École Polytechnique Fédérale de Lausanne

Smallworld: Gossiping with Raspberry Pis Locally

by Romain Küenzi (continued by Guilhem Niot)

Bachelor Project Report

Professor Anne-Marie Kermarrec
Project Advisor

Dr. Erick Lavoie
Project Supervisor

EPFL IC IINFCOM SaCS
BC Building, Office BC 347
CH-1015 Lausanne
Switzerland

July 7, 2021

Contents

Chapter 1

Introduction

In high-income countries, the ubiquity of Internet access encourages users to rely on online centralized services when direct connectivity between devices would be sufficient: when exchanging files or collaborating on shared documents for example. This increases the energy usage of the network and cloud infrastructure; it also introduces privacy compromises, as many of the online services monetize user behavior and interests to cover their operating costs.

In low-income countries, Internet connectivity may be unaffordable to many so direct connectivity may become the only affordable option. And in situations of humanitarian crises, the required telecommunication networks may simply no longer be available so direct connectivity may become the only possible option.

While mobile devices used in all the previous situations most often have the required physical communication capabilities for direct connectivity, through protocols such as Wi-Fi [2] and Bluetooth [6], different mobile vendors implement different higher-level protocols that are not interoperable. As an example, Apple devices use their own proprietary protocol, AWDL [4], whereas Android devices use the Wi-Fi Direct standard [1].

The objective of this project is to tackle afore mentioned issues: simplify device-to-device communication in an affordable way. Connection should require minimal interaction from the users and, once it is established, allow automatic sharing of data between them using peer-to-peer protocols and applications.

With that goal in mind, we will first have to make a few technology choices to make our solution as available as possible and ideally compatible with all existing devices. To achieve this compatibility, the first choice is an hardware platform: the Raspberry Pi. Then we need to choose standards communications protocols. At the physical layer we will rely on Wi-Fi, which enables the usage of the the Internet protocol suite in higher layers. We will then demonstrate in which scenario the resulting device can be used. The actual solution will be described in two steps, first listing all the required software tools, then explaining the implementation of a simple script that

enables to create the communication networks. Finally we will demonstrate the effective usage of the device with an existing peer-to-peer application called Secure-Scuttlebutt [5].

Chapter 2

Technology Choices

This first chapter describes and justify the technology choices that were made, showing their advantages and explaining why other solutions were dismissed.

2.1 Hardware Platform

Raspberry Pis are single-board computers developed by the *Raspberry Pi Foundation* whom have an objective to make digital technologies accessible and affordable. Multiple models are available, with the most standard coming at a price of 35$. The particular model used in this project is the *Raspberry Pi Zero W*. It is a smaller and cheaper (10$) board and therefore has a bit less computing power, but it still includes Wi-Fi and Bluetooth capabilities. With this the affordability requirement is met[1].

Choosing the RPi also came with other various advantages in the research and implementation of a solution. First, finding ressources is fairly simple as the RPi has a big community of users and a complete online documentation. The board used in this project has everything we need for multiple connectivity options, namely micro-USB, Wi-Fi and Bluetooth modules. Other boards also have USB and Ethernet ports. As mentioned earlier, the RPi is a single-board computer and it usually runs on the *Raspberry Pi OS*, an operating system based on Debian. That gives access to the standard tools you have on any computer running a Linux distribution.

Lastly, having a dedicated device for direct connectivity might sound unnecessary when the devices you own already have the capabilities to communicate together, but as already pointed out in the introduction, devices tend to implement different incompatible protocols. As shown in Figure 2.1, if you want to allow connectivity between any devices, you have to solve a complex

[1] All the Raspberry Pi products and their reference prices can be found on the Raspberry Pi website at `https://www.raspberrypi.org/`

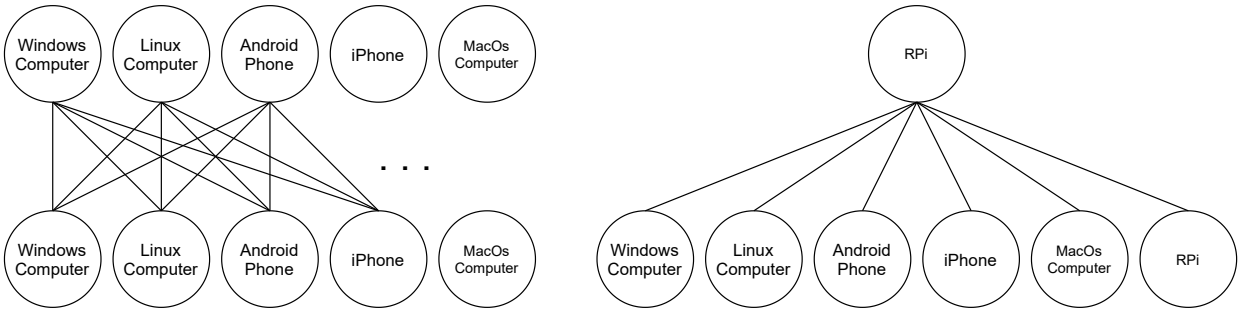


Figure 2.1: Many-to-Many vs. One-to-Many compatibility graph

many-to-many compatibility graph, whereas if you use a dedicated device you only have to solve one-to-many.

2.2 Communication Protocol: Wi-Fi and IP

The next step is to choose which communication protocol to use. Always aiming for simplicity and maximum compatibility, the best solution is to use what already exists and is widely implemented: Wi-Fi and the *Internet Protocol* (IP). These choices come with a lot of advantages.

For end users, Wi-Fi is a technology they are used to, so it should not be hard to understand how the device work. The wide usage of Wi-Fi and IP also made it easier to find ressources, tools and already compatible applications.

Wi-Fi network are used to create *Local Area Networks* (LAN). IP is used for giving addresses on the Internet but it also does it in LAN. This address is used to communicate with other devices on the Internet, but it works the same in LAN, if the IP of another device is known, you can start sending it messages. On the Internet, the *Network Address Translation* (NAT) performed by home routers makes it complicated to locate a specific device. But on a LAN there exists multiple ways to do this, for example *Avahi*[2] lets you announce to everyone that you are available on the network to deliver specific services (and therefore communicate). SSB, the protocol we will use in Chapter 5 to verify the functioning of the device, also implements a way to discover all peers on a LAN by broadcasting UDP messages.

To facilitate Wi-Fi setup on our device, we used the *Wi-Fi Protected Setup* (WPS) standard [2]. It enables devices to create secure connections that require less configuration steps from the users. In our case, we use the push-button method, which sets up a Wi-Fi connection between the host and the client when a button is pressed on both devices.

[2]Avahi facilitates service discovery on LAN : `https://avahi.org/`

2.3 Discarded Option: Bluetooth

The other option available for wireless communication is Bluetooth but the idea of using it was quickly dismissed. It is not appropriate for the following reasons:

- It has a low transfer rate (< 2 Mbits). We want to be able to transmit any kind of data and it includes potentially large files. Wi-Fi as available on the RPi Zero W can transmit up to 54 Mbit/s and other devices with more recent Wi-Fi chips can go above this limit.
- Bluetooth is designed to implement application-specific protocols in higher layers which means that, among other things, it does not work easily with IP networking.

