Radboud University Nijmegen

Faculty of Science

Profiling mobile users: user identification through analysis of LTE app traffic

Tracking users based on their installed apps

Thesis for MSc Computing Science, specialisation Cyber Security

Author:
David de Muinck Keizer

Supervisor:
Güneş Acar

Second reader:
Katharina Kohls

12 August 2023

Abstract

In this thesis, we examine the possibility of fingerprinting users, based on the apps that they have installed on their phones. We research whether a set of apps will generate specific patterns in LTE traffic, when users are not actively using their phone. We repeatedly alternate between different sets of apps and record their passive LTE traffic in separate recordings. By extracting Autonomous System (AS) numbers to create a vector for each recording, we can apply different distance metrics on the vectors to determine the similarity between recordings. By applying this method on 60 minute recordings, we demonstrate that we can derive what set of apps was being used in 100% of the recordings. When using a shorter recording time of only 5 minutes, we show that it is still possible determine the correct set of apps in 98%-100% of the cases, by choosing the right distance metric. This shows that an attacker with eavesdropping capabilities only requires a very short period of time to be able to identify the user that has generated that LTE network traffic. This constitutes a serious privacy infringement that requires the implementation of appropriate countermeasures.

Contents

1 Introduction

Long-Term Evolution (LTE) or 4G is the fourth generation communication standard used in mobile networks. It has many benefits compared to its predecessor (3G), including significantly higher bandwidth, lower latency and the ability to connect more devices simultaneously. While the fifth-generation (5G) has also already been actively being deployed for a couple of years now, LTE continues to play an important role in the realm of mobile communication. Enrollment and adaption of 5G takes time and is not economically feasible to deploy everywhere. As a result, numerous areas in the world are still lacking 5G coverage and some regions might never gain access to this technology. But even in areas where it is available, many older mobile devices are in use that still only support LTE. One of the factors that refrain mobile users from making the switch to 5G in such areas, could be the fact that the available 5G is still non-standalone 5G. This is a commonly used version of 5G where the infrastructure still runs on LTE, which might cause mobile users to question the current benefits of upgrading to 5G. In the Netherlands for example, as of the time of writing, it will still take a year before standalone 5G will be available [46]. Even though 5G adaption increases, we will focus on LTE in this research due to its important role in the telecommunications industry. Because of its extensive adoption and widespread use as a standard, it is an attractive target for attackers. Examples of attacks include downgrade attacks (e.g. [24, 15, 25]), denial of service attacks (e.g. [26, 27, 28, 29, 30, 31, 32]), localization attacks (e.g. [8, 9, 10, 11, 12, 13]) and fingerprinting attacks. Examples of fingerprinting attacks are website fingerprinting attacks [1] or app fingerprinting attacks [5]. In these attacks, the goal is to derive what websites were visited, what apps were visited or even what actions were performed inside those apps. This is achieved by prerecording great amounts of known network traffic and using Machine Learning (ML) to train on that data to recognize patterns in that traffic. This allows an attacker to classify any newly recorded traffic, by examining its similarity to the known patterns. In this paper, we will perform a similar attack on traffic generated by different sets of apps. The average user has around 40-80 apps installed on their phone [6, 7], which means that the set of apps they have installed on their phone is most likely unique for each user.

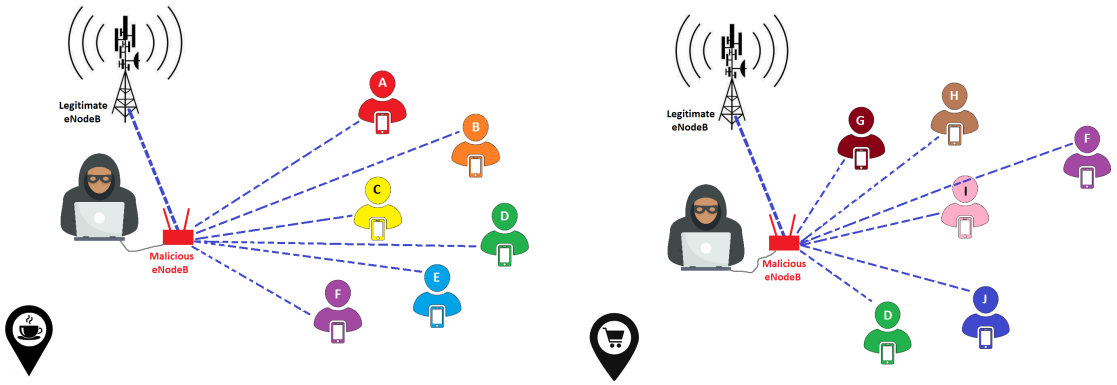
Each time we record the LTE network traffic, we will create a vector based on the Autonomous System (AS) numbers that we derive from the requested IP addresses in the traffic of that recording. By comparing these vectors of different recordings, we can determine their similarity and uniqueness. This allows us to investigate whether these vectors can serve as unique fingerprints, making it possible to distinguish different users based on the patterns in their network traffic. In other words, the goal of this research is to discover whether a unique set of apps also results in unique traffic patterns in their LTE traffic. This opens up the possibility to distinguish users based on the apps they have installed on their phones. Moreover, we will focus on achieving this using passive LTE traffic only. This means that the users do not need to be actively using their phone for our attack to work. The ability to track users this way, could be used for various purposes. For example, it allows to recognize the presence of known criminals in a network and thus at a certain location, which could aid law enforcement and intelligence agencies. It could also be used for user authentication purposes or targeted advertising. However, it is important to realize that this information might also be used for the following purposes:

- Recognize the same user in multiple locations (tracking)
- Derive their interests from the obtained location data (profiling)
  For example, if the same user is often detected at locations close to a festival, this reveals that persons interest in festivals.
- Surveillance and monitoring (espionage)

Therefore, the successful deployment of this attack would also introduce serious privacy concerns that imply the necessity of appropriate countermeasures. Motivated by its potential consequences and the profound implications, the purpose of this research is to thoroughly investigate the feasibility of this attack. In section 2, we will elaborate more on the attack scenario. Section 3 gives an overview of related work and how our work differentiates from that. We will illustrate how to prepare the environment that closely resembles the attacker scenario in section 4, after which we reveal our final setup in section 5. In section 6, we provide an explanation on how to analyze the traffic that we will record in our experiment. After having performed our experiment, we proceed with presenting the results of our analysis in section 7 and our discussion in section 8. Finally, we conclude our research in section 9.

2 Attack scenario

In our attack, we picture an attacker recording LTE traffic at multiple locations. Mobile network traffic could be recorded using a passive sniffer. While this sounds like a feasible option, it would require a way to decrypt the traffic, as the intercepted LTE traffic is encrypted. Therefore, a better way would be to set up a false LTE base station. A false base station (FBS) - also referred to as a fake base station, a rogue base station or a malicious *eNodeB* - is a device of an attacker that mimics a cell tower, with the purpose of tricking mobile users into connecting to it. He can achieve this by sending a stronger signal than any legitimate LTE cell tower (eNodeB). Once the users are connected, they will have an encrypted connection with the fake base station. At the same time, the attacker pretends to be a client to the legitimate base station, with whom it also establishes an encrypted connection. While the connections of the fake base station with both the users and the legitimate base station are encrypted, this Man-in-the-Middle (MitM) construction allows the attacker to act as an involved communication party in both connections. The attacker can therefore decrypt both connections and look inside the LTE packets. This could allow the attacker to recognize user specific patterns inside the decrypted LTE packets and he might be able to use that information to identify connected users. By recording traffic at multiple locations, an attacker could recognize the same patterns at those locations, allowing him to become aware of a users whereabouts. Therefore, this allows the tracking of a user. The victims that are successfully tracked in this attack do not need to have actively used their phone. Since we use passive traffic for our attack, any user can be become a victim, even if they have been keeping their phone in their pockets the entire time. Therefore, the users are unlikely to notice that they are being tracked.



(a) In a coffee shop

(b) At the mall

Figure 1: Recording traffic at multiple locations using a fake base station

In figure 1, the attacker is secretly recording LTE traffic in a coffee shop (1a) and later at the mall (1b). For every user that inadvertently connects with his fake base station, he notices different patterns in the traffic. However, after having compared the recordings of the coffee shop with those of the mall, he also notices that the patterns are not different for every user. Two of the users that he recorded at the mall turn out to have highly similar (or even identical) patterns to two users he recorded at the coffee shop earlier. He then knows that those two users have been at both the coffee shop and the mall when he was recording. When assigning labels to the discovered users, the attacker makes sure that these two users have the same label at both locations. In this example, he labels them user D and user F. Because the attacker has recognized users D and F at both locations, he knows when and where they have been. He has successfully tracked these users.
In this simplified example, the attacker has only recorded LTE network traffic at two locations, which has only resulted in discovering two people who coincidentally appeared at both locations at the same time. In a real-world scenario, an attacker might record traffic across a substantially greater number of locations simultaneously. While it may seem unlikely for a single person to use a large number of these advanced devices for tracking purposes, there are other potential adversaries with greater access to this type of equipment. For example, mobile network operators (MNOs), especially the ones that the mobile user did not subscribe to. Recording traffic at a multitude of locations simultaneously will result

into significantly more matches, allowing the attacker to track many people. The ability to fingerprint users and keeping track of their locations and interests this way is a great privacy infringement.

3 Related work

Numerous studies have examined the possibilities of exploiting flaws and performing attacks on LTE. In the context of tracking users, there are several areas that overlap with our research. For example, localization attacks in LTE (e.g. [8, 9, 10, 11, 12, 13]) attempt to discover the physical location of a user that is connected to an LTE network. While the aim of these attacks is to determine the location of a specific user, our goal is to recognize similar traffic patterns at multiple locations. This allows us to identify the same users at different locations and thereby becoming aware of a users whereabouts. Another way to identify the user at multiple locations, is using their IMSI. Similar to our attack, IMSI catchers (e.g. [15, 16]) also make use of a rogue base station. Given the fact that our goal is similar and we also have to setup a fake base station, it would make sense to think that the most rational course of action is to also catch the user's IMSI, instead of putting in more effort by proceeding to analyze their traffic while essentially achieving the same. However, while a user's IMSI is considered a permanent identifier, it could still be subjected to change. In the Netherlands for example, 55% of mobile phone users had a sim-only subscription [17] in 2022. This type of users will only have to swap their sim card if they switch to a different network operator. Whenever they do that, their IMSI will change, but their phone will keep generating the same traffic. This means that an IMSI catcher would no longer be able to identify it as the same user, while an attacker using our method will. Moreover, our attack does not require the attacker to actively prompt the victim to share its IMSI. Since the attacker only needs to record the LTE traffic, he can still identify the users later, when analyzing the recordings. This means that our attack can also be applied on previously recorded LTE traffic that has been collected for other purposes.

Another area that overlaps with our research are fingerprinting attacks and traffic classification techniques. For example, [2, 4] demonstrate the feasibility of classifying LTE traffic into categories. It is also possible to apply LTE traffic classification to derive what website is being visited [1]. This website fingerprinting attack on LTE works by focusing on various characteristics of the metadata and recognizing patterns. Similarly, app fingerprinting attacks have shown to successfully derive individual apps used by users, by only looking at the metadata of LTE traffic of those users. This could also be done in different ways, e.g. by extracting `Transport Block Size (TBS)` information from decoded `Downlink Control Information (DCI)` messages, applying deep learning algorithms and thereby classifying a set of six apps [4] or by creating a model based on the packet sizes and the probability that it appears [3]. Apart from the individual apps, the user actions that are performed inside apps, can also be retrieved. [5].
All of these aforementioned studies focus on active traffic, i.e. traffic that results from user activity. In our research, we will focus on passive traffic. Our goal is to be able to track users, while they are not actively using their phone. Rather than the type of traffic, websites or individual apps, we will focus on combinations of apps and the traffic that they generate. If we can derive patterns in the LTE network traffic that are different for each combination of apps, this means we can distinguish different users as they all have their own unique set of apps installed on their phone.

