdsourcing has been shown as one of the better ways of performing network measurements despite certain implementation and usability challenges. This study aims to develop a monitoring system specifically designed to address the needs of community network users. In an attempt to find the best possible design, we present an empirical analysis of alternative techniques for measurement scheduling and synchronization, and

study the trends in performance of these strategies with varying parameters that constitute internet measurements.

1.3 Research questions

The main objective of this research is to answer the question *'what design considerations are necessary for a low cost internet measurement system for under-resourced networks'*. Our study aims to answer this question by focusing on measurement scheduling and synchronization strategies. First, we survey related literature and determine *'what coordination, synchronization and scheduling strategies would be suitable for measurement and data collection in community internet measurement platforms'*. After this, we implement these strategies in our own measurement platform and evaluate them on a synthetic community network testbed and we conduct relevant experiments to find out *'how are measurement results affected by changes in scheduling strategies and measurement specifications'*.

1.4 Research approach

The first part of this research dealt with the implementation of four measurement scheduling algorithms that include Round Robin (RR), Earliest Deadline First (EDF), Ascending order of subvertices degree (AOSD) and descending order of subvertices degree (DOSD). Our implementation of these algorithms was based on the assumption that the execution

time of measurements is always fixed. Therefore, to minimize the overlapping of measurements with each other we conducted a series of preliminary experiments to determine maximum execution times (in milliseconds) of five types of measurements. These measurement types were pings, DNS lookups, traceroutes, HTTP downloads and TCP speed tests (both uplink and downlink). For the ease of implementation, these values were rounded to the next integer so that we can have 1 millisecond as the size of one time slot. The algorithms were implemented as part of a measurement orchestration server. The server was capable of accepting measurement specifications from a user (a researcher or community network manager) and store the collected network data in a central repository available to all users. The system was also designed to support passive measurements in the form of usage summary data from the smartphones of community network users.

The second part of our research focused only on TCP speed test performance due to their high execution time and throughput consumption. Motivated by the recent advancements in Software defined networking, we implemented a virtual network testbed that aimed to emulate a community network through the use of software defined networks. Community networks are typically implemented as a wireless mesh with one or more access points connected with each other for redundancy[19]. Thus, to virtualize such a network, we implemented switches that were capable of handling multipath routes. To compare our results with the case where there's little or no scheduling, we also implemented an approach

that randomly assigned measurements to the measurement nodes.

1.5 Thesis outline

The rest of this thesis is structured as follows. Chapter 2 provides a review of existing community network research. It also provides a comparative analysis of various measurement tools, platforms and scheduling algorithms that have been proposed in literature to address classical network measurement challenges and evaluates their usability in a community network context. Chapter 3 focuses on design and implementation of our measurement platform and the proposed evaluation framework. To this end, we discuss about setup of our community network testbed and define experiments and metrics to evaluate the performance of selected measurement scheduling algorithms. In chapter 4 we present the results of our experiments. Lastly, chapter 5 concludes the dissertation by detailing the findings of our study and discusses about what could be done as future work.

2

Background and Related Work

This chapter begins by highlighting some of the important community networks located all over the world. Section 2.1 also provides insights into different sub-areas of exisitng community network research. We then describe some of the existing measurement tools and platforms for classical networks in section 2.2 with a focus on the scheduling strategies used for measurements. In section 2.3, we describe different measurement scheduling that have been proposed to address the problem of managing resources in active network measurements. This is followed by an overview of research in software defined networking and a description of its relevance in this study in section 2.4. We end this chapter by providing some conclusions in section 2.5.

2.1 Community Network Research

Community networks have evolved into a variety of shapes and sizes with respect to their network technologies, their offered services and their organizational structure [4]. All over the world, citizens and organisations pool their resources and coordinate their efforts to

build network infrastructures [20]. At present, numerous community networks are operational all over the world, including those in the below subsections that have been surveyed for the purpose of this study.

2.1.1 iNethi

iNethi * is deployed in Ocean View and Masiphumelele regions in Cape Town, South Africa. It was created to offer cheaper internet access and encourage content sharing among members of the community [21]. The network uses mixed 2.4 GHz and 5GHz Wifi mesh networking powered by LibreMesh to connect WiFi hotspots [22]. It utilises open-source software to share files and to allow people to communicate using OwnCloud and Diaspora respectively [21].

2.1.2 Zenzeleni

Zenzeleni mesh network (ZMN)† is a community-driven, self-organized, decentralized and bottom-up mesh network deployed and operational in the Mankosi administrative area of the Eastern Cape Province, South Africa [23]. The network runs on low-cost mesh potato devices with solar power, enabling communication at affordable prices.

*https://www.inethi.org.za/
†https://zenzeleni.net/

2.1.3 Guifi.net

Based in Spain, Guifi.net [24] consists of more than 27,000 operational nodes, which makes it the world's largest community network in terms of the number of nodes and coverage area. Most of the links in Guifi.net are wireless. There's now an increasing number of fibre links being used [25].

2.1.4 Athens Wireless Metropolitan Network (AWMN)

Located in Greece, AWMN* is one of the largest community mesh networks in the world. As of 2011, it had over 9000 registered nodes, with more than 2400 of them being active. Some of the services in AWMN include mail, FTP, web hosting and game servers, VOIP, P2P file sharing, etc [26].

2.1.5 FunkFeuer

FunkFeuer † is composed of multiple smaller networks in Austrian cities like Graz and Wien. It is interconnected to Guifi.net and AWMN via CONFINE project ‡.

*http://www.awmn.net.
†https://funkfeuer.at/
‡http://confine-project.eu/

2.1.6 WLAN SLOVENIJA

wlan slovenija * is linked to FunkFeuer and AWMN and has more than 42,000 active users. An open source system, nodewatcher has been developed to allow coordination between participants in the network without requiring them to know each other.

2.1.7 WasabiNet

WasabiNet † is a mesh-based Wireless Internet Service Provider (WISP), operating low-cost Wifi in the St. Louis area in United States. Each mesh node works with the other nodes to find the shortest and fastest path through the mesh, thereby establishing a decentralized, ad-hoc infrastructure for routing internet traffic.

2.1.8 Bogota-Mesh

Bogota-Mesh's ‡ main objective is to link free technology to offer mass media a totally open, being a platform that decreases the digital divide in Bogota, Colombia. It is an autonomous network that can be used for different forms of communication, and sharing of social, cultural, and scientific projects.

*https://wlan-si.at/
†https://gowasabi.net
‡https://wiki.p2pfoundation.net/Bogota_Mesh

2.1.9 ITC e-choupal

ITC e-choupal provides a virtual marketplace where farmers can transact directly with a processor and can make better profits for their outputs[27]. ITC Limited is a private sector company behind this initiative that provides the farmers with computers and internet access across various villages in India.

2.1.10 AirJaldi

AirJaldi* provides high-quality broadband connectivity to rural areas in India at reasonable rates. It connects large and small clients from the corporate, civil society and private sectors. This initiative has its presence in 9 Indian states.

2.1.11 Wireless for Community (W4C)

W4C† is an init of Digital Empowerment Foundation (DEF) and the Internet Society (ISOC) that has been supported by various partners over the years. Launched in 2010, Wireless for Communities or W4C aims to connect rural and remote locations of India and Pakistan, where mainstream Internet Service Providers (ISPs) are not willing to provide internet connectivity as they feel their operations would not be commercially viable.

*https://airjaldi.com/
†https://wforc.in/

2.1.12 ZittNet

ZittNet's* objective is to improve access to communications in Kafanchan community of Nigeria by providing intranet and internet access to local partners. Some of the stakeholders in this initiative are educational institutions, faith based institutions, health services, small enterprises and individuals.

2.1.13 SNET

SNET or Street Network connects tens of thousands of residential users across Havana, Cuba. SNET is the standalone internet access option for most of it's users as it hosts over hundreds of websites including, a vast number of information and communication services [28].

A lot of research in the area of community networks is related to providing open access to internet data. Braem *et. al* [3] present a case of research for community networks and the relation to Community-Lab. Community-lab is an a open infrastructure that provides to researchers and experimenters a testbed to carry out experiments within wireless community networks. It consists of a set of nodes integrated into the existing community networks to give researchers access to the network and to allow them to perform experiments. As of 2013, this testbed was deployed in different cities across Europe, connecting multiple com-

*https://www.fantsuam.org/project/ict4d-ff

munity networks. Rameshan *et. al* [29] demonstrate a monitoring system tailored to meet the specific requirements of the community network testbed and propose an architecture for self management to automate management. Similar to the work by Braem *et. al*, the proposed architecture provides open access to other researchers to experiment with the data generated by the monitoring system.

Braem *et. al* [20] uses Community-Lab to gather data and focuses on the end-to-end quality of internet access in community networks. M-Lab's NDT was deployed on all available nodes in the Community-Lab testbed. The data collected through this work was logged to M-Lab's servers and was accessible via Google's BigQuery service. Download tests were conducted in Guifi.net, AWMN and Ninux* community networks and measurements like Round Trip Time (RTT) and throughput were collected. In this work, a comparative analysis with ISP per country is also presented. The analysis shows the effectiveness of community networks in providing satisfactory services to users, particularly effective for underserved and rural areas.

Some of the community network research is focused on providing access to cloud services from within the community network, instead of the internet. Jiménez *et. al* [30] describe the deployment of clouds in a community network. Their approach extends existing platforms and cloud software systems to achieve a feasible deployment of a cloud system in the Guifi.net community network. Baig *et. al* [31] show how cloud infrastructures have

*http://www.ninux.org

been made operational in a community wireless network, as a particular case of a community cloud, developed according to the specific requirements and conditions of the community. Unlike Jiménez *et. al*, they evaluate the usability, operation and sustainability of the cloud deployment and argue that other parameters like cost, security and innovation opportunity should be considered as well. Apolónia *et. al* [32] in a more advanced work, propose a system architecture for applying machine or container virtualization to the low-cost hardware used in community networks. Their comparative analysis with the current infrastructure in Guifi.net gives evidence of how devices can concurrently run multiple services. Their findings also highlight the tradeoffs between the number and resource requirements of services and the degradation of quality in services.

In developing regions, a large portion of community network research focuses on their establishment and operational challenges. Surana *et. al*[33] argue that most community internet infrastructures fail to stay sustainable over the long term, and do not go beyond the pilot phase. Besides providing a detailed list of operational challenges, they propose a monitoring system to capture active and passive network metrics in two rural networks: Aravind Eye Hospital in Southern India, and the AirJaldi community network in Northern India. ITC e-Choupal is another Indian community network, researched upon for it's sustainability [27, 34] and contribution in the Indian agriculture sector. Pujol *et. al* [28] present the first detailed characterization of SNET, Cuba, also known as Street Network. They describe

the network's infrastructure and map its topology, measure bandwidth, available services, usage patterns, and user demographics. In addition, scalability, security, and organizational challenges have also been discussed.

2.2 Measurement tools and platforms

An internet measurement platform is an infrastructure of dedicated probes that periodically run network measurement tests on the internet [35]. An internet measurement platform makes use of a variety of tools to provide information to the users about different aspects of network performance. Internet measurement tools range from simple commands included in common operating systems, like ping, traceroute, etc., through open-source applications to commercial packages and systems [36].

With the goal of building a measurement platform specifically for community networks, we surveyed the architectural features of popular measurement tools and platforms. We describe some of these platforms in more detail in the below subsections.

2.2.1 M-Lab

Measurement-Lab [37] is an open, distributed server platform for researchers to deploy active internet measurement tools. M-Lab's objective is to provide internet users with useful information about their internet performance. All data collected via M-Lab platform is openly available, and all the measurement tools used to collect this data are open source

*. Measurements are conducted between a user's device and the M-Lab servers to measure the end-to-end network performance [37]. M-Lab maintains a repository of existing tools and one of the most commonly used tools among these is the Network Diaganostic Tool (NDT). NDT measures TCP throughput between a client running at user's host and an M-Lab server [37]. An NDT test provides detailed packet level information along with kernel-level statistics on how the TCP connection performed in the given path. NDT helps determine the causes of slow speeds, as well as checks for proxies, NAT devices between the machine running the tests and the M-Lab server. The data collected is logged into M-Lab servers, which can be queried using Google's BigQuery †. One limitation of NDT [38] is that it evaluates network traffic between a mobile and the nearest M-Lab server, and not any arbitrary server. To avoid the over-utilization of resources, M-Lab recommends that all of its client integrations should use a Poisson [39] process to schedule at most four tests per day at random times.

MobiPerf ‡ is another open-source tool which was earlier hosted on M-Lab. It supported several types of network performance measurements. The MobiPerf application was developed to measure network performance and diagnose problems with application content delivery on mobile devices. It is based on the Mobilyzer [16] library that provides measurement isolation (only one experiment is active at a time), which avoids bandwidth con-

*https://www.measurementlab.net/
†https://cloud.google.com/bigquery/docs/reference
‡https://sites.google.com/site/mobiperfdev/

tention and radio power state transitions across experiments [38]. Throughput measurements are not carried out in parallel as the results may suffer from a bias [40]. MobiPerf takes comprehensive measurements of a smartphone's network properties, such as HTTP benchmark downloading latency (ms) and bandwidth (kbps), traceroute with latency (ms) to different hops, ping latency (ms), DNS lookup latency (ms), TCP uplink and downlink throughput (Kbps/Mbps), and IPv4/IPv6 compatibility. Throughput is measured by transmitting random data to a nearby M-Lab server for 16 seconds and then computing uplink and downlink throughput from packet traces. To obtain metrics like uplink and downlink UDP packet loss, out-of-order delivery and one-way latency, MobiPerf sends a series of UDP packets to a nearby M-Lab server, where the network metrics are calculated from packet arrival time and order [41]. Mobiperf allows the experiments to run in the background, which assists the researchers in monitoring long-term network performance. The current version of MobiPerf records a user's email address to access measurement history, if a user consents to provide one.

2.2.2 MySpeedTest

Unlike M-Lab, MySpeedTest is a standalone tool that measures network performance of mobile devices [42]. MySpeedTest can be used to perform both active and passive measurements. Passive measurements include total number of bytes received and transmitted by each active application, battery and package information, and application status(active

vs background). These metrics allow a user to understand which applications consume the most data and power, and the dependencies among the applications in terms of performance. Actively, MySpeedTest measures TCP uplink and downlink throughput, inter-packet delay and packet loss. Along with active measurements, application usage data is collected every 15 minutes [42]. The MySpeedtest application collects personally identifiable information like phone number, IMEI and device location, which we seek to avoid in our current work.

2.2.3 MONROE

MONROE * is another platform that solves a wide variety of hardware as well as software challenges associated with mobile networks. MONROE measures performance of Mobile Broadband (MBB) networks from the end-user perspective, using highly distributed measurements from fixed and mobile nodes [43]. MONROE makes use of an AngularJS-based user access and scheduling system for handling measurements. In the underlying nodes, MONROE leverages the use of container technology as it allows agile reconfiguration and flexibility in terms of adding more containers to deploy additional experiments [44]. Users can run baseline experiments like HTTP download, ping, and passive measurements using this platform [45]. All the produced data is stored in a non-relational database, that stores experiment measurements and metadata in separate collections [45, 44]. MONROE's vi-

*https://www.monroe-project.eu/

sualization tool comprises of time-based performance characteristics as line plots, and aggregated values as guage/pie charts, along with device tracking information in the form of indexed tables [44]. One of the key features of MONROE is it's ability to run real-time user-defined measurements on selected nodes. This aspect of MONROE's design can be replicated in a community network and nodes can be remotely triggered at scheduled time intervals to collect data.

2.2.4 Portolan

While most of the previous tools and platforms focus on measuring end-to-end performance of mobile application communications, Portolan is a tool that allows developers and researchers to learn more about the state of network infrastructure and configurations that affect transmission of application traffic [38]. Portolan takes advantage of crowdsourcing approach for measuring large scale wired and wireless networks [15]. The platform uses a software client that one can install on Android phones [35]. The client treats a smartphone device as sensors that can measure network-related properties. A central server is responsible for converting a large measurement task into microtasks and distributes them to users' smartphones through the help of dedicated proxies. Measurement tests like ping, traceroute, maximum throughput, and detection of traffic shaping of BitTorrent traffic can be conducted using Portolan. The Portolan mobile application limits the bandwidth usage to 2 MB per day and postpones the experiments when the battery level drops below 40%.

In our current work, we seek to adapt Portolan's deisgn, introduce passive measurement capability and allow the collected data to be publicly available to the stakeholders.

2.2.5 Ricercando

Having measurement tools is only the beginning of a successful network monitoring effort. To leverage the full capacity of these platforms, an equal amount of effort should be spent on gathering, transforming and visualizing the collected data. Ricercando [46] is a network data analysis framework that deals with the problem of detecting and explaining Mobile Broadband network performance issues. Ricercando enables multi-staged data analysis by introducing four key features. First, it introduces a data representation scheme that takes data from different sources and merges them into a time series-based representation, which is suitable for querying at different levels of granularity. Second, it provides interactive visualization of geographical and temporal multi-dimensional data. Third, Ricercando implements anomaly detection methods that pinpoint where network performance indicators deviate significantly from expected values. Fourth, Ricercando provides a machine learning pipeline designed to help with the identification of key factors leading to the observed anomalies. Ricercando's code-base is open to extension by the research community. This framework can be enhanced by introducing features that would enable the collection of experiment data around an anomalous point by triggering automated measurements.

Each of the platforms described above have their own advantages and disadvantages. In

this study, we attempt to adopt some aspects of the above platforms' designs in a community network monitoring platform. We will be considering a centralized crowdsourcing based architecture similar to Portolan. In order to reduce installation and maintenance costs, task allocation can be restricted to only the centralized server without the use of strategically located proxies. For active network monitoring, MobiPerf can be used in the measurement nodes as it supports a rich set of measurements and is also open source. Passively, data can be collected from the network firewall in the form of files and can be analyzed within the centralized server. Another form of passive data collection can be performed by measuring application internet consumption in the users' smartphones, similar to MySpeedTest. Further, the measurement nodes can be triggered at periodic intervals by adopting MONROE's approach and the measurement results thus obtained can be processed and visualized using Ricercando.

2.3 Scheduling algorithms

Round Robin [47, 48] and Earliest Deadline First [49] are some of the most commonly used scheduling algorithms with applications in process and network packet scheduling. While Round Robin algorithm lets the tasks take turns for execution, the Earliest Deadline First scheme selects tasks based on their deadlines. Both of these algorithms are designed for uniprocessor systems and cannot be used in a concurrent context. Therefore, in order

to schedule tasks in nodes scattered within a community network, these algorithms would require appropriate modifications. In this subsection, we provide details about suggested enhancements in these algorithms in order to solve the problem of network measurement scheduling, discuss about additional distributed algorithms that deal specifically with network monitoring, and comment on their usability in low resource networks.

2.3.1 Controlled Random Scheduling

Controlled Random Scheduling (CRS) [50, 51] is a distributed scheduling algorithm designed to avoid network congestions by reducing the amount of concurrent measurements. CRS assumes that each node can be either in a measurement state or sensor state at a given point of time. Switching between these states is performed by using a controlled random function that makes use of pseudo randomizers to generate a random number and then comparing it against a threshold. This algorithm requires that each node must know how to reach the other nodes at the time of scheduling.

2.3.2 Controlled Priority Scheduling

Controlled Priority Scheduler (CPS) is a suggested improvement over the CRS algorithm. CPS inherits the distributed and concurrent properties from CRS, where each node alternates between a measurement state and sensor state at a given random period. However, the CPS algorithm uses a priority based scheme to decide which monitor/sensor pairs to

measure. The CPS algorithm is designed to prevent starvation and get a more consistent monitoring period between all the monitor/sensor pairs.

