Topology-Aware Measurement Scheduling Strategies in Low Resource Networks

Taveesh Sharma*, Josiah Chavula†
Department of Computer Science, University of Cape Town
Cape Town, South Africa
*shrtav001@myuct.ac.za, †jchavula@cs.uct.ac.za,

***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. Our results show that a graph colouring algorithm (AOSD) that arranges network measurement jobs in ascending order of their number of conflicts performs better than other scheduling algorithms like Round Robin (RR) and Earliest Deadline First (EDF).**

***Index Terms*—Internet Services, End User Applications, Internet Performance**

I. Introduction

Community networks are open, free and neutral network infrastructure built and maintained by citizens and organisations who pool their resources and coordinate their efforts [1]. These networks are often run by non-profit organizations. Further, community services like local networking, voice connections and internet access can be developed in cooperation with local stakeholders [2]. ITU statistics [3] for South Africa report that there were around 165.6 mobile-cellular subscriptions per 100 inhabitants as of 2019. Of these 165.6 subscriptions, around 102.2 correspond to mobile broadband subscriptions. Also, about 92.6% of total active internet users worldwide, and 94.7% of internet users in South Africa accessed internet through their mobile phones as of January 2021 [4]. The usage of smartphones as a computing and networking device thus presents an opportunity for constructing mobile crowdsourcing applications [5]. Such applications can be used as a platform to conduct research to improve community networks.

System architectures based on crowdsourcing are generally more complex as compared to systems where design, implementation and execution are centralized [5]. Other challenges arise due to adopting smartphones as an execution unit. It is difficult to precisely capture internet performance data through smartphones because the conditions, like location and bandwidth, are always changing over time, and as the subject moves with the device.

Although the load of executing measurements is shifted to smartphones in a crowdsourcing-based architecture, a centralized server should be able to allocate the measurements intelligently, 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 [5], i.e a bias in the measurements due to the measurement infrastructure itself. Measurement processes that are executed in common points and links could contend for shared network resources. This contention for resources is also called measurement conflict problem [6]. Thus, scheduling and synchronization of measurements among the smartphones is important to ensure proper resource utilization and accuracy of results.

The costs associated with building large-scale active internet measurement platforms that provide open access to anonymized data to researchers are generally very high [7]. Networks may become overly congested and additional data costs on users' side may be incurred as a result of the injection of probing packets. A major reason behind this phenomena lies in the skewed distribution of measurement jobs towards a few vantage points. We posit that these costs can be reduced by ensuring proper scheduling and distribution of these measurements. Our study thus introduces mobile crowdsourcing in the context of a low cost monitoring system for low-resource networks with a focus on measurement scheduling strategies. In pursuit of finding the best possible design, we present an empirical analysis of alternative techniques for measurement scheduling and synchronization. These techniques are compared on the basis of evaluation metrics like platform delay, waiting time and node busy time ratio.

II. Related work

Most of the research on community networks has been done on the basis of Guifi.net [1], [8], [9] (the largest community network) or other interconnected European community networks [10]–[12]. Limited research has been performed on monitoring network characteristics in the context of community networks for developing regions [13]–[15], and measurement scheduling aspect of systems 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 known in prior, 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.

Scheduling of active internet measurements in classical networks has been studied by few researchers. Some of these works [16], [17] have not given much attention to conflicts between individual measurements. Calyam *et. al* [18] propose a novel enhanced-EDF heuristic that allows concurrent execution of measurements to address the measurement conflict problem. Scheduling algorithms for both periodic and on-demand jobs have been proposed in this work, and have been proven to perform better than existing algorithms through computer simulations as well as in an internet testbed. Qin *et. al* [6] compare the enhanced-EDF scheme with a novel graph-coloring based approach, and through computer simulations proves to achieve effective contention resolution and low execution delays. In addition to the graph coloring approach, the original Round Robin algorithm has also been modified by the authors to work well in a concurrent context. Two variants of the proposed graph coloring scheme are considered in our implementations - 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 [19], is a drawback. However, this could benefit our centralized measurement platform. In addition to proposing AOSD and DOSD algorithms, the original Round Robin algorithm has also been modified to run in a concurrent context.