4 Preparing the setup

In this section, we will discuss how to build our experimental setup the that we are going to use in our research. We have an SDR (USRP B205 mini-i [33]) and a phone (POCO F2 Pro [34]) at our disposal. Our goal is to create an LTE network where the SDR functions as a fake base station. The phone will be used to simulate users connecting to our LTE network. We start by preparing the phone to be ready to connect to the network that we are going to setup.

4.1 Installing a programmable SIM card

For our phone to connect with our custom LTE network that we are going to setup, we need a programmable SIM card. We will use the sysmoISIM-SJA2 [35]. To program the SIM card, we require

- a card reader, e.g. the HID OMNIKEY 3121 [36]
- software to program the card, such as PySIM [37]

After installing the software and its dependencies and connecting the card reader, we can program the SIM card as shown in the example below (based on an example from [38]):

```
# ./pySim-prog.py -p 0 -x 101 -y 02 -t sysmoISIM-SJA2 -i 101020000000003 -s 8988211000000012345
-op=11111111111111111111111111111111 -k 8BAF473F2F8FD09487CCCBD7097C6862 -a 53770832
```

If we then insert the SIM card into the card reader, this will be the output:

```
Insert card now (or CTRL-C to cancel)
Generated card parameters :
> Name : Magic
> SMSP : e1ffffffffffffffffffff0581005155f5ffffffffff000000
> ICCID : 8988211000000012345
> MCC/MNC : 101/2
> IMSI : 101020000000003
> Ki : 8BAF473F2F8FD09487CCCBD7097C6862
> OPC : 8e27b6af0e692e750f32667a3b14605d
> ACC : None

Programming ...
Done !
```

We need the OPc later, when editing the srsRAN configuration files in section 4.2.3.

4.2 Creating an LTE network

To perform our attack, we need to set up our own an LTE network. Our attack involves the use of a malicious eNodeB, which we are going to simulate using our SDR. We will also use a computer, on which we run software called `srsRAN` [39]. This software creates an LTE core network on the computer. It also turns the SDR that we connect to the computer into an eNodeB, but in order for that to happen, we first need to install a hardware driver for the SDR called `UHD` [40].

4.2.1 Installing UHD

In order to set up UHD, we will perform the instructions listed on [41] on a machine that runs Ubuntu. We start with the following commands:

```
# sudo add-apt-repository ppa:ettusresearch/uhd
# sudo apt-get update
# sudo apt-get install libuhd-dev uhd-host
```

We can use this command to make sure the installation of UHD is successful and the SDR can be detected on the computer:

```
# uhd_find_devices
```

In our case, we have the following output:

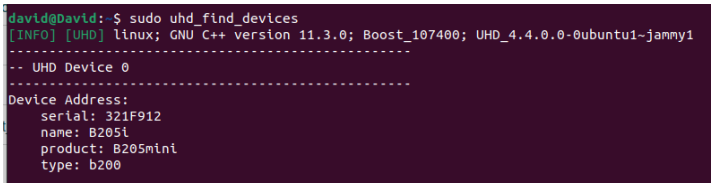


Figure 2: The output of the command uhd_find_devices

This shows that UHD is installed correctly and our SDR has been detected.
If any errors are displayed at this point, it could be the result of broken packages or unmet dependencies. In those cases, the problem could be resolved by uninstalling and reinstalling UHD while resolving any errors that might occur during this process. If UHD was installed using the command above, we could uninstall it using:

```
# sudo apt remove libuhd-dev uhd-host
```

4.2.2 Installing srsRAN

Installing srsRAN can be done using the source or using package installation. [42]. We will use the source installation, for which we will first need to install the required packages:

```
# sudo apt-get install build-essential cmake libfftw3-dev libmbedtls-dev libboost-program-options-dev
  libconfig++-dev libsctp-dev
```

We can now download and build srsRAN from source

```
# git clone https://github.com/srsRAN/srsRAN_4G.git
# cd srsRAN_4G
# mkdir build
# cd build
# cmake ../
# make
# make test
```

After building, we can install srsRAN:

```
# sudo make install
# srsran_4g_install_configs.sh user
```

4.2.3 Managing configuration files

We need to make sure that the configuration files for srsRAN are created using this command:

```
# sudo srsran_4g_install_configs.sh
```

We can then view the configuration files, which are located in the folder **/etc/srsran_4g** or in the folder **/etc/srsran/** when using an older version of srsRAN.
The relevant files in this folder are the following:

- epc.conf
  The file epc.conf is the configuration file for the core network. It is important to keep these lines in mind:
  $mcc = 001$
  $mnc = 01$
  $apn = srsapn$

- enb.conf
  We have similar entries in the file enb.conf, which is the configuration file for the eNodeB. We need to make sure that the values for mcc and mnc in enb.conf match the values in epc.conf. We also need the mcc, mnc and APN later, when we are changing the phone settings in section 4.4.

  In order to record the traffic that flows through our malicious eNodeB, we can tell srsRAN to save the traffic into a PCAP file. We can achieve that by uncommenting the following lines in enb.conf and specifying the desired filename:
  $[pcap]$
  $enable = true$
  $filename = enb.pcap$

- user_db.csv
  In user_db.csv, we need to add an entry with the following details:
  - Name: Any human readable value
  - Auth: Authentication algorithm (xor/ mil)
  - IMSI: UE's IMSI value
  - Key: UE's key, hex value
  - OP Type: Operator's code type (OP/ OPc)
  - OP: OP/ OPc code, hex value
  - AMF: Authentication management field, hex value must be above 8000
  - SQN: UE's Sequence number for freshness of the authentication
  - QCI: QoS Class Identifier for the UE's default bearer
  - IP Alloc: IP allocation strategy for the SPGW

  The values that we add here should match the values of the same parameters that we used in section 4.1 to program the SIM card, such as the IMSI and the OPc.
  However, the example shown in section 4.1 is not our own example, but from another website [38].
  In our case, we will add this line to the file:

  ```
  Magic,mil,001010000000002,4baee4d1600b1d2f6ece6373a59a1a14,opc,
  358422278845a5632bbfb7b354db103a,9000,00000000634e,9,dynamic
  ```

4.3 Running our LTE network

In order to start running the LTE network that we have just created, we have to open up two terminal windows and enter a command in each of them:

1. Start our LTE core network:
   Enter the following command in the first opened terminal window:
   ```
   # sudo srsepc
   ```

2. Start our eNodeB:
   This command should be entered in the second terminal window:
   ```
   # sudo srsenb
   ```

   This, of course, requires the SDR to be connected to the computer on which we execute the command

In order to stop our LTE network from running, we just have to stop the programs that are running in both terminals. This could be done using ctrl+c in each of them, for example.

4.4 Connecting the phone to our LTE network

So far, we have already been taking the first steps in connecting the phone to the network, namely programming the SIM card with the right parameters (in section 4.1) and adjusting the configuration files to match those parameters (in section 4.2.3). Now we need to make sure that the phone can indeed connect, by changing the settings on the phone. We proceed with the following steps:

- Add an APN
  If we go to Network Settings > Access Point Names on the phone, we will find the option add a new APN.
  It is important to make sure that we add the same information for mcc, mnc and APN as we specified in epc.conf in see section 4.2.3.
  As shown in figure 3, we did not specify a "Bearer", but if the phone does not connect, it might be necessary to set the bearer to **LTE**.

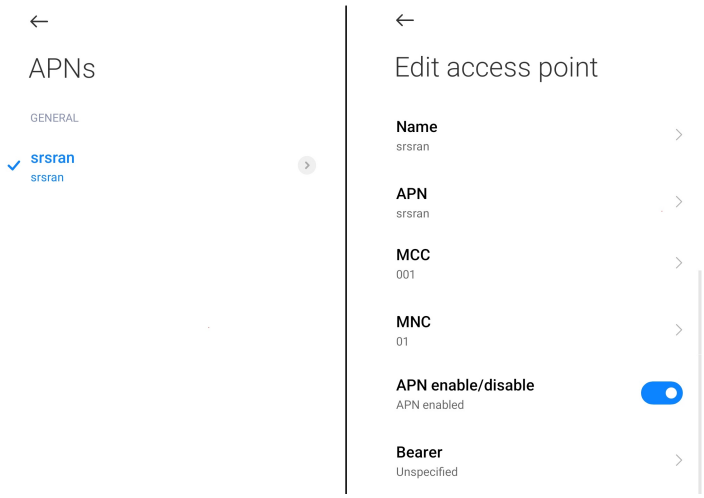


Figure 3: The settings for the newly added APN

- Let the phone access the internet
  To make sure that the phone can use mobile internet, we need to connect our own LTE network to the internet. Since our LTE network runs on a computer, we can use the computer's internet connection for the LTE network.
  This can be achieved using the masquerading script [43] which we can call using the following command:
  `# sudo [srsran folder]/srsepc/srsepc_if_masq.sh [interface]`
  Here [srsran folder] is the path to the folder that we used to install srsRAN and [interface] is the name of the network interface of the computer (e.g. "wlp2s0" or "eth1"). After this command, the LTE network has to be restarted (similar to section 4.3).

- Use airplane mode to force connecting
  At this point, we should be able to connect our phone to our LTE network. If this does not happen automatically, it can be enforced by enabling airplane mode and then disabling it again. After it is disabled, the phone will try to connect to a network again.
  It might also work to manually search for mobile networks in the network settings and then select the one created with srsRAN. However, our network was not visible in that list, while using airplane mode to force reconnecting did trigger a successful connection with our network.

- Optional: hidden menu
  If the phone still does not connect to the LTE network at this point, it might be necessary to access a hidden menu. The way to access such a menu depends on the brand of the phone. In our case, we could dial ***#*#4646#*#*** to open the menu. In this menu, it is important to enable LTE connections, even if it is already enabled in the regular phone settings.
  We noticed that this could make a difference in our early attempts to establish a successful connection.

4.5 Simulate multiple users

In a real world scenario, we would record the data of multiple users simultaneously. Using a malicious base station or a sniffer, we would observe and process the data of all phones that are connected to the base station. In this thesis, our setup is limited to one phone only. This therefore requires us to simulate multiple users onto a single phone. We describe our method to achieve this below.

4.5.1 Preparing the phone

The first thing we do is installing the 500 most downloaded apps worldwide [44] on the phone. We included the list of these 500 apps and their package names in Appendix C.

4.5.2 Installing the apps

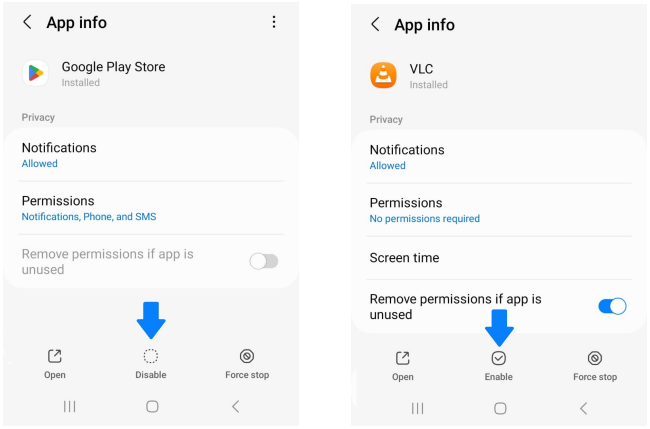
Installing 500 apps by hand would be an immensely time-consuming process. Therefore, we want to automate this process using `adb`. `Adb` [45] is software that makes it possible to control a phone using a computer (through a USB connection or SSH). Using a script that executes `adb` commands, we can open each app that we want to install in the Google Play store and then automatically hit the install button from there. This means that we need a script with the following abilities:

- Execute system commands, including commands that use `adb`
- Emulate a user that taps the screen at specific coordinates
- Open the Play Store and go to the specific app that we want to install
- Find the install button and then tap it

Appendix A.1 shows a simplified version of such a script. Using this script, we can then call the function `installapp()` for every package name on the list of package names of the apps that we want to install. However, it is important to note that while running a script like the one suggested in Appendix A.1, one will notice that several errors can occur. For example, not every app is available in the Google Play Store. This can be due to several reasons, including country specific restrictions. To avoid these restrictions and other problems that might occur during the process of installing 500 apps, extensive error handling is required that causes the actual code to be significantly more complex than this simplified example. For example, some apps have to be installed through a 3rd party source, which requires keeping track of which apps did not successfully install through the Play Store and creating an additional script that retrieves them from another source. Therefore, the purpose of the provided script is only to give a general idea how one might proceed in installing the 500 apps using a script.