2.3.3 Round Robin Scheduling

The Round Robin (RR) algorithm [52] runs a job for a time slice, or scheduling quantum and then switches to the next job in the run queue. During this quantum, only one job is executed while the rest of the jobs in the queue wait for their turn. When a quantum reaches its allocated limit, the next scheduled job will be executed. Round Robin algorithm ensures that no process is starved, and there are no measurement conflicts among the processes. Since concurrent measurements cannot be performed, the only way to decrease the time to reach full monitoring coverage would be to decrease the time it takes to monitor each scheme [53]. This algorithm does not scale when the number of jobs to be executed increases. One drawback of round robin algorithm is that it cannot be utilized in a concurrent context. Qin *et. al* [18] suggest an improvement in the original round robin scheme (Figure 2.1) and enhance it with concurrent execution capability. The improved scheme selects tasks for execution by following a pre-defined order, provided as an input along with conflict relationship of the jobs. A scheduling point is defined as any time instance when a new job gets available for execution or one of the current jobs finishes it's allotted time slice. At a scheduling point, if there is no conflict with the current on-going task, the job is scheduled, otherwise it is kept in the job queue for execution at the next scheduling point.

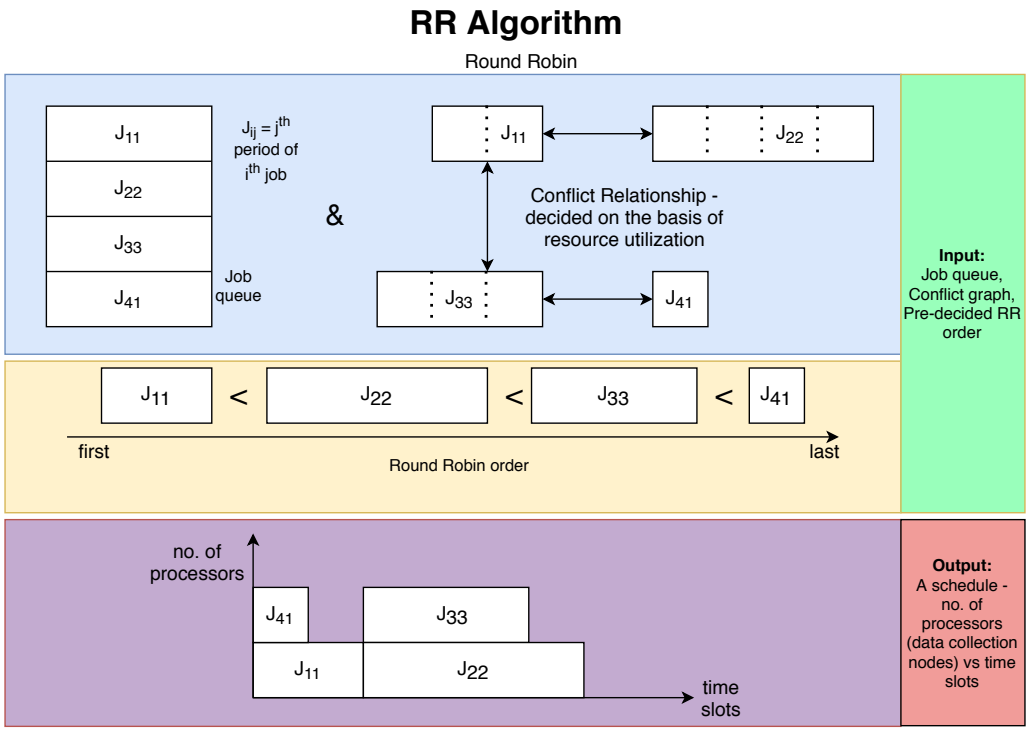


Figure 2.1: A figure showing the modified version of Round Robin Algorithm, adapted from Qin *et. al* [18]

2.3.4 Earliest Deadline First Scheduling

The Earliest Deadline First (EDF) algorithm selects a job with the earliest deadline. This algorithm assumes that whenever the execution of a job is ended, the next job can start without wasting any time. However, this algorithm does not avoid deadline-misses [54]. Thus, the execution of a job can start in an iteration but it may not necessarily end in the same iteration. This algorithm has been shown to be optimal [55] and comes with a variety of variations in real-time systems. One of the variations considered for this study has been

proposed by Calyam *et. al* [49] (Figure 2.2). A novel mechanism to flexibly use the offline schedule for minimizing the response time of on-demand measurement jobs has also been proposed. This variation of the EDF algorithm allows concurrent executions, if possible, to construct a schedule for a periodic measurement task set. Similar to concurrent round robin algorithm, this algorithm also accepts a conflict relationship among the jobs as an input.

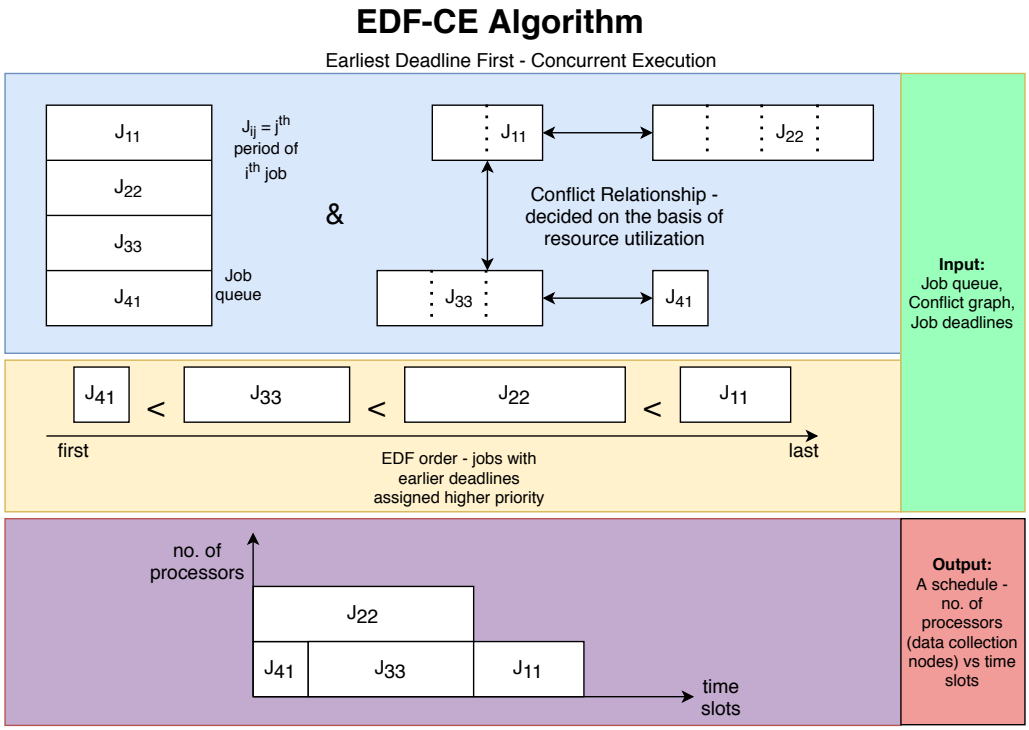


Figure 2.2: A figure demonstrating the EDF algorithm [49]

2.3.5 Graph coloring-based Scheduling

Qin *et. al* [18] propose an active measurement scheduler for both periodic and on-demand tasks. This concurrent algorithm is driven by a graph colouring perspective, called ascending order of the sum of clique number and degree of tasks. The algorithm focuses on reducing the average waiting time for periodic monitoring while reducing measurement conflicts. For on-demand tasks, the proposed scheme minimizes the waiting time of inserted on-demand tasks while keeping time space utilization high. This algorithm accepts just the conflict relationship among the jobs as an input, and does not use any user-defined heuristic to generate a schedule. Two variants of the proposed scheme are considered for evaluation - AOSD and DOSD, corresponding to ascending and descending order of subvertices' degree respectively. Both these schemes rely on a centralized point of scheme generation and task reporting, which according to Mathew Clegg [53], is a drawback. However, this could benefit this particular study as a centralized measurement system is being developed.

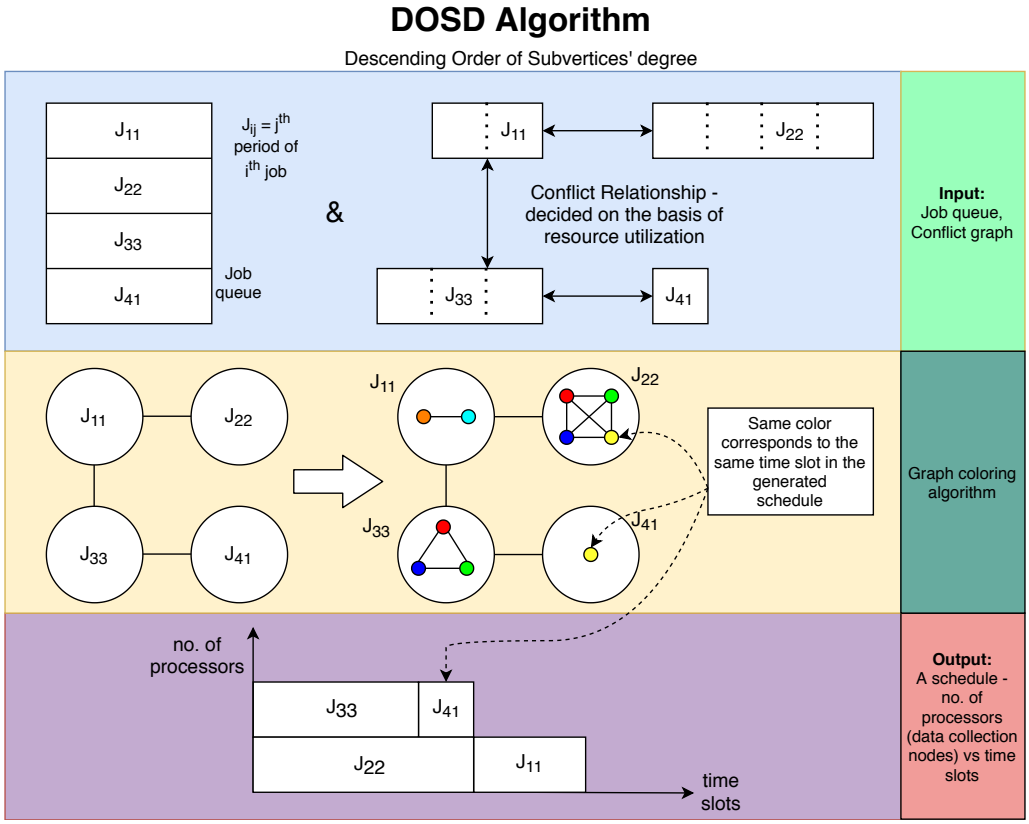


Figure 2.3: A figure showing the DOSD algorithm, proposed by Qin *et. al* [18]. AOSD algorithm works in a similar manner with the difference that it arranges the subvertices in ascending order of their degree.

Both CRS and CPS algorithms are based on a common scheme that requires the measurement nodes to be reachable from one another. If implemented in a low resource context, this would increase the amount of traffic and could lead to bias in measurements. On the other hand, the concurrent version of RR, EDF, AOSD and DOSD require a conflict relationship between the jobs as an input, which makes these algorithms easy to compare. Also, AOSD has been shown to outperform EDF and round robin with the help of

computer simulations, it's implementation on a non-synthetic testbed has not been yet explored. Therefore, we will consider concurrent RR, EDF, AOSD and DOSD schemes for comparison in our work.

2.4 Software defined networking

Software defined networking (SDN) is a network design paradigm that aims to address the limitations of existing networks by separating their control plane from the data plane [56, 57]. This separation of concerns provides a number of important advantages. Not only does SDN provide the ability of programmability to networks, it also simplifies network management by logically centralizing the control plane [58]. Additionally, modification of network policies becomes easy and accurate by the use of high-level languages and software components, as compared to device-specific configurations in traditional networks. In traditional networks, where the control plane and data plane are not separate, automatic configuration and dynamic enforcement of policies is not possible. On the other hand, in software defined networks, a centralized controller is able to access each switch's control plane. The policy decisions made at the controller are installed as forwarding rules in the data plane through the SDN's northbound API. This helps in performing configurations as and when required [59].

A software defined network (Figure 2.4) comprises of 3 well-defined layers: the applica-

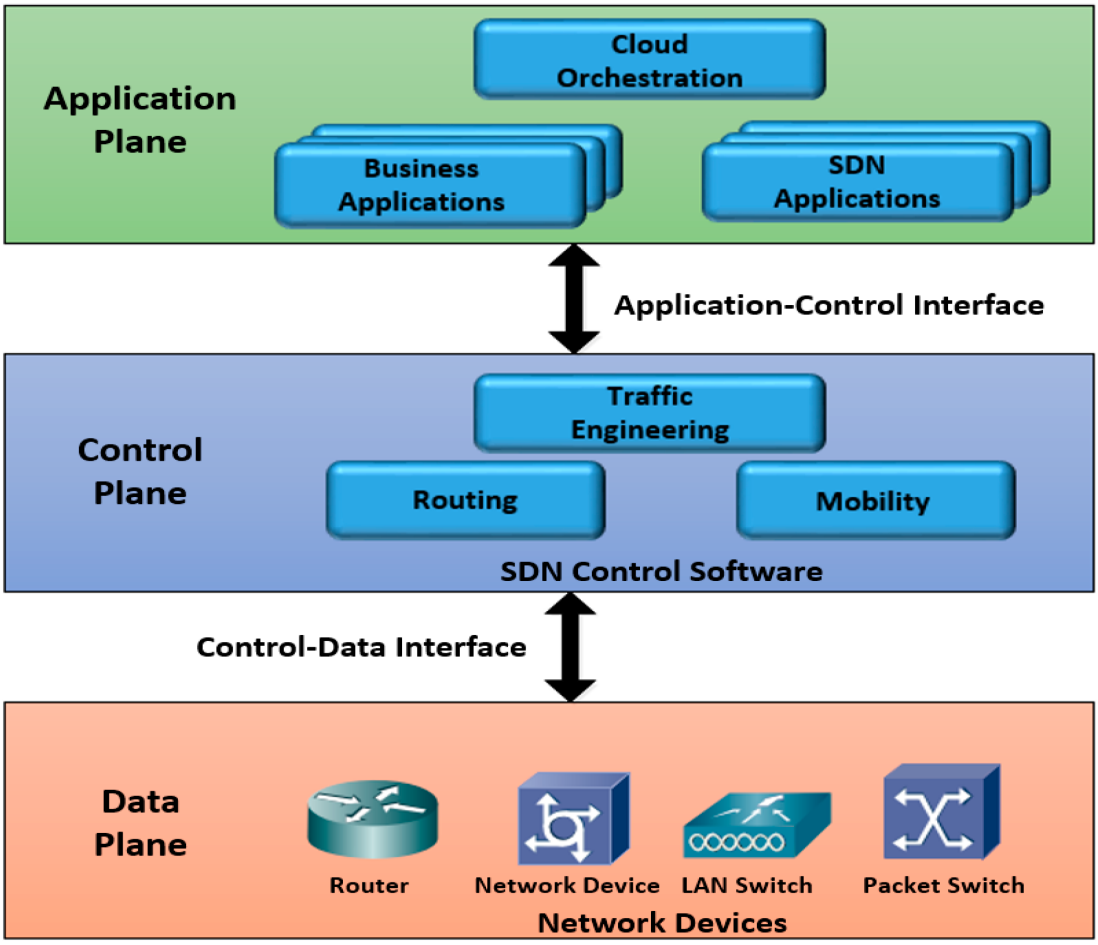


Figure 2.4: Architecture of a Software defined network [60]

tion plane, the control plane and the data plane. Forwarding devices such as switches and routers reside in the data plane, while one or more centralized controllers manage these devices by implementing application-specific policies. A controller is typically a multi-threaded program that communicates with the data plane using the OpenFlow protocol [10]. Depending on the type of application, a controller is able to access and modify forwarding tables in the data plane devices. Since the controller has a global view of the entire

network, it can implement the logic to determine the forwarding path of each flow in the network and modify the tables at runtime. The control plane usually exposes REST APIs to a variety of applications like routing algorithms, intrusion detection systems and load balancers [61].

Salman *et al.* [62] provide a comparative analysis of a number of network controllers and state that the choice of an SDN controller depends on several criteria and the user. For example, controllers like OpenDayLight * and Floodlight † have been used in large scale networks like data center networks, while other controllers like Ryu ‡ are designed for less specific environments like Campus networks. Also, a performance comparison [63] of Floodlight, OpenDayLight and Ryu has shown that Ryu's CPU consumption is much lower than OpenDayLight and FloodLight. Community networks are built using off-the-shelf hardware with limited network and CPU resources. Therefore, the network controller scripts for this dissertation were chosen to be written using Ryu's API.

2.5 Conclusions

In this chapter, we first provided an overview of existing community network research. Then, we provided a discussion on measurement platforms that have been used to actively measure classical networks. We also discussed the applicability of these measurement plat-

*https://www.opendaylight.org/
†https://github.com/floodlight/floodlight
‡https://ryu-sdn.org/

forms in community networks. Then, we discussed about various scheduling algorithms that have been proposed to address the problem of resource utilization in active measurements. Lastly, to design an evaluation testbed for the system, we survey existing software defined networking literature and evaluate different design choices in a low resource context.

Most of the QoS monitoring research on community networks has been done on the basis of Guifi.net (the largest community network) or other interconnected European community networks. Limited research has been performed on monitoring network characteristics in the context of community networks for developing regions, and measurement scheduling aspect of frameworks that characterize these networks has mostly been overlooked. In classical networks, the design and evaluation of measurement scheduling algorithms is based on computer simulations or synthetic testbeds. Also, these algorithms rely on the assumption that the execution time of measurements is a constant, contrary to a real-world scenario where it could vary because of several factors like signal strength, geolocation and time of the day. In our work, we make an attempt to apply the principles used in literature to develop a QoS measurement platform that can be used and tested in any real-world low resource network.

With the increasing popularity of SDN as a network management paradigm, we aim to conduct performance evaluation of our measurement system in an SDN-emulated testbed.

The primary objective of these evaluations will be to minimize the error in active measurements due to the observer effect. As part of the measurement system, we implement four measurement scheduling algorithms that formulate this minimization problem as a conflict-aware job scheduling problem as proposed by Qin *et. al.* We also implement an additional scheduling approach that assigns measurements immediately to end user devices to baseline our results.

3

Methodology

This project aims to evaluate existing network measurement scheduling strategies in the context of community networks. To this end, we develop a measurement system for the iNethi community network, located in Ocean View, Cape Town. The design principles that we use in developing this system in this chapter have found their utility in other related research studies [64, 65]. Our approach is divided into four main parts. The first part comprises of the implementation of four existing measurement scheduling algorithms as guided by Section 2.3, namely, Round Robin (RR), Earliest Deadline First (EDF), Ascending Order of Subvertices Degree (AOSD) and Descending Order of Subvertices Degree (DOSD). They are implemented as part of a centralized server that distributed measurements to data collection nodes.

The second part deals with the development of an application that could utilize these algorithms to schedule measurements on end user devices. Past research deals with implementation and testing of the above algorithms on synthetic testbeds but our aim was to evaluate the complete measurement system as a single unit.

The third part of our methodology is the development of an end user application that could execute measurements and send the results to a central repository. For this purpose, the Android platform is chosen as about 66% of users in our target community network, iNethi had access to Android phones [66].

