Bootstrapping the Development of Services for Wireless Community Networks

Keegan White*, David Johnson*, Melissa Densmore*, Hafeni Mthoko†
*Department of Computer Science, University of Cape Town, South Africa
[1]keeganthomaswhite@gmail.com
[2]david.lloyd.johnson@gmail.com
[3]melissa.densmore@uct.ac.za
†Office of the Pro-Vice Chancellor: Research, Innovation and Development, University of Namibia, Namibia
[4]hafenimthoko@gmail.com

***Abstract*—Affordable Internet access has been critical for continued remote education and work-from-home environments during the COVID-19 pandemic. However, localized services made available through a community network can also provide high value for education, work and entertainment in communities that lack affordable access. Creating these localized services is technically challenging and in this paper, we take the first attempt at describing the development issues faced when creating these services as well as the first attempt at fostering a collaborative approach to developing community networks in Africa. We examine a case study relating to the development of a localized service followed by detailing an improved containerised architecture for easily bootstrapping new localized services using a combination of a virtual machine, Docker containers and the Traefik HTTP reverse proxy. Through the creation and subsequent sharing of our codebase and containerised architecture, we aim to set a precedent for resource sharing and collaboration amongst developers in the African wireless community network space.**

***Index Terms*—Wireless Community Networks, Open Source Development, Virtualisation**

I. Introduction

As South Africa continues to develop, access to the internet has become an ever more pivotal tool and resource for many population members. However, mobile data usage in low-income areas in South Africa is limited. This is due to the high cost of data relative to the per capita income in these areas and variable mobile signal coverage, which results in low-quality connections [1] [2] [3]. As a result, a large portion of the country's population is left with limited or no access to the internet.

To rectify the above disparities, there has been an increase in the creation of Wireless Community Networks (WCNs) throughout South Africa. WCNs are created to connect communities to the internet while also offering zero-rated local services [4]. However, this movement struggles from a lack of collaboration and subsequently a lack of shared resources to educate and build off of. Rey-Moreno et al. [5] examined over thirty WCNs based in Africa, where more than sixty percent were based in South Africa and found that the majority of parties involved in the creation of WCNs were unaware of other projects similar to their own. He concluded that there is the need for collaboration and resource sharing to facilitate a collective movement towards bringing communication technology to resource-constrained environments [5]. Creating a starting point for collaborative development in this space is the main contribution of this paper. By sharing our entire codebase we aim to set a precedent and motivate other WCN owners to do the same, thus forming a collaborative ecosystem for WCN development. While there have been numerous papers addressing the benefits of WCNs in Africa and examining their architecture [5] [6] [7] [8], to the best of our knowledge, this is the first attempt at examining the development process taken on by contributors to an open source WCN platform and a first attempt at documenting a freely available containerised WCN server. Thus, there is no relevant related work to our project.

This paper will focus on the iNethi project, an open source platform that allows communities to quickly bootstrap a community-built internet service provider (ISP) developed with Docker containers. iNethi was started by a group of researchers, developers, entrepreneurs, and community activists. It aims to facilitate the development of WCNs simply and intuitively while also abstracting the complex networking knowledge needed to carry out such a task. Furthermore, by facilitating an easier and more efficient way to produce a WCN, these networks can become community-owned entities and, thus, create a sense of ownership and control over users' services and data. There is an active deployment of iNethi in Ocean View (OV), a low-income area in Cape Town, South Africa. There are currently ten Libremesh WiFi access points in the community, which roughly three-hundred residents use to access local services or the internet at R20 per gigabyte.

The open source nature of this project means that the code for the system is openly available online, with anyone able to contribute to the project. Students at the University of Cape Town have been actively participating in the expansion of iNethi, with numerous postgraduate projects being based on the OV network and the university's Developers' Society actively recruiting members to participate in iNethi. Such projects have shown a need for a simple and efficient way to set up a development environment that contributors can work on individually and test their services before adding them to the code base. We will examine a case study developed using the original HAProxy architecture to illustrate the Docker container architecture development process and challenges encountered, followed by an improved architecture that sim-

plifies configuration complexity. This case study outlines the need for a testing environment that nullifies the expansive knowledge of networking needed to participate in WCN service development while also emphasising the importance of WCNs in addressing the digital divide in low-income areas in South Africa. Additionally, it highlights the complex nature services can adopt when catering for the needs of members of low-income areas.