2.4 Wi-Fi or Wi-Fi Direct

Initially, one of the objectives was to implement a solution that used Wi-Fi Direct. The main advantage would have been a simpler user experience. Wi-Fi Direct removes the need to choose which device hosts the network, as the protocol determines it automatically. The resulting device would have had a single use case: connect to another device. In fact WPA Supplicant, the tool used to manage Wi-Fi networks (described in more detail in Section 4.1), offers Wi-Fi Direct functionalitie. But even after some research we were not able to connect devices with it. In the end, loosing this option was not critical because, despite being broadly implemented by Android phones, it is not available on iOS devices. Linux PCs should be capable to use Wi-Fi Direct through WPA Supplicant but they do not always have a user interface to use it. In any case, Wi-Fi Direct could always be implemented later as an optional feature.

2.5 Ethernet-over-USB

The Raspberry Pi Zero W, along with *Raspberry Pi 4*, also offers the possibility to implement Ethernet-over-USB[3]. Ethernet, like Wi-Fi, works well with the Internet Protocol. As it can be implemented with just a few configuration changes on the RPi, this option is also set up. However the compatibility is not as good as Wi-Fi. It was only verified to work with a Linux laptop. We also tested it with a Windows desktop and an Android phone and it did not work. Although it might work with some more configuration on these devices, it does not fill our criteria of simplicity. This functionality has not been tested with other devices.

[3] Other models of Raspberry Pis do not offer this feature

Chapter 3

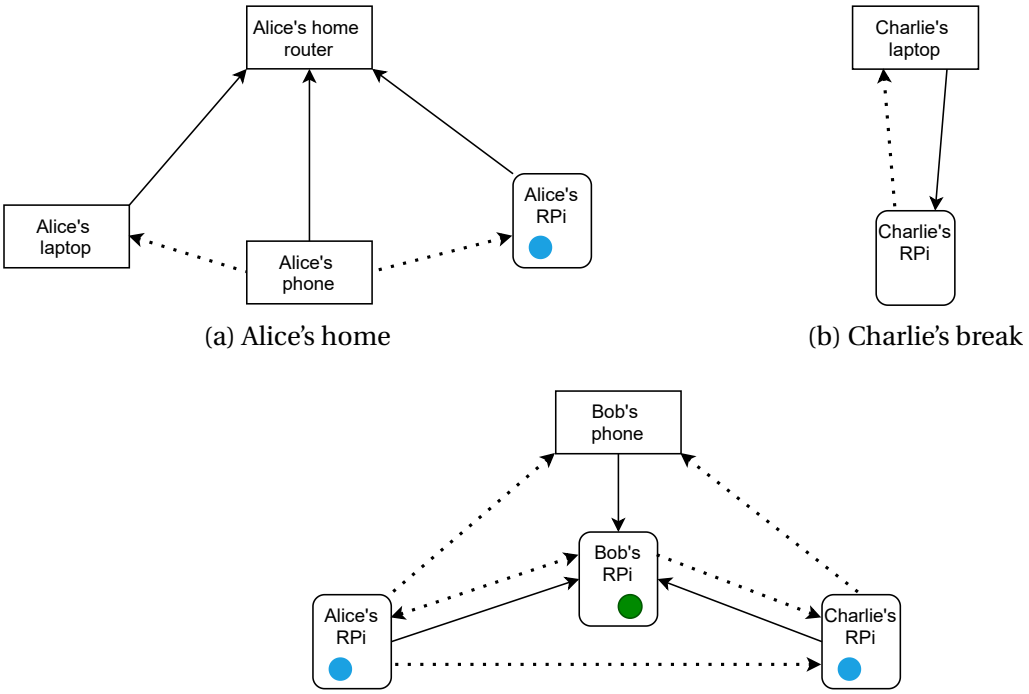
User Experience

This chapter presents a scenario that highlights all the uses of the device that we envisioned. The scenario assumes that each participant uses an application that replicates the data they want to share to all their friends, and also their own devices. We will explain in more detail in Chapter 5 how this scenario can be implemented with the SSB protocol [3]. For now, we present the scenario from the perspective of end-users, without reference to SSB-specific implementation details. On the figures, arrows indicate that two devices are connected to each other and the device it points to hosts the network. Doted arrows indicate when data flows from one device to another. For RPis, blue dots indicate that it is connected to a Wi-Fi network and green dots indicate that it is hosting its own Wi-Fi network.

Alice likes to take a lot of photos on her phone and share them with friends. At home she always connects all her devices to the Wi-Fi network. As Figure 3.1a shows, it is quite convenient because her photos get replicated to both her laptop and her RPi. With her laptop she can edit them, create albums, ..., and with her RPi she can easily share them with friends when they meet. This show the first usage of the device where it is simply connected to an existing network and synchronizes with other devices on the network.

When she goes out to meet friends she takes her RPi along to be able to share her photos. Today, she meets two friends, Bob and Charlie, and during the discussion, she talks about the last pictures she took. Bob and Charlie really want to see the photos so they all bring out their devices. Bob chooses to enable the Access Point on his device so he can connect his phone easily and take a look at Alice's photos right away. The other two connect their devices to Bob's network. As shown in Figure 3.1c every device receives the pictures including Bob's phone, so he can take a look at the pictures with Charlie. This illustrates two other uses of the device: they can connect to each other, and other devices can be connected to them in the same way these other devices connect to any Wi-Fi network. Once connected, all the devices are able to communicate.

During their meeting, Charlie and Alice also received some data from Bob. After the other two leave, Charlie still has some work to do so he takes out his laptop. A bit later, while taking a



(a) Alice's home

(b) Charlie's break

(c) Friends meetup

Figure 3.1

break, he recalls that Bob likes to write blog posts about his favorite turtle pet, so he brings out his RPi and USB cable and plug his device to his laptop. As shown in Figure 3.1b, he receives all the data he got earlier from Alice and Bob on his laptop. As expected, Bob wrote a new blog post, so Charlie spends the rest of his break reading funny stories about Bob's turtles. This show the last connection option using a USB cable. The laptop connects to the RPi network in the same way it connects to a network using an Ethernet cable, and the two devices can communicate.

Chapter 4

Design and Implementation of Connectivity

4.1 Software Tools

The first step toward the construction of a solution is to find the appropriate tools to manage the technologies mentioned in Chapter 2. Here is a list of the tools used to manage connectivity on the Raspberry Pi:

WPA Supplicant[1] is a service (or "daemon" program) that runs in the background to manage wireless connections. It can be controlled through a C library (to develop your own frontend) but also comes with a text-based frontend (wpa_cli) and a GUI (wpa_gui). It is made to support most Wi-Fi related standards.

Hostapd[2] is a service to create and manage Wi-Fi access points and authentication. It also comes with a text-based frontend (hostapd_cli).

DHCP is the protocol used to assign addresses on an IP network. To connect to existing IP networks, a DHCP client is necessary so the RPi comes with a default DHCP client called *dhcpcd*[3]. In some situations, we want the RPi to control a network, therefore it needs to assign itself and other devices on the network an IP address. *Dnsmasq*[4] provides multiple network infrastructure tools for small networks, including an easy to configure DHCP server that we use to assign IP addresses on the networks we create with the RPi.

[1] `https://w1.fi/wpa_supplicant/`
[2] `https://w1.fi/hostapd/`
[3] `https://roy.marples.name/projects/dhcpcd/`
[4] `https://thekelleys.org.uk/dnsmasq/doc.html`

```
apt-get update
apt-get install hostapd dnsmasq
# Unmask hostapd with systemd
systemctl unmask hostapd
```

Configuration 4.1: Hostapd and dnsmasq installation

Systemd[5] is the service manager for Debian. It starts background scripts on boot and manage them during execution: we use it to run WPA Supplicant, the Smallworld script we developed (Section 4.2), and the hostapd service.