Lastly, we develop a testbed for the performance analysis of our system. It is designed as a containerized application that made it possible to be deployed in any real world community network. Additionally, we develop an SDN controller application for the community network. We test the system on a virtual network prior to a real-world deployment. This approach also allows us to make use of different network topologies during the evaluation phase. The scripts for performance analysis are written in Python as there are Python libraries that provide extensive support for interacting with docker containers.

3.1 Proposed measurement system

The main component in the proposed architecture is a measurement server, which is responsible for orchestrating measurement experiments across data collection points distributed within a community network. The measurement server can be configured to run one of the scheduling algorithms described above for providing orchestration to user-specified experiments. Each user-specified experiment is converted into a job by the measurement server and is allowed to execute in one or more measurement nodes depending

upon their availability. We implement RR, EDF, AOSD and DOSD algorithms for implementation within the measurement server. Additionally, we use a randomized scheduling approach for the purpose of performance comparison. This approach assigns an available job to any random device that is connected to the server. In addition to the measurement server, a user interface is developed to allow researchers schedule experiments, visualize the collected data and perform analysis. Through this interface, network managers will be able to investigate performance-related issues and also monitor the measurement nodes and their corresponding access points. Community members will be able to visualize application usage on their smartphones in terms of number of bytes transmitted and received across the network.

Figure 3.1 shows the architecture diagram of the proposed system. The key components in the system as shown in the figure are described in the subsections below.

3.1.1 User Interface

The user interface consists of measurement initiator and data visualizer. Using the measurement initiator, a researcher can schedule experiments in the job queue. The data visualizer component is where the researchers will be able to graphically view the results of all the experiments scheduled by them. For development of the user interface, ReactJS was used, which is a component based library deployed for the development of interactive user interfaces [67]. ReactJS uses an efficient and lightweight document object model, which boosts

performance. Also, it is supported by vast community of individual developers and organizations. Thus, ReactJS is a good choice for the user interface.

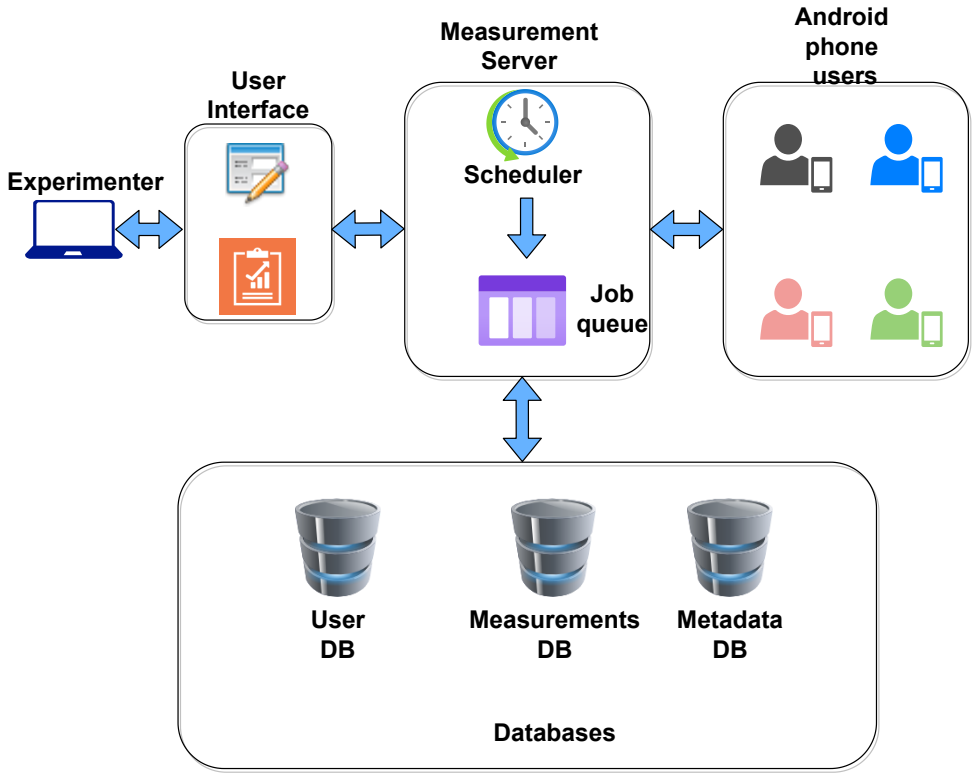


Figure 3.1: A figure showing the architecture of QoSMon, the proposed measurement system

3.1.2 Measurement Server

The measurement server is the main component of the system that receives and handles requests from users. It has a job scheduler that is responsible for scheduling measurement

experiments that users want to run. The scheduling algorithm to be used in the job scheduler can be configured by an administrator. Algorithm 1 shows a generic implementation of the four algorithms used. While RR and EDF algorithms sort the jobs in order of their arrival and deadlines respectively, DOSD and AOSD construct a subgraph using the conflict matrix and colour it appropriately. The colour of a particular node in the subgraph determines which time slot is assigned to the job in the generated schedule. The implementation of these algorithms is adapted from Qin *et. al* [18]. We include additional implementation details like the job assignment criteria in Algorithm 1. Devices take turns in executing the jobs in a circular fashion. This design choice has been made to ensure that the measurement load is shared equally among the devices in all circumstances.

Lastly, the measurement server handles the logic for communication with databases. Any data manipulation or aggregation logic is also handled in this component. To implement the measurement server, we use Java's Spring Boot [68] framework. The primary reason for using Spring Boot for development of our system is that it provides embedded HTTP servers like Jetty, Tomcat etc., which reduces the development and testing time. To ensure portability, performance and ease of deployment in the community network cloud, the measurement server will be containerized using docker.

Algorithm 1 A generic conflict-aware scheduling algorithm implemented as part of the measurement server

Input: $J, \mathcal{M}, D, E$ ▷ List of jobs, Conflict Matrix, List of devices, Assumed execution times
Output: $\mathcal{A}$

```
device_index ← 0                          ▷ Index of the device to be assigned to an outgoing job
J_p ← []                                  ▷ An empty list to hold parallelly running jobs
A ← {}                                    ▷ An empty mapping for job assignments
P ← PriorityQueue()                       ▷ Holds scheduling points
while P is not empty do
    C ← P.poll()                          ▷ Current scheduling point
    for all j in J_p do
        if j.dispatchTime + E[j.type] = C then Remove j from J_p
        end if
    end for
    for all j_1 in J do
        if j_1 not in A then
            hasConflicts ← False
            for all j_2 in J_p do
                if M[j_2] contains j_1 then
                    hasConflicts ← True
                end if
            end for
            if not hasConflicts and |J_p| < |D| then
                device_id ← D[device_index]
                device_index ← (device_index + 1)%|D|
                j_1.dispatchTime ← C
                A[j_1] ← {C, device_id}
                P ← P + C + E[j_1.type]
                J_p ← J_p + j_1
            end if
        end if
    end for
end while
```

3.1.3 Data collection points

The data collection points consist primarily of Android phones but our platform's support can easily be extended to Raspberry Pis as well. The Raspberry Pis execute special-

ized scripts for periodical as well as on-demand collection of QoS data, while the Android phones run a customized version of MobiPerf. The communication between the data collection points and the measurement server is achieved through HTTP websockets. This is our modification over original MobiPerf's polling mechanism where the data collection points had to check-in with the server at regular intervals. In our implementation, measurement requests are dispatched to the data collection points in the form of jobs as and when they are ready to be executed.

3.1.4 Databases

Three types of databases are used to store the information related to the measurement system.

3.1.4.1 User database

The user database stores information to the role of the user (researcher, network admin or community member), the username and password, both in encrypted format. MongoDB*, a document-based database has been used for this purpose as it provides great performance, allows fast querying, is horizontally scalable and provides well-documented Java libraries.

*https://www.mongodb.com/

3.1.4.2 Measurement database

This database is responsible for storing information related to experiment results. Each recurring job is assigned a unique key by the system. To separate the individual instances of each job, we use the instance's end timestamp as an identifier. Time-series databases are known to store measurements indexed by their timestamps and allow efficient real-time analysis for large volumes of time-series data. Therefore, we use influxDB *, a widely used time-series database for this purpose.

3.1.4.3 Metadata database

The metadata database stores the past experiment requests by the researchers. These experiments can be shown under the profile of the user on the user interface. The requests will be assigned a unique key against which the experiment parameters will be stored in a database for fast retrieval. Therefore, for this purpose we use MongoDB database.

3.2 Community network testbed design

Figure 3.2 depicts the architecture of the testbed deployment of our system in the iNethi [22] community network. This testbed is composed of two measurement servers, each located at the University of Cape Town and the community network respectively. Two

*https://www.influxdata.com/

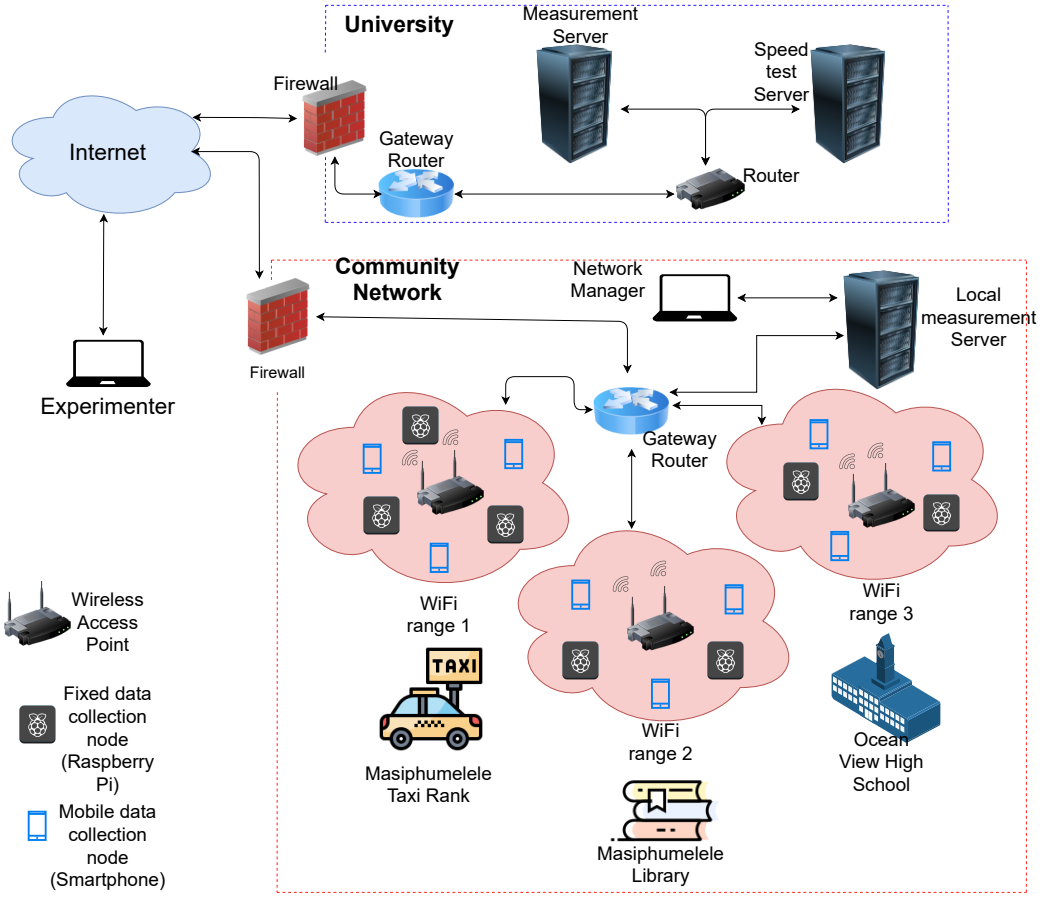


Figure 3.2: A figure showing the testbed set up in iNethi community network for the proposed system

instances of the measurement server are necessary to separate the measurements that will be launched by researchers from those by network managers. Additionally, a speed test server is used to measure the end to end throughput between the community network and the university. This throughput value will be of special interest to researchers as it will provide them with valuable insights about how the community network is performing. A researcher does not have to be within the university campus to launch the measurements on

the community network. This aspect of the testbed's design allows authorized experts of the world to conduct research on the network and help it grow further to support multiple communities. The measurements will run on the Android devices of iNethi users and multiple Raspberry Pis will also be deployed at different access points.

3.3 Performance testbed design

Performance analysis for the system was divided into two phases. In the first phase, the evaluation of the system was based on a simulated topology. A detailed analysis of the experiments conducted during this phase is discussed in our prior works [69, 70]. Our implementation of the network topology was restricted to an adjacency list and the end-user devices used were smartphones. The choice of an adjacency list allowed us to model a real-world community network as a logical connection of smartphones with access points. In the second phase, we built a virtual network using the SDN mininet emulator. To run measurement scripts inside SDN hosts, we extended our platform to run measurements within docker containers. We used Containernet *, a version of the Mininet † network emulator, to run our containerized Mobiperf clients.

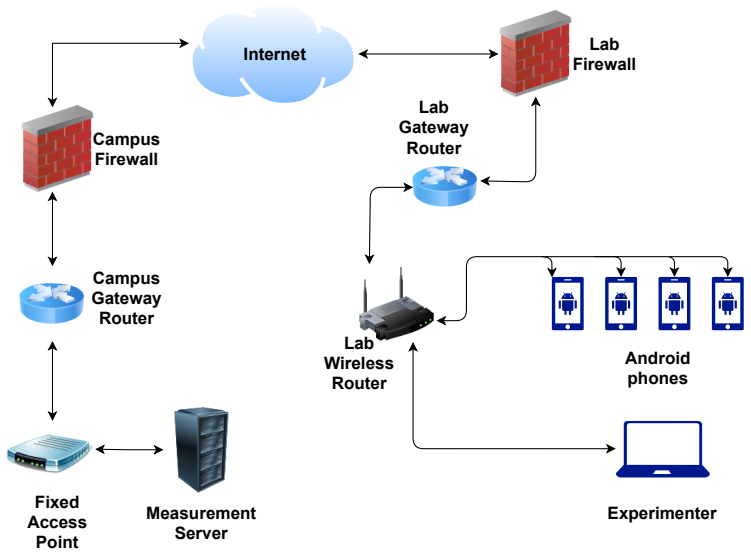


Figure 3.3: A figure showing the testbed for QoSMon in the first phase of performance evaluation.

3.3.1 Phase 1 Evaluation

Figure 3.3 shows the performance testbed for the first phase of evaluations, i.e evaluation of the simulated topology with actual smartphones. Four Android phones, co-located with an experimenter in a lab were used to run measurements. We first conducted a set of preliminary tests to determine the execution time of each job type and calibrate our scheduling algorithms. We assumed the maximum time across each job type to be the expected value of execution time for any upcoming job in our system. The maximum value of execution time

*https://containernet.github.io/
†http://mininet.org/

would be a good estimate for setting a safe time window so that the next job will execute after the previous job has finished execution.

The key aspects of this testbed's design include the selection criteria of devices and conflict determination among measurements, selection of network topology, and the choice of measurement profiles. Each of these aspects are described below.

3.3.1.1 Conflict Determination and Device Selection

Conflicts between individual measurement instances are first decided on the basis of target server. If two measurements are destined to the same target server, it is highly likely that they use some common links in the network. In a low resource set up, we would want the target server to handle the least load possible and not get overwhelmed by large number of measurement requests. Thus, we use a pairwise binary matrix for representing conflicts between active measurements. This binary matrix is then used to build an undirected conflict graph of the measurements and supplied to the scheduling algorithms as input.

Our implementation is also capable of addressing conflicts that arise due to the topology of the network. During initialization phase of the measurement server, we load the network topology as an undirected graph and then calculate the cost of scheduling on each measurement node. A drawback of this approach is that the measurement server is not sensitive to sudden changes in the network topology. We address this problem in a later implementation.

The scheduling cost is calculated by summing up the number of affected links in the network. We thus calculate all simple paths from a measurement node to the gateway access point and add the number of edges in the network graph. Fig. 3.4 shows an example of a network topology in which AP3 is the gateway access point. Requests from M4 to a target server anywhere on the internet can be routed through one of the paths in the set {M4 → AP6 → AP2 → AP4 → AP3, M4 → AP6 → AP2 → AP1 → AP3, M4 → AP6 → AP2 → AP1 → AP5 → AP3, M4 → AP6 → AP5 → AP3, M4 → AP6 → AP5 → AP1 → AP3}. We calculate the scheduling cost by adding up the number of links in these paths irrespective of the number of times they appear in each path. Therefore, the scheduling cost for M4 is 20 in this example. We then assign the first available device with lowest value of scheduling cost to an executing measurement.

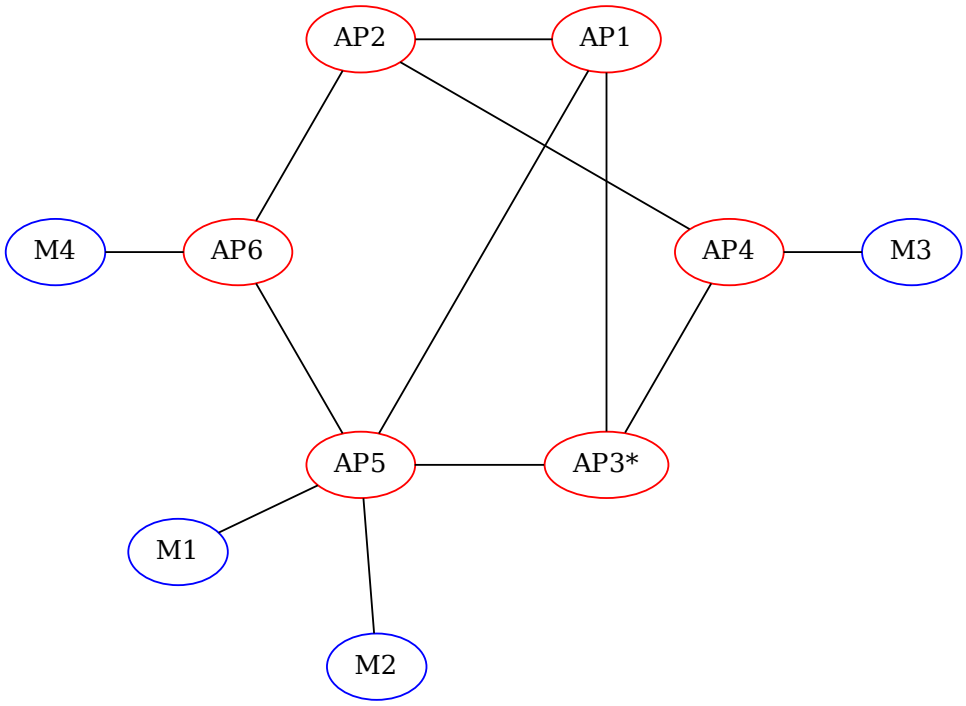
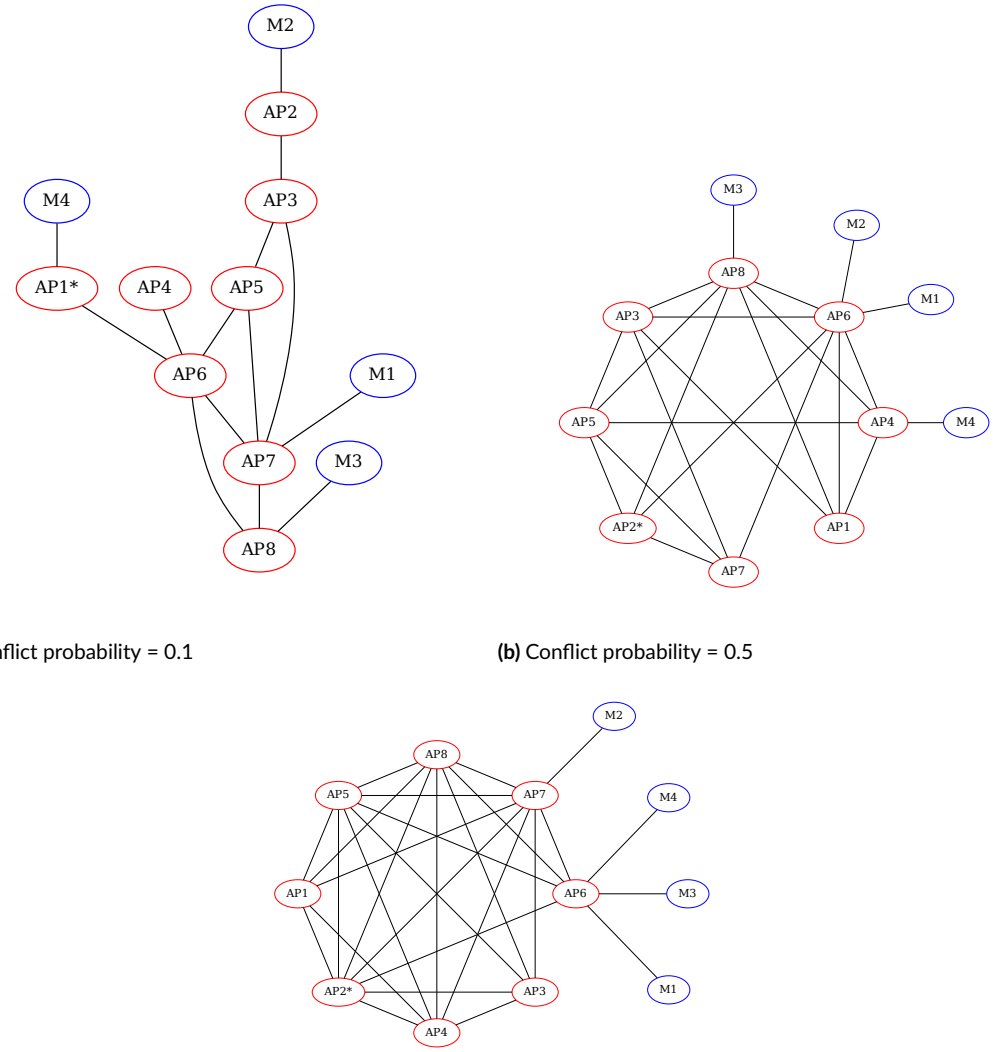


Figure 3.4: A figure showing an example topology to illustrate the calculation of scheduling costs. In this figure, blue ovals represent data collection nodes and red ovals represent access points in the network. The access point marked with an asterisk (*) represents a gateway access point, i.e an access point that is connected directly to the network's firewall.