To solve the aforementioned issues, we modified our deployment environment's architecture and built an Ubuntu-based Virtual Machine (VM), that is freely available online and mirrors the OV deployment environment. The pre-configured nature of the VM allows contributors to start developing immediately and nullifies the need for a complex understanding of the deployment environment. Therefore, the VM facilitates collaboration through simplification and improves the chances of successful deployments by mitigating potential differences between development and deployment environments. Additionally, the open source nature of the configuration files and code used to create this VM makes the process of creating a functional WCN server as simple as running pre-existing build scripts. The VM uses Dnsmasq and Traefik Docker containers to route traffic and create the links needed throughout the Docker network to build a working environment.

The remainder of this paper is organised as follows: Section II outlines the background of the iNethi project, Section III motivates the need for the VM system in open source projects like iNethi, Section IV presents an iNethi case-study that exemplifies the need for the VM, Section V describes the VM framework, and section VI concludes the paper.

II. Background

This paper focuses on the iNethi deployment in OV, which launched in 2018. The network is owned by OVCOMM Dynamic, an entirely community-owned cooperative. The directors of OVCOMM Dynamic come from various backgrounds, including education, art and media, youth development, and radio communication. In addition, they live in OV and are therefore representative of the community.

The iNethi infrastructure consists of several Libremesh-based WiFi access points located in the community. Libremesh allows the network to make use of layer-two and layer-three mesh protocols. This makes it possible for users to move between access points without losing their connection. Additionally, it creates room for network expansion and accommodates scalability as the mixture of the two routing approaches allows inter-cluster links to be established over dedicated radio links. The aforementioned community access points allow users to utilise the iNethi network and internet through a voucher-based system. Vouchers cost R20 per gigabyte, making them at least five times cheaper than the lowest one-gigabyte data bundle offered by South African mobile service providers [9].

WCNs, such as the deployment of the iNethi in OV, are important pillars in societal communication architecture. The creation of WCNs lessens the digital divide by establishing more affordable internet access and communication technologies. This has become even more important in under-resourced areas in light of the COVID-19 pandemic. E-learning has become an important tool in combating the spread of COVID-19 in South Africa. At OV secondary school, a staggering 36 percent of respondents to a school-wide survey could not access online learning material during the nationwide lockdown [9]. This led to the creation of the iNethi NextCloud platform, an open source cloud hosting service similar to Dropbox that can be hosted on Linux servers, such as the server located in OV. iNethi runs a global and local instance of NextCloud that allows teachers to upload content to the global server even when they are not within the confines of the community. This content is synced with the local instance on NextCloud, and students can access it via an app or through a browser-based interface.

The creation of iNethi NextCloud stemmed from a need within the community and was expanded upon in community-led workshops with OV-based educators. This is an important aspect of iNethi and its principles; all service creation and additions to the network must stem from a need identified by those within the community rather than a project which an outside source deems interesting or useful.

The creation of iNethi and the local content distributed within the community draws on the concept of locality of sharing that Phokeer et al. [1] found apparent in low-income areas, with OV being one of the areas they examined. This concept highlights how members of such communities communicate with people within their locality. This has motivated iNethi to be a means of access to the global internet and offer locally-hosted messaging, file sharing, video streaming, music sharing, and the aforementioned e-learning platform to community members. Instead of using the most popular mainstream applications to message a neighbour via an international expanse of networks, users' can use a locally-hosted service. This thereby nullifies the need for a message to be sent via expensive external data links. This is a key factor in reducing the cost of communication and media consumption within low-income areas. Therefore, sharing the code and framework used to create and host features such as iNethi NextCloud will allow other WCNs to adopt such a service, thus benefiting the global WCN ecosystem as a whole.

There are currently few contributors to the iNethi project, meaning that there are numerous important services that the community has requested that have not been started and numerous projects that have been started but never reached deployment. The lack of continued involvement by contributors and the number of unfinished projects can be attributed to the skill gap that many contributors face and a general tendency towards a lack of commitment. These are issues that have already been documented in many other open source projects [10] [11] [12].

