Machine-to-machine communication in rural conditions: Realizing KasadakaNet

Fahad Ali
VU Amsterdam
m.f.y.ali@student.vu.nl

Supervised by Victor de Boer

ABSTRACT

Contextual constraints play an important role in ICT for Development (ICT4D) projects. These ICT4D projects include those whose goal is to enable information and knowledge sharing in rural areas while keeping constraints such as lack of electricity and technological infrastructure or (technical) illiteracy of end-users in mind. The Kasadaka project offers a solution for locals in rural areas in Sub-Saharan Africa to share knowledge. Due to a lack of technological infrastructure, networks and internet connections are often not available. Therefore, many ICT implementations in those areas are not able to share data among each other. This paper explores the possibilities of a machine to machine communication method to enable information sharing between geographically distributed devices. We have developed a Raspberry Pi-based device called the Wifi-donkey that can be mounted on a vehicle and facilitates information exchange with nearby devices, using the built-in wifi card of the Pi 3. We evaluated the solution by simulating a low resource setting and testing it by performing so-called "pass-bys". Results show that the system works fairly reliably in the simulated setting. The machine-to-machine communication method can be used in various ICT4D projects that require some sort of data sharing functionality.

1. INTRODUCTION

Bringing Information and Communication technology to rural areas of the world has proven to be quite a difficult task. The scientific research field that deals with these types of problems is called Information and Communication Technologies for Development (ICT4D). In general, ICT4D projects need to keep end-users and surrounding contexts in mind, even more so than traditional IT projects. Off-the-shelf solutions have proven to be ineffective when implemented in rural countries (Heeks, 2008). Circumstances such as a lack of financial resources, electricity, (technological) infrastructure, or low literacy rates and technical skillsets of end-users should not be overseen as they often have a significant impact on the outcome of projects.

Over the past few years, members of the Web Alliance for Regreening in Africa (W4RA)[1] at the Vrije Universiteit in Amsterdam have conducted research and implemented technologies in the context of ICT4D in a number of rural areas in Africa (de Boer et al., 2012; Guéret, Schlobach, De Boer, Bon, & Akkermans, 2011; de Boer et al., 2015). One of the explored concepts is using Semantic Web technologies to facilitate knowledge sharing in rural areas (Guéret et al., 2011).

One of the resulting products that came out of this research is the Kasadaka[2], a low-resource Raspberry Pi-based device that provides an infrastructure on which voice-based applications can be built and deployed locally. These applications for the Kasadaka are usually custom-built for specific needs and use cases. New applications can also be created fairly easily, making the platform ideal for rapid prototyping (Baart, 2016). Essentially, these Kasadaka's are deployed on a one-per-village basis, giving each village access to its own little piece of technology that facilitates local information and knowledge sharing.

In their current state, these Kasadaka's do not communicate with each other, making it difficult to send data from one Kasadaka to another, or to accumulate data contained on geographically distributed Kasadaka's in one place. Machine-to-machine (M2M) communication could offer some valuable insight for ICT4D projects that deal with similar communication issues. M2M communication is, by definition, communication between machines and therefore requires little to no human intervention (Cha, Shah, Schmidt, Leicher, & Meyerstein, 2009). It is widely applied in Internet of Things (IoT) use cases such as smart homes. M2M communication methods could aid in elevating low resource knowledge sharing use cases to the next level.

This paper presents an approach to these types of problems. Our approach is based on a method that facilitates machine-to-machine communication through what we call "pass-bys", therefore enabling the sharing of semantic data across different Kasadaka's. We thus pose the following research question: "How can machine-to-machine communication and semantic data sharing be achieved using a pass-by communication method in such a way that it matches requirements from ICT4D use cases?".

The rest of this paper is structured as follows: section 2 describes related works, such as the Kasadaka and some of its applications, machine-to-machine communication in general, and other projects that dealt with similar problems. Section 3 offers some more depth into the problem description, effectively scoping the problem area and explaining our approach to tackle it. Section 4 describes our solution from a

[1]`http://w4ra.org/`

[2]`http://kasadaka.com/`

technical perspective. In section 5, our solution is evaluated through experiments and evaluation results are presented. Section 6 discusses the meaning of the results of the evaluation as well as general aspects of the system. Furthermore, limitations and known issues and bugs are listed. Section 7 concludes the paper with a brief summary and how future works can build upon the research presented in this paper.

2. RELATED WORKS

This section covers some existing literature surrounding the project presented in this paper. Topics related to the Kasadaka and its applications, machine to machine communication, and networks that can operate offline are covered.

2.1 The Kasadaka

The Kasadaka is a low-cost device based on a Raspberry Pi[3] and is used to bring information and communication technology to several developing countries in Africa. It is built to conform to the technical and non-technical constraints of those countries and keeps the end-users' situation in mind. The Kasadaka uses a GSM dongle to connect to local GSM networks (Baart, 2016). This allows applications that run on the Kasadaka to implement some form of machine-to-human communication by exposing data stored on the machine. This is often done through a voice interface. The GSM dongle, along with Asterisk[4], essentially allows people to use their mobile phones to "call" the Kasadaka device and retrieve information stored on it or create new information. Internally, the Kasadaka features a triple store to store data and a number of Python programs to retrieve and exchange data between the data store and Asterisk, which handles the phone calls.

2.2 Applications

Several use cases have been developed into applications using the Kasadaka already (Lô & Blankendaal, 2016; de Boer et al., 2012; de Boer, Bon, WaiShiang, & Gyan, 2016). These applications implement common information services, such as a marketplace and a veterinary information service, among others. These simple applications can be very impactful for the locals in rural communities as they take their needs into account, which is often the need to share knowledge. The shared principle behind these applications is to make information — depending on the use case —available to people by creating a service within an application that runs on the Kasadaka. Because of a lack of technological infrastructure and low literacy rates, information is often spread through speech. Keeping those constraints in mind while also making use of existing GSM networks often results in applications having voice interfaces. Several of these applications are highlighted below.

2.2.1 RadioMarché

RadioMarché (de Boer et al., 2012) is a Market Information System (MIS) that facilitates agricultural trade in rural communities of developing countries. It allows people to put up offerings of the products that they want to sell. These offerings, along with contact details of the seller are all stored in a triple store in RDf format. The stored data is aggregated and eventually sent to broadcasting stations on a periodical basis. These broadcasting stations broadcast product and seller information regularly, allowing potential buyers to listen in to the radio stations and contact a seller if they are interested in their product.

2.2.2 Digivet

DigiVet (Lô, 2016; Lô & Blankendaal, 2016) is a service that provides information about animal diseases. The application allows animal-owners and farmers to diagnose a sick animal by answering questions. The diagnoses on the application are stored in a decision tree structure, certain answers will result in certain diagnoses. It is also possible to contact a veterinary doctor if necessary, or provide the right care for the animal themselves. The DigiVet application has multiple ways of exposing information to a user. It contains a voice-based interface that makes use of a GSM dongle with Asterisk and the Kasadaka's functionalities, as well as visuals that can be displayed with an attached LCD screen for the Raspberry Pi.

2.3 Machine-to-machine communication

Machine-to-machine communication (M2M) refers to machines communicating with each other without the need of a human element. Typical examples include data transfer over the internet, or smart home devices communicating with each other over a local Wifi or Bluetooth connection. A particularly interesting example of M2M is the Nintendo 3DS Streetpass functionality[5]. A Nintendo 3DS is able to communicate and exchange information with another 3DS simply by passing by it, without having an internet connection available. When the lid of the 3DS is closed, it actively looks for other nearby 3DS devices over a wifi-like network[6]. When another 3DS is in range and detected, some data is exchanged between the two devices, thus creating a "streetpass".

M2M communication could be very useful in constrained ICT4D projects, as it allows, by example of the 3DS' Streetpass, communication and exchange of data between devices without the presence of an internet connection. M2M communication in rural settings has been researched before, even with the Kasadaka as a specific use case. The related work described in the next subsections each show examples of M2M communication.