3.3.1.2 Choice of Network Topology



(a) Conflict probability = 0.1

(b) Conflict probability = 0.5

(c) Conflict probability = 0.9

Figure 3.5: Choice of network topologies for our experiments with 8 access points and 4 measurement nodes

To evaluate the system's performance on different types of network architectures, we decided to execute our experiments on sparsely, moderately and densely connected network topologies. To achieve this, we associated a conflict probability between the hosts and access points in our virtual network. The higher this probability, the denser would be the connectivity in the network. For lower values of conflict probability, there was a risk of introducing a connectivity gap within the network. We prevented this by allowing at least one gateway access point to be available in each connected component. Figure 3.5 shows the network topologies generated using the conflict probability as 0.1, 0.5 and 0.9 respectively.

3.3.1.3 Measurement Profiles

For each scheduling algorithm, we chose 20 measurements for execution in 2 hours. The measurement types were chosen randomly from the five available types- ping, DNS lookup, traceroute, HTTP download and TCP throughput. The period of each job was chosen uniformly from the integral range of 5 to 10 minutes. All TCP speed test jobs had the same target server, which corresponded to the speed test server from figure 3.3. For the rest of the job types, the target servers were chosen uniformly from top 8 Alexa global websites [71]. At the start of each experimental iteration, the measurement server was configured with the combination of a scheduling algorithm and network topology. Thereafter, the jobs were allowed to run until 2 hours elapsed. Results stored in InfluxDB were extracted into CSV files and were used for evaluation. This process was repeated until all possible combinations

of scheduling algorithms and topologies were exhausted.

3.3.2 Phase 2 Evaluation

The results from the first phase of evaluation suggested a number of changes in the system's design. First, it was found that the algorithms performed similarly on all three types of topologies. This motivated the full virtualization of the community network using SDNs. Secondly, TCP speed test jobs were found to consume the most network and CPU resources, which led us to shift our focus only on these type of jobs. Another reason for focusing just on these jobs was the idea that a speed test server can also be made part of the network topology. Since network emulators like containernet allow us to create point-to-point links with fixed bandwidth and delay, we could compare the speed test results against the throughput value that we fix on the links.

In this project, the ability of the system to schedule measurements is of a greater interest to us than network metrics like throughput, latency, packet loss and jitter. So we evaluate the system by using metrics that can highlight the effectiveness of the scheduling algorithms used. We describe these metrics below.

Figure 3.6 shows an SDN-based testbed for the measurement system. In this experimental setup, the measurement server and the emulated network reside within a single host. The default network interface of this host has been bridged and a virtual machine is connected through it. The virtual machine is equipped with containernet which allows us to

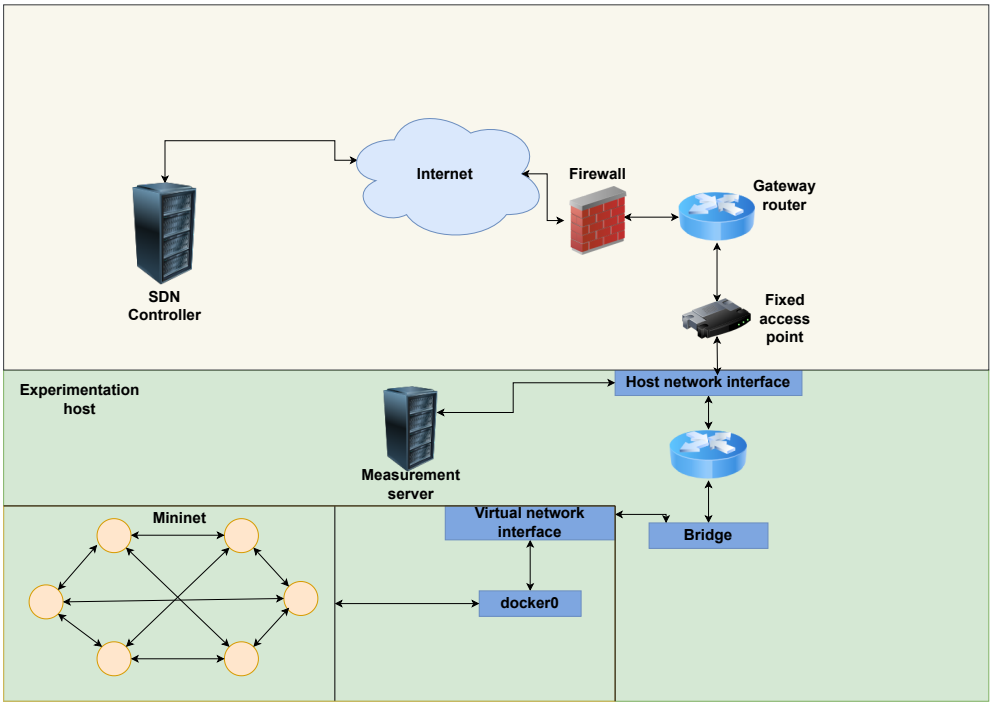


Figure 3.6: A figure showing the testbed for QoSMon in the second phase of performance evaluation.

create custom network topologies with hosts able to run measurements against the measurement server. The controller for the emulated network was chosen to be remote and it implemented a multipath routing algorithm [72]. We used breadth first search to discover the shortest path from a given source to a destination. The reason for choosing breadth-first search was that the network topologies used were represented as unweighted graphs and breadth first search by design finds the shortest path to a given node being visited. In case of a weighted graph, the implementation can be modified to use Dijkstra's shortest path algorithm.

3.3.2.1 Conflict Determination and Device Selection

In this phase of evaluation, since only TCP speed test jobs were used and were directed to a single target server, we decided to make the jobs to be pairwise conflicting with each other. As a result, the resultant schedules were expected to be non overlapping with each other as no two jobs could be scheduled parallelly on two devices. Devices were decided to take turns for executing a newly available job as it allowed an even load distribution among the nodes.

3.3.2.2 Choice of Network Topology

Previous work [73] states that wireless community networks are typically designed in such a way that multiple nodes are connected to each other through a single link. The only way to represent such links in a software-defined context is to have point-to-point links in which one node is individually connected to all the other nodes forming a clique (Figure 3.7). Our target network iNethi has 9 access points meshed together, so we decide to build clique-based topologies with number of access points ranging from 4 up to 15. This helped us not only in finding out trends in scheduling algorithm performance with varying network sizes but also tackle the issue of scalability if the network size grows in future.

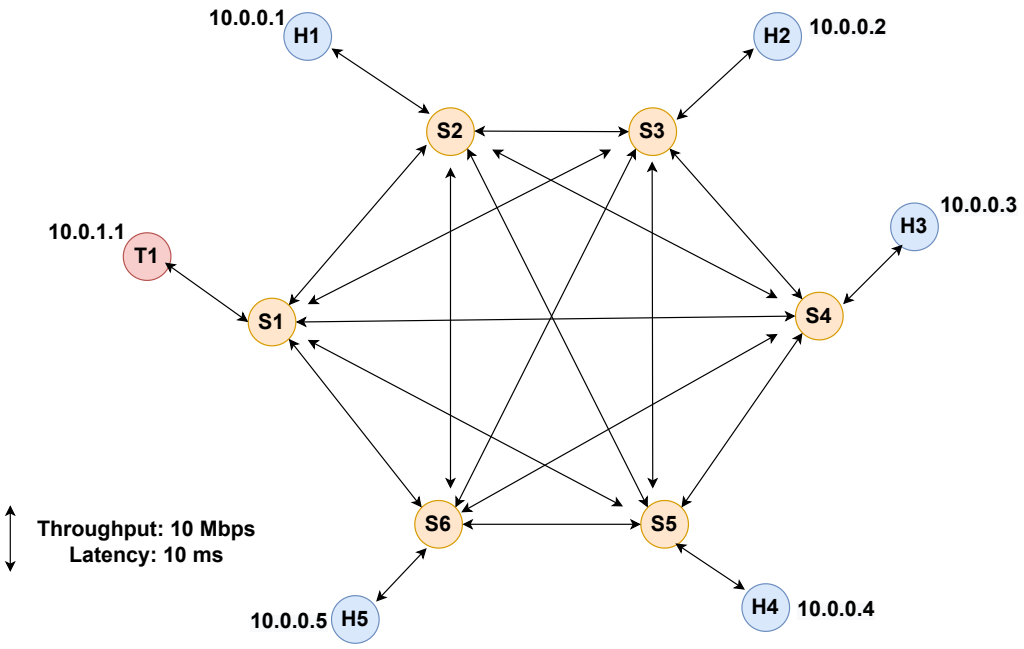


Figure 3.7: A clique with 6 access points. One of the access points is connected to the speed test server (T1) while the rest of them are connected to the measurement nodes. Traffic flows from a measurement node (H1 through H5) towards the speed test server or vice versa, depending on whether an uplink or downlink speed test is scheduled. Each link was allocated a throughput of 10 Mbps. This allowed us to have a fixed value of end-to-end throughput for every combination of source and destination.

3.3.2.3 Measurement Profiles

For each scheduling algorithm, we scheduled 50 measurements for execution in 3 hours. All of these measurements were either uplink or downlink speed tests, chosen randomly. The period of each job was chosen uniformly from the integral range of 10 to 20 minutes. Like in the first phase, these TCP speed test jobs were destined to the same target server. This server was assigned a unique IP address of 10.0.1.1 in all topologies. The key difference between performance evaluation in the first and second phases was that in the second phase

the specifications of each job were kept the same across all configurations of the measurement server.

3.3.3 Performance metrics

The following metrics were used to evaluate the performance of the measurement system. We consider the average value of each of these metrics during evaluation. Since the jobs in consideration are periodic, this average is calculated across all instances once the measurement server is configured with a particular scheduling algorithm.

3.3.3.1 Measurement Success Rate

This metric provides the fraction of measurements that executed successfully by taking all of their individual instances into account. For example, if a measurement M_i can theoretically run for 10 times during the experiment duration but it successfully executes only 8 times, then it's success rate would be 80%. Measurement success rate, MSR can be calculated using the formula below.

$$MSR = \frac{Number\ of\ successful\ measurement\ instances}{Total\ number\ of\ instances} \times 100 \tag{3.1}$$

Here, we calculate the total number of instances using Algorithm 2.

Algorithm 2 Algorithm to calculate the total number of measurement instances

Input: $\mathcal{M}$ ▷ This holds measurement metadata
Output: *number_of_instances*
  current_time ← $\mathcal{M}$.*start_time*
  number_of_instances ← 0
  while *current_time* ≤ $\mathcal{M}$.*end_time* **do**
    number_of_instances ← *number_of_instances* + 1
    current_time ← *current_time* + $\mathcal{M}$.*period*
  end while

3.3.3.2 Measurement Error

This metric provides insights into how the observer effect operates when a certain number of measurements are scheduled within our virtual network. This metric was used in the second phase of evaluations. The measurement error, ME for a single speed test measurement is defined as:

$$ME = \frac{\textit{Baseline value of throughput} - \textit{Observed value of throughput}}{\textit{Baseline value of throughput}} \times 100 \qquad (3.2)$$

As stated previously, the baseline value of throughput was kept as 10 Mbps across all links in the experimental network topologies.

3.3.3.3 Waiting Time

Waiting time for a measurement is defined as the time it spends in the waiting queue before it is dispatched to a device. It is useful to measure the waiting time because it gives us

the opportunity to find out the precise reason why a particular measurement's result was delayed or whether or not a measurement misses its deadline. Waiting time, WT is defined as:

$$WT = \textit{Dispatch time of a measurement} - \textit{Enqueuing time of the measurement} \tag{3.3}$$

3.3.3.4 Platform Delay

Platform delay for a job is defined as the time it takes for its result to be stored in the measurements database starting from the arrival time of the job. This metric determines how quickly the measurement system completes periodic measurements. Along with waiting time, platform delay can also be used to calculate the time that a job spends out of the measurement system. Platform delay, PD is defined as:

$$PD = \textit{Completion time of a job} - \textit{Enqueuing time of the job} \tag{3.4}$$

3.3.3.5 Node Busy Time Ratio

This metric determines the distribution of jobs among the measurement nodes. The closer the busy time ratios for nodes are to each other, the better is the distribution of jobs among them. For example, in an experiment with 3 nodes, a distribution of $33 : 33 : 34$ is better as

compared to 60 : 30 : 10. The busy time ratio for a node, NBTR is defined as:

$$NBTR = \frac{Time\ spent\ by\ the\ node\ in\ job\ execution}{Total\ duration\ of\ the\ experiment} \tag{3.5}$$

3.3.3.6 Link Utilization

To measure the impact that the system has on the underlying network, we calculate the transmitted and received megabytes of traffic through each link in the network. This is only calculated in phase 2 of experimentation as in this implementation we have control over the end-to-end traffic path between a measurement node and speed test server. We also measure the traffic distribution between the links connecting the switches to measurement nodes as well as the links connecting switches to each other.

3.4 Conclusion

In this chapter, we proposed the design of a community network measurement platform. The platform uses intelligent scheduling algorithms to minimize the conflicts for resources among the measurement nodes. We evaluate this system in two phases. In the first phase, we conducted evaluations over a logical map of the community network topology. These logical maps varied among each other in terms of the density of connections, which we fixed using a conflict probability. We used 5 types of measurements during these evalua-

tions in a lab setup. The second phase involved a testbed in which software defined network topologies were used. The controller of the SDN was chosen to be a remote machine and only TCP speed tests were used during the experiments. The performance metrics used during both phases of evaluations were aimed to measure scheduling algorithm performance.

4

Research Findings

Chapter 3 presented our research methodology for utilizing four measurement scheduling algorithms in the context of low resource networks. Round Robin (RR) algorithm schedules jobs in the order that they arrive, while Earliest Deadline First (EDF) algorithm schedules the jobs in the order of their deadlines. The remaining two algorithms, AOSD and DOSD schedule jobs by coloring a conflict subgraph that is constructed using resource constraint relationships among the jobs.

In this chapter, we present the performance analysis of the system based on the metrics described in section 3.3.3. We begin with describing the results for the first phase of implementation in section 4.1. Then, in section 4.2 we discuss the motivation behind altering the implementation for the second phase. Lastly, we describe the results for the second phase of implementation in section 4.3. A discussion on phase 2 implementation results is provided in a later chapter.

4.1 Phase 1: Adjacency list Implementation

As indicated in Section 3.3.1 , we executed 20 periodic measurements in three types of community network topologies logically represented as an adjacency list. These jobs were uniformly selected from the available job types and they were destined to target servers uniformly chosen from top-8 Alexa global websites. The chosen topologies consisted of 8 access points and 4 mobile phones. For a given pair of scheduling algorithm and the conflict probability chosen to build the topology, the experiment was allowed to run for 2 hours and the below metrics were calculated using the measurements' metadata.

4.1.1 Measurement Success Rate

To measure the success rate, we calculate the number of successful instances per job and average it over the 20 jobs for a single iteration for each scheduling algorithm and conflict probability. The theoretical number of job instances are calculated by dividing the time elapsed between start and end time by the measurement period. The ratio of number of successful instances to theoretical number of instances is then used to calculate the overall success rate. Figure 4.1 shows a plot of average success rate for each scheduling algorithm against the conflict probability. We notice that a near 100% success rate is achieved for all scheduling algorithms used. There were few missing instances in every case, which can be attributed to few periodic measurements missing their deadlines, i.e, missing the second

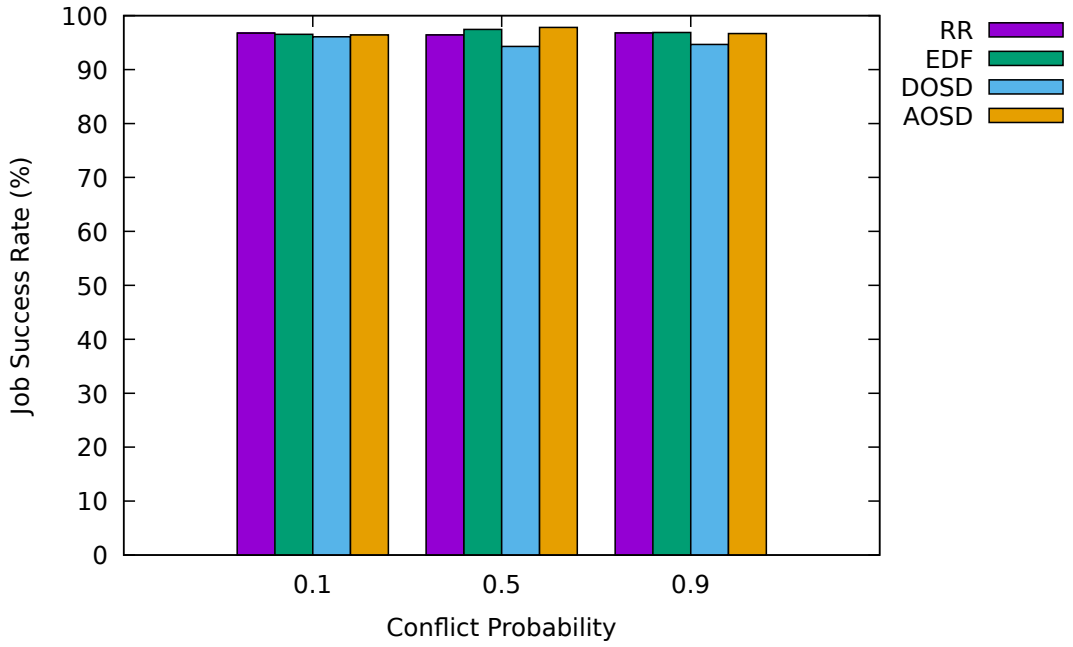


Figure 4.1: Measurement success rate for the first phase of experimentation

execution among 3 consecutively scheduled executions. We investigate this further in the next phase of evaluation. We also observe that DOSD algorithm achieves a marginally lower success rate which could be a result of the random nature of measurement specifications.