Two of these tools, hostapd and dnsmasq, are not preinstalled on the RPi. They have to be downloaded and installed with *aptitude*, the Debian package manager, with the commands in Configuration 4.1.

4.2 The Smallworld script

Smallworld is a python script that establishes and monitors wireless connections between devices in close proximity. It runs as a background service that starts when the RPi boot. The script, along with example configuration files, is available at `https://github.com/rkuenzi-epfl/smallworld`.

4.2.1 Design

The current design comprises a Raspberry Pi Zero W with two push buttons, two 220 Ohm resistors and two LEDs connected to its GPIOs. Figure 4.1 illustrates the electrical schema of the connected components. The buttons control the device and the LEDs give feedback on its current connection status. The AP button is connected to GPIO 13, and the LED giving feedback for the AP is connected to GPIO 19. The WPS button is connected to GPIO 6. The LED on GPIO 26 gives feedback on the connection status of the device in both host and client mode.

Figure 4.2 illustrates the complete behavior of the daemon. When the RPi starts, the *Access Point* (AP) is disabled. The device eventually connects to known Wi-Fi networks listed in the `wpa_noAP.conf` file if any is available. When the user presses the WPS push-button, the RPi tries to connect to an existing Wi-Fi network. The *Connection State* LED blinks until it finds an available Wi-Fi to connect to then stays on if the connection is successful. If it does not find a Wi-Fi waiting for a WPS connection, it stops searching and times out after two minutes. When pressing on the AP button, the device starts its own Wi-Fi network (and turn on the AP LED). Any Wi-Fi device can connect to this AP the same way they connect to a home network as long as the passphrase is

[5] `https://systemd.io/`

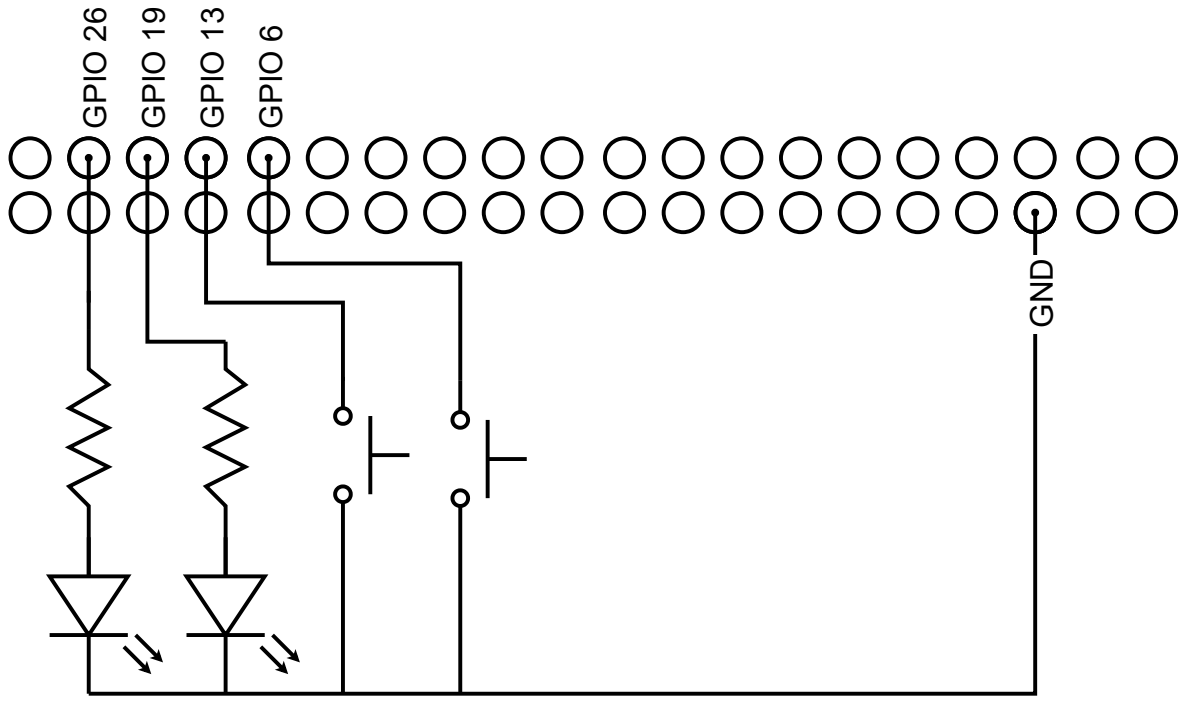


Figure 4.1: Electrical schema of the final device

known. If the other device implements a WPS push-button, it can also be connected by starting a WPS search on both devices. When the AP is enabled the *Connection State* LED stays on as long as there is at least one device connected to the network, and always blinks during WPS discovery.

4.2.2 Configuration

An important part of the solution relies on specific usages and configurations of the tools mentioned in Section 4.1. The script sometimes relies on "default behaviors" resulting from these specific configurations, which make them all the more important. Configurations presented here are meant as examples and most of them can be modified to the user preferences.

First we configure the DHCP client and server[6]. Dnsmasq makes the server configuration quite easy. After its installation you add the following line `dhcp-range=wlan0,192.168.4.2,192.168.4.20,255.255.255.0,24h` to the dnsmasq configuration file. In order, the arguments define the followings: interface to distribute the addresses on, first address to serve, last address to serve, address mask and leasing time. A DHCP request happens whenever the RPi creates or connects to a network, so the DHCP client needs to be configured for both of these situations. Configuration 4.2 shows the configuration for the

[6]Files for both dhcpd and dnsmasq are found under the /etc folder and follow the <software_name>.conf convention for their configuration filenames

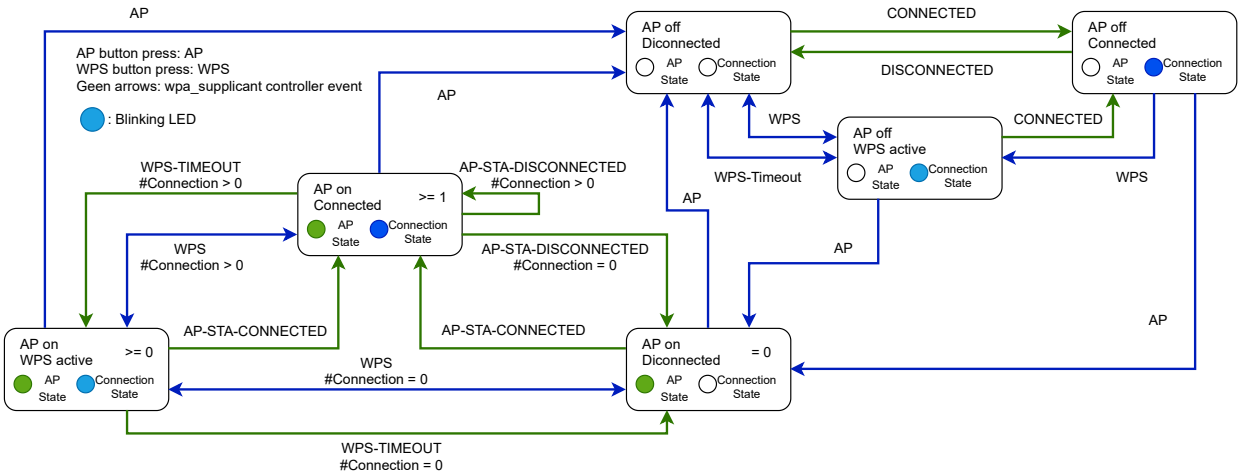


Figure 4.2: Smallworld state machine

```
profile ap_wlan0
static ip_address=192.168.4.1/24
static routers=192.168.4.1
nohook wpa_supplicant

interface wlan0
fallback ap_wlan0
```

Configuration 4.2: DHCP client configuration

DHCP client. The default configuration for `interface wlan0` runs a standard DHCP request, but if it fails it fallbacks to `profile ap_wlan0`. That happens when the RPi creates its own Wi-Fi network. The ap_wlan0 profile has to use the first address on the subnet defined by dnsmasq. This configuration should be appended to the dhcpcd configuration file.