4.5.3 Enabling and disabling apps

On Android, it is possible to enable or disable apps. When an app is disabled, it cannot be used and will not generate any traffic. However, an app can be re-enabled at any time, allowing the user to use the app again and generate traffic.



(a) Enabled app with disable option

(b) Disabled app with enable option

Figure 4: An app can be disabled or enabled in the settings of the app

We are going to use this principle to simulate different users on one phone. The idea is that every user has a unique set of apps that will result into unique traffic patterns. To simulate that, we disable all of the 500 apps and then only enable a specific, randomly generated subset of apps. By doing so, we simulate one user. If we want to simulate another user, we will disable all of the apps again and enable another randomly generated subset of apps. We will call these randomly generated subsets of

apps that imitate users "profiles".
While figure 4 shows how to enable and disable apps in the settings, we will do this with a script. This script will be using the following `adb` commands:

- Enable an app:

```
# adb shell su -c pm enable [package name]
```

- Disable an app:

```
# adb shell su -c pm disable [package name]
```

It is important to note that the phone needs to be rooted in order to be using these su commands. This can be a laborious process, especially with brands that purposely make it almost impossible to achieve this. However, it is worth investing the effort, given the fact that it is an essential step in being able to disable/enable apps and therefore being able to use profiles.

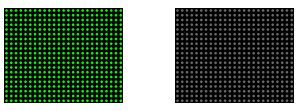
4.5.4 Profiles

As mentioned earlier, we will be creating "profiles", which are randomly generated subsets of the total of 500 apps that we have installed on the phone. By disabling all of those 500 apps, except for the 50 apps in a profile, we can generate traffic for a specific set of apps and thereby simulate a user. If we then later also disable all the apps in that profile and enable the apps inside a different profile, we can simulate another user. This way, by repeatedly disabling all apps and then enabling the apps inside a profile, we can simulate different users.

In other words, we will:

1. Disable all 500 apps
2. Activate one of these profiles
   (by enabling the selection of the 50 apps inside it)
3. Capture the LTE network traffic generated by the apps in the activated profile
   (for 1 hour)
4. Deactivate the profile
   (by disabling all apps again)
5. Repeat this process for the next profile

We can represent the 500 apps that we will install as dots in a matrix: a green dot for an enabled app and a grey dot for a disabled app. Figure 5 shows what that matrix would look like if we enable all apps (figure 5a) or disable all apps (figure 5b).

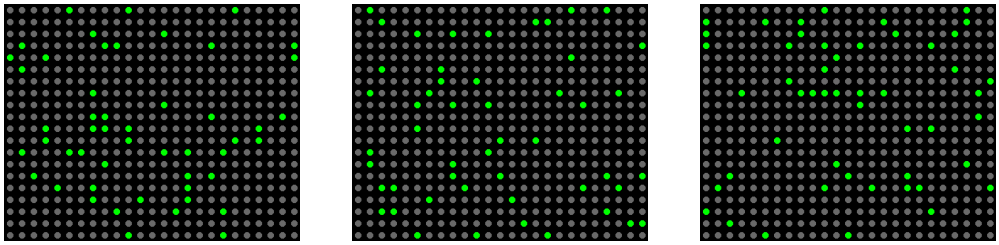


(a) All apps enabled

(b) All apps disabled

Figure 5: A matrix of 500 dots, representing the 500 apps we will install

After having disabled all apps (figure 5b), we will re-enable a random selection of 50 apps. In other words, we will be activating one of the profiles. Figure 6 shows three examples of such randomly generated profiles.



(a) Profile 1 (b) Profile 2 (c) Profile 3

Figure 6: Examples of random subsets (profiles) of 50 apps that we can enable on the phone

While these are only three examples, we will be using 20 profiles. Appendix D shows the 20 randomly generated profiles that we will be using in our research. We will record the data of each profile for 1 hour. We will repeat this entire process 9 times, such that the traffic of each profile is being recorded 10 times in total. This means that we will end up with 20 profiles $\times$ 1 hour per recording $\times$ 10 times = 200 (hours of) recordings. Each recording will have its own PCAP file, so we will also end up with 200 separate PCAP files. When we are going to analyze these PCAPs later, our goal will be to be able to correctly determine what profile was being used in each PCAP file, without using prior knowledge.

4.6 Account for software or hardware crashes

As mentioned in section 4.2, we will use software called `srsRAN` and an SDR to create the LTE network. We noticed that both this software and hardware can be unstable, meaning that it is highly likely that our setup will freeze at some point. It is important to take this into account, such that our setup can recover from this state whenever this occurs. In that light, we will use this section to discuss different types of failures, how to detect them and how to recover from them while running our setup, such that we can continue our experiments without human interaction. Moreover, we will present a state transition diagram that shows how to automate this process.

4.6.1 Detect a crash using the phone's connection state

In our setup, we can use the phone's connection state to see if the phone is still connected to the SDR. By connecting the phone directly to the laptop using a USB cable, we can use `adb` to see if the phone has an LTE connection:

```
# adb shell dumpsys telephony.registry
```

This is an example output:

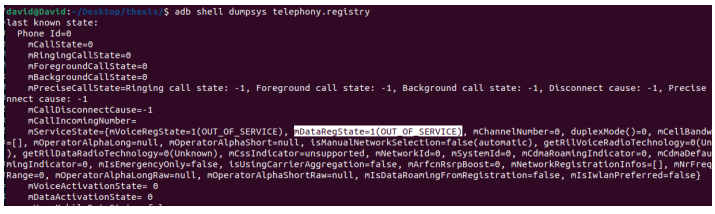


Figure 7: Output when the phone is not connected

Despite the wealth of data that is being returned after using this command, we direct our attention only to what is highlighted in this screenshot: mDataRegState=1. This means that the phone is not connected. If the phone is connected to the LTE network, its value would be 0:

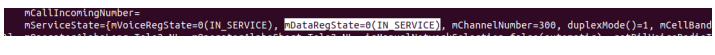


Figure 8: Output when the phone is connected

To only extract this value, we can use this command:

```
# adb shell dumpsys telephony.registry | grep -i -o '\smDataRegState=.'
```

We can incorporate this command in our setup, by using this command in the Python script that will run our setup. Please refer to Appendix A.2 to see an example code of a Python function that uses this command. It extracts and returns the relevant value from the output of the command.

4.6.2 Recover the connection using Airplane mode

Since a crashed SDR would result in a disconnected phone, we can use this value to detect a crashed SDR. However, it is important to note that a phone can be disconnected for several reasons. This means that the absence of phone connectivity alone does not necessarily imply a crash of the SDR. But if the SDR did not crash, there is high likelihood that we would succeed in reconnecting the phone using airplane mode. With `adb` we can:

- pull down the top menu in Android
- activate airplane mode
  (by automating a tap on the airplane icon)
- wait a couple of seconds
- disable airplane mode
  (by tapping the airplane icon again)

Leaving airplane mode will force the phone to make a reconnecting attempt. It is not uncommon that this will only work after two or more attempts. Therefore, if this strategy seems ineffective, we still have to repeat this process several times in order to make sure that it is not the phone's fault that it has lost connection.

4.6.3 Recover the connection by killing srsRAN

If only trying to reconnect the phone does not seem to be effective, it still does not necessarily indicate that the SDR has crashed. It could also be that it is just the software srsRAN that has entered a state that it cannot recover from. Therefore, we need to kill and reboot srsRAN. As explained in section 4.3, we run our LTE network using two srsRAN programs: srsepc and srsenb. In order to kill srsRAN, we need to end both of them. Since we want to automate this process, we do not do this by pressing ctrl+c in the terminal windows of both programs. Instead, we can kill both of them with the following command:

```
# kill [pid]
```

Here, [pid] is the process identifier: a number that is a reference to the program that we would like to kill. So we first need to find the `pid` for both srsepc and srsenb, such that we can execute that kill command for both of them. We can find the `pid` of srsepc using this command:

```
# sudo ps -aux | grep srsepc
```

However, this returns a list of multiple processes that have "srsepc" in the name. In order to kill srspec, we also need the one that is ran by root and has "S+" in the 6th column. But since this could still be the grep process from the command itself, we need to make sure that the last column does not contain "grep" or "–color=auto". Taking all of this into account, we can create the Python script from listing 5 in Appendix A.3 to get the pid of srsepc.
If we run that script, then this will the output:

```
        [COMMAND]  sudo ps -aux | grep srsepc
        Done.
Overview of processes matching ' srsepc ':
root       27347  0.0  0.0  13968  5328 pts/1    S+   16:52   0:00 sudo srsepc
root       27348  0.0  0.0  13968   680 pts/2    Ss   16:52   0:00 sudo srsepc
root       27349  1.7  0.4 238104 64972 pts/2    Sl+  16:52   0:21 srsepc
david      27862  0.0  0.0   2888   948 pts/0    S+   17:12   0:00 /bin/sh -c sudo ps -aux | grep srsepc
david      27864  0.0  0.0   9212  2276 pts/0    S+   17:12   0:00 grep srsepc

====================
['root', '27347', '0.0', '0.0', '13968', '5328', 'pts/1', 'S+', '16:52', '0:00', 'sudo', 'srsepc']
The pid that we need is: 27347
```

Figure 9: The output of our Python program to find the pid of srsepc

What we see in this screenshot is that the command to find the pid returned multiple lines. The first line is the one that satisfies our requirements, which will therefore be selected. That line is split into a list that we see at the end. The pid is the second element in that list: 27347. Similarly, we can retrieve the pid of srsenb by calling `getPid("srsenb")` using the script above. Now that we have retrieved the pids of both srsepc and srsenb, we can kill them both using the kill command that we mentioned earlier. Listing 6 in Appendix A.3 provides a code sample with a `killprogram(keyword)` function and a `killsrsRAN()` that calls both `killprogram("srsenb")` and `killprogram("srsepc")`. After both programs have successfully been killed with the `killsrsRAN()` function, we can reboot them using the commands `sudo srsepc` and `sudo srsenb` as explained in section 4.3.

4.6.4 Maintain the state before killing srsRAN

Whenever we have to kill and reboot srsRAN as explained above, we want to make sure that we continue our recording at the moment it stopped. For example, if we had to kill srsRAN after 37 minutes of recording profile 7 (repetition 3), we want to make sure it continues with profile 7 (repetition 3) for another 23 minutes after we have rebooted srsRAN. This way, we can still achieve the 1 hour recording for each profile during each repetition. To achieve this, we can rename the PCAP that is created by srsRAN to `profile-7-repetition-3-part1.pcap`, right after we kill srsRAN. We also keep track of how many minutes it has recorded so far in a separate text file. Whenever our script reboots srsRAN, it sees that the most recent file is `profile-7-repetition-3-part1.pcap`. It then realizes that the recording has stopped unexpectedly and it has to look in the separate text file to calculate how many minutes it still has to record. After it starts recording again and the rest of the recording is finished, it will be saved to `profile-7-repetition-3-part2.pcap`. We can later merge `profile-7-repetition-3-part1.pcap` and `profile-7-repetition-3-part2.pcap` into one file named `profile-7-repetition-3.pcap`. Next, it will start with profile 8 (repetition 3). To ensure that the right profile is activated when we continue after a crash, we will close and disable all of the apps again and then re-enable and open the apps inside the current profile. This will make sure that we do not falsely assume that certain apps are enabled or disabled, while that does not reflect the actual state of the phone. However, there is a downside to this approach, as closing, disabling, enabling and opening 50 apps in a row using `adb` commands can be time-consuming.