III. Measurement System Overview

This project will investigate strategies to measure under-resourced networks by deploying a containerized measurement system within iNethi [20], a localized content sharing and services platform being developed in Ocean View, a township in Cape Town, South Africa.

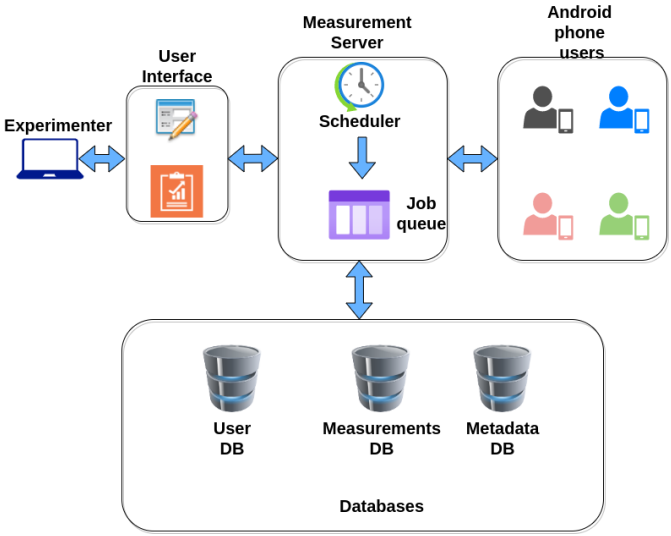


Fig. 1. A diagram showing the architecture of QoSMon, the proposed measurement system

Fig. 1 shows the architecture of the system. A key component in our architecture is a measurement server, which is responsible for orchestrating measurements like ping, DNS look-ups, HTTP downloads, and TCP throughput tests. For community network users, our system is capable of collecting and storing network usage data of applications in a phone. The data for apps is aggregated within a phone and sent to the server every 24 hours for storage. If the sending of usage data fails due to connectivity issues on a day, the data for this particular day is sent along with the batch for the next day. We implement this using a 3-step communication process. First, the phone sends a request to the server for a latest timestamp from which usage data is required for that phone. Second, the server responds with the desired timestamp. If no timestamp is returned, the phone sends usage data to the server for the last 24 hours. The measurement server is capable of running one of RR, EDF, AOSD and DOSD algorithms (Section II). Measurements are conducted in users' mobile phones distributed within the community network. Only Android phones are able to run the measurements in our current implementation. The communication between phones and measurement server is facilitated through HTTP websockets, whereby the server sends measurement jobs to the phones as and when they are ready to be dispatched. In addition to the server, a user interface is developed for community network administrators and researchers to schedule experiments, visualize the collected data and perform QoS analysis. Community network members will be able to visualize application usage through their phones.

IV. Implementation Details

A. Conflict determination

Conflicts between individual job instances are first decided on the basis of target server. If two jobs are destined to the same target server, it is highly likely that they use common links in the network. Thus, we use a pairwise binary matrix for representing conflicts between active jobs. This binary matrix is then used to build an undirected conflict graph of the jobs and supplied to the scheduling algorithms as input.

Our system 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. This 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. 2 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 $\rightarrow$ AP6 $\rightarrow$ AP2 $\rightarrow$ AP4 $\rightarrow$ AP3, M4 $\rightarrow$ AP6 $\rightarrow$ AP2 $\rightarrow$ AP1 $\rightarrow$ AP3, M4 $\rightarrow$ AP6 $\rightarrow$ AP2 $\rightarrow$ AP1 $\rightarrow$ AP5 $\rightarrow$ AP3, M4 $\rightarrow$ AP6

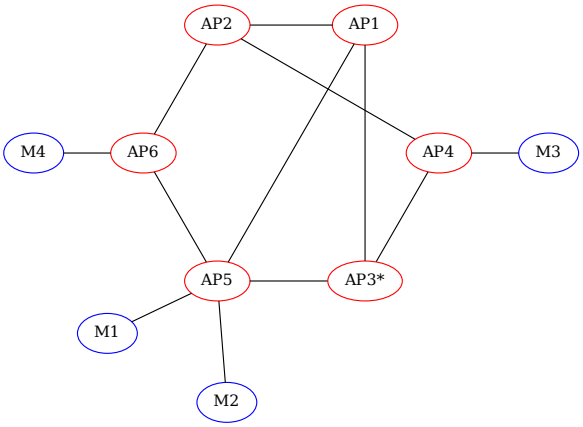


Fig. 2. A network topology with 6 access points and 4 measurement nodes. AP3 is chosen as the gateway access point, i.e, an access point having direct connection to the internet. All requests originating from other nodes in the network are routed through AP3.