Then we configure WPA Supplicant. The default WPA Supplicant configuration is in `/etc/wpa_supplicant/wpa_supplicant.conf`, but we do not modify it directly. Instead the small-world script uses two configuration files, `wpa_AP.conf` and `wpa_noAP.conf`, that overwrite the default WPA Supplicant configuration during execution. The base for both of them is shown in Configuration 4.3. If we want the RPi to automatically connect to specific Wi-Fi networks we can add configurations for those in `wpa_noAP.conf`. Configuration 4.4 gives a template to configure a standard WPA2 network that should work for most home networks[7]. However `wpa_AP.conf` has to stay empty for the Access Point to start properly[8].

The next step is to configure hostapd. Configuration 4.5 gives a complete example to setup a Wi-Fi network using WPA2 security, which is equivalent to what is used for home networks. Otherwise, only lines 11 to 14 are specific to the implementation, because they enable WPS push-

[7]We can append as much of these network configurations as we want to the file and WPA Supplicant will connect whenever one of them is available.

[8]We did not manage to run hostapd without running WPA Supplicant with this empty configuration file... but it might be possible.

```
1  ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
2  update_config=0
3  country=CH
```

Configuration 4.3: Base WPA Supplicant configuration file

```
network={
  ssid=<network_ssid>
  psk=<network_password>
}
```

Configuration 4.4: Home network configuration template

button. The only things to complete in the file are the `<network_ssid>` and `<network_password>`.

Finally, we write Configuration 4.6 in `/lib/systemd/system/smallworld.service`. It runs the smallworld script on boot and restart if it stops for any reason. For this service to work we have to put a few files in the right place: `wpa_AP.conf` and `wpa_noAP.conf` should be moved to the `/etc/smallworld/` folder and *smallworld*, as well as another script called *smallworldForwardEvent*, should be moved to `/usr/sbin/`. Once everything is in place, running `systemctl enable smallworld` will start the script at every boot.

4.2.3 Implementation

First, the *smallworld* script connects to the WPA Supplicant control interface to react when an event happens, such as connection/disconnection from a Wi-Fi network or any WPS event. The WPA Supplicant client offers a method to run a script, with the event informations as parameters, every time an event happens with the Wi-Fi controller (Configuration 4.7, line 8). This method is used to run the second script, *smallworldForwardEvent*. This script sends a UDP message on the local machine with the informations received from WPA Supplicant. The smallworld script runs a UDP server to receive theses messages and react accordingly.

The smallworld script also reacts to user interactions and gives feedback on its state. It uses the *GPIO Zero* package[9] to handle the RPi GPIO pins and consequently, buttons and LEDs.

When the user enables the AP by pressing the appropriate button the following sequence happens:

1. WPA Supplicant configuration file is overwritten with `wpa_AP.conf`.
2. The script forces WPA Supplicant to reread its configuration file (Configuration 4.7, line 10).

[9]Part of the default installation of the Raspberry Pi OS, documentation is available at `https://gpiozero.readthedocs.io/en/stable/`

```
country_code=CH
interface=wlan0
ssid=<network_ssid>
hw_mode=g
channel=7
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=<network_password>
wpa_psk_file=/etc/hostapd/hostapd.psk
wps_state=2
ap_setup_locked=1
config_methods=label display push_button keypad
ctrl_interface=/var/run/hostapd
ctrl_interface_group=netdev
eap_server=1
wps_pin_requests=/var/run/hostapd.pin-req
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Configuration 4.5: Complete Access Point configuration

```
[Unit]
Description=Smallworld script
After=multi-user.target

[Service]
Type=simple
StandardOutput=journal
RuntimeDirectory=smallworld
WorkingDirectory=/run/smallworld
ExecStart=/usr/sbin/smallworld
Restart=always

[Install]
WantedBy=multi-user.target
```

Configuration 4.6: Service file for smallworld

3. Hostapd service is turned on with the command written in Configuration 4.8.
4. Hostapd starts forwarding its control interface events (Configuration 4.7, line 17).
5. It turns the AP LED on.

When enabling the AP, reconfiguring WPA Supplicant has various useful side effects that depend on the configurations described in Subsection 4.2.2. It disconnects the Wi-Fi, which is necessary to allow hostapd to create its own Wi-Fi network. For the same reason, it is important that the configuration file used when reconfiguring is empty (`wpa_AP.conf`), otherwise the AP only starts if no available Wi-Fi network is found. The sequence when disabling the AP slightly differs:

1. Hostapd service is turned off with the command written in Configuration 4.8.
2. WPA Supplicant configuration file is overwritten with `wpa_noAP.conf`.
3. The script forces WPA Supplicant to reread its configuration file (Configuration 4.7, line 10).
4. It turns the AP LED off.

Stopping hostapd in step one also stops the message forwarding for hostapd events. Reconfiguring allows WPA Supplicant to connect to the networks added to `wpa_noAP.conf` if there is any.

Pressing the WPS button runs the appropriate command of line 12 or 19 from Configuration 4.7 and the connection LED starts blinking until the WPS search ends. If the WPS search is successful, an event is received through the UDP server and the connection LED stays on. If the WPS search times out, an event is received and the LED stays off. If the user presses the button again, the WPS search is canceled with one of line 14 or 21 from Configuration 4.7 and the LED stays off.

In more standard use cases, WPS only simplifies the configuration process to connect to a Wi-Fi network, therefore the resulting configuration is usually memorized. In this implementation, we decide that WPS connections only allow the devices to connect together once, so connections are only initiated from an explicit action of the user. This behavior depends on some of the configurations mentioned in Subsection 4.2.2. Line 2 of Configuration 4.3 prevents WPA Supplicant from writing to its own configuration file on execution so it cannot save a Wi-Fi configuration after a successful WPS connection. In Configuration 4.5 line 11 indicates where the shared keys are stored after a successful WPS connection. To prevent other devices from reconnecting with the same shared key, this file content is deleted after each successful connection.