4.6.5 Recover the connection by unplugging the SDR

If we cannot recover the phone's connection state using Airplane mode or by killing srsRAN, it is very likely that the SDR has crashed. In that case, we need to unplug the SDR, wait a while and then reinsert the the USB-cable into the computer. Performing this action manually would slow down the data collection process immensely, as we would have to notice it first and be in the position to act on it before our setup can continue. To avoid delays like this, we also want to automate this process.

Steps in reconnecting the SDR to the computer
In order to simulate the disconnecting and reconnecting of the USB cable of the SDR, we have to do the following things:

1. Kill srsRAN (as explained in section 4.6.3)
   to stop any software from trying to communicate with the SDR
2. Reset the SDR
   by cutting the power to the SDR through USB, leave it powered off for certain amount of time and restore the power to the SDR.
3. Reset the USB connection with the SDR on the computer
   as suddenly cutting the power of the SDR results in difficulties communicating between the computer and the SDR
4. Reboot srsRAN

To cut the power to the SDR, we can make use of a "smart" or "managed" USB hub. With additional software, we can disable and restore the power on an individual USB port on that hub. In our setup, we use an USB hub from Amazon Basics together with a tool called Uhubctl [47].

This USB hub also has another advantage apart from the ability to reset the SDR if errors occur. We noticed that the SDR sometimes experiences issues related to not getting sufficient power from the USB port. This resulted into errors, that we no longer experienced as soon as the SDR was connected to the USB hub. We assume that this is the result of the USB hub having its own power source, which also causes the SDR to always be provided of sufficient power.

4.6.6 Killing srsRAN vs. unplugging the SDR

We have described the possible solutions of killing srsRAN and unplugging the SDR in section 4.6.3 and section 4.6.5, respectively. As discussed earlier, killing srsRAN is also a part of the process of unplugging the SDR. As this unplugging process encompasses the steps of rebooting srsRAN and also additional ones, it requires more effort than just rebooting srsRAN. In that light, it may seem intuitive to only start the unplugging process if killing srsRAN has shown to be ineffective. However, the most time-consuming part of unplugging the SDR is rebooting srsRAN (due to all the steps mentioned in section 4.6.3 and section 4.6.4).
If we would start the unplugging process only after rebooting srsRAN has shown to be ineffective, this means that we would have to perform the time-consuming reboot of srsRAN twice: once to conclude that it is ineffective and once as part of the unplugging process. Therefore, significant time savings can be achieved by only rebooting srsRAN once. We can achieve this by skipping our attempt to only reboot srsRAN and instead directly initiating the unplugging process after deactivating airplane mode has shown to be ineffective to recover the connection. This means that we might unnecessarily unplug the SDR while just rebooting srsRAN would have sufficed. While this may be a minor inconvenience, it pales in comparison to the time and effort it takes to reboot srsRAN twice.

4.6.7 Automate the above

As explained in sections 4.6.2, 4.6.3 and 4.6.5, we have three solutions to recover from a potential crash: using airplane mode, killing srsRAN and unplugging the SDR. Figure 10 shows how we combine these solutions (marked in orange) in our setup, to enable timely detection and recovery from any crashes:

- While running our setup, we will periodically verify that the phone is connected (every 5 seconds).
- If the phone is not connected, we perform several attempts to restore the connection using airplane mode (as discussed in section 4.6.2).
  If the phone's connection is restored during those attempts, the counter will be reset.
  If the connection is still not restored after 5 attempts, we continue to the next step.
- The state of the recording will be saved (as explained in section 4.6.4), we will initiate both the solution of killing srsRAN and unplugging the SDR (see section 4.6.6) and we will then reboot our setup and recover the state.

While it is highly unlikely that even this last step is not an effective remedy to recover from a crash, it is not impossible. If that occurs, our setup will retry all of the aforementioned steps, increasing the likelihood of recovering from the crash even further. If, however, our setup still has not recovered after several times unplugging the SDR, it is probable that other factors than a crash are the reason for the phone being unable to reconnect. In those cases, we will undertake a careful evaluation of the situation, and take appropriate measures based on the nature of the issue at hand.

4.7 Configuring the permissions of the apps

To make sure that the 500 apps that we installed on the phone are actually sending and receiving data from the servers that they connect with, we aim to eliminate any obstacle that might prevent that from happening. Depending on how each individual app is built, such an obstacle might be the fact that an app has not been granted the permissions it requires to operate. To ensure that this does not happen, we will grant every app all of the permissions that it wants. However, doing this manually by enabling all permissions in the settings of each app would be an exceedingly time-consuming process. Therefore, we want to automate this process. To achieve that, numerous extensive steps have to be

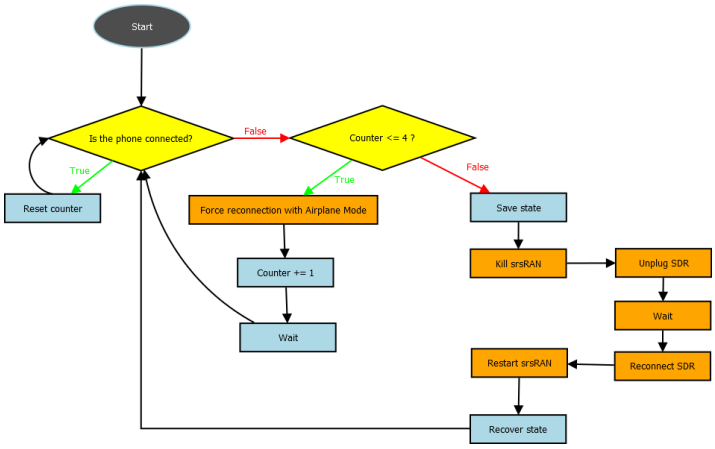


Figure 10: Flow chart demonstrating how to recover from a crash

taken. Appendix B presents a comprehensive outline of the procedural steps involved in this automated process.

It is important to realize that assigning any permission to an app on a phone, without questioning its relevance, can result into various parties being able to collect unnecessary great amounts of user data. This undermines the intended functionality of the permissions system and poses a great privacy threat to the phone user. While this practice supports our research, it should not be applied outside the scope of a research environment.

4.8 Other things to account for

We have explained how to account for crashes in our setup in section 4.6, and in section 4.7 and Appendix B, we showed how to take the permissions into account. Apart from these crashes and permissions, there are other things we need to take into account to make sure our setup can run successfully:

- Root the phone
  As mentioned in section 4.5.3, the phone requires to be rooted in order to use commands that enable and disable apps. Without those commands, we cannot activate a profile, which makes rooting the phone a crucial step. In our case, we had to engage in an elaborate process to make this happen. Xiaomi, which is the manufacturer of our POCO F2 Pro, had created several obstacles that made it almost impossible. Steps included in this process were changing hidden settings, installing fastboot, unlock the bootloader, installing Magisk, installing a custom ROM, using the Miui unlock tool, creating a Xiaomi account with personal details, waiting a couple weeks, etc. All of these steps have to be taken in the right order and even then errors can occur that have no other solution than retrying. For creating a setup like ours, it is important to keep in mind that the process of rooting the phone can be one of the most time-consuming parts.

- Sleep mode on the computer
  Both the phone and the SDR will be connected to a laptop that runs srsRAN. Since we are recording for period of time that extends beyond the usual interval that a computer hibernates, we have to disable the sleep mode to avoid interruptions.

- Overheating
  Since our setup will run continuously for more than 200 hours, devices in our setup may experience an elevation in temperature beyond the standard operational range. To prevent any device from shutting down due to overheating, we need to take appropriate measures to control the temperature of the devices.
  For that matter, we will place several heat sinks on the devices. Another possible solution could be to cool the devices using liquid nitrogen [49].

5 The attack setup

In section 4, we have discussed the tools we need and all kinds of things we need to take into account when creating our setup. In this section, we will demonstrate the result.

5.1 Diagram of the setup

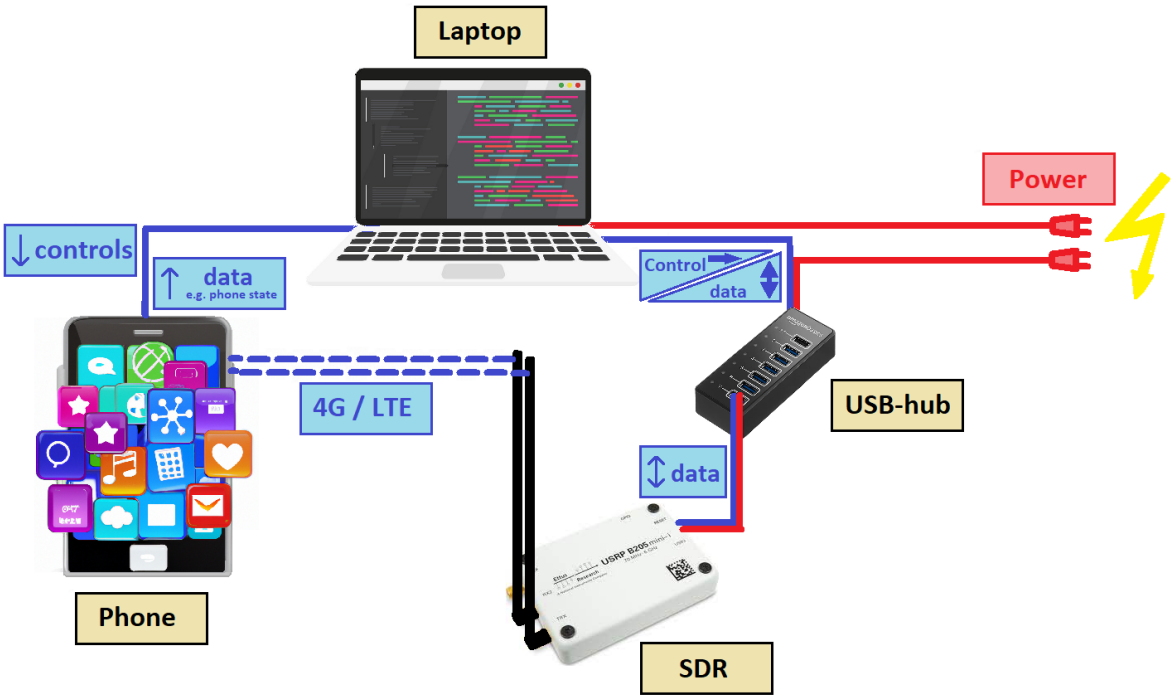


Figure 11: Diagram of the setup that was used

Figure 11 shows us a diagram of the setup that we are using.

5.1.1 Components

The setup comprises the following components:

- A phone
  to simulate different users generating LTE traffic using their apps (see section 4.5). We are using the Poco F2 Pro.
- An SDR
  We use a USRP B205-mini-i as an eNodeB for our LTE network.
- A laptop
  To establish our LTE network:
  - Running srsRAN to create a core network for our LTE network and to use the SDR as an eNodeB;
  - The internet connection of the laptop is used to provide an internet connection to the phone that is connected to our LTE network;
  - Capture the LTE traffic and save it into a PCAP file.

  And to control the phone:
  - Enabling and disabling apps on the phone, such that we can switch profiles (see section 4.5.4);

  - Read the connection status of the phone
    (see section 4.6.1);
  - Use airplane mode to restore LTE connection with the SDR in case it is disconnected
    (see section 4.6.2).

- A USB hub
  To provide sufficient power for the SDR and for automating the process of unplugging the SDR and reinserting its USB cable in case the SDR has crashed (see section 4.6.5).

5.1.2 Connections

The phone is connected to the laptop with a USB cable and to the SDR with an LTE connection. The SDR is connected to the USB hub with a USB cable. The USB hub has two cables: one USB cable to the laptop and one power cable that we connect to a wall outlet. The USB hub is part of the USB connection that is used for the the laptop and the SDR to communicate with each other.