$\rightarrow$ AP5 $\rightarrow$ AP3, M4 $\rightarrow$ AP6 $\rightarrow$ AP5 $\rightarrow$ AP1 $\rightarrow$ 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 job.

B. Job life-cycle

When a measurement job is requested to be scheduled, a unique identifier is assigned to the job by our system. After the job passes the criteria for a valid MobiPerf job, it gets stored into the metadata database. Next, the job is taken up by the scheduler service after which it is added into a global queue of active jobs. At this point, the job is assigned an instance number of 1. The time at which the job is added to the queue is recorded in the metadata database. A tracker thread executes within the server every 20 seconds that checks whether any of the jobs in the queue is ready to be fed to the underlying scheduling algorithm. This thread is also responsible for checking if a job is ready to be removed or reset. If a number of jobs are past their start time, they are supplied to the scheduling algorithm. The scheduling algorithm assigns a dispatch time and a mobile device to each job. A job is dispatched to the assigned device when the present time exceeds the dispatch time.

After the job finishes execution in the assigned mobile device, the job's results are sent to the measurement server through the websocket connection. The server checks the instance number of the job and records the time at which job's result was received. It also captures the execution time in milliseconds of the job in the mobile device. Starting from this time, when the tracker thread executes for the next time, it resets the job by updating the start time and incrementing the job's instance number. If the end time of the job is attained, it is removed from the job queue.

C. Device-Server communication

When our android application is installed in a phone, it establishes an HTTP websocket connection with the server. This connection is kept alive until the server is shut down manually or the app is manually stopped within the phone. Our android application is robust enough to handle connectivity changes in the community network. If the internet connectivity changes or the user switches to a different type of internet connection in the phone, the phone establishes a new websocket connection with the server. When new jobs are ready to be dispatched from the server, the server publishes the list of jobs to a fixed endpoint. The phone subscribes to this endpoint and receives jobs as and when they are made available.

V. Performance Evaluation

Before the deployment of our system in the community network, we performed evaluations in a lab setup. Fig. 3 and Fig. 4 show the testbeds of our measurement system. In the lab setup, four Android phones were used to conduct our experiments. We first conducted a set of preliminary experiments to determine the execution time of each job type and calibrate our scheduling algorithms. Then we assumed the maximum time across each job type as the expected value of execution time for any upcoming jobs in our system. We argue that the maximum value of execution time would be a good estimate so as to have a safe time window for the next job to execute when the previous job has not finished execution.

The target servers for ping, DNS lookup, traceroute and http were chosen uniformly from Alexa top 8 global websites [21]. TCP speed tests were run against a local speed test server running on the same test network. The periods of the jobs were chosen uniformly from the range [5, 10] minutes.

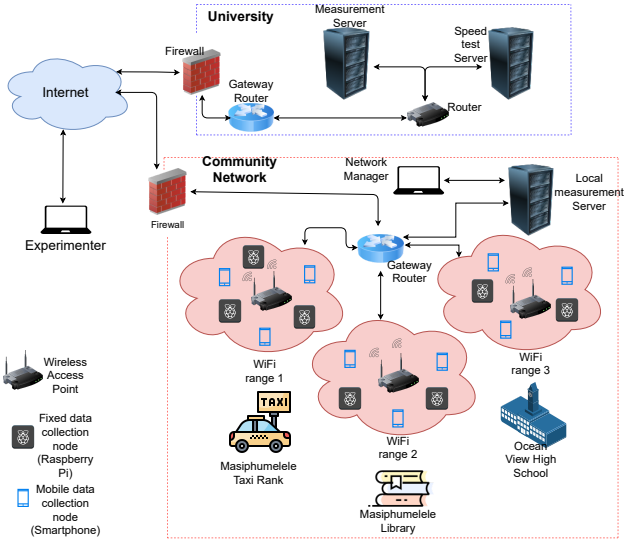


Fig. 3. A diagram showing the testbed for QoSMon set up in the iNethi community network