2.3.1 SMS-based M2M communication

Valkering et al. introduced and implemented a method for machine-to-machine communication using SMS messages as a substitute for the Web to transfer semantic data in RDF format (Valkering, de Boer, Lô, Blankendaal, & Schlobach, 2016). With the Kasadaka as a use case, semantic data was succesfully transferred from one Kasadaka to another through SMS messages. The SMS messages essentially contain compressed SPARQL queries and their results. These queries and their responses are sent back and forth over SMS, thus transferring data in a similar way HTTP requests and responses would. In order to reduce the number of SMS messages required, various data compression methods were

[3]`https://www.raspberrypi.org/`
[4]http://www.asterisk.org/

[5]`http://www.nintendo.com/3ds/built-in-software/streetpass`
[6]`http://3dbrew.org/wiki/StreetPass`

tested. The SMS-based communication method has been applied and tested on the DigiVet and the RadioMarché applications that both run on the Kasadaka platform.

The SMS-based solution proposed by Valkering et al. aims to solve the same goal; sharing data across devices without using the internet. In comparison with the SMS-based solution, this paper presents an alternative solution to achieving the aforementioned goal. The SMS-based solution is particularly limited by the fact that SMSes can only contain a set number of characters, and cost a set amount of money. The solution presented in this paper will not rely on SMSes as a method of data transfer, but still uses the Kasadaka as a baseline.

2.3.2 *Entity registry system*

The Entity Registry System (Charlaganov et al., 2013) describes a way of setting up a network such that it allows devices to communicate offline locally, as well as communicate online when an internet connection is available. It consists of three core types of components: Contributors, Bridges, and a global server.

Contributors essentially create, update, retrieve and delete data locally; these are the elements of a network that generate data that essentially needs to be available to other Contributors. Bridges ensure some extent of data sharing by connecting isolated clusters of Contributors. Data can be shared through a synchronization that consists of two steps: Sending new information from the contributors to the Bridge, and sending data from the Bridge's local storage to the Contributors. Finally, the Global server collects everything from each contributor through their Bridge, and makes it accessible in read-only form from the Web. Bridges and the Global server are optional components and are specifically used to enhance data availability. The Global server makes data on the ERS available to the outside world, while the Bridges make data available to other Contributors. This essentially corresponds to global and local sharing of data.

The ERS aims to synchronize data across all Contributors through the Bridges, which means there is a flow of data between Bridges and Contributors. The solution presented in this paper takes some inspiration from this. In terms of Bridges and Contributors, our solution uses just a single, but mobile Bridge. This Bridge passes by statically positioned Contributors periodically, in order to push and pull data to and from them.

2.3.3 *The last inch of the last mile*

DakNet, an ad hoc network that provides asynchronous networking capabilities to wireless devices in rural areas is a working example of M2M communication in rural areas (Hasson, Fletcher, & Pentland, 2003; Pentland, Fletcher, & Hasson, 2004). DakNet uses a "store-and-forward" networking concept that involves physically moving devices that send out a wifi network, also known as mobile access points. In the United Villages project (Hasson, 2010), DakNet was implemented in rural parts of India, Cambodia, Paraguay, Rwanda and Costa Rica. The projects included mounting a wifi access point device on vehicles which would drive around villages. Each village was equipped with one computer that would broadcast its cached data and applications over a local wifi hotspot. The villagers would have wifi-enabled devices with which they could communicate with the computer that broadcasts the hotspot, thus making use of the data and applications contained on the computer. The mobile access point vehicles would periodically visit the villages and exchange data with the local computers that broadcast the hotspots, making it available to other devices through the hotspot. This effectively created a network to bring information and communication technology to rural communities.

The solution presented in this paper is heavily inspired by, and very similar to DakNet; a moving vehicle that sends out a local Wifi network and periodically sends data to or receives data from other devices are similarities in both solutions. A key difference, however, is that our solution uses the Kasadaka as a baseline which is built using open standards, whereas DakNet is a commercial solution. The Kasadaka by default uses GSM networks and voice interfaces to make information available, which means that end-users do not need a wifi-enabled phone to communicate with Kasadakas. In the case of DakNet, the local computer in each village sends out a hotspot network, making wifi-enabled devices a requirement to retrieve information.

2.3.4 *Offline communication and file sharing*

An interesting example of a network that can work completely offline is FireChat,[7] a social networking app that allows users to send public and private messages to other users of the app. Using Bluetooth and peer-to-peer Wifi, it creates a mesh network between phones that have the FireChat app installed. Messages and files can be sent from one node in the network to any other node. The message is sent from node to node in the network until it is delivered to the intended recipient. The range of communication between two nodes can be up to 200 ft/60 meters. End-to-end encryption allows only the sender and the intended recipient to see the message. If at any point a node connected to the network goes online, then that node can be used to send long-range messages as well. The message can then be sent to another online node that is geographically closer to the intended recipient, or even the intended recipient itself if they are connected to the internet.

A different example of an offline-capable networking device is the PirateBox[8]. The PirateBox is a DIY-device that can be built with cheap hardware components. Two devices can be used as a base for a PirateBox: either a router or a Raspberry Pi. The PirateBox hosts a Wifi network that other devices can connect to. When connected, the PirateBox offers functionalities like chat rooms and file sharing. The files will be stored locally on the PirateBox and available to other devices that are connected to it. The system can easily be made portable as well, for example by powering a Raspberry Pi with an external battery, solar panel, or a powerbank. This makes the system mobile, meaning that it's possible to move around with the PirateBox, offering its functionalities to different users in different locations. Furthermore, the PirateBox's software is completely open source, making it possible to customize and adapt for certain situations,

[7] `https://play.google.com/store/apps/details?id=com.opengarden.firechat&hl=en`
[8] `https://piratebox.cc/`

possibly low-resource settings as well.

Both examples of offline communication share a certain aspect in common: physically moving components in order to bring data from one device to another. In the FireChat app, data is stored on a node of the network and passed on to other nodes when they are in close proximity, which happens over Bluetooth or local Wifi. The PirateBox can do the same; it contains data and files created by users, and if the physical PirateBox moves elsewhere, new users can gain access to the data and files created by others. The previously mentioned example of the 3DS's StreetPass functionality also contains physically moving components to exchange data. This type of data sharing is inspired by sneaker nets, which boils down to transferring data by physically bringing it from place to place instead of over a computer network. Sneaker net approaches can still be highly effective when transferring large volumes of data (Gray, Chong, Barclay, Szalay, & Vandenberg, 2002). Obviously, transferring data this way comes with a high latency. This, however, is not necessarily always a problem.

The solution presented in this paper is inspired by aforementioned examples of offline-capable networks. It is built using the PirateBox as a starting point to host local Wifi networks. Combining the PirateBox with a sneakernet approach results in physically moving it to exchange data with other devices. On paper, it seems like such an approach can work under rural conditions. Furthermore, since the PirateBox is an open source solution, it can be customized to fit our specific needs. It contributes to keeping overall costs low as well, as it is built with the same cheap hardware as the Kasadaka.

3. PROBLEM DESCRIPTION

As mentioned previously, ICT4D projects tend to involve the creation of tools that are tailored specifically for the target audience. This is crucial, as end-users often do not have access to technological infrastructure and have a limited technical skillset. The Kasadaka platform is one of those types of tools; one that facilitates information sharing in rural areas. Applications for the Kasadaka can be custom built, meaning that wherever there is a particular need, an application that offers a solution and satisfies that need can be built. However, due to limitations of the Kasadaka platform itself, not every need can be satisfied. In particular, use cases that require Kasadaka's to communicate or exchange data between themselves or with other devices can be problematic due to a lack of network infrastructure that Kasadaka's could use efficiently to communicate. The goal of this project is to develop a method that makes it easier for Kasadaka's to share their data, thus enabling new use cases for the platform as a whole.

Using the Kasadaka as a baseline, this project will develop an extension for the Kasadaka platform that solves the communication problem to some extent. Similar to the SMS-based solution (Valkering et al., 2016), this project will tackle the same M2M communication problem. The difference, however, lies in the fact that completely different techniques will be used to approach the problem. The approach used here is based off of pre-existing work and concepts, specifically the notions of sneakernets[9], DakNet, and the Wifi-donkey[10] are used as an inspiration to create a M2M communication method that can facilitate data-sharing in rural conditions. The common denominator between those concepts is the fact that there is a moving component in the system as a whole. Typically, transferring data goes through the internet. In rural conditions, however, the internet is either not available or not reliable enough, which sparks the development of alternative communication methods.