5.1.3 Script

The main component of our setup is the laptop as it will control everything in our setup: the phone, the SDR, the USB hub and srsRAN. In order to achieve that, we create a Python script that has the following abilities:

- Run srsRAN (srsepc and srsenb)
- Disable and unable apps on the phone in order to switch profiles (see section 4.5.4)
- Recover from a crash as shown in figure 10, which includes:
  - Reading the phone's LTE connection state (see section 4.6.1)
  - The ability to enable and disable airplane mode on the phone (see section 4.6.2)
  - Stopping and rebooting srsRAN (see section 4.6.3)
  - Unplugging the SDR and reconnecting it using the USB hub (see section 4.6.5)
  - Save the current state of the recording and restore the state (see section 4.6.4). This means:
    * Keeping track of what profiles have been recorded
    * How many times each profile has been recorded
      (i.e. in what iteration we currently are)
    * The duration of the current recording before we lost connection
    * The part number of the current recording.
      For example, if this is the first time srsRAN crashes during this recording, we call this part1 of the recording. If we later continue, then that will be part2 of the recording.

6 Attack analysis

In this section, we will discuss how we are going to process and analyze the LTE data that we will be recording.

6.1 The output data of our setup

After having run our setup successfully, we have collected the data that is needed for our attack. The data is saved in PCAP files that are named according to the following convention:
`profile-[profile number]-repetition-[repetition number].pcap`.
Some files may also have part numbers in their name, as the setup might have been crashed and recovered while collecting the data. Those files can be merged into a single file and renamed conform the earlier mentioned convention. Since we are recording 20 profiles for 10 repetitions, we will end up with 200 PCAP files.

6.2 Extracting relevant data from the PCAP files

Each PCAP file contains a large amount of data. Processing all of this data could be time-consuming, while it may not necessarily align with our intended objectives. In our case, it is sufficient to only extract a relevant portion of the data, rather than processing the entire data set. To this end, we first have to identify the relevant data that we want to extract from each PCAP file.
The relevant data will be something that allows us to distinguish the traffic of different users, while simultaneously revealing similarities that can identify traffic originating from the same user. The network traffic that we will be looking at, is the result of apps on a users phone communicating with their server, for purposes such as providing accurate and up-to-date information, authentication, tracking, advertisements, etc. App providers are often also the ones that control those server, such that they can handle user data securely, ensure availability of their services, implement specific configurations required for their app and scale their infrastructure based on demand. Therefore, the servers and the requests to those servers can be very specific for each app. This means that we might be able to distinguish traffic based on what server is being requested. To this end, DNS queries might seem like a viable option. However, due to DNS caching and the fact that we are only using passive data, we expect that the majority of DNS lookups are already stored in the cache. Therefore, it is probable that our data will contain very few, if any, DNS queries. Consequently, IP addresses are likely to be a more promising starting point. When using IP addresses to distinguish traffic, it is important to realize that multiple IP addresses could be associated with the same autonomous system (AS). This means that we should convert IP addresses to AS numbers (ASNs) first, before drawing conclusions from it. Otherwise, we might unjustly distinguish traffic with requests to different IP addresses, while they might request data from the same AS. Upon detecting the ASes within the recorded traffic, we can then differentiate users by the ASes they interact with.

6.3 Looking at data from multiple users simultaneously

In our setup, only one profile at the time is activated, which means that each PCAP file only contains the traffic of one user. However, in a real world scenario, an attacker would record the data of multiple users simultaneously. To be able to only look at the data generated from one user at a time, he could distinguish the traffic by looking at some variable that distinguishes the different flows of traffic. For example, the Radio Network Temporary Identifier (RNTI) is a temporary identifier that the attacker can use to temporarily distinguish the traffic of multiple users. The reason why the attacker does not just use this identifier to identify users instead of our attack, is because this identifier is only temporary. It changes every session, e.g. when the user connects to a different cell tower. The exact duration of a session depends on the user activity, network conditions and the type of communication (e.g. web browsing or voice calls or video streams). For example, a Voice over Internet Protocol (VoIP) session could potentially last for hours, whereas a session for quick data transmission might only take seconds. The RNTI could also be reset at any moment. This makes it too volatile for the attack to depend on it long term.

6.4 Extracting IP addresses from the PCAP files

To obtain the ASNs from a PCAP file, we first have to extract the IP addresses from that PCAP. This means that we need to read the PCAP file with a script. Using Python, this could be achieved using the Scapy library [50]. However, in our case, it experienced difficulties in correctly interpreting the LTE protocol while reading our PCAP files. Therefore, we used TShark [51] together with system calls instead. To make sure that TShark can successfully read the PCAP files, we first need to configure its settings on how to interpret LTE data. It is complex to do this in TShark itself, due to the absence of a graphical user interface (`GUI`). Fortunately, Wireshark does have a GUI while it uses the same dissection engine as TShark. This means that we can configure the dissectors in the Wireshark [52] GUI and as a result, these changes will also affect TShark [51].

6.4.1 Altering dissectors in Wireshark

In Wireshark, we proceed with the following steps:

1. In the menu that appears after selecting
   `Edit > Preferences > Protocols > DLT_USER > Edit...`,
   we can add a new entry to the table where we set DLT to 149 and enter "udp" in the `Payload protocol` column.

2. In `Analyze > Enabled Protocols`,
   we can select and enable the following settings:

   - `MAC-LTE > mac_lte_udp`
   - `MAC-NR > mac_nr_udp`

3. When opening a PCAP file with captured LTE traffic in Wireshark, we can right click on any MAC-LTE packet and go to: `Protocol preferences > MAC-LTE > "Source of LCID -> drb`. Here we can make sure it is set to "`From configuration protocol`" (instead of "`From static table`").

If we apply these settings, the data in any PCAP with LTE data looks similar to this:

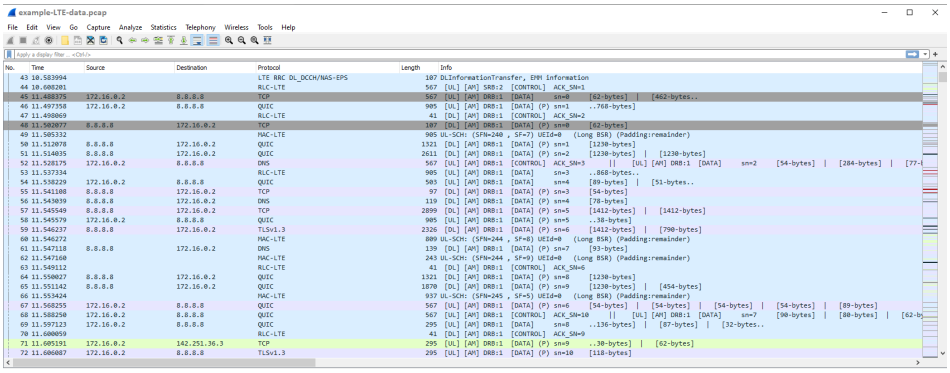


Figure 12: Example of a PCAP file with LTE data, viewed in Wireshark

Because Wireshark and TShark share the same dissection engine, these changes now also directly affect TShark. We can therefore now also read the LTE data in the PCAPs using TShark.

6.4.2 Using TShark to extract IP addresses

Using the following command, we can extract IP addresses from a PCAP file:

```
# tshark -r "example-LTE-data.pcap" -T fields -e 'frame.time_relative' -e 'ip.src' -e 'ip.dst'
```

This will print the relative time of each frame, a tab, the source IP address, another tab and the destination IP address.
Since our example file `example-LTE-data.pcap` also has some frames without a source and destination IP, it is possible that some lines in this output only contain the relative time. It is also possible that a frame contains multiple packets. In that case, multiple source and destination IP addresses will be displayed, separated by comma. Here are five lines from the output of the command above that display all of these examples:

```
...
11.724233000
11.725123000      8.8.8.8       172.16.0.2
11.725198000      172.16.0.2    8.8.8.8
11.726064000      172.16.0.2,172.16.0.2      8.8.8.8,8.8.8.8
11.727148000
...
```

We can process the output of this command with Python as follows:

```
    #The shell command to extract the relevant columns of our PCAP file
    tsharkcommand = "tshark -r 'example-LTE-data.pcap' -T fields -e 'frame.time\
    _relative' -e 'ip.src' -e 'ip.dst'"
    output = systemCmd(tsharkcommand)

    #Convert the output to a list of lines
    lines = output.decode("utf-8").split('\n')

    #Iterate over the lines
    for line in lines:
        #Convert the line into a list of fields
        fields = line.split("\t")

        #Iterate over the different fields from this line
        for i, field_value in enumerate(fields):
           if(i == 0):
              #Process the relative time field
              ...
           if(i == 1):
              #Process the source IP field.
              src_ips = field_value.split(',')
              ...
           if(i == 2):
              #Process the destination IP field.
              dst_ips = field_value.split(',')
              ...
```

Listing 1: Extracting PCAP data in Python

While we extracted both source and destination IP addresses in this example, we will eventually find the same IP addresses in both fields, as each app and the server are both involved in the same connection, in which they will be acting as both sender and recipient. Therefore, it does not really matter whether we look at source IP addresses, destination IP addresses or both. We will find the same IP addresses in the PCAP file.

6.5 Converting IP addresses to AS numbers (ASNs)

Now that we can extract IP addresses from the PCAP files using TShark and Python, we can use those IP addresses to convert them to AS numbers (ASNs). To convert an IP address to an ASN in our Python code, we need an external library such as GeoIP2 [53] and a database such as GeoLite2 ASN database [54].
These allow us to create the following Python function that converts an IP address to an ASN:

```
def IPtoASN(ip):
    geolitedatabase = "GeoLite2-ASN.mmdb"
    with geoip2.database.Reader(geolitedatabase) as reader:
        #Try look up the IP in the database. If it fails, return None
        try:
          response = reader.asn(ip)
          return response.autonomous_system_number
        except ValueError:
          pass
        except TypeError:
          pass
        except geoip2.errors.AddressNotFoundError:
          pass
    return None
```

Listing 2: Function to derive an AS number from an IP address using GeoLite2 ASN database

This function will return the ASN if no error occurs. Otherwise, it will return None. Errors could occur if, for example, the provided IP address is an empty string (ValueError) or a None type (TypeError). It is also possible that no ASN can be found for the provided IP address, which is most likely the result of that IP address being an internal IP address inside our LTE network, e.g. 172.16.0.2. This IP address is used as an internal IP address for the phone that is connected to our LTE network.

Figure 13 shows the process of retrieving AS numbers from a PCAP file. We will create a Python script - called `PCAP-to-ASN.py` - that uses TShark to extract the IP addresses. The script then converts those extracted IP address to AS numbers, using GeoIP2 and the GeoLite2 ASN database.

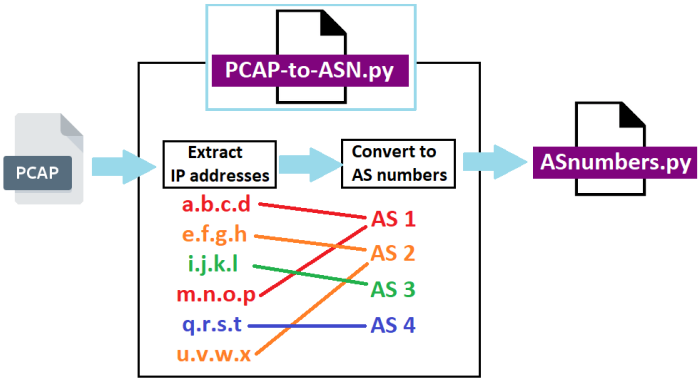


Figure 13: Extract AS numbers from each PCAP file and append them to ASnumbers.py

The number of resulting ASNs can be less than the amount of IP adresses we extracted, since multiple IP addresses can belong to the same AS. The script creates a list of the following format:

[[profile number, repetition number], [list of obtained ASNs]]

The profile number and repetition number are retrieved from the filename of the PCAP. We will then append this list to a file called ASnumbers.py. Since we will be performing these steps for all PCAPs, this means that ASnumbers.py will eventually contain a list with sublists of ASNs for each PCAP. So even though we will feed 200 PCAP files to PCAP-to-ASN.py, we only have one output file (ASnumbers.py) that contains all of the extracted AS numbers. This file is also the script that we will include in our next step (as shown by figure 19 in section 6.9).