4.1.2 Platform Delay

The overall delay between a measurement's start time and completion time is referred to as platform delay in this dissertation. Similar to average success rate, the average platform delay was calculated by first calculating platform delay per measurement, which was then averaged over all combinations of scheduling algorithm and conflict probability. The plot

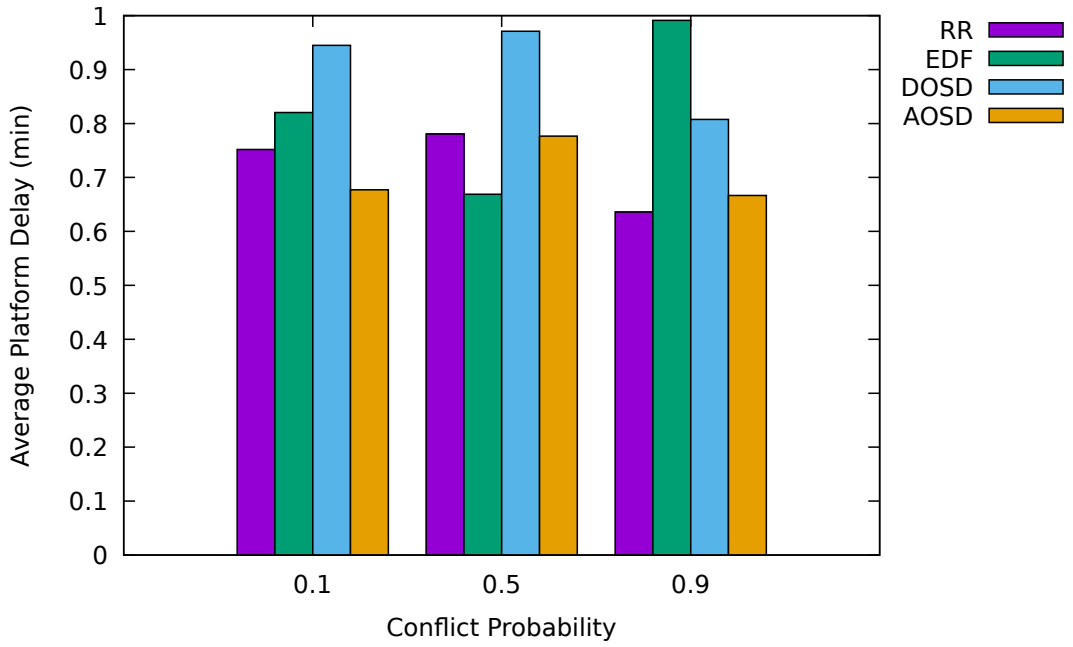


Figure 4.2: Average platform delay for the first phase of experimentation

of average platform delay (Figure 4.2) suggests that in all scenarios, a measurement completes execution within 1 minute of its start. If a larger number of conflicting measurements are scheduled simultaneously, this delay is likely to increase. The aim of our experiments was to determine the limits of the measurement system in scenarios where there is a greater measurement load. Therefore, in a real-world deployment where there's a greater likelihood of a lower measurements-to-devices ratio, lower values of platform delays are expected.

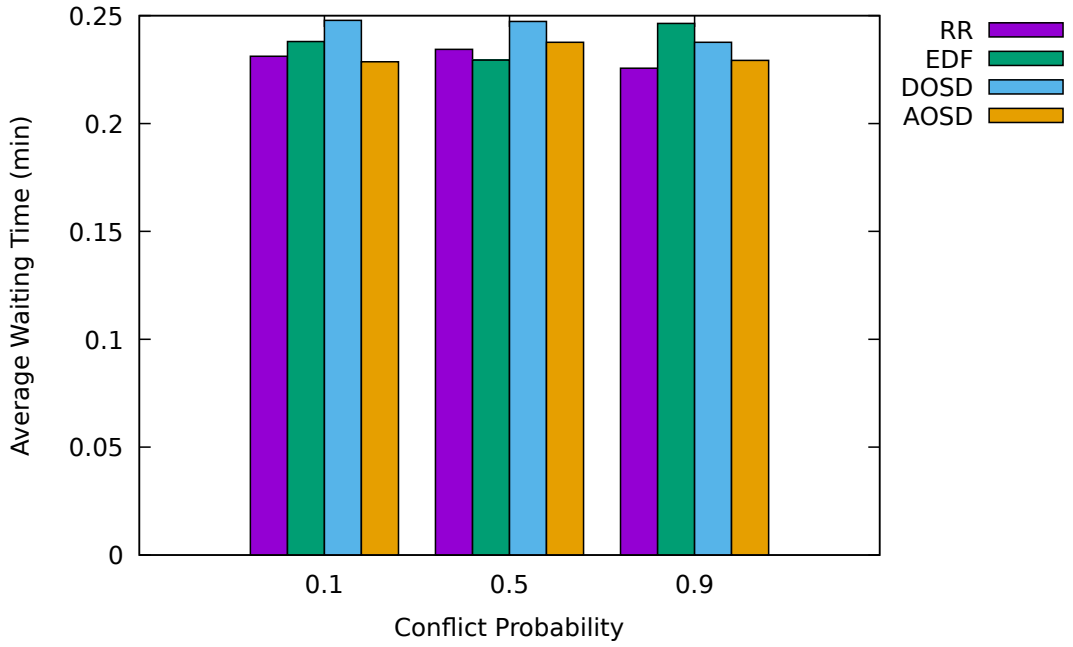


Figure 4.3: Average waiting time for the first phase of experimentation

4.1.3 Waiting Time

To determine the factors that contribute significantly towards platform delay, we calculate the average waiting times, i.e the average time that a measurement spends in the job queue before being dispatched. We observe in Figure 4.3 that the waiting time of the jobs was almost unaffected by network's topology but it contributed quite significantly to the overall platform delay. We observed that 33% of the platform delay in AOSD algorithm was due to waiting time on an average. This percentage was 27%, 29% and 32% for DOSD, EDF and RR respectively. The remaining portion of platform delay was due to factors like network delay and scheduling delay within the phones due to uneven distribution of jobs.

4.1.4 Node Busy Time Ratio

Node busy time ratio helps in determining the distribution of jobs among the measurement nodes. We observed a highly skewed distribution of jobs among the phones in the case of DOSD algorithm (Figure 4.4). RR and EDF had similar load distribution patterns while the most even load distribution was achieved in AOSD, where one of the phones executed about 50% of the total job load in all three topologies. This confirmed that the higher contribution of factors like network delay and scheduling overhead towards platform delay in RR, EDF and DOSD was indeed due to a skewed distribution of jobs among the phones.

4.2 Discussion on Phase I results

Our results show that AOSD algorithm provided better results compared to the rest of the chosen algorithms in terms of efficient distribution of jobs among measurement nodes and job success rate. A success rate as high as 97.3% is useful in areas where internet connection is unstable due to a high proportion of wireless links in comparison to wired links. Even though this algorithm shows great promise in terms of measurement distribution and success rate, it still distributes about half of the total load to a single smartphone. This could lead to scalability issues because the number of users in a community network keep fluctuating. If there are less smartphones and more measurements, then users might experience

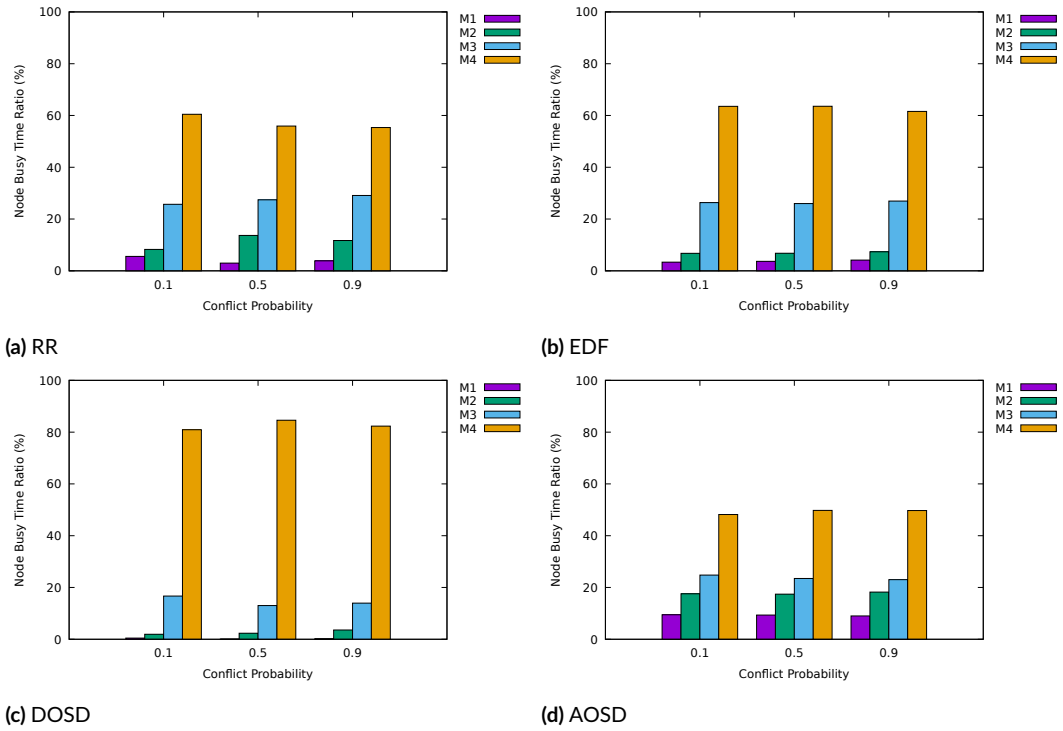


Figure 4.4: Node busy time ratios for 4 mobile phones used in the first phase of experimentation

issues like high data usage and battery drainage. Also, the results suggest that the scheduling performance does not depend on the network topology chosen.

The problem related to distribution of measurements to devices can be addressed by ensuring that measurements are assigned uniformly to the available phones. Therefore, the implementations of each of the scheduling algorithms can be modified to assign measurements to a random device that's connected to the measurement server. For the purpose of evaluation, if we limit the job types to speed tests only, we can put a limit on end-to-end throughput by fixing the throughput of each link. This will allow us to measure the error

in measurements against the link's fixed throughput value as well.

4.3 Phase 2: SDN Implementation

In the second phase of evaluation, the implementation of RR, EDF, DOSD and AOSD algorithms was altered to allow a more even distribution. Additionally, a randomized algorithm was also included within our system which dispatched any available job without a delay to a random device. We also extended the support for our system to docker containers which allowed us to run measurements inside a virtual machine. This virtual machine was equipped with the latest installation of a mininet fork called containernet*. Only complete graph topologies were used to test the system in our experimental setup with the throughput of each link fixed to 10 Mbps. We varied the number of access points from 3 to 15 in our experiments. This was done to ensure that the average number of access points matches the number of access points in our target community network. This meant that for any source-destination pair, the maximum end-to-end throughput was also fixed to 10 Mbps. Before evaluation, we generated a random set of 50 TCP speed test measurements, both uplink and downlink, and used these measurements for all our experiments. All of these were periodic jobs with periods chosen uniformly from the interval [10, 20] minutes. For every combination of scheduling algorithm and number of hosts, our experiments were allowed to run for 3 hours against a remote SDN controller. The following subsections highlight

*https://containernet.github.io/

the results of our experiments.

4.3.1 Measurement Error

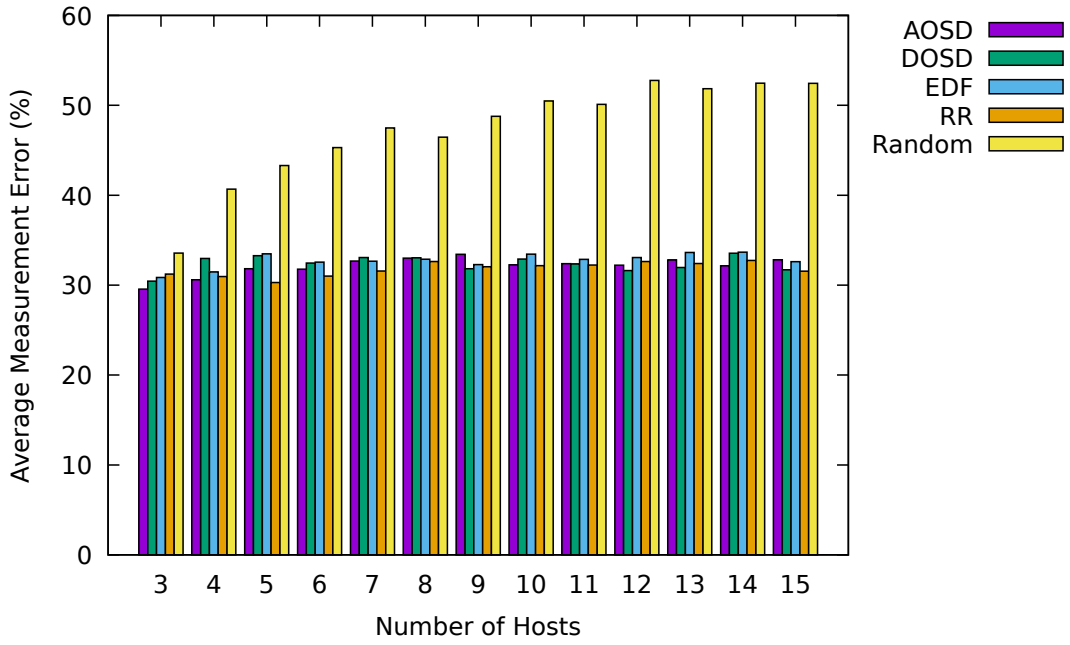


Figure 4.5: Measurement error for the second phase of experimentation

As described in Section 3.3.3, measurement error is the absolute percentage deviation from the actual link throughput. The average measurement error calculated against 10 Mbps link capacity for the randomized approach was found to show an increasing trend with the number of hosts (Figure 4.5). This suggests that the higher the number of measurement nodes, the higher is the probability of conflicting measurements travelling through the same links in the network. In contrast, the measurement error for the other four schedul-

ing algorithms did not increase beyond the 34% mark. It was also found that the error for the remaining scheduling algorithms stayed constant with increasing number of hosts. This shows that irrespective of the structure and orientation of the network, the scheduling algorithms provide our system the capability to conduct conflict-aware speed tests.

4.3.2 Measurement Success Rate

The success rate in phase 2 evaluation was found to be higher than in phase 1 and it did not change much after the number of hosts increased to 4 (Table 4.1). When there are only 3 measurement nodes, the success rate is lower because of a high per-device load, thus a pronounced observer effect. As the number of devices increase, the likelihood of the network becoming congested decreases which results in measurements meet their deadlines. We achieve a near 100% success rate in this implementation as well and it was also significantly higher than the phase 1 implementation.

Number of hosts	Random	RR	EDF	DOSD	AOSD
3	92.53%	94.08%	93.75%	94.00%	93.44%
4	99.77%	99.77%	99.77%	99.77%	99.50%
5	99.77%	99.55%	99.77%	99.60%	99.77%
6	99.45%	99.77%	99.47%	99.60%	99.77%
7	99.48%	99.77%	99.57%	99.40%	99.77%
8	99.40%	99.77%	99.77%	99.57%	99.77%
9	99.60%	99.77%	99.77%	99.77%	99.77%
10	99.65%	99.77%	99.21%	99.38%	99.77%
11	99.49%	99.77%	99.77%	99.77%	99.77%
12	99.77%	99.77%	99.44%	99.77%	99.77%
13	99.77%	99.59%	99.77%	99.55%	99.77%
14	99.77%	99.99%	99.37%	99.77%	99.77%
15	99.48%	99.77%	99.77%	99.57%	99.77%

Table 4.1: A table showing the measurement success rate with varying number of hosts in our experiments. The success rate for 3 hosts was found to be significantly lower as compared to a higher number of hosts. As the number of hosts increased, the success rate approached 100%.

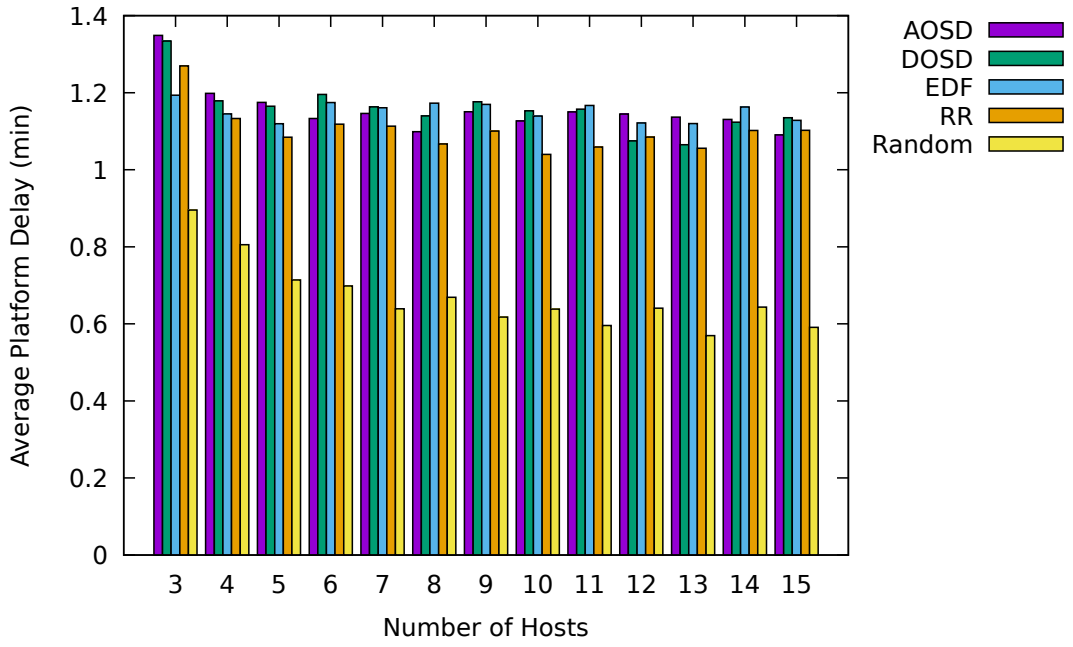


Figure 4.6: Platform Delay for the second phase of experimentation

4.3.3 Platform Delay

Measurements in randomized approach are dispatched as and when they are required to, we observe that this approach results in the least platform delay as compared to the other algorithms (Figure 4.6). We observe an increase in the platform delay as compared to phase 1. This can be attributed to the greater number of measurements scheduled in phase 2. A scheduling delay is caused because of conflicting jobs getting postponed beyond their expected start times. One key observation is that a trade-off can be observed between accuracy of measurements and the platform delay. Since the error in measurements did not increase with varying number of hosts, our recommendation would thus be to prioritize accuracy

over platform delay.