```
# -i interface to run the command on, by default WPA Supplicant as access
#    to another interface so we need to specify it for wpa_cli commands
# -B runs the command in the background so we immediately
#   return to the python script
# -P path to create a PID file, used in the script to check if the command
#   was already runned once in case smallworld restart after a crash
# -a is the path to the script you want to run when there is an event
wpa_cli -iwlan0 -B -P<pid_path> -a<run_script>
# Force WPA Supplicant to reread its configuration file
wpa_cli -iwlan0 reconfigure
# Start WPS Search
wpa_cli -iwlan0 wps_pbc
# Stop WPS Search
wpa_cli -iwlan0 wps_cancel
# Same arguments as wpa_cli equivalent command but hostapd does not
# have access to any other interface
hostapd_cli -B -P<pid_path> -a<run_script>
# Start WPS Search
hostapd_cli wps_pbc
# Stop WPS Search
hostapd_cli wps_cancel
```

Configuration 4.7: WPA Supplicant and hostapd clients commands

```
# (Re)start hostapd service
systemctl restart hostapd
# Stop hostapd service
systemctl stop hostapd
```

Configuration 4.8: System commands to start and stop hostapd

```
profile usb0
ip_address=192.168.6.1/24
static routers=192.168.6.1
```

Configuration 4.9: DHCP client configuration, Ethernet-over-USB

4.3 Ethernet-over-USB

The Raspberry Pi Zero W has a micro-USB port that can use the *USB-On-The-Go* (USB-OTG) specification, which allows a USB device to act as both a host and a peripheral. It can be used to let the RPi act as any kind of USB device, e.g. audio, mass storage, mouse or keyboard, . . . , but in our case we are interested by the Ethernet option. Of the other RPi models, only the most recent *Raspberry Pi 4* can use one of its USB ports in OTG mode.

4.3.1 Enable Ethernet-over-USB

Thanks to the contributions of the Raspberry Pi community, enabling OTG mode and its Ethernet option on the RPi is done fairly easily. You have to change two files used by the RPi during boot. The first one is accessed under `/boot/config.txt`. Appending `dtoverlay=dwc2` at the end of the file (or after the `[all]` line) enables OTG-mode. Then editing the `/boot/cmdline.txt` file[10], by adding `modules-load=dwc2,g_ether` between `rootwait` and `quiet`, selects the Ethernet option of the OTG mode.

4.3.2 Use Ethernet-over-USB

Depending on the device used with the RPi, the previous configuration might be enough, because the hosting device automatically creates a network when you plug in. But our device should not depend on another device's configuration. So instead the RPi creates the network. To configure this we use a similar configuration as for the Wi-Fi network. Appending `dhcp-range=usb0,192.168.6.2,192.168.6.20,255.255.255.0,24h` to the dnsmasq configuration file and Configuration 4.9 to the dhcpcd configuration file.

[10] This file is said to be picky with its formatting and needs exactly one space between arguments and no newlines.

Chapter 5

P2P Application: Secure-Scuttlebutt

We now validate whether the implementation of Smallworld has the required functionalities to support peer-to-peer applications. To do so, we test it with an existing peer-to-peer protocol called *Secure-Scuttlebutt* (SSB) [3, 5]. SSB is a protocol designed to build social applications. It replicates data between users that choose to follow each other. By default, it also replicates the data of people followed by people you follow. We refer to this property as *follow of a follow*. This is already a transitive behavior in itself, but SSB also allows another transitive behavior, namely, when two users have an interest in the same person, either directly or through follow of a follow, they exchange the data from this person, even without following each other. We refer to this property as *common interest*. Both of these behaviors are illustrated in Figure 5.1.

5.1 Software

SSB applications come in two parts: a server and a client. The server implements the actual protocol and has the task to replicate the messages between users and update the local database. The client accesses the database content, in general it implements a GUI for the users to see their content in a nice way. There exist multiple implementations of both. On the RPi, to demonstrate

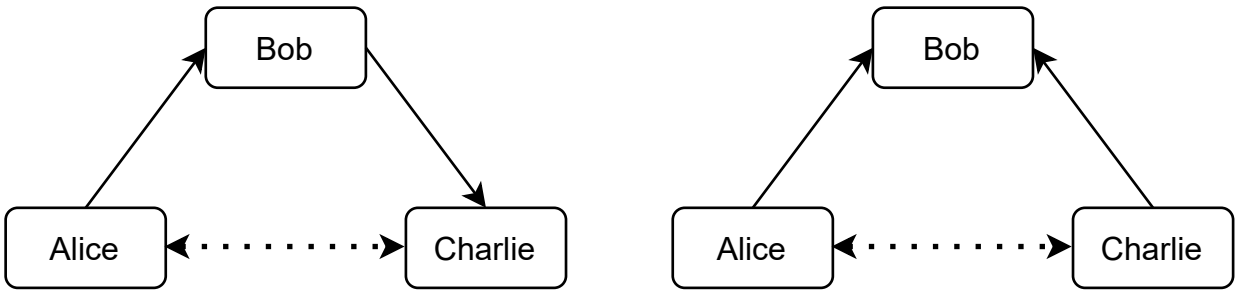


Figure 5.1: SSB transitive behaviors

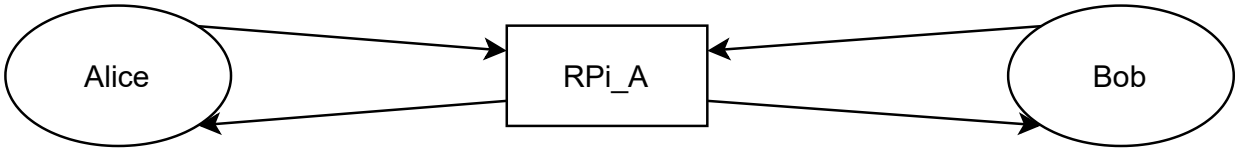


Figure 5.2: Simple follow graph using a single RPi

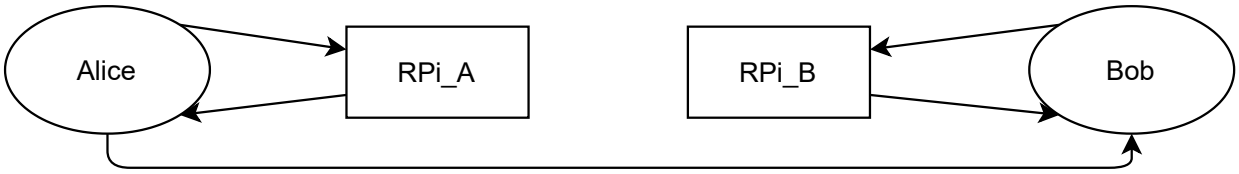


Figure 5.3: Follow graph taking advantage of the transitive properties of SSB

the device usage, we use sbb-server[1], the JavaScript reference implementation for a SSB server. We only use the server because a graphical interface is not necessary on the RPi. To properly verify the full functionality we also use the Oasis client[2] on other devices.

5.2 Scenario: Follow Graphs

To confirm the proper replication of users data and present the various possibilities that the device offers when used with the SSB protocol, we try out three scenarios with different initial follow graphs. To simplify the presentation of these follow graphs, we refer to a user by name, e.g. Alice, when we actually mean that user's main device, which has its own SSB identity. A follow from Alice to RPi_A therefore means that Alice's main device's SSB identity follows the SSB identity of RPi_A. This section presents the three scenarios. In each of these scenario, informations should flow from one end user to the other.

The first example in Figure 5.2 only uses one RPi. It represents a situation where only one person in a group has the device and this person shares the Wi-Fi password with trusted people. Of course these connections can also be achieved through WPS with devices that implement it. It can also represent the situation where a single person wants to synchronize some data between several devices.

The second example in Figure 5.3 shows a follow graph where Alice follows Bob, but their RPis do not know about each other. Exploiting the follow of a follow property of the SSB protocol, Alice's device receives data from Bob. But Bob can never receive any data from Alice.

The third and last example in Figure 5.4 is a more complex follow graph where Bob uses multiple other devices with his RPi. But because Alice and Bob RPis follow each other, Alice can

[1]Available at: `https://github.com/ssbc/ssb-server`
[2]Available at: `https://github.com/fraction/oasis`

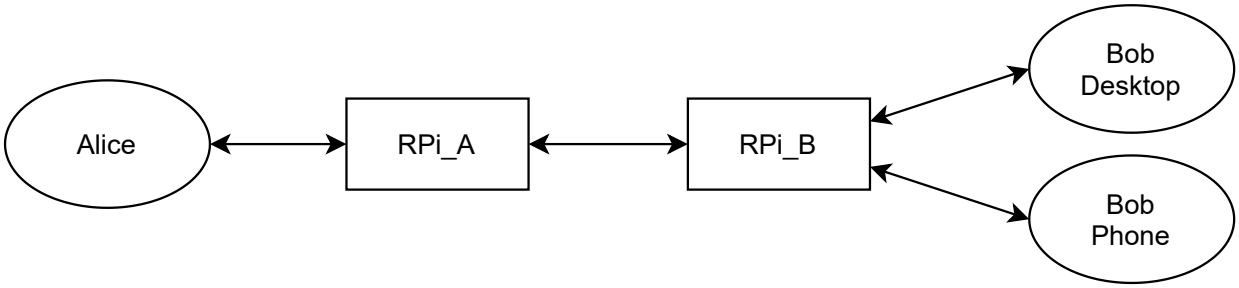


Figure 5.4: Follow graph with multiple devices abstracted by a single RPi