III. Motivation

Up until now, there have only been a handful of long-term contributors to the iNethi project. There have been many

students from the University of Cape Town that have tried to contribute. However, the complex nature of the network has led to a skill gap that many have not been able to overcome. To set up an adequate test environment, contributors need to have experience with low-level network configuration and a good understanding of traffic routing within a network. This is the main issue students, and other contributors face when trying to help contribute to iNethi. In turn, this has made collaborating with new contributors very difficult as the few active contributors have to answer a myriad of questions before these developers can start on projects. While this is possible, active contributors do not always have time to upskill new developers. These issues are best exemplified by examining previous student contributions. In 2020, four honours students attempted to create bespoke services for the iNethi project. Considerable time investment was required for integration and this was only achieved by one student. Another student then was able to leverage this work to create a plausible iNethi-hosted service, while the other two students failed to integrate their solution with the deployment environment.

The aforementioned issues have led to the development of a fully functional VM replica of the iNethi server in OV. The VM uses Dnsmasq and Traefik Docker containers to make up the networking architecture. In addition, the scripts used to set up the VM are also available online so that contributors can build an iNethi environment on their own servers, VMs or machines. While this not only equips contributors with an adequate test environment, it also allows the iNethi project to be easily reproducible. This allows more people to adopt this architecture and thereby is a step towards addressing the lack of shared resources amongst African WCN owners and developers [5].

Traefik is a Hypertext Transfer Protocol (HTTP) reverse proxy and a load balancer designed to deploy microservices, such as Docker containers [13]. It configures itself automatically and dynamically, abstracting the complex network knowledge needed to set up a WCN test environment. Dnsmasq further simplifies this process by acting as a Domain Name Server (DNS) within the VM, resolving requests for services hosted by the VM. In addition, Dnsmasq is preconfigured in the VM and does not require user intervention to run even when new services are added.

Thus, using Traefik in conjunction with Dnsmasq allows a contributor that is new to Docker and networking to start a hosted service with a Docker compose file easily. Consequently reducing the skill gap required to contribute to iNethi.

IV. iNethi Case Study

In 2020 we conducted an honours project that focused on creating a tailor-made music sharing service for the iNethi architecture in a co-design fashion. This project was proposed by members of the OV community and aimed to create the aforementioned music sharing service with specific features to allow local artists to share their music with residents of their community and people residing outside of OV in a simple, cost-effective and data-efficient way. The purpose of this website was not only to encourage local content creation and celebrate the local talent but also to stimulate the local economy by providing the musicians with a way of making money and marketing themselves online. This was addressed by incorporating social media style profiles where artists can advertise themselves, accept donations and share contact details. Additionally, the platform included an e-commerce component that allowed users to purchase coupons to access restricted content or download music marked as freely downloadable.

Background research was done before interviews with OV-based musicians to gather context and design well-refined questions. Three musicians were interviewed, and an OV-COMM Dynamic director who has a passion for music and interacts with the artists regularly.

This project set out to solve an important problem specific to low-income areas in South Africa, exemplifying the significance of not only iNethi but WCNs in general.

A. Background and Community Insights

From interviews with the musicians based in OV, it was apparent that the biggest issue faced by South African musicians in low-income areas is their inability to share their music. This is due to a lack of an established platform that aligns with the data restraints they and their fellow community members face.

This has lead to music sharing in low-income areas being dominated by Bluetooth and WhatsApp file sharing [14] [15]. There has only been one previous attempt to facilitate music sharing in these communities, which resulted in the development of a website called KasiMP3. It was designed for use on feature phones with data constrained users in mind [15] [16]. At its peak, KasiMP3 had over fifty-thousand artists on their platform. However, this platform is no longer running.

During the feature selection process, the OV-based musicians suggested that the music sharing system incorporated a feature that allowed them to sell access codes, referred to as coupons, for the music they chose to place behind a paywall. This would replace their need to sell compact discs at live performances, which they said had become less lucrative over the years as digital music consumption has become more prevalent.