The approach we use is rather similar to the way StreetPass on the Nintendo 3DS works. As two devices pass by each other, there is just a brief moment where the devices are in range of each other. During this short timeframe, data can be exchanged. Kasadaka's are usually geologically spread out and stationary. For a pass-by to occur, a second device —something that can communicate with Kasadaka's in some way— would need to physically move through the range of a stationary Kasadaka. Given the simplicity of the task, equipping a donkey with a small device may not be such a bad idea at all. Often times NGO's are also active in these rural areas. If they are willing to collaborate, using their vehicles may also be a realistic possibility.

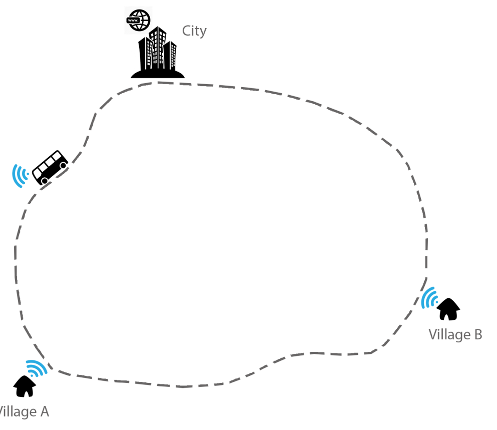


Figure 1: Example situation

Figure 1 visualises the core of the problem. A city with internet access and two rural villages are depicted. The villages are both equipped with a Kasadaka. Over time, data is generated locally on each Kasadaka by the users in that village. Up until recently, getting data from one Kasadaka to another or to the city where it can be published online was not easy. The SMS-based solution (Valkering et al., 2016) facilitates this to some extent, but is limited in the sense that it costs a set amount of money per SMS, and only certain types of content can be transferred over SMS, which does not include audio or image files for example. This is where the Wifi-donkey, depicted as a bus in figure 1, could make a difference. When the bus is in range of a Kasadaka, data exchange is possible. This happens for every village the bus visits. Eventually the bus will reach the city again, where, for example, data stored on the Wifi-donkey can be made

[9]https://what-if.xkcd.com/31/
[10]http://techland.time.com/2012/08/21/wi-fi-donkey-brings-new-tech-to-an-old-world/

available online or all the aggregated data can be analyzed.

3.1 Example use cases

Having a method to push and pull data to and from geographically distributed Kasadaka's will open up the Kasadaka platform to new use cases. This section details two example use cases that could be fulfilled with the Wifi-donkey.

The first use case we describe is that of aggregating data of multiple villages. For example, if somebody wants to know how much of a particular product, say milk, was sold in each village through the RadioMarché application. In this case, a SPARQL query that retrieves the milk sales data can be written and loaded on the Wifi-donkey. Then, as the Wifi-donkey-equipped bus visits each village, the query is executed on each Kasadaka. The query results, which should contain milk sales data, will be stored on the Wifi-donkey. When the bus has made a full circle, the milk sales data from each village will be ready for analysis.

A second use case is where each village's Kasadaka can be "updated", in the sense that new data can be pushed to its triple store. This would work similar to the first use case; there is no change in the fact that the Wifi-donkey is mounted on a vehicle which visits each village. The difference lies in the query that is loaded on to the Wifi-donkey initially; with an INSERT statement in the SPARQL query, data can be inserted into each Kasadaka's triplestore.

4. KASADAKANET

In this section we present our solution to the communication problem. Our solution, KasadakaNet, is a prototype for a machine-to-machine communication method that facilitates offline data sharing between graphically distributed devices. It consists of two main components: The Wifi-donkey, which is a customized PirateBox device, and a set of geographically distributed Kasadakas. This section covers the technical aspect of KasadakaNet. Technical challenges and their solutions, design decisions that were made along the way, and an overall view on the internal workings of KasadakaNet are discussed.

4.1 Setting up the foundation

The development phase of this project was done in several iterations. This allowed us to start with a solid foundation and continue to build upon that. Preparatory research and brief exploration of different concepts and ideas revealed several different approaches, such as sending data through soundwaves[11] or through radio frequencies[12]. One of the other approaches we found was the PirateBox[13], an open source project that essentially creates an offline networking device using relatively cheap hardware components. As explained in detail in the Related Works section, the PirateBox was an ideal approach as it takes some of our constraints into consideration, such as being cheap, open sourced and low-maintenance. Development thus started with a clean installation of the PirateBox project using a Raspberry Pi 3.

[11]https://www.chirp.io/

[12]https://www.youtube.com/watch?v=Ueb5JG0dCL8

[13]https://piratebox.cc/start

4.2 Technical challenges

A simple file-based M2M communication method could already be realized with just a PirateBox, as the PirateBox by default hosts a Wifi network and allows connected devices to upload and download files over HTTP. However, manual file up- and download was not enough. Since we were using the Kasadaka platform as a use case, we needed to involve Semantic Web technologies in some way. File up- and download is a nice-to-have, but more importantly we needed the ability to exchange semantic data over HTTP. In light of M2M communication, we needed to remove the human element as much as possible as well; the system should be able to exchange data without a human manually triggering actions. These were the two main challenges faced during development of KasadakaNet.

In order to solve the first challenge, ClioPatria[14] was installed and set up on the Wifi-donkey. ClioPatria is a webserver that includes many Semantic Web related functionalities, most importantly the ability to send and receive SPARQL queries over HTTP and is also installed on the Kasadakas. Therefore it made sense to keep consistent with tools that were already in place. Internally, data on a typical Kasadaka is stored in triples which are exposed through a ClioPatria webserver. This data can be queried using SPARQL, and new data can be created as well. We used this to facilitate the exchange of semantic data between the Wifi-donkey and a Kasadaka.

For the second challenge, we made some alterations to the Kasadaka. Since the Wifi-donkey was hosting a local Wifi network, we needed the Kasadakas to detect this network when it is in range. Once detected, the Kasadakas should be able to connect to the network, thus enabling the ability to exchange data with the Wifi-donkey. This challenge was solved by means of a continuously running script that checks whether the Wifi-donkey's network is in range and connects to it if it is. Specific details of this script can be found in section 4.3.2.

Additionally, the Wifi-donkey was also made to detect when a client device connects to its network. When that happens, it automatically sends out pre-defined SPARQL queries over HTTP to the newly connected device. In doing so, we managed to further automate the exchange of semantic data within the system.

4.3 System as a whole

The entire system consists of two major components: A set of Kasadakas that are geographically spread out, and the Wifi-donkey that is mounted on a vehicle. In simple terms, the Wifi-donkey needs to host an offline network, and the Kasadakas need to be able to detect that network and connect to it when it is in range. When a connection is established, the Wifi-donkey can exchange data over HTTP with the connected Kasadaka. This overview is depicted in figure 2. This section describes the internal workings of the two parts of the system and how this was achieved. The scripts and configuration files for both the Wifi-donkey and the Kasadaka are available on github at https://github.com/

[14]http://www.swi-prolog.org/web/ClioPatria/Overview.html

`fahad105/masterproject`.

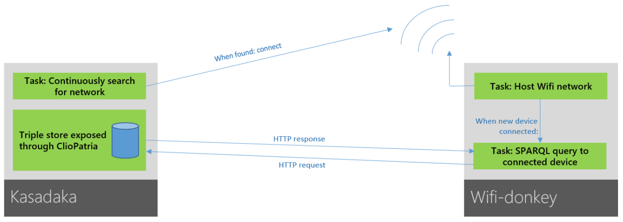

When found: connect
Task: Continuously search for network
Triple store exposed through ClioPatria
Kasadaka
HTTP response
HTTP request
Task: Host Wifi network
When new device connected:
Task: SPARQL query to connected device
Wifi-donkey

Figure 2: System overview

4.3.1 The Wifi-donkey

In a nutshell, the Wifi-donkey has two responsibilities. These are 1) hosting a wireless network and 2) running a SPARQL query on any Kasadaka that connects to its wireless network.