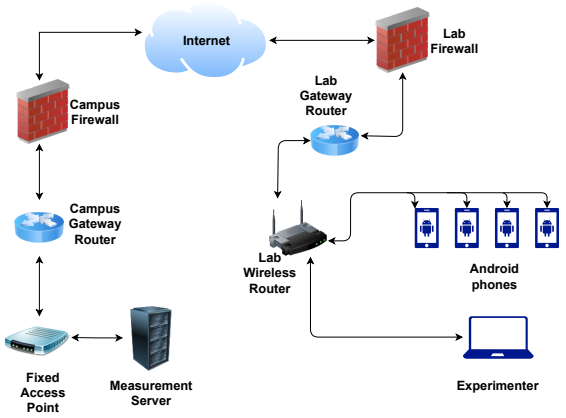


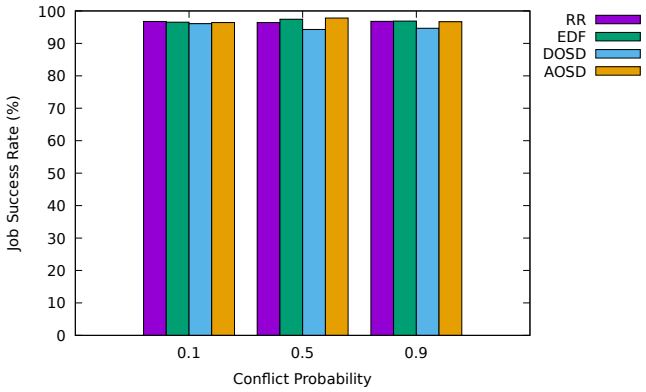
Fig. 4. A diagram showing the testbed for QoSMon set up in a lab environment

For our main experiments, we executed 20 periodic jobs with randomized network topologies containing 8 access points and 4 measurement nodes. Every topology had a conflict probability associated with it. A conflict probability of p indicates that there is an edge between any two access points with a probability of p. Topologies with lower value of conflict probability were likely to have more than one connected component. Therefore, we decided to randomly assign one gateway access point to each connected component in the generated topologies. All four scheduling algorithms were allowed to run with the same topology and set of jobs for 2 hours each. For the next iteration of our experiments, we changed the conflict probability and generated a new topology. The choice of p was made from the set $\{0.1, 0.5, 0.9\}$ so as to ensure that we could capture results for sparsely, moderately and densely connected topologies respectively.

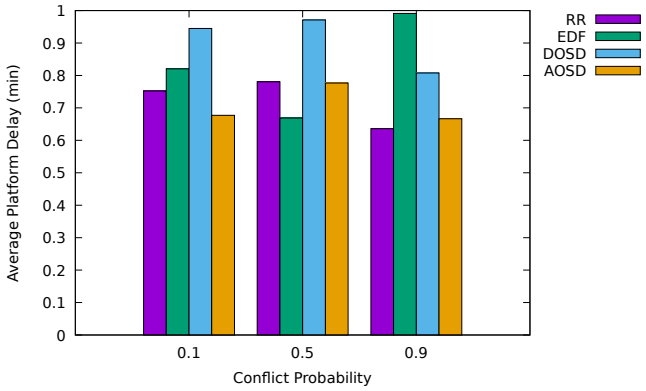
Scheduling algorithms were first compared in terms of their job success rate. For a single periodic job, success rate is defined as:

$$JSR = \frac{Number\ of\ successful\ instances}{Total\ number\ of\ instances} \times 100 \quad (1)$$