B. Implementation

To allow musicians to grow their audience outside of OV, two instances of the music sharing website were created. The first was a version that would only be seen when someone accessed the website using a wireless access point within the community. The second was hosted on an Amazon Web Services (AWS) server and was accessible by anyone. These two versions will be referred to as the local and global deployments, respectively. Each site only differed in terms of feature accessibility. The global website only allowed users to sign up as a customer and download music. In contrast, the local website allowed users to sign up as customers or musicians and upload their music.

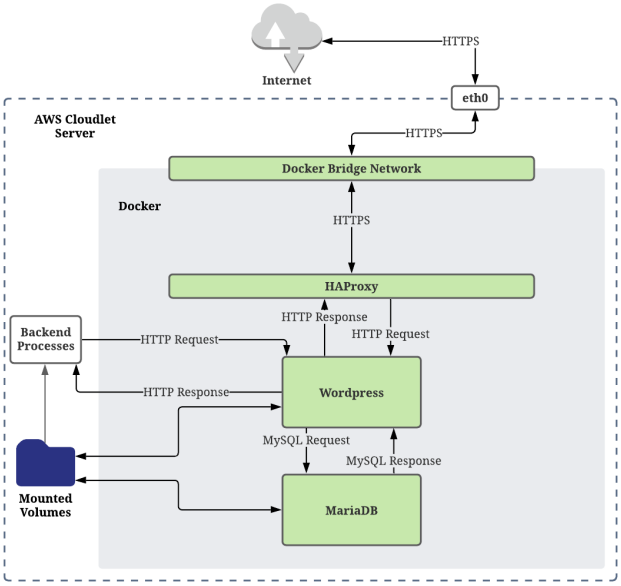

Fig. 1. Architecture of the global music sharing website

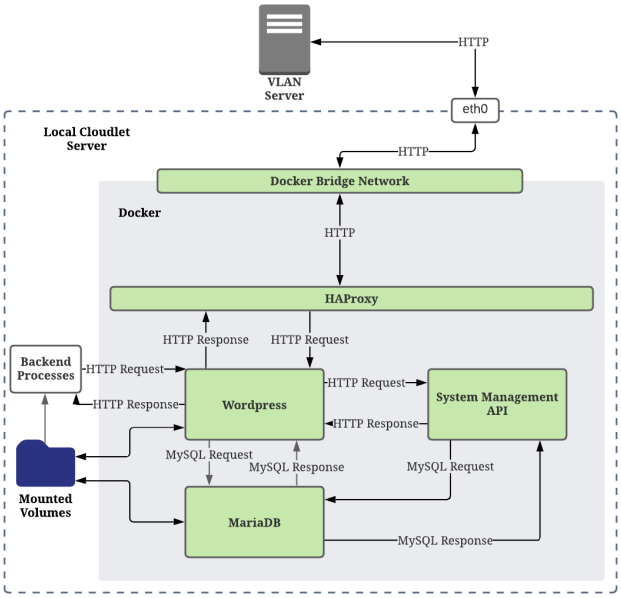

Fig. 2. Architecture of the local music sharing website

These websites were built using various Docker containers. The website front-end was built using WordPress to allow for easy customisation from someone without computer science knowledge. This aligns with the goal of iNethi to build community-owned networks. In essence, much of this project was built in the simplest way possible to allow for community maintenance and ownership. The system consists of a WordPress Docker container, a MariaDB Docker container, and a custom-built Python flask-based Application Programming Interface (API) housed within a Docker container, known as the system management API. The system management API handles the coupon creation process and updates the download statistics for each song.

When the system is built, the Docker containers are connected to a Docker bridge network, which allows them to communicate with each other. An HAProxy Docker container was run as a reverse proxy server on both the AWS and local servers. This takes in Hypertext Transfer Protocol Secure (HTTPS) requests for the AWS server, verifies and decrypts them and then passes them to the WordPress Docker container as an HTTP message, see Fig. 1. In the local network, the HAProxy container routes HTTP messages to the WordPress container or system management API, see Fig. 2.

On both servers, the WordPress container carries out the request and returns an HTTP response to the HAProxy container, which is then encrypted in the case of the AWS server and sent as an HTTPS response back to the user. In the case of the local server, the process is the same, except the response is sent in HTTP format. Optionally, the WordPress container queries the connected MySQL database, hosted in a MariaDB container, if data is needed from it.