4.3.4 Waiting Time

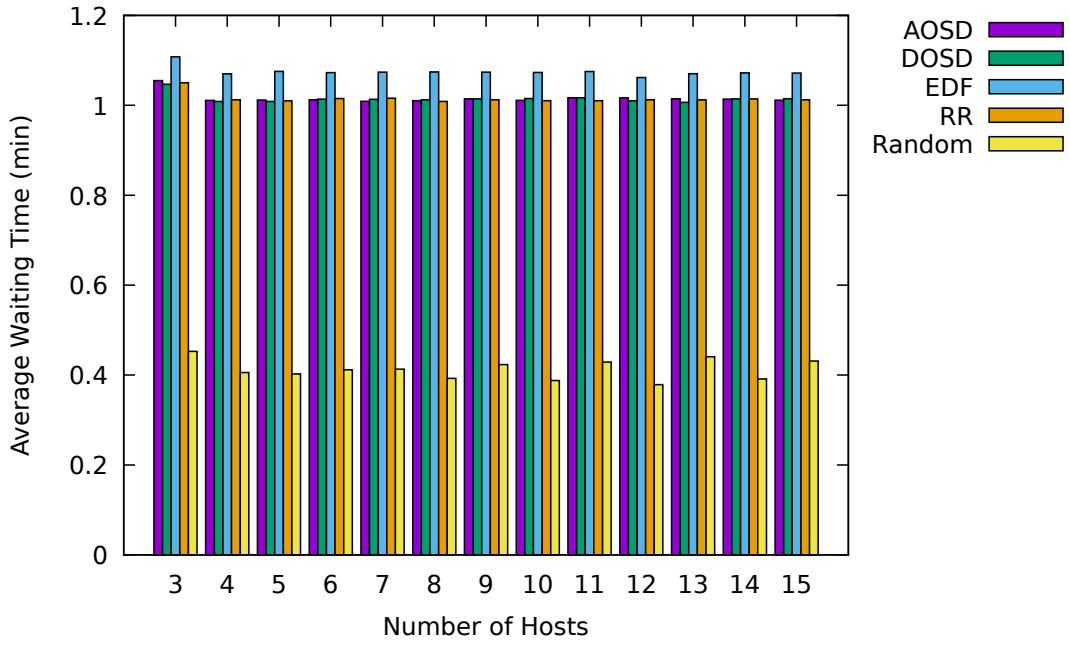


Figure 4.7: Waiting time for the second phase of experimentation

Figure 4.7 shows that the waiting time remains constant with varying number of hosts for all types of scheduling algorithms. For the randomized algorithm the average waiting time is 59.98% lower as expected to the other algorithms on an average. The waiting time for EDF is slightly higher because the jobs are arranged in the order of their deadlines before being supplied to the scheduling algorithms. It can also be inferred that in phase 2 evaluation, the majority of the platform delay is due to waiting time, i.e the delay caused by the

scheduling algorithms. If the execution time of the speed test measurement is known beforehand for a given load pattern, this delay can be minimised. We thus hypothesize that this delay can be minimised by not relying on the assumption of a fixed execution time.

4.3.5 Node Busy Time Ratio

Figure 4.8 provides insights into the measurement execution times for each algorithm as the number of hosts vary in the virtual network. It is worth noting that the total time calculated in these plots exceeds the total experiment duration of 3 hours. This is because a large number of measurement instances run in parallel on different measurement nodes at overlapping time slots. We observe that the overall load of measurements is spread evenly among different devices in all cases. Even though the randomized approach assigned measurements randomly to devices, it is still able to distribute the load almost evenly. The distribution of measurements in the other four algorithms is even more fair as compared to the randomized approach. The higher the load on a single device, greater the likelihood of smartphone battery drainage in a community network deployment. An even busy time ratio thus translates directly to the energy efficiency of our system and the assignment of measurements in turns is a better design choice over phase 1.

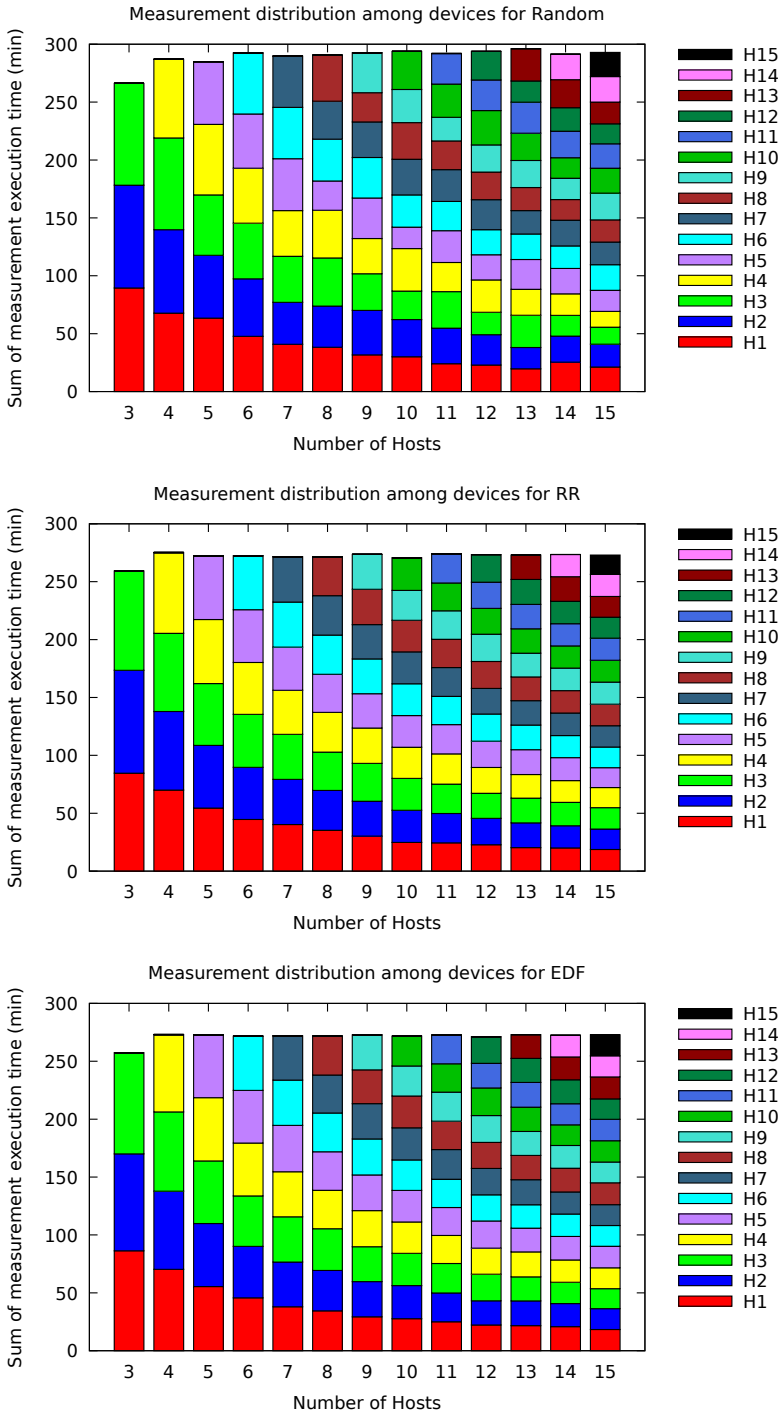


Figure 4.8: Distribution of measurement execution times among nodes for each scheduling algorithm

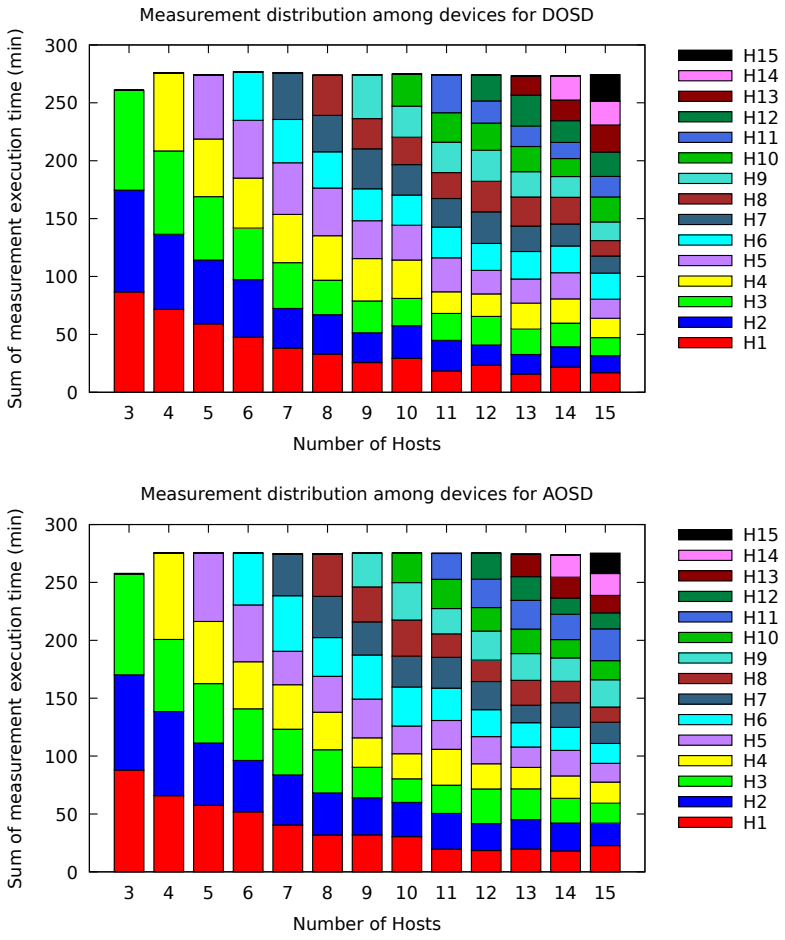


Figure 4.8: Distribution of measurement execution times among nodes for each scheduling algorithm (cont.)

4.3.6 Link Utilization

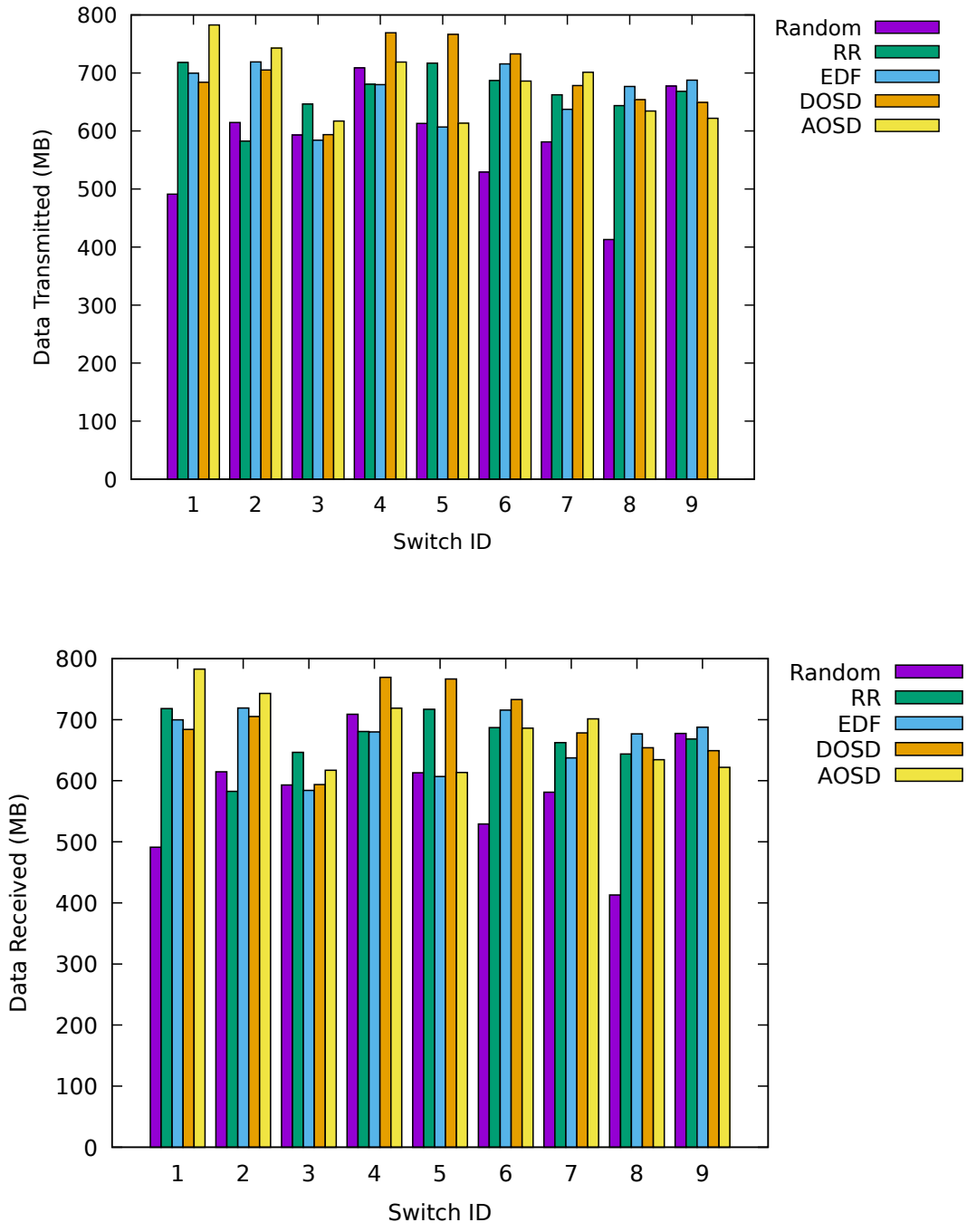


Figure 4.9: Traffic transferred through the link between each switch and the immediately connected host in a topology with 9 hosts and 1 speed test server. x-axis denotes the ID of the link. The switch with ID 10 corresponds to the link between S_{10} and the speed test server. Similar to Figure 3.7, each host H_i was immediately connected to a switch S_i.

Figure 4.9 represents the transmitted and received traffic through each switch. We observe a balanced distribution of traffic among the measurement nodes as there's no visible spike in the graphs for any of the algorithms. The link between the speed test server and switch number 10 is heavlily utillized because of its direct connection to the speed test server. The values of link utilization between received and transmitted plots differ due to the distribution of uplink and downlink measurements launched.

We also measured the amount of traffic flown through switch-to-switch links. As indicated in figure 4.10, we again observe a fairly even distribution of traffic between the measurement nodes. The links connected to the tenth switch were also on the shortest path links to the speed test server from each host. Both these plots corresponded to exactly the same values of transmitted and recieved megabytes, which means that no packets were dropped during the process. Another observation was that 99.9% of the total traffic flows went through the links connecting each switch to S_{10}. This is because there was a single speed test server handling the data transfers to and from the measurement nodes.

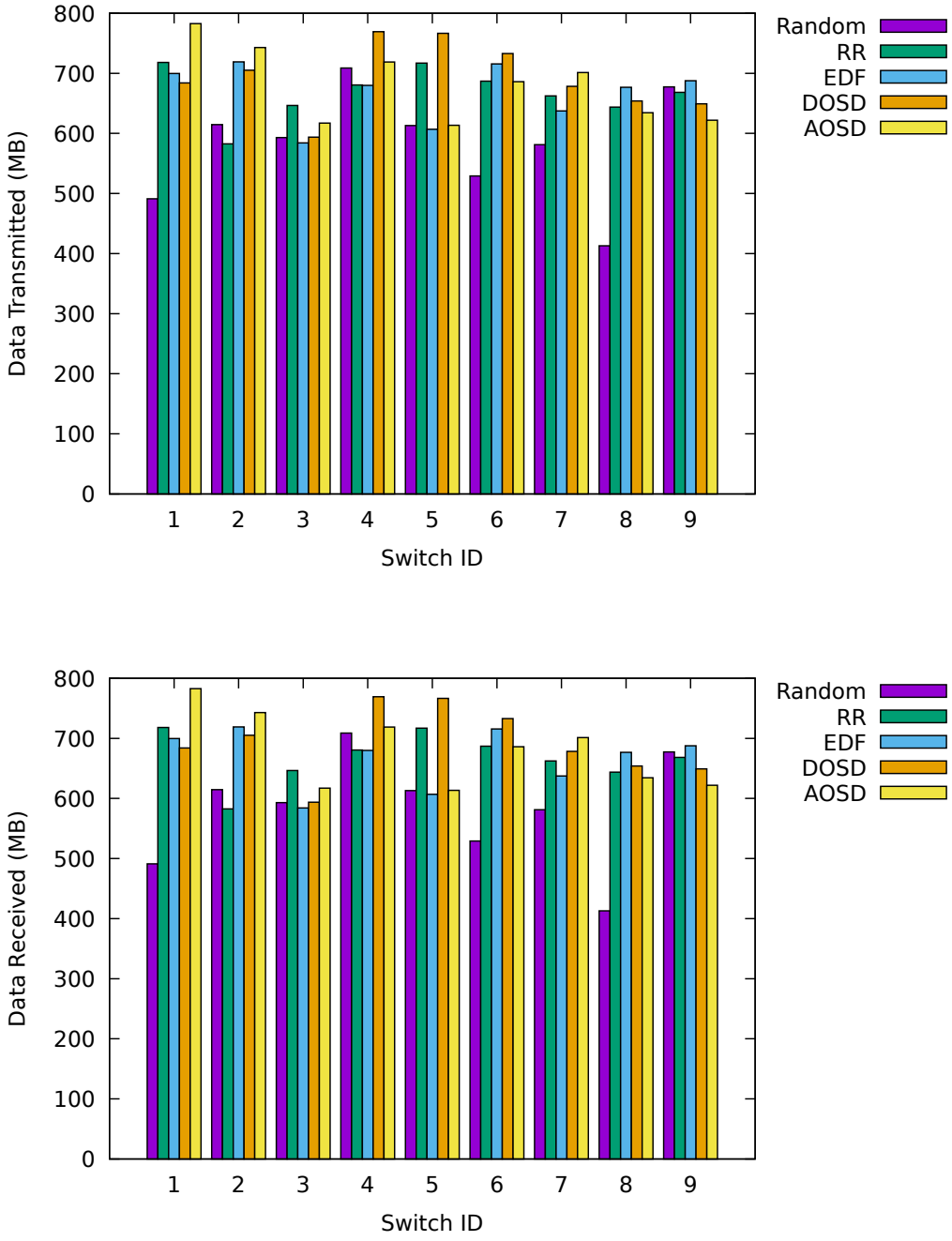


Figure 4.10: A figure denoting the plots of traffic transmitted and received through the direct links from each switch to S_{10}, i.e the switch connected directly to the speed test server.

5

Conclusions

In this chapter, we provide a discussion on the key findings of this dissertation and provide recommendations for future work.

5.1 Key findings

In this study, we implemented four existing scheduling algorithms as part of a network measurement platform. We envision the deployment of this measurement platform in the iNethi community network in Cape Town where the measurements are expected to run in users' smartphones. We conduct the evaluation of this system in two phases. In the first phase, we use four smartphones to run five distinct types of measurements in a lab setup. The outcomes of these experiments suggested to alter our implementation for a more even distribution of measurements. In the second phase, we conducted our experiments in a software-defined testbed while focusing on speed tests only. Our system performed significantly better in terms of accuracy than a "no-scheduling" scenario where measurements are dispatched to random devices as and when they are required to. We also found that

the error in measurements due to sharing of common links stays constant at about 34% as the number of hosts in the network increases. The performance of four scheduling algorithms chosen did not differ much among themselves as they showed similar characteristics in terms of accuracy and platform delay. On an average, the platform provides about 1.2 minutes of delay between the start of a measurement and its result to be received.