```
# This will install ssb-server in a ssb-server sub folder in the current folder:
git clone https://github.com/ssbc/ssb-server.git
cd ssb-server
npm install
```

Configuration 5.1: Install and run ssb-server on the RPi

follow Bob's RPi and then discover all of Bob's devices.

5.3 Configuration

Oasis and ssb-server use *Node.js*[3] (Node). On the RPi, Node is already installed so only ssb-server has to be installed. The commands in Configuration 5.1 install ssb-server. Other devices can download Node on the Node website[3], then install and run Oasis with the commands in Configuration 5.2.

Initially, the RPi and the other device do not follow each other. To have them follow each other, start both Oasis and the ssb-server. The ssb-server is started with line 2 of Configuration 5.3. The command on line 3 of Configuration 5.3 lets the RPi follow the other device. The `device_ID` is found under *profile* on Oasis interface. From Oasis, under *settings*, clicking on *start networking* displays the RPi ID after some time. Clicking on it lets the user follow the RPi.

Additionally, to get situations like Figure 5.4, we use a script written for the project called *ss-*

[3] `https://nodejs.org/`

```
# Install Oasis (globally)
npm -g install fraction/oasis
# Run Oasis
oasis
```

Configuration 5.2: Install and run Oasis on a device with Node.js

```
# From the ssb-server installation folder:
node bin.js start&
node bin.js publish --type contact --contact <device_ID> --following
```

Configuration 5.3: Run ssb-server and follow

```
[Unit]
Description=SSB server
After=multi-user.target

[Service]
Type=simple
User=pi
StandardOutput=journal
ExecStart=node <path-to-ssb-server>/bin.js start
Restart=always

[Install]
WantedBy=multi-user.target
```

Configuration 5.4: Service file for ssb-server

bAutoFollow[4]. This script makes the RPi follow any other device active on the same local network, using the fact that all ssb-server announce their identity on the local network by broadcasting UDP packets at regular intervals.

Finally, Configuration 5.4 and Configuration 5.5 can be copied to `/lib/systemd/system/` to have the ssb-server and the ssbAutoFollow script start on boot. The `systemctl enable <service>` command also needs to be run for both services.

[4]Also accessible on the smallworld repository: `https://github.com/rkuenzi-epfl/smallworld`

```
[Unit]
Description=SSB auto follow
After=multi-user.target

[Service]
Type=simple
User=pi
StandardOutput=journal
WorkingDirectory=<path-to-ssb-server>/
ExecStart=<path-to-ssb-server>/ssbAutoFollow
Restart=always