Hosting a network that other devices can connect to is one of the functionalities that the PirateBox offers out of the box. Because we made use of the Raspberry Pi 3, which has built-in WiFi capabilities, we did not require a separate WiFi dongle. The on-board WiFi card is used by the PirateBox to host a network. This is done through several linux packages, specifically *hostapd*[15] and *dnsmasq*[16]. The *hostapd* package is what turns the Raspberry Pi into an access point. This essentially makes it send out a detectable wifi network. Additionally, the *dnsmasq* package is what handles DHCP leases for devices that connect to the network and provides a local DNS server. Adding a basic webserver along with some simple webpages yields the default PirateBox installation; a device that sends out a network and shows a webpage to connected devices through which they can upload and download files. This functionality was already working neatly out of the box, so no significant changes were made to this.

Running SPARQL queries that target the ClioPatria webserver of any Kasadaka the moment it connects to the Wifi-donkey's network was part of both technical challenges; it involves the exchange of semantic data as well as automating the process of data exchange. The first step was to be able to detect the moment when a device connects to the network. the *dnsmasq* package offers several basic dhcp functionalities, but also allows custom scripts to be executed when certain dhcp events occur. These events could either be the addition of a new dhcp lease, re-activation of an existing dhcp lease, or deletion of a dhcp lease. By default, leases are valid for 2 hours. Changing the lease duration does not have any impact on the way the Wifi-donkey works. It might be a useful parameter if the Wifi-donkey's network would have many devices connected at the same time, but that is not the case for this project.

The code snippet below shows the relevant part of the dnsmasq configuration file. The entire configuration file is available on Github[17].

```
dhcp-script=/opt/piratebox/rpi/bin/runquery.sh
```

The snippet shows that a custom script, called *runquery.sh*, is triggered on dhcp events. This custom script is triggered with several additional arguments. These arguments[18] are: the operation, which is either "add", "del", or "old", the MAC address of the connected device, the assigned IP address of the device, and the hostname of the device if there is one. As an example, the script could be triggered by *dnsmasq* like this:

```
/opt/piratebox/rpi/bin/runquery.sh add
b8:27:eb:5d:7e:74 192.168.77.18 kasadaka
```

The *runquery.sh* script is shown in the code below.

```
1  op="${1:-op}"
2  mac="${2:-mac}"
3  ip="${3:-ip}"
4  hostname="${4}"
5
6  echo "${op}␣|␣${mac}␣|␣${ip}␣|␣${hostname}" >> /opt/piratebox/tmp/dhcpchanges.txt
7  if [[ ${op} = "add" || ${op} = "old" ]] ; then
8    QUERY=$(cat /home/pi/kasadakanet/query.txt)
9    mkdir -m 777 -p /home/pi/kasadakanet/results/${mac}
10   /usr/bin/curl ${ip}:3020/sparql/ --data-urlencode "query=${QUERY}" > /home/pi/kasadakanet/results/${mac}/queryresult.rdf
11   echo "ran␣query␣on␣${ip}" >> /opt/piratebox/tmp/dhcpchanges.txt
12 fi
```

As shown in lines 4 through 7, the arguments the script was called with are now available as variables in the script itself. This script first checks whether the operation was "add" or "old", because if that is the case then either a new device has connected to the network or a device has reconnected to the network. In this scenario we want to do something; we want to run a SPARQL query on the connected device's ClioPatria server using the *curl* package and store the result of the query in a folder somewhere on the Wifi-donkey. By default, all Kasadaka's have a ClioPatria server running on port 3020. As shown previously, we already have access to the IP address, the MAC address and the hostname within this script. That means we know exactly where the ClioPatria server of the connected Kasadaka is running. Going by the previous example, this would be at 192.168.77.18:3020. Knowing this, we can construct a curl command that sends a query to that IP address and port, and store the result in an RDF file. The query itself is stored in a separate text file, and contains a simple SPARQL query. Line 11 of the

[15]`https://wireless.wiki.kernel.org/en/users/documentation/hostapd`
[16]`https://wiki.debian.org/HowTo/dnsmasq`
[17]`https://github.com/fahad105/masterproject/blob/master/donkeybox/dnsmasq_default.conf`
[18]`https://www.systutorials.com/docs/linux/man/8-dnsmasq/`

script retrieves the contents of this text file and puts it in a variable. To ensure that the query result is stored correctly, we need to make sure that the folder where the result will be stored exists and read, write and execute permissions are given for this folder. That is what line 12 does; it creates a directory (only if it does not exist yet) with the MAC address of the connected device as its name, which ensures that the created directory is unique. Read write and execute permissions are then set for this folder. With the IP address, the query, and the folder for the query result ready, we can execute the curl command to run the query and store the result in a file called queryresult.rdf in the appropriate folder. This is done on line 13 of the script. The command also makes use of *–data-urlencode* as the contents of the query's text file is likely to contain spaces, tabs or line breaks. Obviously, if port 3020 at the targeted IP address is not running, which could be because the connected device is not a Kasadaka or the ClioPatria server is not running at that port for some reason, then the queryresult.rdf file will be empty.

With this, the Wifi-donkey is able to accumulate data from different Kasadaka's and store that data in RDF files. The Wifi-donkey hosts its own ClioPatria server as well, a ClioPatria server with some dummy market data from the RadioMarche application was used during development and testing. This server is set up to start on boot, as configured in the *radiomarche_initd_script* script[19] and the radiomarche folder[20] on Github. This script should be placed in /etc/init.d on the Wifi-donkey. Since the Wifi-donkey is running its own ClioPatria server at port 8910, accumulated RDF files can easily be imported. This can be done either through the ClioPatria web interface or through *curl*. Assuming the current directory contains a file called queryresult.rdf, the queryresult.rdf file can be imported with the following curl command:

```
curl -v -F data=@queryresult.rdf -F
    baseURI="http://192.168.77.1:8910/demo
    /result/" http://192.168.77.1:8910/
    demo/servlets/uploadData
```

With this, the Wifi-donkey is able to exchange semantic data through SPARQL queries with the Kasadakas and import the accumulated data into its own ClioPatria webserver, thus solving the first technical challenge. Additionally, the second technical challenge is also partly solved as the Wifi-donkey automatically detects when a device connects to its network and sends out the SPARQL queries. the remaining part of the second challenge is solved on the side of the Kasadaka.

4.3.2 *The Kasadaka*

The remaining part of automating the process of data exchange was solved by altering the Kasadaka. Specifically, the Kasadaka was made to detect when the Wifi-donkey is in range and connect to its network. This was accomplished with a script[21] that runs continuously in the background on the Kasadaka.

The script, called networkmonitor.sh, is triggered on boot on the Kasadaka. this was done by adding the following the line on the *rc.local* configuration file of the Kasadaka. The entire rc.local file can be found on Github[22].

```
/etc/network/networkmonitor.sh &
```

The networkmonitor.sh script is shown in the code below.

```
1  sleep 30
2  while true; do
3    if iwconfig wlan0 | grep -q "KasadakaNet
       " ; then
4      echo "connected␣to␣KasadakaNet"
5      sleep 1
6    else
7      echo "scanning␣for␣KasadanaNet..."
8      if iwlist wlan0 scanning | grep -q "
         KasadakaNet" ; then
9        echo "Found␣KasadakaNet,␣
           reconnecting..."
10       service networking restart
11       sleep 5
12     else
13       sleep 1
14     fi
15   fi
16 done
```

The script starts with a 30 second sleep timer, this is necessary because the script is executed on boot. The 30 second timer is set in order to give other startup services, specifically the networking interfaces of the Raspberry Pi, time to fully get up and running. The script then continues with a while loop that runs on infinitely. Inside the loop, the script first checks whether the Kasadaka is currently connected to the Wifi-donkey's network, which is called KasadakaNet. This happens every second. If at some point the device is not connected to KasadakaNet, then the script checks whether KasadakaNet is in range. If it is not in range, the loop will continue with the next iteration after 1 second. If it is in range, the Kasadaka will restart its networking interfaces, as seen in line 10 of the script. This will connect the Kasadaka to the network once the networking interfaces are back up, as this is configured in the *wpa_supplicant.conf*[23] file of the Kasadaka. This restart of the networking service takes a few seconds, which is why the script waits for 5 seconds before continuing with the loop. If a connection was established succesfully, in the next iteration of the loop the script will detect that the device is connected to the network. The sleep timers inside the while loop are kept as short as possible in order to ensure that a connection with the network is established as fast as possible. However, as this entire script

[19]https://github.com/fahad105/masterproject/blob/master/donkeybox/radiomarche_initd_script
[20]https://github.com/fahad105/masterproject/tree/master/donkeybox/radiomarche
[21]https://github.com/fahad105/masterproject/blob/master/kasadaka/networkmonitor.sh
[22]https://github.com/fahad105/masterproject/blob/master/kasadaka/rc.local
[23]https://wiki.archlinux.org/index.php/WPA_supplicant

runs continuously in the background, shorter sleep timers result in more frequent iterations of the loop, which likely increases resource usage of this script on the entire device.