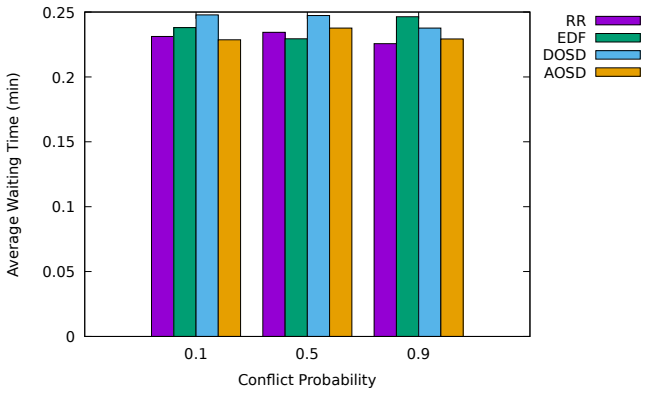
For each 2-hour long iteration on a single topology, we calculated the average JSR over all 20 jobs and plotted it against the chosen conflict probability (Fig. 5(a)). Our first observation was that none of the algorithms achieved a 100% success rate for any given topology. This can be attributed to a few jobs missing their deadlines after being sent to the phones for execution. We argue that the job success rate for each algorithm can be improved by limiting the execution period to a fixed threshold of more than 10 minutes. This will allow the tracker thread to pick up less number of jobs at once and thus reduce the overall load on measurement nodes. Another way to improve JSR would be to increase the number of measurement nodes so that jobs can be distributed in a better way. We also observed that DOSD algorithm achieved the least job success rate of all algorithms and the rest of the



(a) Job success rate



(b) Average platform delay



(c) Average waiting time

Fig. 5. Job evaluation metrics used in our experiments

algorithms had similar performance except for moderately connected topologies for which AOSD algorithm performed better than the rest.

We also calculated average platform delay for each periodic job and plotted it against conflict probability. Platform delay is defined as the time difference between actual and expected completion time of the job. For example, if a job's start time is 6:05 PM but its result is received on server end at 6:10 PM, then the platform delay for that particular instance of the job would be 5 minutes. We calculated average platform delay by averaging the platform delay over all instances of all 20 scheduled jobs (Fig. 5(b)). All four algorithms achieved

an average platform delay of less than 30 seconds, which confirms the absence of any implementation issues within our measurement server. AOSD algorithm outperformed the other algorithms for all three topologies except for the case of $p = 0.5$ where EDF performs slightly better. We argue that this is due to the randomized nature of our selection of jobs. We expected that an average over multiple job distributions would declare AOSD as a clear winner.

For determining the root cause behind platform delays, we calculated the waiting time of jobs in the queue (Fig. 5(c)). We observed 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.

In order to get an idea of the amount of load on the measurement nodes, we calculated their busy time ratios (Fig. 6). This metric translates directly into the energy efficiency of the scheduling algorithms. Better the distribution of jobs to the smartphones, lesser would be the chance of battery drainage. Node busy time ratio for i^{th} node, M_i is defined as:

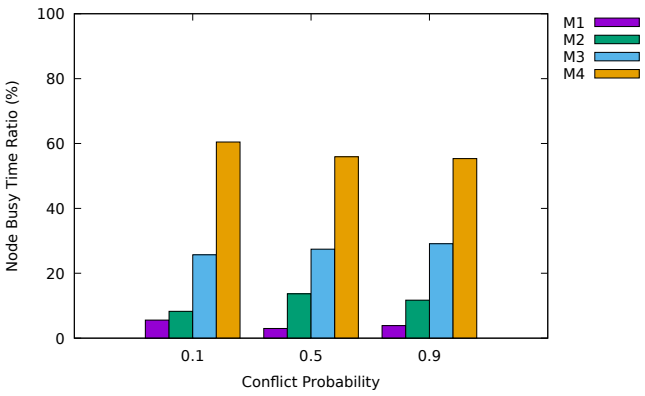
$$NBTR_i = \frac{e_i}{\sum_{i=1}^{m} e_i} \times 100 \tag{2}$$

where e_i denotes the execution time in milliseconds of all job instances in M_i in a single iteration.