Our results show that the algorithms are designed to address the conflicts for network resources between measurements by providing better accuracy than a randomized scheduling approach. However, these results do not declare a single algorithm as a clear winner. A functional implementation of any of these algorithms can thus be used to schedule measurements in a community network. The period of the measurements used in our experiments was kept lower so as to create scope for maximum overlap among the jobs. In a real deployment scenario, there will be multiple measurements running on an even large number of hosts. These measurements are likely to be scheduled by researchers and network managers on an hourly basis to investigate the state of the community network. We thus expect that the system should provide error-free measurements at a high success rate.

We posit that this system has a high applicability in community networks that are looking to expand across hundreds of nodes. The cost to set up our measurement platform is not very high as it has the capability to integrate with any hardware that supports a docker installation. With software defined networking emerging rapidly as a network management

paradigm, we believe that we are not far away from a point where community networks will also adopt this trend. Given that our platform supports not just smartphones but any docker-compatible hardware, its development is thus a significant step towards increasing research efforts on community networks at a reduced cost.

5.2 Future work

The scheduling algorithms used in our study rely on the assumption that the execution time of measurements is fixed. We believe that a thorough analysis involving different network conditions and measurement distributions is required to minimize the observer effect even further and increase the measurements' accuracy. Also, the network topologies used in our study were built using point-to-point links, which is different from wireless mesh architecture. These results present a good starting point in terms of the performance metrics used to evaluate the system. Future work will thus involve testing the system in an actual deployed network with more challenging conditions like wireless hand-offs, mobility, and cross-traffic.

Another interesting addition to this study could be to introduce anomaly detection capabilities in the system. Since the current implementation involves conduction of periodic measurements, a future implementation could involve learning from network data and launching of system generated measurements for a fine-grained network analysis. This

would also involve the adjustment of schedule for existing measurements in the system in accordance with a priority order as proposed in Qin *et. al* [18].

Further, a deeper analysis of the CPU and network footprints of our system would help us in understanding if this system can be deployed alongside already existing community network services without disrupting the end-user quality of experience. White *et. al* [74] have proposed a containerized architecture for seamless deployment of community network services in iNethi. A future work could thus be the integration our platform with this architecture and getting the measurement system deployed in iNethi as per guidelines in our current study.

References

[1] John M Carroll and Mary Beth Rosson. Network communities, community networks. In *CHI 98 Conference Summary on Human Factors in Computing Systems*, pages 121–122, 1998.

[2] Luca Belli, Sarbani Banerjee Belur, Peter Bloom, Anriette Esterhuysen, Nathalia Foditsch, Maureen Hernandez, Erik Huerta, Mike Jensen, Meghna Khaturia, Michael J Oghia, et al. Community networks: the internet by the people, for the people, 2017.

[3] Bart Braem, Chris Blondia, Christoph Barz, Henning Rogge, Felix Freitag, Leandro Navarro, Joseph Bonicioli, Stavros Papathanasiou, Pau Escrich, Roger Baig Viñas, et al. A case for research with and on community networks, 2013.

[4] Panagiota Micholia, Merkouris Karaliopoulos, Iordanis Koutsopoulos, Leandro Navarro, Roger Baig Vias, Dimitris Boucas, Maria Michalis, and Panayotis Antoniadis. Community networks and sustainability: a survey of perceptions, practices, and proposed solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3581–3606, 2018.

[5] Davide Vega, Roger Baig, Llorenç Cerdà-Alabern, Esunly Medina, Roc Meseguer, and Leandro Navarro. A technological overview of the guifi. net community net-

work. *Computer Networks*, 93:260–278, 2015.

[6] Amr Alasaad. *Content sharing and distribution in wireless community networks*. PhD thesis, University of British Columbia, 2013.

[7] Thomas Plagemann, Roberto Canonico, Jordi Domingo-Pascual, Carmen Guerrero, and Andreas Mauthe. Infrastructures for community networks. In *Content Delivery Networks*, pages 367–388. Springer, 2008.

[8] Bob Lantz, Brandon Heller, and Nick 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*, pages 1–6, 2010.

[9] Bruno Astuto A. Nunes, Marc Mendonca, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617–1634, 2014.

[10] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[11] Xuan Nam Nguyen. *Software defined networking in wireless mesh networks*. PhD thesis, MASTER II IFI (Informatique: Fondements et Ingénierie)-Université Nice ..., 2012.

[12] Andrew Coyle and Hung Nguyen. A frequency control algorithm for a mobile adhoc network. 2010.

[13] Peter Dely, Andreas Kassler, and Nico Bayer. Openflow for wireless mesh networks. In *2011 proceedings of 20th international conference on computer communications and networks (ICCCN)*, pages 1–6. IEEE, 2011.

[14] George Parissis, Stamatis Arkoulis, and Theodore Apostolopoulos. An open architecture for monitoring and measuring qos indicators in wireless community networks. In *2008 Next Generation Internet Networks*, pages 284–291. IEEE, 2008.

[15] Adriano Faggiani, Enrico Gregori, Luciano Lenzini, Valerio Luconi, and Alessio Vecchio. Smartphone-based crowdsourcing for network monitoring: opportunities, challenges, and a case study. *IEEE Communications Magazine*, 52(1):106–113, 2014.

[16] Ashkan Nikravesh, Hongyi Yao, Shichang Xu, David Choffnes, and Z Morley Mao. Mobilyzer: An open platform for controllable mobile network measurements. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 389–404, 2015.

[17] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):15–20, 2011.

[18] Zhen Qin, Roberto Rojas-Cessa, and Nirwan Ansari. Task-execution scheduling schemes for network measurement and monitoring. *Computer communications*, 33(2):124–135, 2010.

[19] Pantelis A Frangoudis, George C Polyzos, and Vasileios P Kemerlis. Wireless community networks: an alternative approach for nomadic broadband network access. *IEEE Communications Magazine*, 49(5):206–213, 2011.

[20] Bart Braem, Johan Bergs, Chris Blondia, Leandro Navarro, and Sabine Wittevrongel. Analysis of end-user qoe in community networks. In *Proceedings of the 2015 Annual Symposium on Computing for Development*, pages 159–166, 2015.

[21] Maria Rosa Lorini, Melissa Densmore, David Johnson, Senka Hadzic, Hafeni Mthoko, Ganief Manuel, Marius Waries, and André van Zyl. Localize-it: Co-designing a community-owned platform. In *International Development Informatics Association Conference*, pages 243–257. Springer, 2018.

[22] *Inethi*, Accessed October 15, 2020. https://www.inethi.org.za/.

[23] Carlos Rey-Moreno, Zukile Roro, William D Tucker, Masbulele Jay Siya, Nicola J Bidwell, and Javier Simo-Reigadas. Experiences, challenges and lessons from rolling out a rural wifi mesh network. In *Proceedings of the 3rd ACM Symposium on Computing for Development*, pages 1–10, 2013.

[24] Roger Baig, Ramon Roca, Leandro Navarro, and Felix Freitag. guifi. net: A network infrastructure commons. In *Proceedings of the Seventh International Conference on Information and Communication Technologies and Development*, pages 1–4, 2015.

[25] Albert Domingo, Marlies Van der Wee, Sofie Verbrugge, and Miquel Oliver. Deployment strategies for ftth networks and their impact on the business case: A comparison of case studies. 2014.

[26] D Ztoupis, K Zarifis, Ioannis Stavrakakis, and Christos Xenakis. Towards a security framework for an established autonomous network. In *2008 3rd International Symposium on Wireless Pervasive Computing*, pages 749–754. IEEE, 2008.

[27] B Bowonder, Vinay Gupta, and Amit Singh. Developing a rural market e-hub: The case study of e-choupal experience of itc. *Indian Planning Commission Report*, 2002.

[28] Eduardo E P. Pujol, Will Scott, Eric Wustrow, and J Alex Halderman. Initial measurements of the cuban street network. In *Proceedings of the 2017 Internet Measurement Conference*, pages 318–324, 2017.

[29] Navaneeth Rameshan, Leandro Navarro, and Ioanna Tsalouchidou. A monitoring system for community-lab. In *Proceedings of the 11th ACM international symposium on Mobility management and wireless access*, pages 33–36, 2013.

[30] Javi Jiménez, Roger Baig, Pau Escrich, Amin M Khan, Felix Freitag, Leandro Navarro, Ermanno Pietrosemoli, Marco Zennaro, Amir H Payberah, and Vladimir Vlassov. Supporting cloud deployment in the guifi. net community network. In *Global Information Infrastructure Symposium-GIIS 2013*, pages 1–3. IEEE, 2013.

[31] Roger Baig, Felix Freitag, Agusti Moll, Leandro Navarro, Roger Pueyo, and Vladimir Vlassov. Cloud-based community services in community networks. In *2016 International Conference on Computing, Networking and Communications (ICNC)*, pages 1–5. IEEE, 2016.

[32] Nuno Apolónia, Felix Freitag, and Leandro Navarro. Leveraging deployment models on low-resource devices for cloud services in community networks. *Simulation Modelling Practice and Theory*, 77:390–406, 2017.

[33] Sonesh Surana, Rabin K Patra, Sergiu Nedevschi, Manuel Ramos, Lakshminarayanan Subramanian, Yahel Ben-David, and Eric A Brewer. Beyond pilots: Keeping rural wireless networks alive. In *NSDI*, volume 8, pages 119–132, 2008.

[34] Ravi Anupindi and S Sivakumar. Itc's e-choupal: A platform strategy for rural transformation. *Business solutions for the global poor: Creating social and economic value*, 173:182, 2007.

[35] Vaibhav Bajpai and Jürgen Schönwälder. A survey on internet performance measurement platforms and related standardization efforts. *IEEE Communications Surveys & Tutorials*, 17(3):1313–1341, 2015.

[36] Nevil Brownlee and Chris Loosley. Fundamentals of internet measurement: A tutorial. 2001.

[37] Constantine Dovrolis, Krishna Gummadi, Aleksandar Kuzmanovic, and Sascha D Meinrath. Measurement lab: Overview and an invitation to the research community, 2010.

[38] Utkarsh Goel, Mike Wittie, Kimberly Claffy, Andrew Le, et al. Survey of end-to-end mobile network measurement testbeds. Technical report, arXiv cs. NI/1411.5003, 2014.

[39] Alfréd Rényi. Remarks on the poisson process. In *Symposium on Probability Methods in Analysis*, pages 280–286. Springer, 1967.

[40] Junxian Huang, Cheng Chen, Yutong Pei, Zhaoguang Wang, Zhiyun Qian, Feng Qian, Birjodh Tiwana, Qiang Xu, Z Mao, Ming Zhang, et al. Mobiperf: Mobile net-

work measurement system. *Technical Report. University of Michigan and Microsoft Research*, 2011.

[41] Utkarsh Goel et al. *Internet measurements and application layer optimizations for faster web communications*. PhD thesis, Montana State University-Bozeman, College of Engineering, 2017.

[42] Sachit Muckaden. *MySpeedTest: active and passive measurements of cellular data networks*. PhD thesis, Georgia Institute of Technology, 2013.

[43] Anika Schwind, Michael Seufert, Özgü Alay, Pedro Casas, Phuoc Tran-Gia, and Florian Wamser. Concept and implementation of video qoe measurements in a mobile broadband testbed. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–6. IEEE, 2017.

[44] Özgü Alay, Andra Lutu, Rafael García, Miguel Peón-Quirós, Vincenzo Mancuso, Thomas Hirsch, Tobias Dely, Jonas Werme, Kristian Evensen, Audun Hansen, et al. Measuring and assessing mobile broadband networks with monroe. In *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–3. IEEE, 2016.

[45] Ali Safari Khatouni, Marco Mellia, Marco Ajmone Marsan, Stefan Alfredsson, Jonas Karlsson, Anna Brunstrom, Ozgu Alay, Andra Lutu, Cise Midoglu, and Vincenzo

Mancuso. Speedtest-like measurements in 3g/4g networks: The monroe experience. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, pages 169–177. IEEE, 2017.

[46] Veljko Pejović, Ivan Majhen, Miha Janež, and Blaž Zupan. Ricercando: Data mining toolkit for mobile broadband measurements. *Computer Networks*, 177:107294, 2020.

[47] Tony McGregor, H-W Braun, and Jeff Brown. The nlanr network analysis infrastructure. *IEEE Communications Magazine*, 38(5):122–128, 2000.

[48] *Surveyor: An Infrastructure for Internet Performance Measurements*, Accessed March 10, 2022. https://web.archive.org/web/20080613021845/http://www.isoc.org/isoc/conferences/inet/99/proceedings/4h/4h_2.htm.

[49] Prasad Calyam, Chang-Gun Lee, Phani Kumar Arava, and Dima Krymskiy. Enhanced edf scheduling algorithms for orchestrating network-wide active measurements. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 10–pp. IEEE, 2005.

[50] SM Hoque. *Scalable Network Tomography System*. 2009.

[51] Rezaul Hoque, Andreas Johnsson, Christofer Flinta, Svante Ekelin, and Mats Björkman. A self-organizing scalable network tomography control protocol for active

measurement methods. In *Proceedings of the 2010 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'10)*, pages 65–72. IEEE, 2010.

[52] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 1.00 edition, 2018.

[53] Mathew Clegg. Scheduling network performancemonitoring in the cloud, 2017.

[54] Erhan Okuyan and Bans Kayayurt. Earliest deadline first scheduling algorithm and its use in anka uav. In *2012 IEEE/AIAA 31st Digital Avionics Systems Conference (DASC)*, pages 8B1–1. IEEE, 2012.

[55] John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C Buttazzo. *Deadline scheduling for real-time systems: EDF and related algorithms*, volume 460. Springer Science & Business Media, 2012.

[56] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 43–48, 2012.

[57] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to sdn: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[58] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[59] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE Communications Surveys Tutorials*, 16(1):493–512, 2014.

[60] Yunsick Sung, Pradip Kumar Sharma, Erik Miranda Lopez, and Jong Hyuk Park. Fs-opensecurity: A taxonomic modeling of security threats in sdn for future sustainable computing. *Sustainability*, 8(9), 2016.

[61] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[62] Ola Salman, Imad H Elhajj, Ayman Kayssi, and Ali Chehab. Sdn controllers: A comparative study. In *2016 18th Mediterranean Electrotechnical Conference (MELECON)*, pages 1–6. IEEE, 2016.

[63] Josiah Chavula. Improving pan-african research and education networks through traffic engineering: A lisp/sdn approach. 2017.

[64] Enock Samuel Mbewe, Taveesh Sharma, and Josiah Chavula. Powerqope: A personal quality of internet protection and experience configurator. 2022.

[65] Dominique Oosthuizen, Taveesh Sharma, Josiah Chavula, and Melissa Densmore. Investigating the usability and quality of experience of mobile video-conferencing apps among bandwidth-constrained users in south africa. In *Proceedings of 43rd Conference of the South African Institute for Computer Scientists and Information Technologists (SAICSIT)*, volume 85, pages 243–256, 2022.

[66] Amreesh Phokeer, Senka Hadzic, Eric Nitschke, Andre Van Zyl, David Johnson, Melissa Densmore, and Josiah Chavula. inethi community network: A first look at local and internet traffic usage. In *Proceedings of the 3rd ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 342–344, 2020.

[67] Sanchit Aggarwal. Modern web-development using reactjs. *International Journal of Recent Research Aspects*, 5(1):2349–7688, 2018.

[68] Craig Walls. *Spring Boot in action*. Manning Publications, 2016.

[69] Taveesh Sharma and Josiah Chavula. Investigating measurement scheduling strategies in low resource networks (poster). In *ACM SIGCAS Conference on Computing and Sustainable Societies*, pages 453–456, 2021.

[70] Taveesh Sharma and Josiah Chavula. Topology-aware measurement scheduling strategies in low resource networks. 2021.

[71] *Alexa - Top sites*, Accessed May 26, 2021. https://www.alexa.com/topsites/.

[72] *Multipath Routing with load balancing using Ryu OpenFlow Controller*, Accessed February 15, 2022. https://wildanmsyah.wordpress.com/2018/01/13/multipath-routing-with-load-balancing-using-ryu-openflow-controller/.

[73] *Representing community network topologies on GENI*, Accessed February 15, 2022. https://witestlab.poly.edu/blog/representing-community-network-topologies-on-geni.

[74] Keegan White, David Johnson, Melissa Densmore, and Hafeni Mthoko. Bootstrapping the development of services for wireless community networks. 2021.

A

Deployment of the measurement system

In this chapter, we provide detailed instructions on how to set up the measurement system on any Linux machine. The instructions for other platforms may differ on file paths used and installation procedure for different dependencies.

A.1 Prerequisites

The steps shown in the subsequent sections assume that the host machine has working installations of **docker** and **docker-compose**. The installation steps can be found here.

There's a high likelihood that the current user account may not be able to invoke docker and docker-compose commands without prefixing them with a "sudo". To address this, please review the post-installation steps here.

A.2 Setting up the backend

This component serves as the centralized server and orchestrates the conduction of measurements.

1. Clone the below GitHub repository and change your working directory to **qosmon-request-handler**.

```
$ git clone https://github.com/staveesh/qosmon-request-handler
$ cd qosmon-request-handler
```

2. Create a hidden file **.env**.

```
$ touch .env
```

3. Add the below contents to the **.env** file. These values represent the environment variables and configurations for the system. Defaults for the version used for experiments can be obtained upon request to the authors.

```
INFLUXDB_USERNAME=#value
INFLUXDB_PASSWORD=#value
INFLUXDB_NAME=#value
INFLUXDB_PORT=#value
MONGODB_USERNAME=#value
MONGODB_PASSWORD=#value
MONGODB_NAME=#value
MONGODB_PORT=#value
MONGODB_HOST=#value
HTTP_SERVER_PORT=#value
TCP_SERVER_PORT=#value
FILE_SERVER_HOSTNAME=#value
MILLISECONDS_TILL_RETRY_CONNECT=#value
MILLISECONDS_TILL_ANALYZE_PCAPFILES=#value
MILLISECONDS_INIT_DELAY=#value
FILE_MONITOR_DELAY=#value
NUM_RETRY_CONNECT=#value
SCHEDULING_ALGO_NAME=#value (One of [random, rr, edf, aosd, dosd])
```

4. After the above file is saved with suitable environment variables, the below command can be executed to launch the measurement server. This may take a long time during first time setup.

```
$ docker-compose up --build
```

The measurement server should now be running and ready to receive new measurements. The next step is to setup a measurement node.

A.3 Setting up the speed test server

This component is responsible for sending and receiving speed test data from the measurement nodes. Run the below command within the host on which the speed test server should be launched. For testing purposes, this host can be kept the same as the measurement server.

```
$ docker run -p 6001:6001 -p 6002:6002 -p 6003:6003 -p 31341:31341 staveesh/
    speedserver:latest
```

A.4 Setting up a measurement node

Two platforms are supported as measurement nodes for the platform - Android and Linux. The application has not been tested on Windows/macOS platforms yet but we anticipate that the installation instructions should not difffer much.

A.4.1 Instructions for Android apps

The following instructions require a working installation of Android Studio. Please check here for installation instructions.