During development, reconnection speeds were measured using this script. Consider the following scenario; a Kasadaka and the Wifi-donkey are right next to each other (clearly in range), the Kasadaka is fully booted up and powered on, while the Wifi-donkey is completely turned off. This means that there is no connection between the two devices as the Wifi-donkey is not hosting a network. From the moment the Wifi-donkey is turned on, roughly 30 to 35 seconds are required to establish a connection. This also includes startup time on the Wifi-donkey's side, because the device starts from a complete shutdown. This startup time was estimated at roughly 20 seconds. This was done with line 9 of the networkmonitor script, a simple echo statement which indicates that the network has been detected. This means that from the moment the Wifi-donkey is powered on, roughly 20 seconds were measured until the script echo's that the network was detected. The remaining 10 to 15 seconds were thus measured from line 9 of the script until the next iteration of the while loop (which then echo's that the device is connected to KasadakaNet). Therefore, in an ideal situation where both devices are already powered on but not in range of each other, it will take roughly between 10 and 15 seconds to establish a connection. This does not, however, take into consideration the time it takes for a SPARQL query to be executed and a query result to be passed back. These measurements are key in deciding whether a pass-by will be successful or not, as during a pass-by there is only a limited amount of time in which the two devices are in range of each other. For example, if it takes 15 seconds to establish a connection and, say, 5 seconds to exchange a certain amount of data through SPARQL during a pass-by, we would need the two devices to be in range of each other for at least 20 seconds in order to have a successful pass-by.

4.3.3 Reversing the roles of client and server

Looking at the system from a client-server model perspective, the Wifi-donkey fulfills the role of a client and the Kasadaka would be a server. This is because all transfer of data is initiated by the Wifi-donkey. Specifically this means that the Wifi-donkey, the client, sends all HTTP requests to the servers, which would be the ClioPatria webserver hosted on each individual Kasadaka. Because the Wifi-donkey also runs its own ClioPatria webserver, the opposite is technically also possible; in this case the Kasadaka would be initiating all HTTP requests, and the target of those requests would be the ClioPatria server of the Wifi-donkey. This reverse functionality was also experimented with during this project, but was not of no further use for the system as a whole. It is therefore disabled in this version of the system, but this functionality might be of use for future projects. The rest of this section briefly explains the reverse functionality, e.g. how the Kasadaka can be used as a client that sends requests to the ClioPatria webserver of the Wifi-donkey.

As explained previously, the Kasadaka's only responsibility within the system is to detect the Wifi-donkey's network and connect to it. This is done in the *networkmonitor.sh* script, specifically on line 10 of that script. Once the networking interface is back up, a custom script can be executed. This is done by adding the following line to the *interfaces* file[24] of the Kasadaka:

```
post-up /bin/bash /etc/network/mypostupscript
```

The code snippet below shows a simple example of how the Kasadaka can initiate different curl commands that target the IP address of the Wifi-donkey.

```
if [ "$IFACE" = wlan0 ]; then
  /usr/bin/curl -F "upfile=@/home/pi/kasadakanet/file.txt" 192.168.77.1:8080

    QUERY=$(cat /home/pi/kasadakanet/query.txt)
  /usr/bin/curl 192.168.77.1:8910/demo/sparql/ --data-urlencode "query=${QUERY}" > /home/pi/kasadakanet/queryresult.rdf
fi
```

We first need to check which networking interface triggered the script, because a device can have multiple networking interfaces. Since we use the *wlan0* interface to connect to the Wifi-donkey's wireless network, we can simple do a check on the $IFACE variable, which is passed to the script when it is called. After the check, we can use the Wifi-donkey's local IP address, which is always 192.168.77.1, to make HTTP requests with *curl*. We can, for example, use port 8910 on the Wifi-donkey's IP address to target the ClioPatria webserver just as the Wifi-donkey does with any Kasadaka's ClioPatria webserver. Furthermore, port 8080 can be used to upload files to the Wifi-donkey. This port is configured out of the box by the PirateBox installation. Such a file upload functionality could be useful when there is a need to transfer audio files to the Wifi-donkey, for example.

5. EVALUATION

The Wifi-donkey was used in an experimental setting to evaluate its performance. This experiment was conducted in Amsterdam. A true evaluation would require an experiment in a rural Sub-Saharan setting, however this was not possible within the scope of this research project. Section 5.1 details the experiment's setup, execution, and what variables were measured. In section 5.2, the results of the experiments are presented and discussed. Finally, in section 6 we discuss the overall performance and limitations of the system and to what extent the evaluation results can be expected to transfer to a Sub-Saharan context.

5.1 Experimental setup

The goal of the experiments is to find out to what extent, if at all, a "pass-by" is possible without losing the ability to exchange data. Statistics regarding the system's reliability and data transfer speed were measured during the experiments. Specifically, the success rate of pass-bys, and the amount of time in seconds it takes to transfer different amounts of data

[24]`https://github.com/fahad105/masterproject/blob/master/kasadaka/interfaces`

were measured. Two different experiments were done; one where pass-bys were simulated as closely as possible, and another experiment that specifically measures the manner in which the amount of time scales along with the amount of triples that need to be transferred.

5.1.1 The pass-by experiment

The basic setup of the pass-by experiment consisted of one Kasadaka powered by a 5v powerbank, and one Wifi-donkey powered by a regular 5.2v power supply. As it turns out, the Wifi-donkey does not function properly when powered with a 5v powerbank. While it did send out the KasadakaNet network, connecting to the network and accessing the ClioPatria webserver was not possible. One possible explanation for this is the powerbank not delivering enough electricity to power the Wifi-donkey, which heavily uses the on-board Wifi-module. It is likely that 5v simply is not enough to power a Raspberry Pi 3 in this case, as the official power supply delivers 5.1v as well[25]. The Kasadaka, however, did function properly with the 5v powerbank that was available. As a result, the decision was made to move around with the powerbank-powered Kasadaka during the experiments instead of the Wifi-donkey. Assuming that the communication range of both Raspberry Pi's is the same, this should not have any significant impact on the evaluation results. As