The aforementioned request pattern includes actions like downloading music and browsing the site. These requests are handled by the containers and are thus architecture-independent and run on any server that can run Docker. However, in more complex cases where custom-built solutions are needed to handle user requests, the system management API or server will play a role in carrying out a portion of the request. The two core features of the website fall into these categories. Both the music upload process and coupon creation process are handled by the server and API, respectively.

The music upload process uses backend processes run on the server to coordinate product creation between the local and global server, i.e. creating an entry for a song in the e-commerce store that acts as the music library. These processes include updating the file name, moving the file into a permanent location, syncing files between the servers, reading from the database to get the user's submitted information and then using the WordPress WooCommerce API, the e-commerce store used on the website, to create the product locally and globally.

The coupon generation process uses the system management API to process user input and coordinate coupon creation between the local and global servers. The API uses calls to the WordPress WooCommerce API and information gathered from the local database to create a coupon.

C. Deployment Issues

Although the system was successfully tested throughout development within a local environment, the architecture-dependent nature of the song upload process meant the deployment on the local server was unsuccessful. This was due to compatibility issues with the libraries used for the backend processes. In addition to this, integrating the service's Docker containers, which were architecture-independent, into the deployment environment was difficult and time-consuming. The manual and environment-specific configuration that HAProxy required led to friction when trying to deploy this service be-

fore testing could take place. When this phase was completed, the testing phase brought to light the library compatibility issues.

V. Virtual Machine Framework

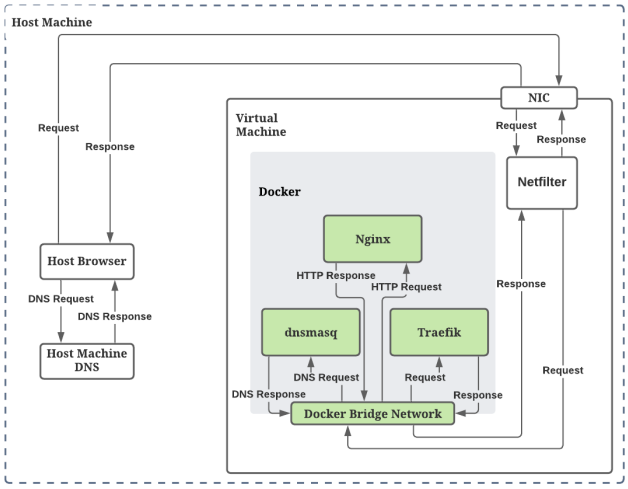

Fig. 3. Architecture of the VM

After the conclusion of the music sharing project, we had seen the shortcomings of the current software development life cycle employed in the iNethi project. It was noted that the chance of an unsuccessful deployment of the music sharing service could have been significantly reduced if we had a development environment that mimicked that of the iNethi deployment environment. To stop projects from experiencing issues similar to those observed during this honours project, the aforementioned VM was created.

The VM is built using Oracle's VirtualBox, a free and open source hosted hypervisor for x86 and AMD64/Intel64 virtualisation [17]. The VM itself runs the same OS as the OV server, a 64-bit Ubuntu 18.0.4 long term support OS. By mirroring the OS and the rest of the network within the VM, we ensure that there will be no unexpected software or library incompatibilities when deploying a new service.

Docker is pre-installed on the VM, and the architecture is pre-configured. When Docker is installed, it creates two custom Iptables chains named 'DOCKER-USER' and 'DOCKER' [18]. It also ensures that incoming packets are always initially checked by these two chains. This allows the host machine and the VM to access the services hosted on the VM without any complex manual configuration. Iptables is a pre-installed firewall system that interfaces with the kernel's Netfilter packet filtering framework. It interacts with traffic through packet filtering hooks in the Linux kernel's networking stacks. Whether incoming or outgoing, every packet will trigger these hooks allowing specific associated programs to interact with these packets at key points in the networking process.

Two Docker containers are pre-configured on the VM to facilitate the networking needed to use the VM as a development environment. Traefik is set up as a reverse proxy, and Dnsmasq acts as the DNS.