6.6 Convert ASNs to Boolean vectors

Now that ASnumbers.py contains the 200 ASN lists (derived from our 200 PCAP files), we will convert each of these lists to a Boolean vector. The achieve this, we use a method that exhibits similarities to the concept of one-hot encoding [64]. To demonstrate how we will be performing this process, we will present a simplified example with only five ASN lists.

Example
Imagine that we have five PCAP files, from which we extracted the following lists of ASNs:

PCAP-1: AS1, AS5, AS6, AS18, AS24, AS35
PCAP-2: AS2, AS3, AS5, AS7, AS24
PCAP-3: AS1, AS3, AS5, AS13, AS35, AS42
PCAP-4: AS6, AS8, AS13, AS27
PCAP-5: AS3, AS5, AS7, AS8, AS18, AS24, AS34, AS42

Figure 14: The PCAP files and their ASN lists

We will now add all of these AS numbers from the different lists into one big list and we will remove duplicates. The resulting list will look as follows:

[AS1, AS2, AS3, AS5, AS6, AS7, AS8, AS13, AS18, AS24, AS27, AS34, AS35, AS42]

Figure 15: The list with all ASNs from each PCAP combined

This new list from figure 15 can now be compared to each of the original ASN lists obtained from the five PCAP files (figure 14) to see which of the AS numbers occur in those lists and which do not. By doing so, we can convert each ASN list into a Boolean vector. It works as follows:

- For each PCAP's ASN list, do the following:
    1. Create a new list, which we will call the **ASN vector**.
    2. For every ASN in the list from figure 15, see if it is also present in the PCAP's ASN list that we are looking at right now:
        - Yes → add a 1 to the ASN vector of this PCAP
        - No → add a 0 to the ASN vector of this PCAP

After completing this algorithm, each PCAP will have its own ASN Boolean vector, which is a list with the same number of items in it as the one from figure 15. But instead of AS numbers, these ASN vectors will contain a 0 for each ASN that does not occur in the PCAP's ASN list and a 1 for each ASN that does occur in it. This will result in the following ASN vectors:

PCAP-1: [1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0]
PCAP-2: [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]
PCAP-3: [1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1]
PCAP-4: [0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0]
PCAP-5: [0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1]

Figure 16: Each PCAP file and the corresponding ASN vector

Now that every PCAP has its own Boolean vector, we can determine the similarity of two PCAP files by comparing their ASN vectors. The goal of comparing two PCAP files this way, is to see whether the same profile was used when the traffic from both files was generated. If the ASN vectors show a high similarity, we know that the same profile was activated when they were generated. This therefore allows us to distinguish users, based on their traffic.

6.7 Determining the similarity of two PCAP files

In order to determine the similarity of two Boolean vectors, we can use a distance metric. There are different distance metrics that we could use and they can all have (slightly) different outcomes. Two PCAP files might score higher in similarity when using one distance metric, while scoring lower in similarity when using another. These slight variations could potentially impact the outcomes of our attack and affect our ability to distinguish users. Therefore, we will evaluate the performance of several distance metrics by comparing their results. Since we are dealing with (ASN) Boolean vectors, we will compare some common metrics that are intended for Boolean vector spaces. For example, we can find eight examples of such metrics in the Scikit-learn documentation [55].

When comparing the values of two Boolean vectors, we will notice that sometimes their values are equal (i.e. both True or both False) and sometimes their value is different (i.e. the value is True in one vector but False in the other vector or the other way around). We can use the following variables as counters to keep track of these different possibilities:

- N: the number of dimensions
- NTT : number of dimensions in which both values are True
- NTF : number of dimensions in which the first value is True, second is False
- NFT : number of dimensions in which the first value is False, second is True
- NFF : number of dimensions in which both values are False

Using these variables, we can list the eight distance metrics that we will use, together with their distance function [55, 63] in table 1:

Distance metric	Abbreviation	Distance function
Dice coefficient	*Dice*	$\frac{NTF+NFT}{2\times NTT+NTF+NFT}$
Jaccard similarity	*Jaccard*	$\frac{NTF+NFT}{NTT+NTF+NFT}$
Kulsinski similarity	*Kulsinski*	$1-\frac{NTT}{N+NTF+NFT}$
Matching coefficient	*Matching*	$\frac{NTF+NFT}{N}$
Rogers-Tanimoto similarity	*Rogerstanimoto*	$\frac{2(NTF+NFT)}{N+NTF+NFT}$
Russell-Rao similarity	*Russellrao*	$\frac{N-NTT}{N}$
Sokal-Michener similarity	*Sokalmichener*	$\frac{2(NTF+NFT)}{N+NTF+NFT}$
Sokal-Sneath similarity	*Sokalsneath*	$\frac{NTF+NFT}{NTT/2+NFT+NTF}$

Table 1: The distance metrics and their distance formulas

In this research, we will compare these different distance metrics and see which of them performs best, i.e. which distance metric results in the highest correctly classified profiles. Given the fact that Rogers-Tanimoto similarity and Sokal-Michener similarity have the same distance function, we expect them to have identical results.

We can call any of these distance metrics in our Python code as follows:

```
from sklearn.neighbors import DistanceMetric
...
#metricstr is a string containing the abbreviation (from the second column of the
    table) of any distance metric
def compareVectors(vector1, vector2, metricstr):
  dist = DistanceMetric.get_metric(metricstr)
  return dist.pairwise([vector1, vector2])[1][0] * -1
```

The exact implementation of these distance metrics themselves can be found on the Scikit-learn Github [63]. In the rest of this document, we will refer to these distance metrics using their abbreviation, specified in the second column of table 1.

6.8 Determining the profile of each PCAP

6.8.1 General idea

By using the earlier mentioned distance metrics on ASN vectors, we can now determine the similarity between two PCAP files. Our objective now is to use this principle to be able to derive what profile was activated when the LTE network traffic from a certain PCAP was generated. So if we have the PCAP `profile-2-repetition-5.pcap` for example, our goal is to be able to determine that this traffic belongs to profile 2. We already know this, since its file name reveals the answer, but we will only use this information for verification. When trying to determine the profile of a PCAP file, we will evaluate its similarity with all other PCAP files. We will use our knowledge of what profile was being used in the other PCAP files, such that we can aggregate the similarity per profile. As a result, each profile will have a certain score, which is the sum of the similarities between the current PCAP file (from which we are trying to determine the profile) and the PCAP files that belong to that profile. The profile with the highest score is most likely the profile that was also used in the current PCAP file. We will then use the information from the file name of the PCAP to verify whether the profile with the highest score does indeed match the profile described in the file name of the PCAP. If that is the case, we have correctly identified a profile and therefore tracked a user. The goal of this research is to discover for how many PCAP files we are able to correctly classify the profile using this method and to see which distance metric aids us the most in doing so.

6.8.2 Example

To demonstrate how we are going to use the ASN vectors in this process, we will present a simplified example with only 15 PCAP files (instead of our 200 PCAP files). In this example,

- we have 3 profiles and 5 repetitions.
  Therefore, we have 15 PCAP files.
- the PCAP files are named "PCAP [number of the PCAP]", where [number of the PCAP] is a number in the range 0-14.
- PCAP 0 to PCAP 5 belong to profile 0, PCAP 5 to PCAP 9 belong to profile 1 and PCAP 10 to PCAP 14 belong to profile 2.
- we have already generated an ASN vector for each of these PCAP files.
- we assign a name to one of the ASN vectors: `ASN vector A`.
  This ASN vector belongs to the PCAP file from which we are going to determine the profile.

Profile used in pcap file	Pcap filename	Similarity with ASN vector A	Sum of similarities per profile	Corrected sum of similarities
	PCAP-0	0.3		
	PCAP-1	0.41		
Profile 0	PCAP-2	0.45	2.26	2.26
	PCAP-3	0.61		
	PCAP-4	0.49		
	PCAP-5	0.78		
	PCAP-6	1		
Profile 1	PCAP-7	0.83	3.94	2.94
	PCAP-8	0.42		
	PCAP-9	0.91		
	PCAP-10	0.22		
	PCAP-11	0.53		
Profile 2	PCAP-12	0.32	1.42	1.42
	PCAP-13	0.15		
	PCAP-14	0.2		

Figure 17: ASN vector A has the highest average similarity with profile 1

Figure 17 shows the similarities we find when comparing ASN vector A with the ASN vectors of all the PCAPs. When we aggregate the similarities per profile, we see that profile 1 has the highest sum of similarities. This means that we now know that profile 1 was activated when the traffic of the PCAP of ASN vector A was generated.
However, we also notice a similarity of 1 with PCAP-6, which is suspicious. Apparently, ASN vector A is identical to the ASN vector of PCAP-6. This is most likely the case due to the fact that the PCAP of ASN vector A is in fact PCAP-6. So we have been comparing PCAP-6 to itself, which might explain why the score of profile 1 is significantly higher than the score of profile 0 and profile 2. To get an objective score, we should avoid comparing PCAP files with themselves. In other words, we should avoid comparisons with two ASN vectors that are 100% identical. While it could theoretically be possible that two distinct profiles have identical ASN vectors, we consider the likelihood of this scenario to be negligible. This is because recorded network traffic always slightly differs among different recordings, due to to the complex interplay of numerous variables that influence the generated traffic (such as apps being in a slightly different state than before). This means that we can safely assume that two fully identical ASN vectors belong to the same PCAP file. Whenever that happens, we will exclude their similarity from the sum of similarities per profile. In this case, that means that we will only use PCAP-5, PCAP 7, PCAP 8 and PCAP 9 to obtain an objective sum of similarities for profile 1. This is the 2.94 we see in the right most column in figure 17. But even then, profile 1 still has a significantly higher score than profile 0 and profile 2, which still makes us arrive at the same conclusion: profile 1 is the profile used in the PCAP of ASN vector A. Once we verify this, we will see that this correct, since ASN vector A belongs to PCAP-6, which does indeed belong to profile 1.

6.8.3 In our setup

As opposed to this example above, we will have 200 PCAPs files (from 20 profiles with 10 repetitions) instead of only 15. However, the method used in the example is identical to the method that we will be using to attempt to classify the profile that was used in each of those 200 PCAP files. We will exclude PCAP files with identical ASN vectors from comparisons, we will use the highest sum of similarities for our classification and we will verify our conclusion using the knowledge we already have (which is the profile that appears in the file name of the PCAP file).

6.8.4 Using shorter traces

In our setup, the data that has been recorded in each PCAP is 60 minutes of passive data. In our analysis, we will be comparing the ASNs that we found during those 60 minutes in each PCAP and see how this attack performs. Another aspect worth researching is how well this attacks performs with data shorter than 60 minutes. With that knowledge, we can conclude whether an attacker can also run its attack for a shorter period of time, making it more feasible to deploy. To this end, we will be taking portions of the PCAP files by comparing only the AS numbers that we find withing specific time intervals in the PCAPs. For example, to see how well this attack performs with only 30 minutes of data instead of 60 minutes, we could limit ourselves to only use the data in the first half of the PCAP for our attack. Another possibility is to use the second half of each PCAP for our attack. However, we expect a difference in performance between these two options. Figure 18 shows the amount of packets over time that we expect to see in each PCAP file. We see that most data is being sent at the beginning of each PCAP and then declines during the rest of the hour that we have recorded. The reason why we expect this, is because we have just activated the profile at the beginning of each recording. This means that whenever we start a recording, the apps inside that profile were just being enabled and opened. All of the 50 apps within the current profile are enabled and opened nearly simultaneously, with a negligible time gap between the opening of the first and the last app, as they are launched sequentially. At this point, they are re-establishing a communication with their servers, causing them to exchange more data than moments where the connection has already been established. Therefore, we expect to find more ASNs in the first half of the PCAP than in the second half, making our attack perform better if we take the first 30 minutes. "Better" means that we will have a higher amount of correct classifications. Therefore, when looking at smaller portions of our PCAP files, it is important to research how well our attack performs with different start times. If t_{start} denotes the start time of the interval that we will be focusing on, this means that we will be comparing the performance of our attack with $t_{start} = 0$ and $t_{start} = 1800$. Similarly, if we want to see how well our attack performs with just 15 minutes of data, we will compare the attacks for $t_{start} = 0$, $t_{start} = 900$, $t_{start} = 1800$ and $t_{start} = 2700$. And when we evaluate the performance of our attack with only 5 minutes of data, we will be comparing our attack for $t_{start} = 0$ and every t_{start} 300 seconds later, until $t_{start} = 3300$.