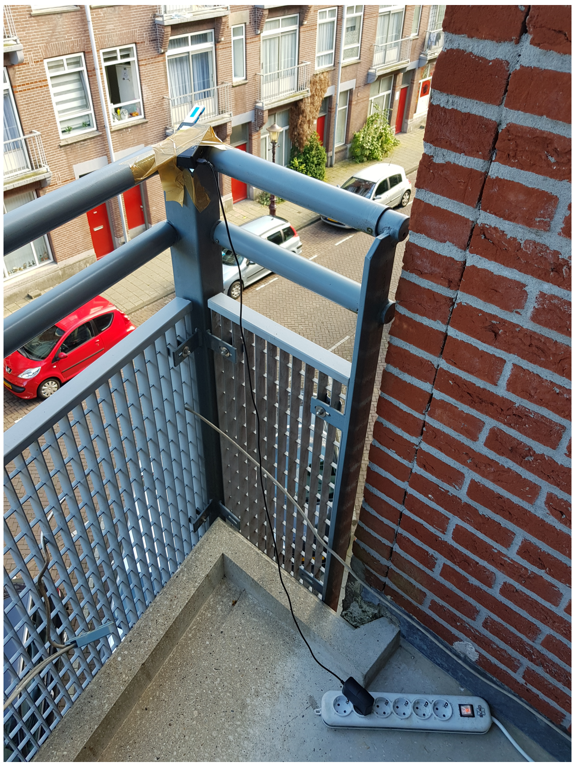


Figure 3: The Wifi-donkey taped to the railing of a balcony

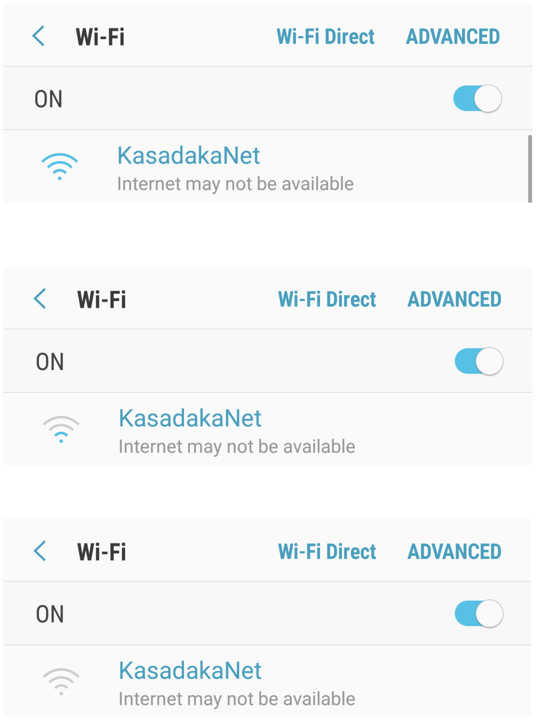
seen in figure 3, the Wifi-donkey—powered by a 5.2v power supply—was taped to a balcony, as far out as possible in order to prevent the brick wall from limiting its range.

[25]`https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md`

With the Wifi-donkey in place and a Kasadaka ready to move, the experiment could be executed. It consisted of two parts: measuring the range of the Wifi-donkey's network, and repeatedly performing a "pass-by" that starts and ends outside the network's range. These steps are explained in detail in the next two paragraphs.

5.1.1.1 Measuring communication range.

Measuring the communication range of the Wifi-donkey was done step by step - literally. By default, if a Kasadaka is not connected through an ethernet cable and the KasadakaNet network is not available, it will host its own network called KasaDaka Wifi Foroba-blon. This network is used for other applications and projects, and of no further relevance to this project. However, the fact that it sends out its own network initially and stops doing that the moment KasadakaNet is in range can be used to estimate the communication range of the two devices. This was done by initially determining an estimate of the outer edge of the network, specifically by using a smartphone's WiFi scan capabilities to detect the network's signal strength. Figure 4 shows three screenshots that were taken increasingly further away from the Wifi-donkey, resulting in a decreasing signal strength. When the signal strength is at its lowest but the network can still be found, we know that we are near the edge of the network's range. This gives a rough indication of where the two devices are in range of each other for the first time during a pass-by.



ON

KasadakaNet
Internet may not be available

ON

KasadakaNet
Internet may not be available

ON

KasadakaNet
Internet may not be available

Figure 4: Screenshots of KasadakaNet's signal strength on a smartphone

The next step was to take several additional steps further out until it is certain that the network is not in range. At this point, powering up the Kasadaka resulted in it hosting the KasaDaka Wifi Foroba-blon network because a) an ethernet connection was not available, and b) the KasadakaNet network was not in range. As explained in section 4.2.2, it can take up to 15 seconds to establish a connection between a Kasadaka and the Wifi-donkey, assuming both devices are powered on, which was indeed the case. Knowing this, the communication range of the Wifi-donkey could thus be estimated more precisely by repeatedly taking one step in the direction of the Wifi-donkey and waiting at least 15 seconds. If after waiting at least 15 seconds the KasaDaka Wifi Foroba-blon network could still be detected on for example a smartphone, then the devices are still outside of each other's range. Repeating this process until the two devices are just barely in range will result in the KasaDaka Wifi Foroba-blon network being turned off and a connection being established between the Kasadaka and the Wifi-donkey. This way, a single point very close to the edge of the network could be pinpointed. This entire process was done two times, each in opposite directions of the Wifi-donkey. Figure 5 shows both spots on a map as blue squared, and indeed roughly in the middle of the distance between the two spots was where the Wifi-donkey was mounted on a balcony. Using Google Maps, the distance between the two blue squares is measured at roughly 120 meters. The red circle estimates the entire range of KasadakaNet's range, obviously assuming the absence of signal interference or other physical obstructions that could limit the network's range.

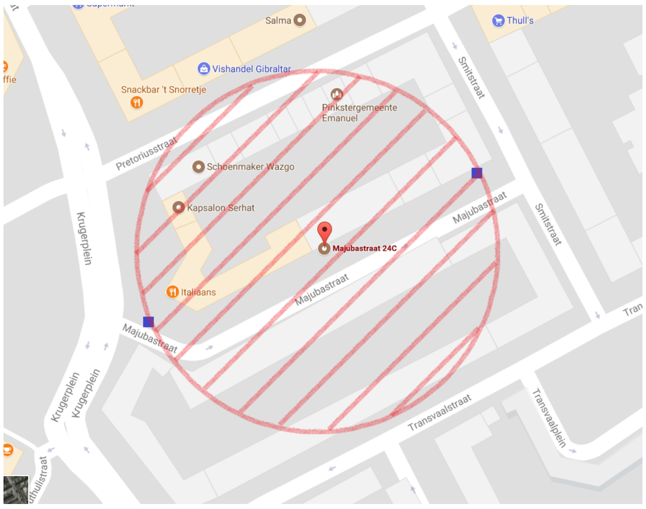


Figure 5: Estimate of the Wifi-donkey's communication range based on two measured spots

5.1.1.2 Measuring performance of the system.

The actual experiment essentially consisted of traversing a path that starts outside of the network's range and continues all the way through it, as seen in figure 6. The time a Kasadaka spends within the network's range during a "pass-by" is an important factor in deciding whether data is retrieved successfully or not. Additionally, how much data needs to be transferred, which obviously varies per query, is also an important factor. A worst case scenario, for example, would consist of a vehicle that travels in and out of the network's range very fast while the Wifi-donkey is configured with a query that would result in a large amount of triples. As such, we define two independent variables for this experiment.

The first independent variable is the speed at which the person carrying the Kasadaka travels. Measuring travel speed precisely is a rather difficult task without appropriate equipment, therefore the distinction in speed was made by either walking at a decent pace, or using a bicycle. In total, the experiment was repeated 12 times, half of which were with a bicycle and the other half without it.