Traefik was chosen as it is a simple and easily configurable reverse proxy that eliminates the need for configuration of each route to connect paths and subdomains to their microservices. Instead, this is done automatically as services are started and removed. It intercepts and routes incoming requests to the corresponding backend services using service discovery to configure itself dynamically from these services. Additionally, Traefik does not need to restart every time a service is added, thereby minimising the overhead of adding new services to the development environment. Furthermore, this choice was motivated by the shortcomings of the old HAProxy architecture as it was difficult to use and required manual configuration to deploy a new service successfully.

Dnsmasq is used as it is a lightweight application that is suitable for resource-constrained environments. Thus, the DNS virtualisation will not take up computing power that can otherwise be used for development [19]. Furthermore, using Dnsmasq means that the VM can act as a self-contained development environment because the hosted services can be accessed from the VM's browser. It also allows the host device to access the VM's hosted services with minimal configuration on the host machine. The Dnsmasq configuration file does not need to be edited by users, which abstracts knowledge of DNS services and network traffic routing that a contributor may otherwise need.

The first step in interacting with the VM's services involves domain name resolution, whether that be for a request sent from the browser of the host machine or from within the VM, see Fig. 3. Incoming requests on port 53 will be sent to the internal Internet Protocol (IP) address of Dnsmasq. This will act as a DNS and confirm that the request has reached the VM. Any request on port 83 or 8080 is forwarded to Traefik, which uses a wild card configuration to ensure that any Uniform Resource Locator (URL) ending in '.inethihome.net' is forwarded to the relevant Docker container's internal IP address where the request will be processed. For example, if a user were to search for 'splash.inethihome.net' in the internet browser on their host machine, the request would be sent to their local DNS and resolve to the IP address of the VM, see Fig. 3. This request is then be sent to the VM via its Network Interface Card (NIC). The incoming packets will activate the hook for port 53 in the Docker chain. This packet will be sent to Dnsmasq, which sends a confirmation response packet to the host machine via the NIC. Following this, the subsequent packets will activate the port 80 hooks in the Docker chain and be sent to the internal IP address of Traefik, which will subsequently send them to the Nginx Docker container. The Nginx Docker container is used to host the 'splash' page - the web page a member of the OV community would see when they first connect to an access point.

This means developers only need to add Traefik labels to their Docker compose file to integrate a service with the environment. Once these are added, the above processes will facilitate their service's networking requirements, and they will be able to test their services in a quasi-iNethi deployment

environment. Once the service has been tested on the VM, it can easily be deployed on the local or global instance without modification.

The VM is also pre-loaded with shell scripts used to build the iNethi environment. A 'build all' script allows the user to select what services they wish to have running on their machine. This script then automatically configures the network and other settings needed to mirror the iNethi deployment environment. Furthermore, these services can be stopped and resumed with other scripts that are freely available online. Therefore the creation of the VM is more than just an effort to allow contributors to test their services. It is a fully configured containerised version of iNethi and allows the machine running it to utilise the VM as a pseudo access point. Because the scripts to build this system are made available to the public, this will allow people to build an iNethi server effortlessly with no manual configuration. This is a significant step in making iNethi a product rather than just a project in the eyes of potential network administrators, as the code that was used to set up the VM can be run on any Ubuntu-based machine to set up a replica of the OV deployment server. Therefore, we are facilitating the first step towards creating a collaborative WCN ecosystem in Africa by facilitating simple external deployments of our framework.

VI. Conclusions

Open source projects can struggle to gain traction and contributors when there is a significant learning curve needed to understand the codebase and deployment environment. Due to the complex networking knowledge and understanding of Docker that is needed to grasp the intricacies of iNethi, there has been an issue in maintaining long-term contributors' interest in the platform and deploying projects. This has led to a small team of contributors and a slow code output rate compared to other open source projects. These factors motivated the development of a VM that imitates the deployment environment of iNethi in OV. This will allow contributors to focus on their individual projects without spending time configuring a suitable development environment. By abstracting the networking knowledge that would otherwise be needed, the skills required to work on iNethi are less specific, and projects that work on a developer's local machine have an increased chance of successfully integrating with the deployment environment in OV.