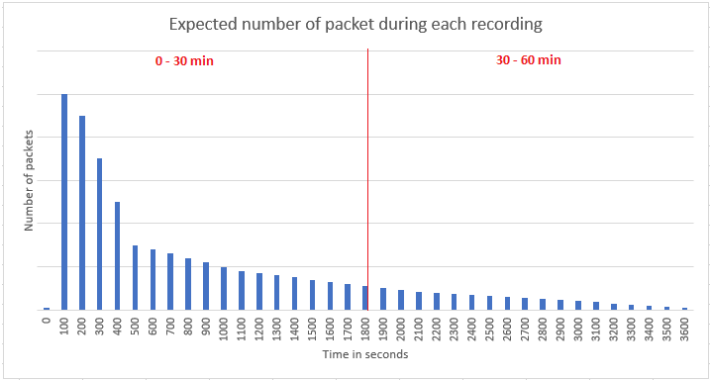


Figure 18: The expected declining number of packets over time in each recording

6.8.5 Accelerating our analysis

In figure 13, we have shown how we can extract IP addresses from the PCAP files, map them to ASNs and append them to a file called ASnumbers.py. However, the process of using TShark to extract the data from a PCAP file and the process of mapping all IP addresses to ASNs are both time-consuming processes. As described in section 6.8.4, we are also interested in using parts of the PCAP files to evaluate the success of the attack with a shorter recording time. This means that we have to repeat the time-consuming process of generating ASnumbers.py every time we want to use a different part each PCAP file. As an alternative, we could extract the relevant data from the PCAP files only once, and save it to a CSV file for each PCAP. So for every PCAP file, we will create an equivalent CSV with the following data for each packet: time the packet arrived, the source IP address, the destination IP addresses and the ASN of both addresses.
Once we have generated our 200 CSV files, we can use those to generate ASnumbers.py. This way, we never have to use TShark or convert IP addresses to ASNs again, as all the relevant data is already present in the CSV files. If we want to see how our attack performs with solely the data that we received during a specific time frame, we could decide to only extract the data in that specific time frame from the CSV files, in order to generate ASnumbers.py. We can then continue our analysis with that new version of ASnumbers.py. This significantly reduces the amount of time required for our analysis due to the fact that we only need to use TShark once, to generate the CSV files.

6.9 Overview of the entire analysis

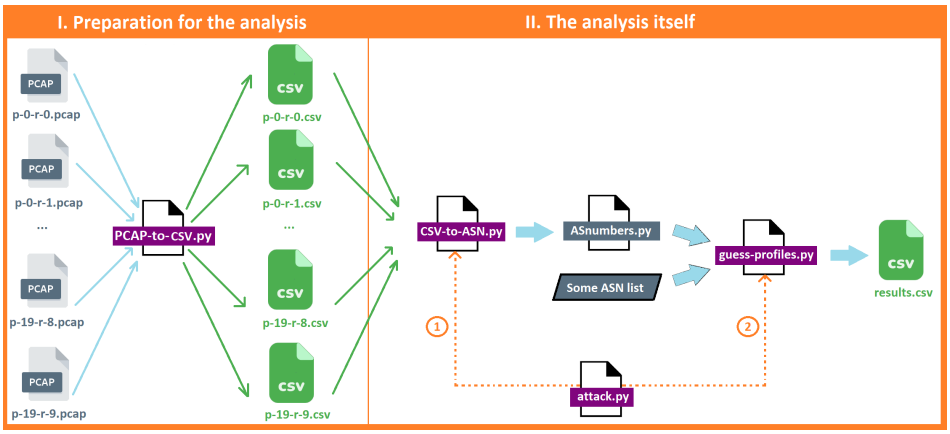


Figure 19: The different steps of the recorded LTE data in our analysis

Figure 19 gives us an overview of the different steps involved in our analysis of the PCAP files. It illustrates two phases in our analyses: phase I to prepare the analysis, and phase II in which the actual analysis takes place. The 200 PCAP files are shown on the left. The filenames displayed below the PCAP icons are shortened in this image, meaning that `p-0-r-0.pcap`, for example, is actual called `profile-0-repetition-0.pcap`. This is our first recording and our last recording is `profile-19-repetition-9.pcap`. We have the following steps in our analysis process.

(i) **Preparation for the analysis**

(a) Convert PCAPs to CSV files
As previously discussed in section 6.8.5, we save the extracted data from each PCAP in a CSV file, such that we only have to perform the time-consuming process of using TShark once. For each PCAP file, there will now be a CSV file with the relative arrival time of each packet, the IP addresses and the ASNs.
We do this with a Python script called PCAP-to-CSV.py.

(ii) **The analysis itself**

(a) Use (parts of) the CSV files to create ASN lists
Using a script called CSV-to-ASN.py, we will create the file ASnumbers.py similarly to how

we explained it earlier in section 6.5. The only difference is that we now extract the data from the CSV files instead of the PCAP files directly. We can either use all the data from the CSV files or only part of it, depending on the time period that we are trying to evaluate the attack for.
ASnumbers.py will now contain a list for each CSV file with the ASNs that we extracted from it, for the specified time interval.

(b) Running guess-profiles.py
Now that we have created the relevant data for our attack in the Python script ASnumbers.py, we can include that script in guess-profiles.py.
Our script guess-profiles.py can then use that data to guess the profile of an ASN list as described in section 6.8.
Figure 19 shows that we can input "Some ASN list" to guess-profiles.py. This list could either be

- one of the ASN lists that is also in ASnumbers.py
- a different ASN list.
  For example, a list of ASNs that we extracted from another PCAP that is not part of the 200 PCAP files that we used for ASnumbers.py.

Guess-profiles.py will then compute the similarities between the provided ASN list and the ASN lists found in ASnumbers.py to determine to which profile the provided ASN list most likely belongs.

Another way to use guess-profiles.py is as follows: rather than providing a separate ASN list to verify, we will iterate over ASnumbers.py and derive the profile for every ASN list in ASnumbers.py. ASnumbers.py also contains the profile and repetition number of each ASN list, which allows us to also verify whether we correctly determined each of the profiles for the ASN lists.
After using guess-profiles.py this way, we will have attempted to determine the profile of all 200 ASN lists in ASnumbers.py and we also verified those 200 attempts. This means that we know how many of those 200 classifications were correct, incorrect or unknown. We can then save these results to results.csv and repeat our analysis for a different part of each CSV file.

(c) The final contents of results.csv
After conducting our analyses, results.csv will contain the results of every analysis. Here is an example of how this file could look:

```
comment,interval,correct,incorrect,unknown
step=3600,0-3600,180,10,10
step=1800,0-1800,170,16,14
step=1800,1800-3600,161,24,15
step=900,0-900,156,27,17
step=900,900-1800,153,29,18
step=900,1800-2700,149,29,21
step=900,2700-3600,120,56,24
...
step=300,3300-3600,90,75,35
```

The first column contains a string for each analysis, specifying the length of the time interval that we used from each recording. "Step=3600" means that we used 3600 seconds of each recording, which is the entire 60 minute recording.
"Step=1800" means that we are using only half of the data of each CSV file (1800 seconds or 30 minutes). The interval column shows which half we used: "0-1800" (the first half) or "1800-3600" (the second half). Since we want to research the results of both halves, both of their results are included in this file.
Carrying forward with the notation, "step=300,3300-3600" means the last 5 minutes of each

recording. The other three values found on each line specify the results of the analysis: the amount of correctly classified profiles, incorrectly classified profiles and how many profiles we could not classify due to some error ("unknown").

In figure 19, we also see a Python script called attack.py. This script will call CSV-to-ASN.py with certain parameters that specify the part of the CSV files that we are going to use for our analysis. It will then call guess-profiles.py, that uses the ASnumbers.py that has just been created by CSV-to-ASN.py. Attack.py will repeat these steps with different parameters, to see how using different parts of the CSV files affect our attack. All results will be appended to results.csv, such that results.csv will look like described in point (ii)c above.

7 Results

In this section, we will present the results of our experiments. We have used different recording times in our experiment between 1 minute and 60 minutes. For each of these durations, we will show graphs that compare the performance of the different distance metrics that we used. We will start with the 60 minutes recording, since that is the entire duration of every PCAP file.
The raw data can also be found in Appendix E.

7.1 File sizes of our recordings

Since we have been recording 20 different profiles for 10 times, we now have 200 PCAP files. If we look at the file sizes of those 200 files, we can already observe remarkable patterns of interest.

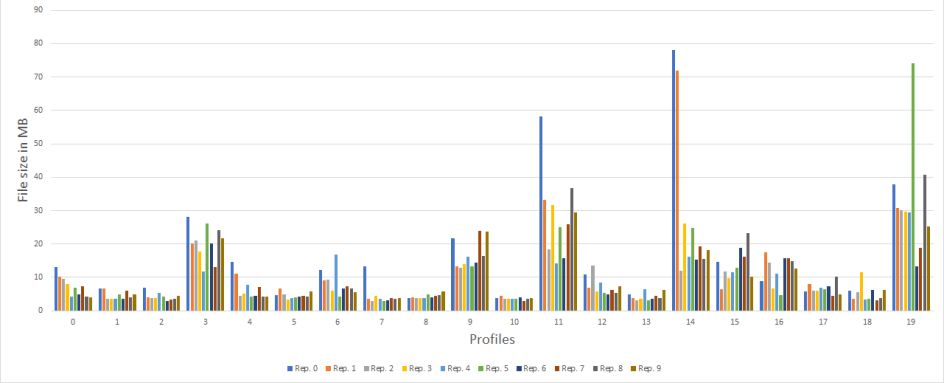


Figure 20: File sizes clustered by profile

Figure 20 shows the file size of each PCAP file. The files have been clustered by profile, such that each bar represents a repetition of that profile. For example, the yellow bar represents repetition 3 for each profile. Therefore, the yellow bar that appears above the number 0 represents the file `profile-0-repetition-3.pcap`.
This graph reveals differences between the profiles in bar height. For example, the bars of profile 3 are significantly higher than those observed for profile 2. This means that the file sizes for the PCAPs in profile 3 are greater than those in profile 2. In other words, the level of network traffic in profile 3 is significantly greater than that observed in profile 2. Apparently, the apps in profile 3 communicate significantly more than the apps in profile 2. Another example is profile 19, that shows higher bars than the rest of the profiles. Since this graph shows differences between profiles, that seem consistent among the different repetitions of each profile, the file sizes seem to be the first indicator that we can discover patterns in the LTE traffic that are specific to what apps are being used.

7.2 Using all 60 minutes of data from the PCAPs

Each PCAP file contains 60 minutes of recorded LTE network traffic. Using all of this network data, we have attempted to derive what profile was used in each PCAP file. Since we have 200 PCAP files,

we will also have 200 classifications. The graph in figure 21 shows the number of correct classified profiles out of those 200 PCAP files for each distance metric.