The amount of data that needs to be transferred is the second independent variable. This obviously depends on the query, as that decides how big the result will be. The following query was used during the experiment:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-
    rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/
    rdf-schema#>
CONSTRUCT {
  ?sub ?pred ?obj .
}
WHERE{
    ?sub ?pred ?obj .
}
LIMIT 30
}
```

This simply retrieves the first 30 triples contained in triple store of the targeted ClioPatria server. In order to vary the amount of data that is transferred during a pass-by, the "LIMIT" parameter of the query was changed to a number larger than the total amount of triples on the server. This means that the two variations of this query retrieved either 30 triples, or all triples on the server, which was 322 triples.

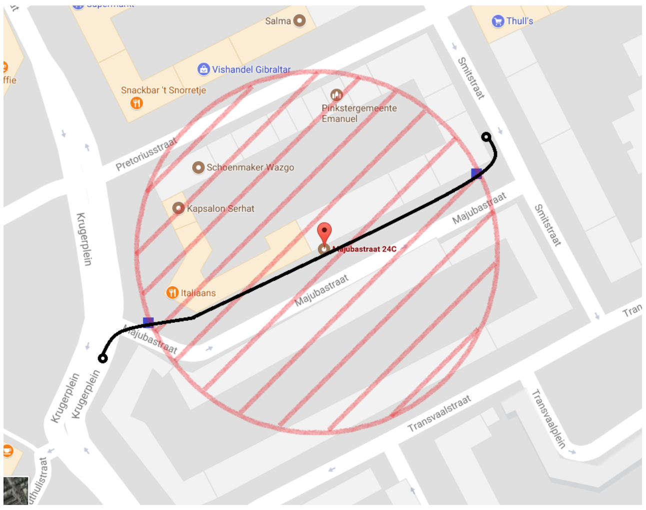


Figure 6: Path traversed during a pass-by

Essentially there were 4 possible combinations of the independent variables; walking + 30 triples, walking + 322 triples, biking + 30 triples, and biking + 322 triples. Each of these combinations was executed 3 times, thus totalling to 12 pass-bys. This includes an initial trial run of the experiment, consisting of 2 pass-bys done on foot.

As mentioned previously, results of the *curl* command on the Wifi-donkey are stored in a file called queryresult.rdf per unique device, based on its mac address. For the purpose of the experiment, the *curl* command that executes HTTP request containing the query was slightly altered. Specifically, the ”-w” argument was configured to append some statistics about the entire command at the end of the result file. Two things were added: the total time in seconds it took to complete the entire command, and the total amount of bytes that were downloaded which equals the size of the queryresult.rdf file in bytes.

By default, any pre-existing file called queryresult.rdf will be overwritten. In order to log the results of each pass-by, the queryresult.rdf file needed to be copied and stored in a separate folder where it wouldn't be overwritten. For efficiency, the experiment was executed with assistance of a second person. One person repeatedly performed a pass-by, and the other ssh-ed into the Wifi-donkey and copied the queryresult.rdf files per pass-by, effectively logging the experiment's results. Throughout the experiment, both persons would communicate and coordinate over the phone.

5.1.2 Transferring data on larger scales

A separate experiment was done to evaluate the performance of the system when larger amounts of data need to be transferred. The setup consisted of one Kasadaka and one Wifi-donkey, both powered by 5.2v power supplies, in close proximity of each other. The experiment did not involve physically moving either of the devices to simulate a pass-by, instead both devices were powered on and in close proximity of each other throughout the entire experiment. HTTP requests were then fired off manually with *curl* after ssh-ing into the Wifi-donkey. During the pass-by experiment, a total of 322 triples were loaded on the Kasadaka, thus limiting the data transferred during pass-bys to 322 triples. In order to measure the request times for larger amounts of data, more triples were required to be loaded on the Kasadaka. For the purpose of this experiment, a dataset of 475000 triples was imported on the Kasadaka's triplestore through ClioPatria, effectively allowing larger queries to be performed.

During the experiment, a total of 35 HTTP requests were sent. These requests were split into 7 categories, each category with a larger query size than the previous. The number of triples requested per category scaled as such: 30, 300, 1k, 5k, 10k, 50k, 100k. Five requests were sent per category, which totals to 35 requests. The request time in seconds and the size in bytes were measured per request. Per category, the average of the time in seconds was computed. Averaging the size in bytes makes little sense, because the size in bytes yields exactly the same value per request within a category, as each category corresponds to a single query, and that same query will always return the exact same result.

5.2 Results

The results of the various experiments are presented in this section. Table 1 contains the results of the pilot pass-by experiment, and table 2 contains the results of the full pass-by experiment. As Table 1 only shows results of 2 pass-bys, the data there is presented in a complete format, whereas Table 2 only presents the success rate, i.e whether a pass-by resulted in a successful transfer of data. All raw data from the pass-by experiment can be found in Appendix A.

Table 1: Pilot pass-by experiment results

Method	Triples	Bytes	Request time	Result
Walking	30	1960	2.859 sec	Success
Walking	322	16718	18.223 sec	Failure

Table 2: Full pass-by experiment results: success rate

Query size / Method	30 triples	322 triples
Walking	2/2	2/2
Biking	3/3	2/3

The results of the scaling experiment are presented in table 3 and plotted in figure 7. Raw data from the scaling experiment can be found in Appendix B.

Table 3: Scaling experiment results: Average time in seconds per category

# of triples transferred	Average time in seconds
30	0,3422
300	0,2582
1k	0,5374
5k	1,7864
10k	3,1558
50k	14,3342
100k	29,4238

As seen in table 1, it took 2.859 seconds to transfer 30 triples, or 1960 bytes. As for the larger query of 322 triples, the request lasted 18.223 seconds and ultimately failed, likely due to a loss of connection between the two devices.

Comparing tables 4 and 5 in the Appendix with table 1, a significant difference in request times can be noticed. Performance was significantly better during the full pass-by experiment than during the pilot experiment. The source of this discrepancy is not clear, however it should be noted that the pilot pass-by experiment and the full pass-by experiment were not conducted on the same day. The Raspberry Pi that was secured to the balcony with tape was also re-taped in between the experiments. We suspect that the discrepancy in the request time results may have been caused by the amount of tape used during the pilot experiment. Pieces of tape might have directly covered the chip antenna of the Raspberry Pi which can hinder signals, though this is mostly speculation at this point. Another potential clarification for the discrepancy could be external signal interference in the street. It is of course quite possible that, depending on the time of day or day of week, there may be more active wireless devices in the vicinity. This might have caused the connection between the two Raspberry Pi's to slow down or even interrupted.

In general, the pass-by experiment's results show that the system as a whole seems to work fairly reliably; only one

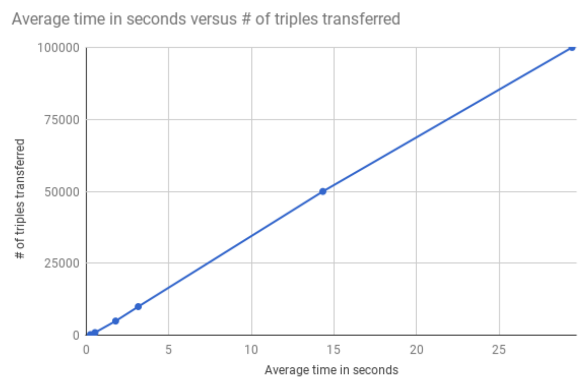


Figure 7: Scaling experiment results: time scales linearly with amount of triples transferred up to at least 100k triples

out of 10 pass-bys failed. As explained previously, once the Kasadaka connects to they Wifi-donkey's network, a SPARQL query is executed. The result of this query is then stored in a file on the Wifi-donkey. In the case of the failed pass-by, the output file was not created, which likely means that, for unknown reasons, the two devices failed to connect during the pass-by.

Another interesting aspect of the pass-by experiment's results is that there is no significant or consistent difference in request times between the 30 triple queries and the 322 triple queries, as some 322 triple queries had shorter request times than some 30 triple queries. This seemed very counter-intuitive at first. The scaling experiment, however, provided an explanation for this. As seen in 7, the request time clearly scales linearly. It seems that it takes some time for transfer speeds to ramp up and reach a maximum; the larger the query, the more data that needs to be transferred, and the more data that needs to be transferred, the more time there is for transfer speeds to ramp up and reach the maximum. Smaller queries are simply completed before transfer speeds reach their maximum, which results in an irregular, non-linear scaling in the request times as shown in the pass-by experiment's results.

6. DISCUSSION

The evaluation shows that the system performs well enough for pass-by communication to work, albeit on travel speeds lower than those of a car or bus. In those situations, where a car or bus passes by without stopping, the success rate is likely to be significantly lower or the system may not work at all. When querying smaller numbers of triples, the vast majority of the total time needed for a pass-by comes from the 10-15 seconds needed to establish a connection between the two devices. To put this into perspective, transferring 1000 triples takes only half a second, which is a result from an experiment conducted in a simulated setting where signal interference could not be measured. This means that if a vehicle passes through the 120 meter range in less time that the time it takes to establish a connection and exchange data, the pass-by will fail. This is a clear point where the Wifi-donkey could be improved; bringing down the time it takes to establish a connection would greatly improve chances of successful pass-bys in cars or other fast vehicles.