Additionally, making this code freely available online facilitates the creation of more WCNs similar to iNethi and allows other networks to use our code in pre-existing infrastructures. Ultimately this will allow for developers and network administrators from different WCNs to create collaborative relationships. This is a vital step in addressing the lack of resource sharing and tools available in the African landscape for deploying WCNs and building services for them.

Acknowledgements

This work was supported in part by the Telkom Centre of Excellence for Broadband Networks and Applications as well as the Telecom Infra Project.

References

[1] A. Phokeer, M. Densmore, D. Johnson, and N. Feamster, "A first look at mobile internet use in township communities in south africa," in *Proceedings of the 7th Annual Symposium on Computing for Development*, 2016, pp. 1–10.

[2] A. Phokeer, D. Johnson, and M. Densmore, "Characterisation of mobile data usage in township communities," *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, 2016.

[3] S. Hadzic, A. Phokeer, and D. Johnson, "Townshipnet: A localized hybrid tvws-wifi and cloud services network," in *2016 IEEE International Symposium on Technology and Society (ISTAS)*. IEEE, 2016, pp. 1–6.

[4] L. Maccari and R. Lo Cigno, "A week in the life of three large wireless community networks," *Ad Hoc Networks*, vol. 24, pp. 175–190, 2015, modeling and Performance Evaluation of Wireless Ad-Hoc Networks. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870514001474

[5] C. Rey-Moreno and M. Graaf, "Map of the community network initiatives in africa," *L. Belli, Community connectivity: building the internet from scratch*, pp. 149–169, 2016.

[6] S. J. N. Noutat, T. D. Ndié, and C. Tangha, "Wireless community network services: Opportunities and challenges for dcs: Case of rural cameroon," in *International Conference on e-Infrastructure and e-Services for Developing Countries*. Springer, 2012, pp. 308–317.

[7] K. Sibanda, H. N. Muyingi, and N. Mabanza, "Building wireless community networks with 802.16 standard," in *2008 Third International Conference on Broadband Communications, Information Technology Biomedical Applications*, 2008, pp. 384–388.

[8] A. Phokeer, S. Hadzic, E. Nitschke, A. Van Zyl, D. Johnson, M. Densmore, and J. 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*, 2020, pp. 342–344.

[9] A. van Zyl and D. L. Johnson, "inethi: locked down but not locked out," *XRDS: Crossroads, The ACM Magazine for Students*, vol. 27, no. 2, pp. 54–57, 2020.

[10] I. Steinmacher, G. Pinto, I. S. Wiese, and M. A. Gerosa, "Almost there: A study on quasi-contributors in open-source software projects," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 256–266.

[11] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel," *Research policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[12] J. Coelho and M. T. Valente, "Why modern open source projects fail," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 186–196.

[13] T. Labs. (2021, jun) traefik proxy. [Online]. Available: https://traefik.io/traefik/

[14] A. Schoon, "Digital hustling: Ict practices of hip hop artists in grahamstown," *Technoetic Arts*, vol. 12, no. 2-3, pp. 207–217, 2014.

[15] ——, "Distributing hip-hop in a south african town: From the digital backyard studio to the translocal ghetto internet," in *Proceedings of the First African Conference on Human Computer Interaction*, 2016, p. 104–113.

[16] M. Mapaya, "Download currency - the kasimp3 story," Johannesburg, 2018. [Online]. Available: https://www.iafrikan.com/publications/DownloadCurrency.pdf

[17] Oracle. (2021, jun) Virtualbox. [Online]. Available: https://www.virtualbox.org/

[18] D. Inc. (2021, jun) Docker and iptables. [Online]. Available: https://docs.docker.com/network/iptables/

[19] S. Kelley. (2021, jun) Dnsmasq. [Online]. Available: https://thekelleys.org.uk/dnsmasq/doc.html

Keegan White is studying his MSc. Computer Science at the University of Cape Town, focusing his research on machine learning in software defined networks.

David Johnson is an adjunct senior lecturer in computer science at the University of Cape Town and the director and founder of iNethi.

Melissa Densmore is an associate professor in computer science at the University of Cape Town and a co-founder of iNethi.

Hafeni Mthoko is a researcher in the Office of the Pro-Vice Chancellor: Research, Innovation and Development at the University of Namibia.