1. Clone the below repository into the file system.

```
$ git clone https://github.com/staveesh/mobiperf
```

2. Open the **mobiperf** directory as a project in Android studio. Then, wait for the gradle build to finish.

3. Browse to the file **mobi/src/main/java/com/mobiperf/Config.java** and change the below parameters.

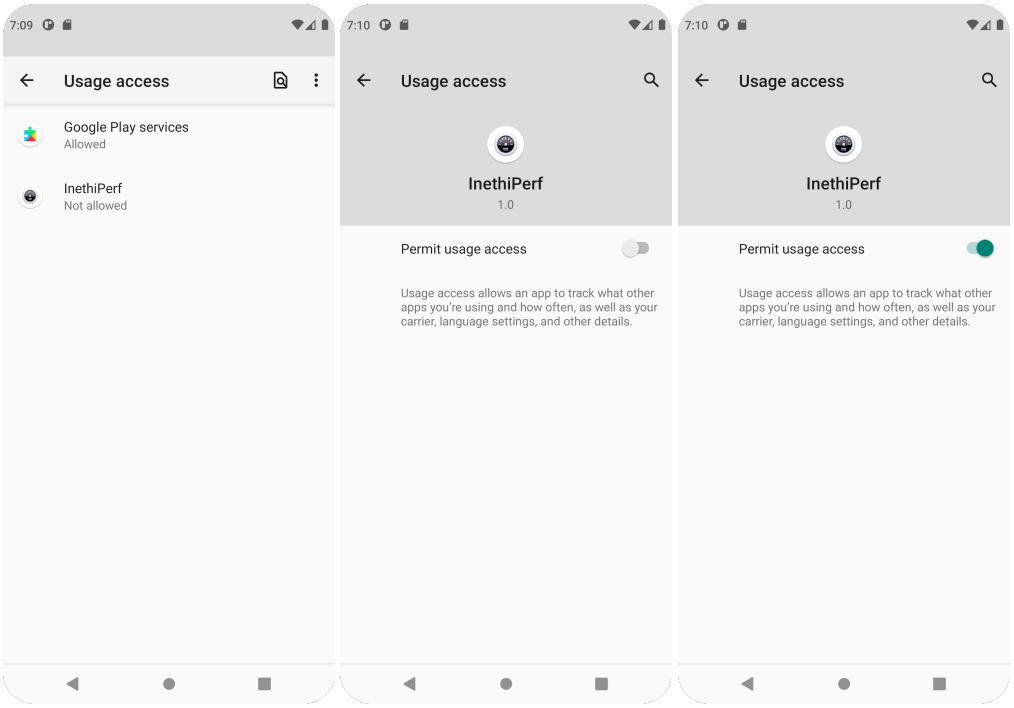
   (a) SPEED_TEST_SERVER_ADDRESS: Address of the speed test server.

   (b) SERVER_HOST_ADDRESS: Change this parameter to the hostname/IP address of the measurement server.

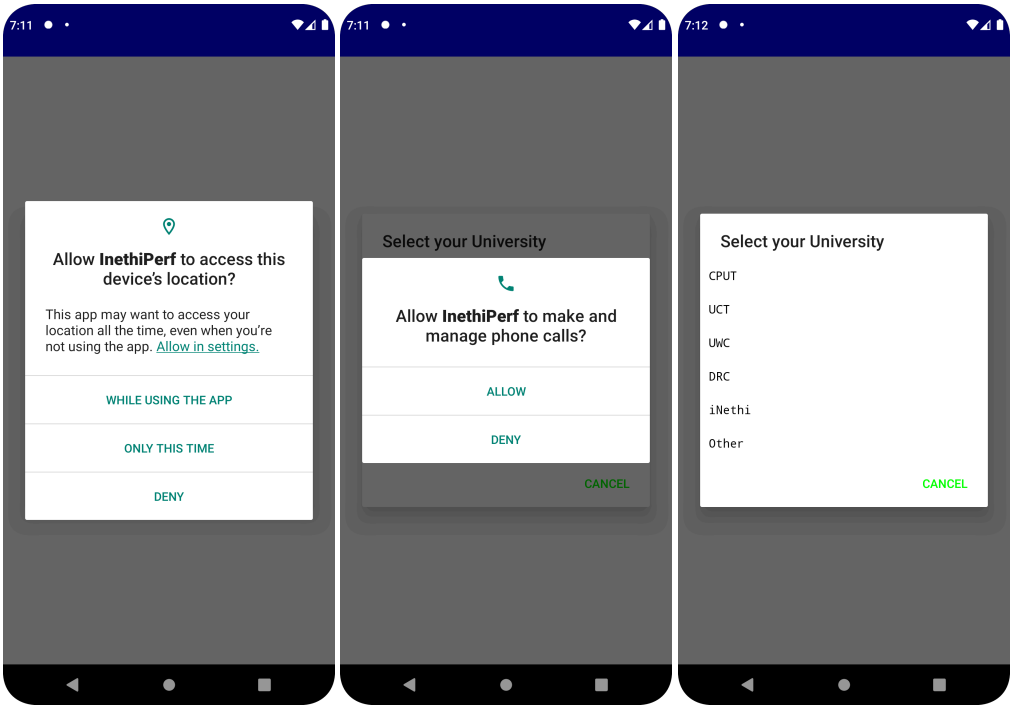
   (c) SERVER_PORT: Change this to the port on which the measurement server was configured to run.

4. Running the app:
   If an Android emulator has not been set up, please follow the instructions provided here.
   To run the app on a physical device, please follow the instructions provided here.

5. First time setup:
   Figure A.1 shows the steps to be followed for setting up the app during the first install.



(a) Select "InethiPerf" **(b)** Turn on "Permit usage access" **(c)** Go back to the InethiPerf app



(d) Select "While using the app" **(e)** Select "Allow" **(f)** Select "iNethi"

Figure A.1: First time install of the app

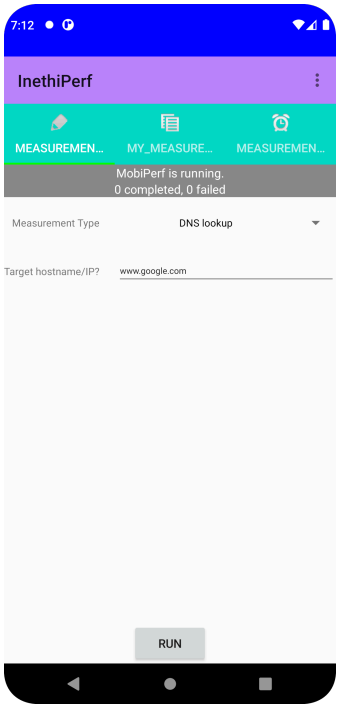


Figure A.2: Main screen of the application

The screen shown in Figure A.2 can be seen after the app is completely set up. Users can launch custom measurements using this interface and also view their results. Scheduled measurements are also supported.

A.4.2 Instructions for Linux

To run measurements from any linux machine, follow the below steps:

1. Clone the below github repository and change current directory to **mobiperf-simulator**:

```
$ git clone https://github.com/staveesh/mobiperf-simulator
$ cd mobiperf-simulator
```

2. Create a **.env** file in this directory.

```
$ touch .env
```

3. Add the below contents to the **.env** file. Use appropriate values for the measurement_server_host and measurement_server_port.

```
SERVER_PORT=#value
MEASUREMENT_SERVER_ENDPOINT=ws://{measurement_server_host}:{
    measurement_server_port}/mobiperf
MOBIPERF_DEVICE_ID=#value (A unique identifier)
MOBIPERF_SCHEDULE_FILE_PATH=/var/lib/schedule
```

4. Launch the below command to start the measurement node:

```
$ docker-compose up --build
```

Like any Android phone, this node should be able to receive measurements from the measurement server.

A.5 Setting up the frontend

The frontend is written in ReactJS. It requires a working installation of NodeJS and the Node Package Manager (npm). Click here to view the installation instructions. Then, the following instructions can be followed to launch the frontend:

1. Clone the below github repository and change current directory to **qosmon-ui**:

```
$ git clone https://github.com/staveesh/qosmon-ui
$ cd qosmon-ui
```

2. Create a **.env** file in this directory.

```
$ touch .env
```

3. Add the below contents to the **.env** file. Use appropriate values for the measurement server's address.

```
REACT_APP_API_URL= # Intended URL for the frontend
REACT_APP_PUBLIC_URL= # URL for the measurement server
```

4. Launch the below command to install frontend dependencies.

```
$ npm install
```

5. Launch the below command to start the frontend webserver.

```
$ npm start
```

The webserver will be up and running and users should be able to visit the frontend in their browser. Users can also Sign Up and Log In to the system using the web forms provided on the homepage.

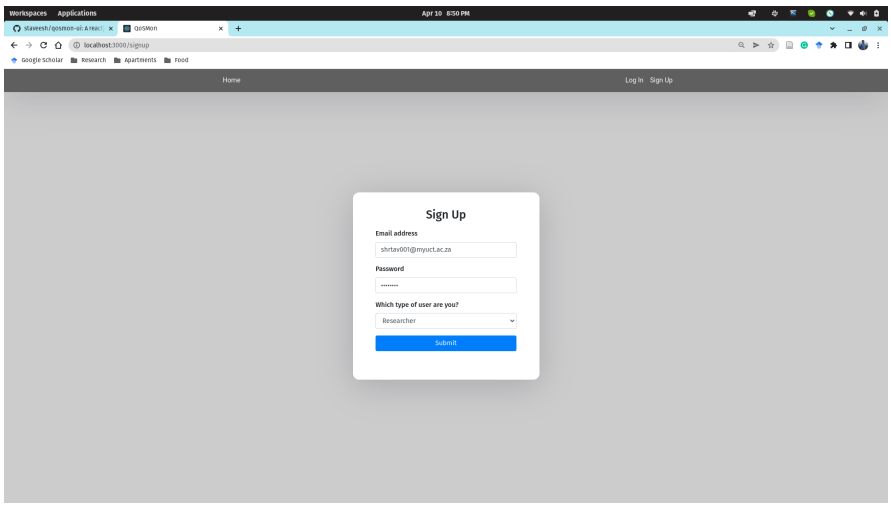


Figure A.3: QoSMon signup page

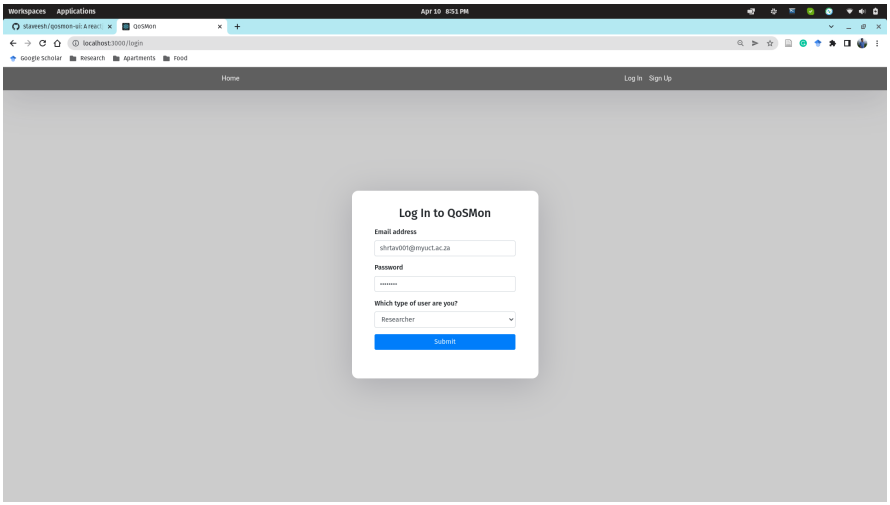


Figure A.4: QoSMon login page

A.6 Launching a measurement

Measurements can either be launched using the frontend or through API invocation.

A.6.1 Launching measurements using the frontend

1. Once a user logs in into the system, the following screen becomes visible:

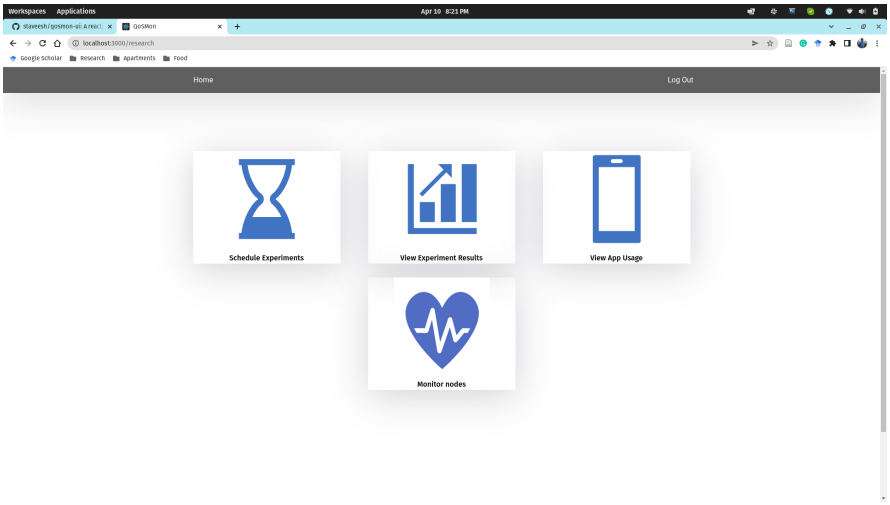


Figure A.5: QoSMon home page

Select “Schedule Experiments” on the screen shown above.

2. On the “Schedule Experiments” screen, fill the form with suitable information and click the “Submit” button.

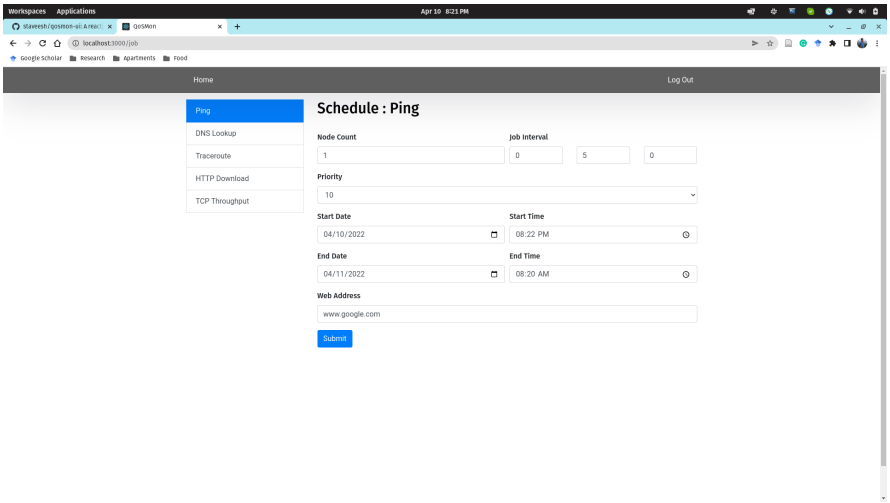


Figure A.6: Launching a measurement in QoSmon through the frontend

A.6.2 Launching measurements using API invocation

A generic JSON payload for a measurement is shown here. To schedule a measurement, a POST request with suitable payload against the /**schedule** API endpoint of the measurement server can be issued.

```
$ curl -X POST http://{measurement-server-host}:{port}/schedule -H 'Content-Type:
    application/json' -d '{payload contents}'
```

A.7 Checking measurement data in InfluxDB

To check the measurement data, issue the below commands within the measurement server. Also check InfluxDB's documentation for detailed instructions related to exporting time-series data into multiple formats.

```
$ docker exec -it influxdb /bin/bash # Opening a shell inside the influxdb docker
    container
<container-id>$ influx # Opening influxdb's command line interface
> use mobiperf; # Specifying the database
> select * from <measurement>; # A measurement can be one of ping, dns_lookup,
    traceroute, influxdb, tcp_speed_test
```

A.8 Visualizing measurement results

1. Login into the home screen of the frontend and select the "View Experiment Results" option.

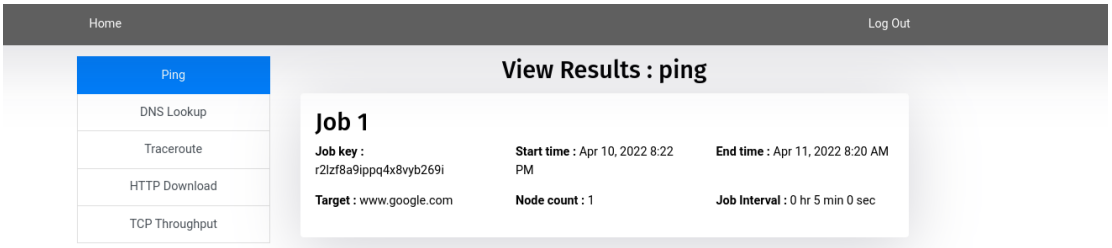


Figure A.7: QoSMon "View Experiment Results" page

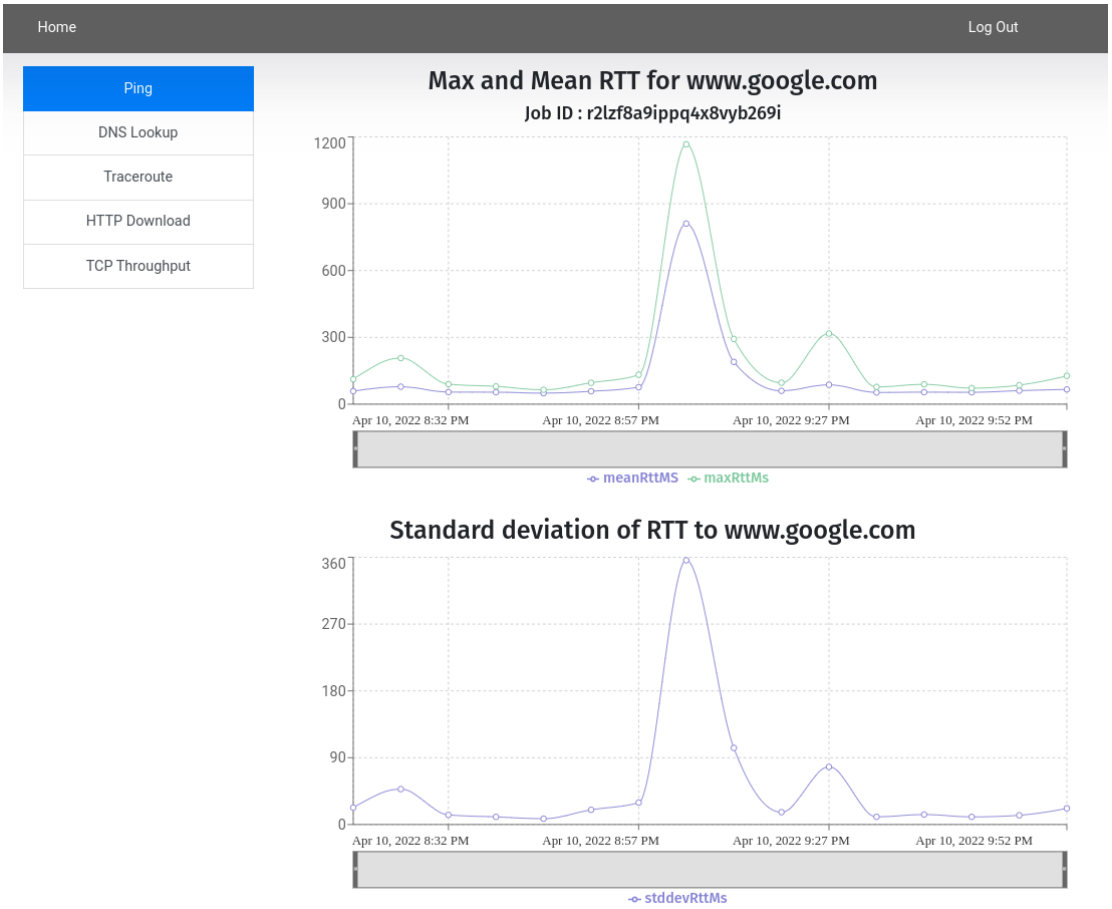


Figure A.8: Sample visualization for a ping measurement is shown here

2. Select the job type from the left pane and the job number from the cards on the right. A graph showing relevant metrics will appear as shown in the figure above.