Signal interference is another factor that might have impacted the experiment's results, it could easily have caused the system to not perform optimally. In real situations in rural parts of the world there is likely to be less signal interference, which can result in higher chances of having a successful pass-by.

The system functions as intended, but there are some limitations as well. The system as a whole has a very high latency for example, albeit by design. It simple takes a while for the Wifi-donkey to complete a full circle in an area, obviously depending on the distance that needs to be traveled. Another limitation is the fact that some human input is still required; as a preparatory step a person with a technical background needs to load a text file containing a SPARQL query onto the Wifi-donkey. This is quite critical, as the system can not do much without a query to execute. Additionally, once the circle is complete and a number of RDF files from different Kasadaka's have been gathered, human input is again required to analyze the acquired data.

As mentioned before, the system was only tested in Amsterdam, and not under rural conditions. Despite this, we know that the system works fairly reliably, at least within the conditions it was tested in. The evaluation shows that the system works in an urban area and can reliably transfer up to 300 triples during a pass-by done on a bicycle. Assuming that there is less signal interference in rural areas and both devices are supplied with enough power during a pass-by, we can deduce that the system should work under rural conditions when transferring a similar amount of triples during a pass-by at similar travel speeds. At faster travel speeds however, pass-bys might fail. In order to accommodate for these scenarios, a possible solution is to instruct vehicles that have a Wifi-donkey mounted on them to briefly stop in range of the Kasadakas. This should theoretically ensure a successful pass-by as there is more time to establish a connection and exchange data. Specific instructions for the drivers of the vehicles would need to be discussed with them directly. Furthermore, NGO's may be willing to assist the project by allowing Wifi-donkey's to be mounted on their vehicles. With these adjustments, this system can be a considered a viable approach for ICT4D projects that need to exchange data across devices under rural conditions.

Finally, there are still some known bugs within the system. The list below details these issues.

- Queries seem to execute more than once. Each time a query is executed, a line of text is written to a log file. We noticed that sometimes multiples lines were written during what should have been a single pass-by. Each query's output is however always written to the same file name, so every time the query executes, any pre-existing file with the same file name will be overwritten. The script that executes the queries is likely triggered multiple times. It should trigger only once when a connection between two devices is established, however, it seems to occur multiple times. This bug does not render the system unusable, but we did notice that this occurs.

- Two pass-bys failed during the experiments. Once in the pilot experiment, and once in the full experiment. The failed pass-by in the pilot experiment resulted in an empty output file being created. This likely means that the connection was lost after about 18 seconds, possibly due to the the portable device moving outside of the range of the stationary device. The other failed pass-by, however, yielded no output file at all. This means that for some unknown reason the two devices did not manage to establish a connection at all. This could be a severe bug, however more information regarding the cause of this is needed.

7. CONCLUSION

The research presented in this paper shows that pass-by M2M communication is a viable approach in extending knowledge sharing systems. Looking at the communication method as part of the larger Kasadaka platform, having the ability to exchange data between Kasadakas opens up new possibilities and use cases. Both existing applications as well as new applications for future use cases of the Kasadaka platform can adopt the ability to share semantic data among devices. Aggregating and analyzing data originating from geographically spread origins is also a real possibility.

Finally, we can conclude that M2M communication can be of great use in ICT4D projects, especially those designed to focus on local knowledge sharing. Our approach, which creates a sneakernet using local Wifi networks, is simple, yet effective and uses low-cost hardware and open source software to enable M2M communication between Wifi-enabled devices. We have shown that this approach works in an urban setting. Provided that devices that need to communicate with each other have a steady power supply when in range of each other, there is no reason this approach would not work under rural conditions as well. It should not be too complex or expensive to implement similar sneakernet-type M2M communication methods in other ICT4D projects, as long as the devices that need to communicate with each other have Wifi capabilities.

References

Baart, A. (2016). Creating a flexible voice service framework for low-resource hardware: extending the kasadaka.

Cha, I., Shah, Y., Schmidt, A. U., Leicher, A., & Meyerstein, M. V. (2009). Trust in m2m communication. *IEEE Vehicular Technology Magazine*, *4*(3).

Charlaganov, M., Cudré-Mauroux, P., Dinu, C., Guéret, C., Grund, M., & Macicas, T. (2013). The entity registry system: Implementing 5-star linked data without the web. *arXiv preprint arXiv:1308.3357*.

de Boer, V., Bon, A., WaiShiang, C., & Gyan, N. B. (2016). *Proceedings of the 4th workshop on downscaling the semantic web (downscale2016).* figshare. Retrieved from `https://dx.doi.org/10.6084/m9.figshare.3827052.v1` (Retrieved: Jan 28, 2017)

de Boer, V., De Leenheer, P., Bon, A., Gyan, N. B., van Aart, C., Guéret, C., … Akkermans, H. (2012). Radiomarché: distributed voice-and web-interfaced market information systems under rural conditions. In *International conference on advanced information systems engineering* (pp. 518–532).

de Boer, V., Gyan, N. B., Bon, A., Tuyp, W., van Aart, C., & Akkermans, H. (2015). A dialogue with linked data: Voice-based access to market data in the sahel. *Semantic Web*, *6*(1), 23–33.

Gray, J., Chong, W., Barclay, T., Szalay, A., & Vandenberg, J. (2002). Terascale sneakernet: Using inexpensive disks for backup, archiving, and data exchange. *arXiv preprint cs/0208011*.

Guéret, C., Schlobach, S., De Boer, V., Bon, A., & Akkermans, H. (2011). Is data sharing the privilege of a few? bringing linked data to those without the web. *Proceedings of ISWC2011-" Outrageous ideas" track, Best paper award*, 1–4.

Hasson, A. A. (2010). The last inch of the last mile challenge. In *Proceedings of the 5th acm workshop on challenged networks* (pp. 1–4).

Hasson, A. A., Fletcher, R., & Pentland, A. (2003). Daknet: A road to universal broadband connectivity. In *Wireless internet un ict conference case study* (pp. 1–9).

Heeks, R. (2008). Ict4d 2.0: The next phase of applying ict for international development. *Computer*, *41*(6).

Lô, G. (2016). The power of knowledge sharing: innovative icts for the rural poor in the sahel.

Lô, G., & Blankendaal, R. (2016). Digivet: a knowledgebased veterinary system for rural farmers in northghana.

Pentland, A., Fletcher, R., & Hasson, A. (2004). Daknet: Rethinking connectivity in developing nations. *Computer*, *37*(1), 78–83.

Valkering, O., de Boer, V., Lô, G., Blankendaal, R., & Schlobach, S. (2016). The semantic web in an sms. In *Knowledge engineering and knowledge management: 20th international conference, ekaw 2016, bologna, italy, november 19-23, 2016, proceedings 20* (pp. 697–712).

APPENDIX

A. PASS-BY EXPERIMENT RAW DATA

Table 4: Results from pass-bys on foot

# of triples	Success/failure	Request time	Size in bytes
30	success	0.531 sec	1960
30	success	1.048 sec	1960
322	success	0.184 sec	16718
322	success	2.219 sec	16718

Table 5: Results from pass-bys on bike

# of triples	Success/failure	Request time	Size in bytes
30	success	0.663 sec	16718
30	success	0.538 sec	16718
30	success	0.736 sec	16718
322	success	0.591 sec	16718
322	failure	-	-
322	success	0.449 sec	16718

B. SCALING EXPERIMENT RAW DATA

Table 6: Category 1: transferring 30 triples

Time in seconds	Size in bytes
0.458	1960
0.500	1960
0.098	1960
0.401	1960
0.254	1960

Table 7: Category 2: transferring 300 triples

Time in seconds	Size in bytes
0.319	15596
0.299	15596
0.293	15596
0.281	15596
0.099	15596

Table 8: Category 3: transferring 1k triples

Time in seconds	Size in bytes
0.512	120947
0.528	120947
0.488	120947
0.476	120947
0.683	120947

Table 9: Category 4: transferring 5k triples

Time in seconds	Size in bytes
1.599	728446
1.572	728446
2.027	728446
1.876	728446
1.858	728446

Table 10: Category 5: transferring 10k triples

Time in seconds	Size in bytes
3.004	1489426
3.310	1489426
2.955	1489426
3.332	1489426
3.178	1489426

Table 11: Category 6: transferring 50k triples

Time in seconds	Size in bytes
14.708	7644038
14.151	7644038
14.145	7644038
14.381	7644038
14.286	7644038

Table 12: Category 7: transferring 100k triples

Time in seconds	Size in bytes
30.121	15393807
31.266	15393807
28.374	15393807
28.767	15393807
28.591	15393807