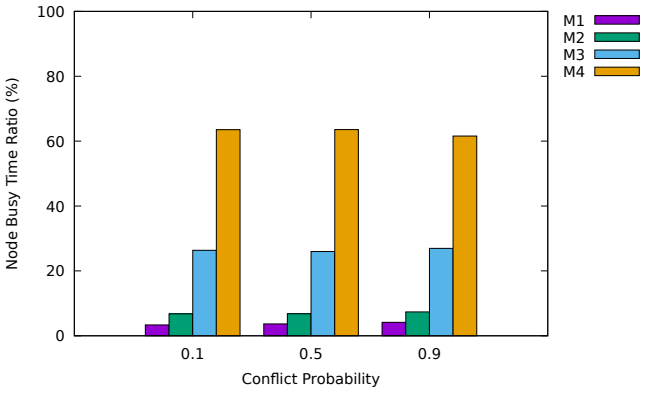
We observed a highly skewed distribution of jobs among the phones in the case of DOSD algorithm. RR and EDF had similar load distribution patterns while the best load distribution was achieved in AOSD, where one of the phones executed about 50% of the total job load in all three topologies. This confirmed that the higher contribution of external factors towards platform delay in RR, EDF and DOSD was indeed due to a skewed distribution of jobs among the phones.

VI. Conclusion

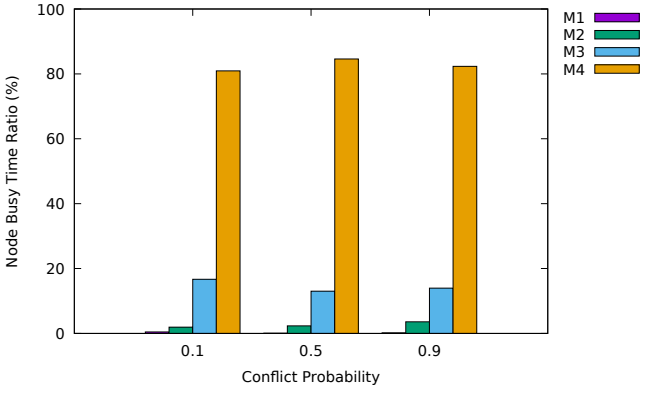
We compared our implementations of four network measurement scheduling algorithms on three different virtual network topologies. We designed our system to schedule network measurements by accounting for conflicts between individual jobs as well as conflicts that arise when active measurements route through common links and access points in wireless networks. Our results show that AOSD algorithm is superior than 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 extremely useful in areas where internet connection is unstable due to a high proportion of wireless links in comparison to wired links. Our system thus has high applicability in wireless community networks, especially in communities where users access internet through their smartphones.



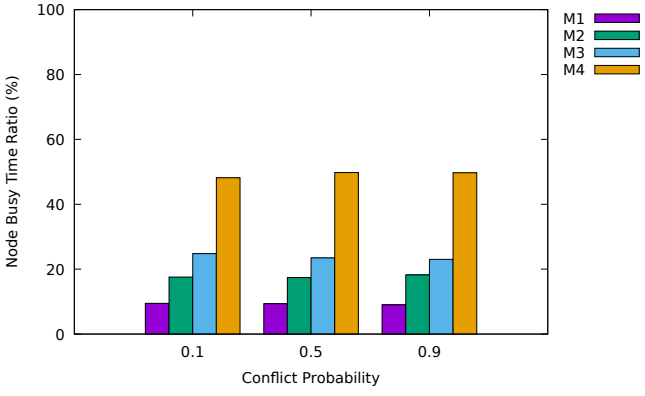
(a) RR



(b) EDF



(c) DOSD



(d) AOSD

Fig. 6. Node busy time ratios for 4 mobile phones used in our experiments

VII. Future Work

In our current work, we make an attempt to find an effective scheduling strategy for low resource network measurements. In future work, the algorithms can be coupled with a job assignment algorithm to ensure even better allocation of jobs to the measurement nodes. A future implementation of our system can also be made to support fixed nodes, like Raspberry Pis coupled with a mininet-based implementation of the network topology. For performance evaluation, dependant variables like number of jobs, number of measurement nodes, time of the day and mobility of the devices etc. will be considered. Another interesting addition to this paper would be the support for on-demand measurements and the adjustment of job schedules in accordance with a custom priority order.