[Install]
WantedBy=multi-user.target
```

Configuration 5.5: Service file for ssbAutoFollow

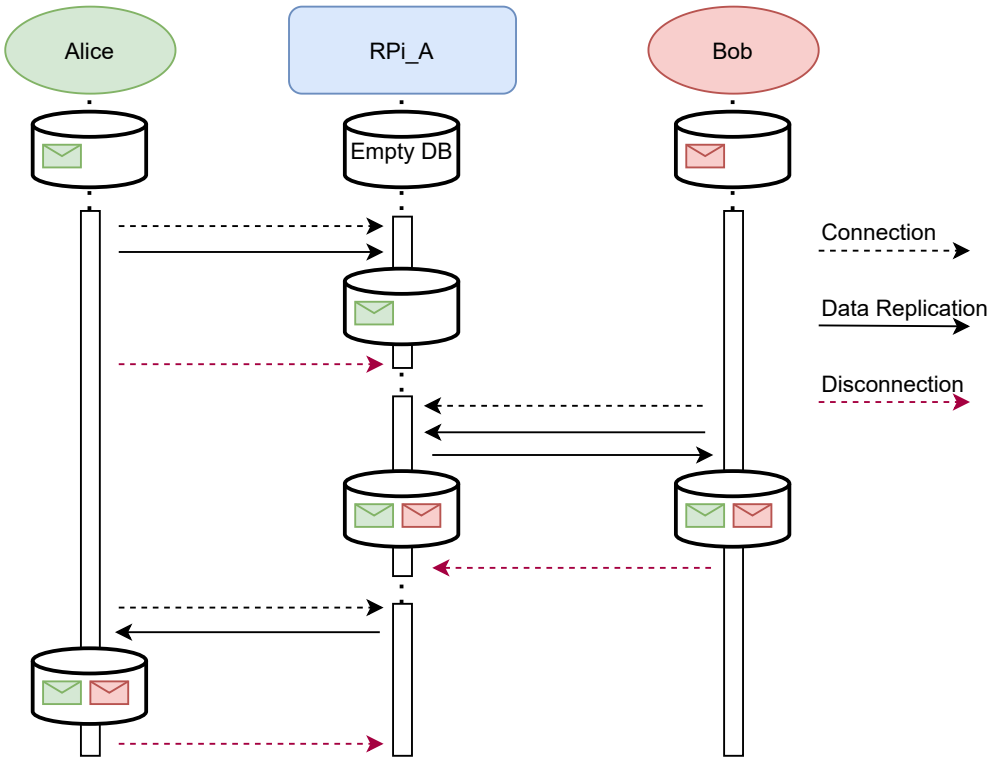


Figure 5.5: Data replication with a single RPi using the follow graph of Figure 5.2

5.4 Results

This section illustrates how data effectively replicate between user devices in each of the scenario presented in Section 5.2. For each scenario, we present one representative out of all possible sequence of connections that could lead to the same final state of data replication. In these diagrams, black dashed arrows indicate when a device connects to another, black arrows indicate when data get replicated from one device to another and red dashed arrows indicate when a device is disconnected from another.

Figure 5.5 demonstrates how the RPi gets the data from each user when they connect, using the follow graph of Figure 5.2. If the RPi has some data that one of the user does not possess yet it sends these data to the user. In this scenario Alice and Bob do not follow each other, but they still exchange their data using the follow of a follow property. Now they have some data from one another so they can follow each other. Note that the same result is achieved if they both connect to the RPi at the same time. The only difference may be the order of replications.

Figure 5.6 demonstrates how the transitive properties of SSB can help RPis replicate data of users who know each other, without the need for RPis to follow each other, which corresponds to the follow graph of Figure 5.3. In this scenario, Alice's RPi does not follow Bob's RPi, but it knows that Alice follow Bob. According to the follow of a follow property, it means that Alice's RPi as

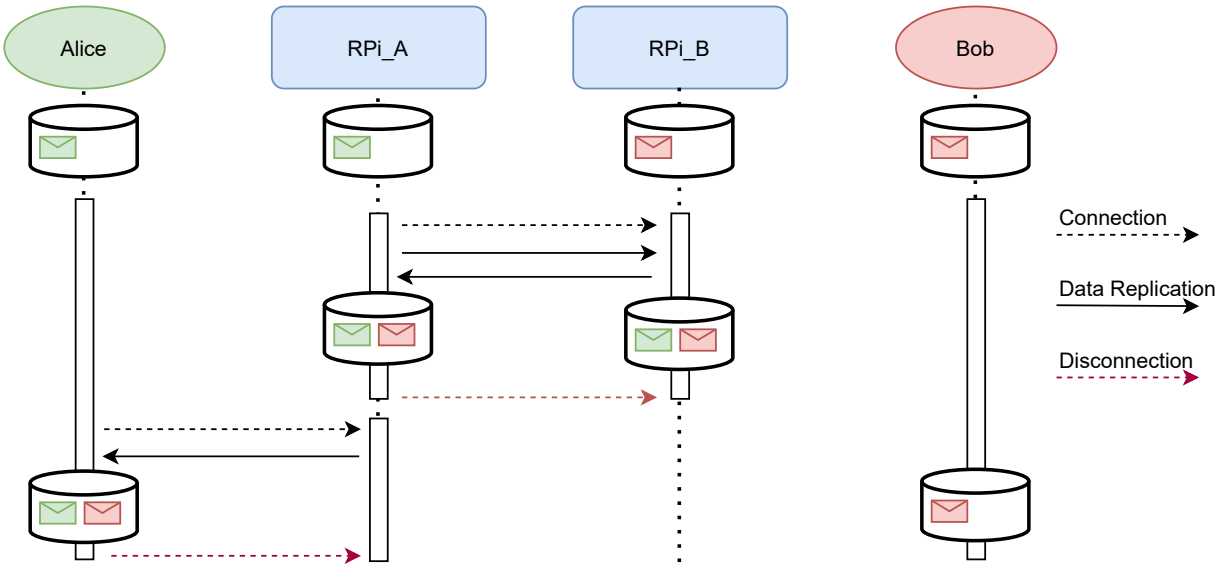


Figure 5.6: Transitive data replication using the follow graph of Figure 5.3

an interest in Bob's data. Because of the common interest they have for Bob's data, when Alice's RPi connects to Bob's RPi, Bob's data get replicated to Alice's RPi. Alice can then retrieve Bob's data when she connects to her RPi. On the other hand, Bob's RPi does not replicate Alice's data because it does not have any interest in it.

Figure 5.7 demonstrates how the automatic follow script can help discover many other peers. At first in this scenario, Alice does not know anything about Bob, as presented by the follow graph of Figure 5.4. Because the RPis automatically follow one another when they connect, Alice's RPi gets all the data from Bob's devices by exploiting follow of a follow. Then Alice can decide to follow Bob's RPi. Once again taking advantage of the same property, Alice gets all the data from Bob's devices. From there she can decide to follow one or all of Bob's devices. The resulting follow graph is presented in Figure 5.8.

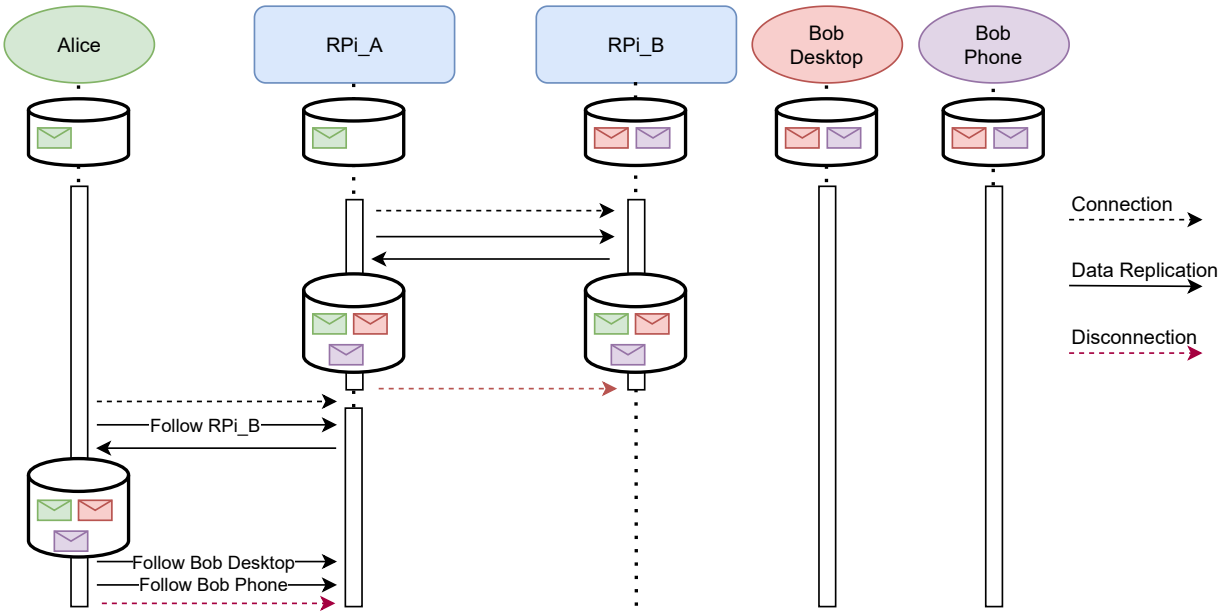


Figure 5.7: Data replication with RPi automatic follow using the follow graph of Figure 5.4

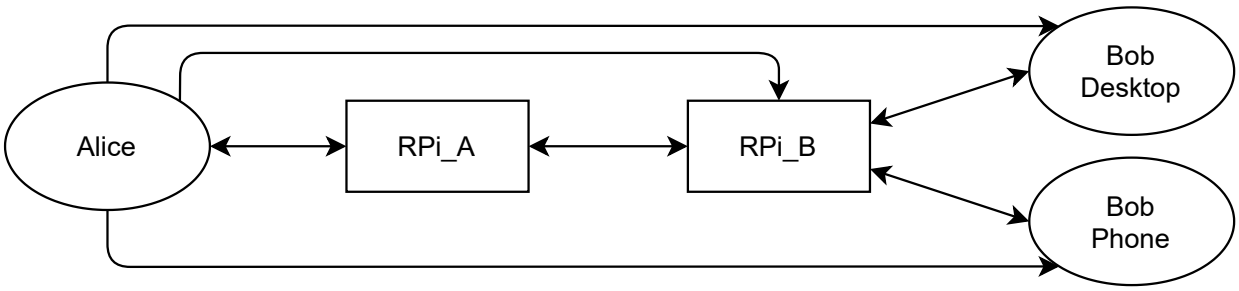


Figure 5.8: Resulting follow graph taking advantage of automatic follow on RPis

Chapter 6

Another application: Dat Protocol

Dat Protocol is a peer-to-peer protocol that allows sharing files in a distributed network.

It differs from BitTorrent in that contents can be updated by their author.

6.1 Installation

The installation of Dat is quite straightforward. The commands in 6.1 are used to install dat-cli.

It must be installed on both the emitter and the receiver devices.

6.2 Using dat

The dat-cli command utility allows to simply interact with Dat files.

It is possible to create a shared folder using *dat share* in the folder to share. This command will give a dat url starting with *dat://* that can be used to retrieve the files on other clients.

A set of useful commands:

```
wget -qO- https://raw.githubusercontent.com/datproject/dat/master/download.sh | bash

# or
npm -g dat-cli
```

Configuration 6.1: Installing dat-cli

- *dat share*, to share a folder (starts a daemon that waits for incoming connexions)
- *dat sync*, tries to retrieve newer version of the files, and share the files for incoming connexions
- *dat clone <url> <folder>*, retrieves files from a dat url
- *dat pull*, tries to retrieve a newer version of the files

6.2.1 Real time synchronization with dat-share

Dat does not directly provide a daemon to run in background and automatically synchronize folders for you.

A tool to do just that is *dat-store*. It provides a service that can be run in background, can be remotely controlled, and does all the work for you.

It should be installed on both the client and the Raspberry Pi using

```
npm -g dat-store
```

You should then configure a systemd service to run the store in background. The configuration on the user's device is

```
npm -g dat-store
```

You should then configure a systemd service to run the store in background. The configuration on the user's device is done according to figure 6.2. The configuration on the Raspberry Pi is achieved following figure 6.3.

Dat-store uses the concept of provider. Providers are instances of dat-store running either locally or remotely. The default one is your local instance, but you can also configure other providers, like the Raspberry Pi.
This notion is particularly useful for remote control of other providers.

Dat-store provides interesting commands that we will use in our demo:

- *dat-store add <url|path> [provider]*: Adds a folder or a dat url to the dat-store of the specified provider.
- *dat-store set-provider <url> [provider]*: Sets the url of the specified provider.
- *dat-store list [provider]*: Retrieves the list of available Dats in the specified provider.

```
# This will create the service file.
sudo cat << EOF | sudo tee /etc/systemd/system/dat-store.service > /dev/null
[Unit]
Description=Dat storage provider, keeps dats alive in the background.

[Service]
Type=simple
# Check that dat-store is present at this location
# If it's not, replace the path with its location
ExecStart=$(which dat-store) run-service
Restart=always

[Install]
WantedBy=multi-user.target
EOF

sudo chmod 644 /etc/systemd/system/dat-store.service

sudo systemctl daemon-reload
sudo systemctl enable dat-store
sudo systemctl start dat-store

sudo systemctl status dat-store
```

Configuration 6.2: Configure dat-store systemd service on the user device

```
# This will create the service file.
sudo cat << EOF | sudo tee /etc/systemd/system/dat-store.service > /dev/null
[Unit]
Description=Dat storage provider exposed to the internet (for the raspberry pi).

[Service]
Type=simple
# Check that dat-store is present at this location
# If it's not, replace the path with its location
ExecStart=$(which dat-store) run-service --expose-to-internet
Restart=always

[Install]
WantedBy=multi-user.target
EOF

sudo chmod 644 /etc/systemd/system/dat-store.service

sudo systemctl daemon-reload
sudo systemctl enable dat-store
sudo systemctl start dat-store

sudo systemctl status dat-store
```

Configuration 6.3: Configure dat-store systemd service on the raspberry pi

- *dat-store clone <path> <url> [provider]*: Clones <*url*> into a local folder.

We can then imagine the case where we have two clients A and B and one Raspberry Pi used as a permanent store to synchronize a folder from A to B.

First, A executes the following commands:

```
dat-store set-provider http://raspberry.local:3472 raspberry
mkdir mydat
dat-store add ./mydat
dat-store add ./mydat raspberry
```

Configuration 6.4: Commands executed by client A

Then, client B executes:

```
dat-store list raspberry
dat-store clone ./mydat <url obtained from the previous command>
```

Configuration 6.5: Commands executed by client A

Chapter 7

Conclusion

In this project, we designed and implemented Smallworld, an affordable device to simplify local peer-to-peer connectivity. The first step was to select which technology we would use. We settled on using Raspberry Pis for their low cost, and the multiple connectivity options they offered. This decision also allowed to focus on developping a single solution that is compatible with all existing devices, instead of trying to solve the incompatibility between them. Then we illustrated how users can use the device.

We implemented a solution for simple local peer-to-peer connectivity that works over Wi-Fi and Ethernet-over-USB in multiple steps. First, we selected relevant software tools, tested a number of possible configurations and explained those we retained. Second, we designed and implemented a user interface composed of two buttons and two LEDs in hardware, and a state machine that controls the behavior of the device in software which puts the device either in Wi-Fi host or client mode. The use of WPS greatly simplified the exchange of keys required to secure the connection. Third, we tested a wired connection to the device with Ethernet-over-USB, which for now only works with Linux laptops. Together, these steps enable any peer-to-peer applications that works with the Internet Protocol to communicate and replicate data with a Smallworld device.

The last step was to test that Smallworld could support a variety of replication scenarios of Secure Scuttlebutt (SSB), an existing peer-to-peer messaging application which transitively replicates data according to user-defined interests, as represented by follow graphs. We illustrated the data replication behavior for different follow graphs under representative connection scenarios. To simplify the initial configuration of those follow graphs, we have created a utility that automatically follows devices on the same local network, as happens when user or Smallworld devices are connected to one another.

7.1 Possible Improvements and Future Work

Our work builds a strong foundation upon which many improvements and extensions can be made. We list some of them in this section.

On the current device, the status of replication between devices is implicit. The device could provide an additional status LED, that would stay on while replication is in progress and turn off once replication is completed.

Because user may want more control over their device, a graphical user interface could also complement it. We can easily imagine two features that could be provided by such interface. The first one would let users have a better control over their follow graph, allowing them to choose who is followed and, as SSB also implements this feature, who is blocked by their Raspberry Pis. The second feature could give a more complete set of options to use the Wi-Fi controller. It could make it easier for users to add new Wi-Fi networks, and change their own network name and password. It could also allow to preserve the keys of trusted devices after a successful WPS connection.

Bibliography

[1] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. "Device-to-device communications with Wi-Fi Direct: overview and experimentation". In: *IEEE Wireless Communications* 20.3 (2013), pp. 96–104. DOI: 10.1109/MWC.2013.6549288.

[2] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". In: *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)* (2016), pp. 1–3534. DOI: 10.1109/IEEESTD.2016.7786995.

[3] Anne-Marie Kermarrec, Erick Lavoie, and Christian Tschudin. "Gossiping with Append-Only Logs in Secure-Scuttlebutt". In: *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good*. 2020, pp. 19–24. DOI: 10.1145/3428662.3428794.

[4] Milan Stute, David Kreitschmann, and Matthias Hollick. "One Billion Apples' Secret Sauce: Recipe for the Apple Wireless Direct Link Ad hoc Protocol". In: *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*. 2018, pp. 529–543. DOI: 10.1145/3241539.3241566.

[5] Dominic Tarr, Erick Lavoie, Aljoscha Meyer, and Christian Tschudin. "Secure scuttlebutt: An identity-centric protocol for subjective and decentralized applications". In: *Proceedings of the 6th ACM Conference on Information-Centric Networking*. 2019, pp. 1–11. DOI: 10.1145/3357150.3357396.

[6] Martin Woolley. *Bluetooth Core Specification Version 5.2 Feature Overview*. Accessed: June 08, 2021. 2020. URL: https://www.bluetooth.com/wp-content/uploads/2020/01/Bluetooth_5.2_Feature_Overview.pdf.