References

[1] R. Baig, R. Roca, L. Navarro, and F. Freitag, "guifi. net: A network infrastructure commons," in *Proceedings of the Seventh International Conference on Information and Communication Technologies and Development*, 2015, pp. 1–4.

[2] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Viñas *et al.*, "A case for research with and on community networks," 2013.

[3] (2020) Statistics. [Online]. Available: https://www.itu.int/en/ITU-D/Statistics/Documents/statistics/2020/MobileCellularSubscriptions_2000-2019.xlsx

[4] (2021) Internet users in the world 2021 — Statista. [Online]. Available: https://www.statista.com/statistics/617136/digital-population-worldwide/

[5] A. Faggiani, E. Gregori, L. Lenzini, V. Luconi, and A. Vecchio, "Smartphone-based crowdsourcing for network monitoring: opportunities, challenges, and a case study," *IEEE Communications Magazine*, vol. 52, no. 1, pp. 106–113, 2014.

[6] Z. Qin, R. Rojas-Cessa, and N. Ansari, "Task-execution scheduling schemes for network measurement and monitoring," *Computer communications*, vol. 33, no. 2, pp. 124–135, 2010.

[7] G. Aceto, A. Botta, W. De Donato, P. Marchetta, A. Pescapé, and G. Ventre, "Open source platforms for internet monitoring and measurement," in *2012 Eighth International Conference on Signal Image Technology and Internet Based Systems*. IEEE, 2012, pp. 563–570.

[15] M. R. Lorini, M. Densmore, D. Johnson, S. Hadzic, H. Mthoko, G. Manuel, M. Waries, and A. van Zyl, "Localize-it: Co-designing a community-owned platform," in *International Development Informatics Association Conference*. Springer, 2018, pp. 243–257.

[8] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, and L. Navarro, "A technological overview of the guifi. net community network," *Computer Networks*, vol. 93, pp. 260–278, 2015.

[9] L. Cerda-Alabern, "On the topology characterization of guifi. net," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2012, pp. 389–396.

[10] L. Maccari and R. L. Cigno, "A week in the life of three large wireless community networks," *Ad Hoc Networks*, vol. 24, pp. 175–190, 2015.

[11] P. A. Frangoudis, G. C. Polyzos, and V. P. Kemerlis, "Wireless community networks: an alternative approach for nomadic broadband network access," *IEEE Communications Magazine*, vol. 49, no. 5, pp. 206–213, 2011.

[12] J. Avonts, B. Braem, and C. Blondia, "A questionnaire based examination of community networks," in *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2013, pp. 8–15.

[13] E. E. P. Pujol, W. Scott, E. Wustrow, and J. A. Halderman, "Initial measurements of the cuban street network," in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 318–324.

[14] C. Rey-Moreno, Z. Roro, W. D. Tucker, M. J. Siya, N. J. Bidwell, and J. 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*, 2013, pp. 1–10.

[16] R. L. Cottrell, C. Logg, and I.-H. Mei, "Experiences and results from a new high performance network and application monitoring toolkit," in *Passive and Active Measurement Workshop*, 2003.

[17] M. Luckie and A. McGregor, "Ipmp: Ip measurement protocol," in *Passive and Active Measurement Workshop*, 2002.

[18] P. Calyam, C.-G. Lee, P. K. Arava, and D. Krymskiy, "Enhanced edf scheduling algorithms for orchestrating network-wide active measurements," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 2005, pp. 10–pp.

[19] M. Clegg, "Scheduling network performance monitoring in the cloud," 2017.

[20] "About the Object Management Group," accessed on October 15, 2020. [Online]. Available: https://www.inethi.org.za/

[21] "Alexa Internet," accessed on May 26, 2021. [Online]. Available: https://www.alexa.com/topsites

Taveesh Sharma is a Masters in Computer Science student at the University of Cape Town (UCT). His research topic is Investigating Optimal Internet Data Collection in Low Resource Networks. He obtained his honours degree from the Birla Institute of Technology and Science in India.

Josiah Chavula is a lecturer and researcher in Computer Science at the University of Cape Town. He received a PhD in Computer Science from UCT (2017), and an MSc in Networking and Internet Systems from Lancaster University (2011). His research focuses on performance of internet systems in Low Resource contexts.