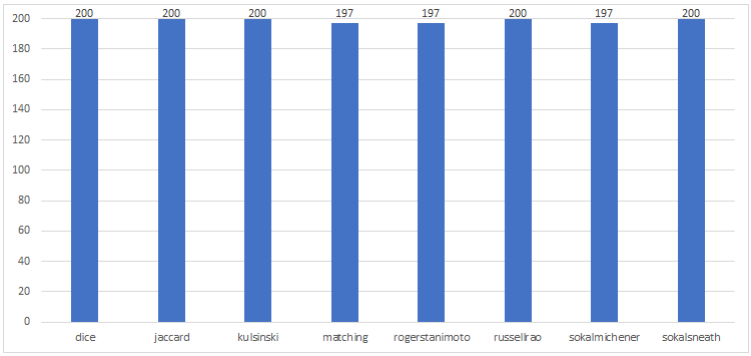


Figure 21: The amount of correctly classified profiles using different distance metrics (60 minutes of data)

We see that dice, Jaccard, Russellrao and Sokalsneath could correctly classify 200 out of 200 profiles (100%). Matching, Rogerstanimoto and Sokalmichener all classified 197 out of 200 profiles correctly (98.5%), which means that only a small amount of 3 classifications were incorrect.

7.3 30 minutes of data

To analyze a recording of 30 minutes of LTE data, we can use part of the 60 minute PCAP files that we already have. For example, we could take the first 30 minutes or the last 30 minutes (or something in between). We did both, which is why the graph in figure 22 shows two bars for each distance metric. The first bar shows the results when using the first 30 minutes of each PCAP (0-1800 seconds), while the second bar represents the results when taking the last 30 minutes of each recording (1800-3600 seconds).

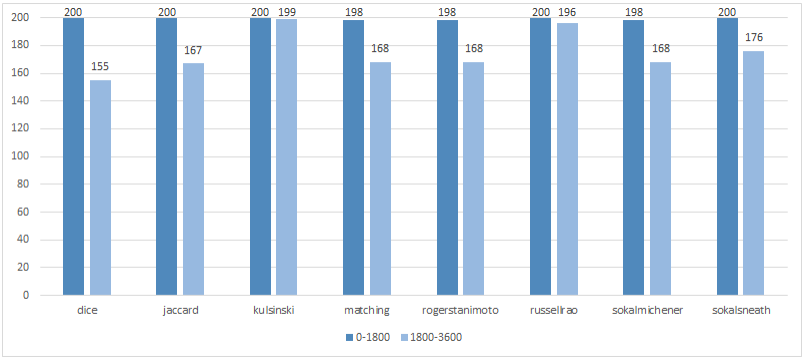


Figure 22: The amount of correctly classified profiles using different distance metrics (30 minutes of data)

Figure 22 shows similar results for both time intervals while using Kulsinski and Russellrao. For the other distance metrics, we observe a significant lower number of correct classifications when using the last half hour in each PCAP as opposed to using the first half.

7.4 20 minutes of data

To see how well our attack performs with only 20 minutes of LTE data for each recording, we can take a portion of 20 minutes from our 60 minutes recording. For example, we could take the first 20 minutes (0-1200 seconds), the last 20 minutes (2400-3600 seconds) or something in between (e.g. 1200-2400 seconds). We compare the results for each of those time intervals, while using different distance metrics.

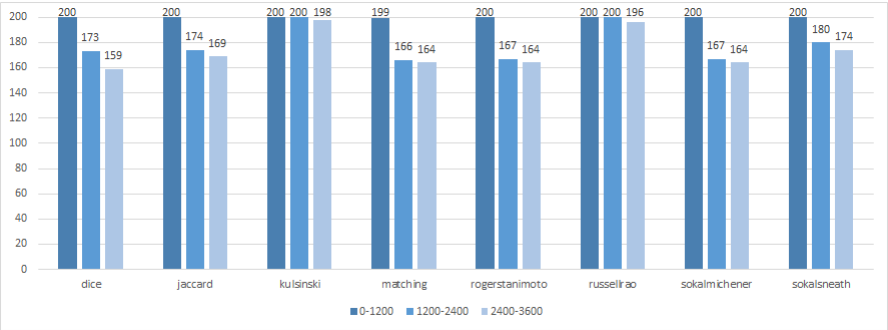


Figure 23: The amount of correctly classified profiles using different distance metrics (20 minutes of data)

Again, we only observe an almost negligible decline in correct classifications for Kulsinski and Russellrao and only when using the last 20 minutes of each recording. The other distance metrics show a notable decrease in accurate classifications, already starting from the second time interval.

7.5 10 minutes of data

To see how well our attack performs on 10 minutes of recorded LTE data, we will take portions of 10 minutes from our 60 minute recordings. We will compare the performance of our attack for using the first 10 minutes of each recording (0-600 seconds), the second 10 minutes (600-1200 seconds), the third 10 minutes (1200-1800 seconds), the fourth 10 minutes (1800-2400 seconds), the fifth 10 minutes (2400-3000 seconds) and the last 10 minutes (3000-3600 seconds). In the graph from figure 24, we omit the data labels on top of each bar to improve readability. Although the primary focus of our analysis is on differences in the bar heights, we refer to appendix E for the exact numerical values.

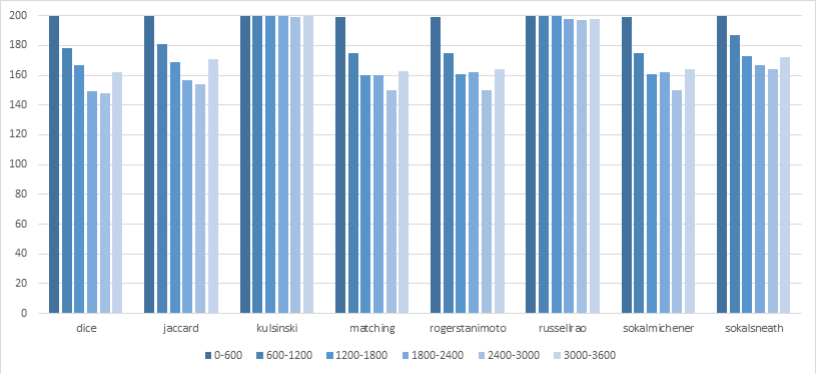


Figure 24: The amount of correctly classified profiles using different distance metrics (10 minutes of data)

We observe a significant decreasing trend for all distance metrics, except Kulsinski. The amount of correct classifications declines as our start time increases. However, this trend does not apply to the latest 10 minutes of each recording, as we observe a higher last bar than the bar before for each distance metric.

7.6 5 minutes of data

To use 5 minutes of recorded LTE data for our attack, we could divide our 60 minute recordings into 300 seconds time intervals. We can then compare the performance of those time intervals for each distance metric.

While we can still observe a decreasing trend for later time intervals of 300 seconds, we can also spot an increasing trend at the latest time intervals. This effect is more difficult to spot for Kulsinski and Russellrao. Those two distance metrics outperform the other distance metrics again. They perform

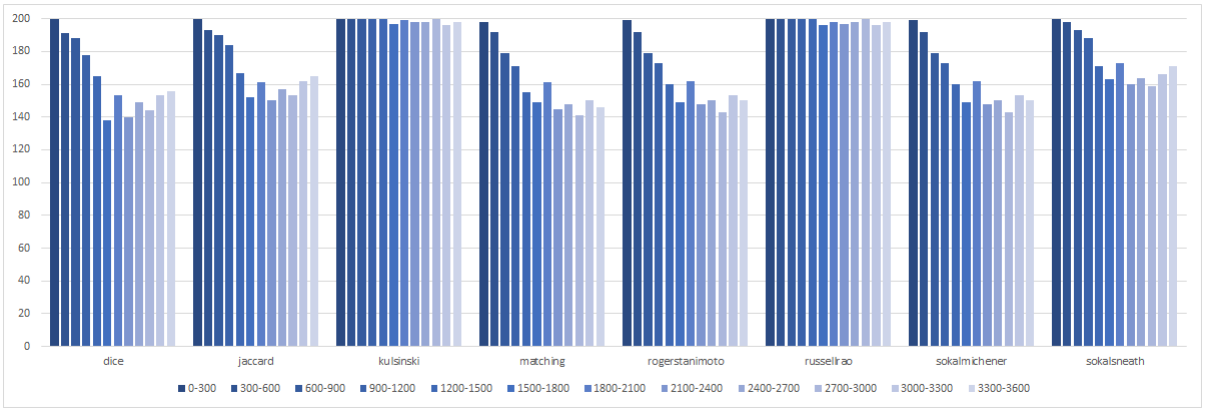


Figure 25: The amount of correctly classified profiles using different distance metrics (5 minutes of data)

almost equally, but appendix E reveals that Kulsinski does have one more correct classification than Russellrao in the time intervals 1200-1500, 1500-1800 and 1800-2100.

7.7 1 minute of data

We can also take as little as one minute of data from each of our 60 minutes recordings and evaluate the performance of our attack for only that one minute. If we do this for every minute in our recordings, we can compare the number of correct classifications for each of those one minute time intervals, for each distance metrics. For readability, we have omitted the legend of the graph, but each bar represents a one minute time interval: 0-60 seconds, 60-120 seconds, 120-180 seconds, ..., 3480-3540 seconds or 3540-3600 seconds.

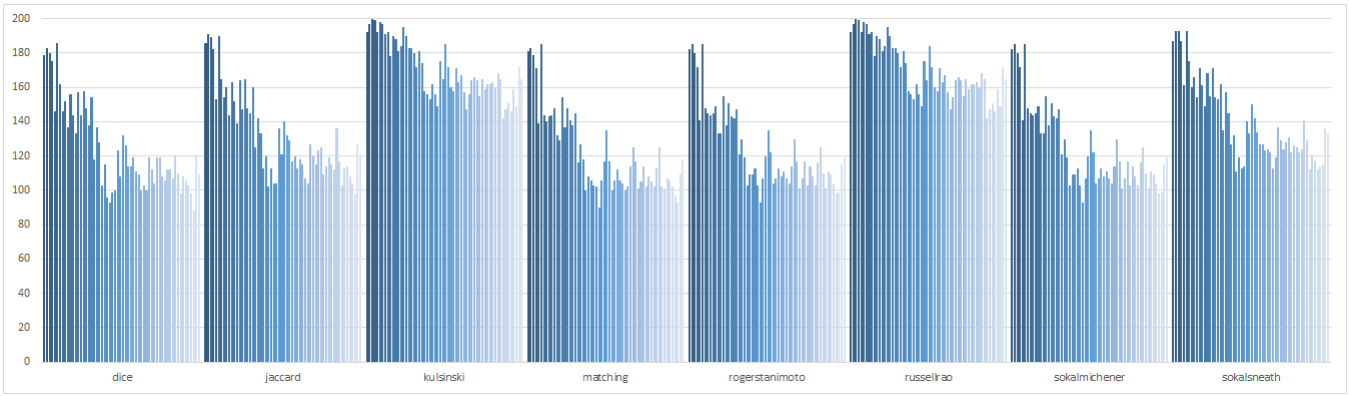


Figure 26: The amount of correctly classified profiles using different distance metrics (1 minute of data)

While Kulsinski and Russellrao still have the highest number of correct classifications for each one minute time interval, we see a significant decreasing trend for every one minute time interval in the first half of our 60 minutes recording. In the second half, the numbers go up and down, but the trend is not as clear. Kulsinski and Russellrao perform almost identical, except for the time intervals 1740-1800, 1800-1860 and 2340-2400 where Kulsinski performs slightly better.

8 Discussion

The results of this research show that it is possible to correctly classify the profile being used in each PCAP file in 200 out of 200 cases (100%), when using 60 minutes of recorded LTE data. We also observe that a reduction in recording time of LTE data is accompanied by a decline in the corresponding

success rate of our attack. We evaluated the success rate of a shorter recording time by only using the data within a certain time interval and we observed that it depends on both the start time of the time interval and the distance metric that we used. A later start time means that there is a greater time difference between the moment that the profile was activated and the start of our recording. We observed a negative correlation between this time difference and the success rate of our attack, indicating that a larger time difference results in a lower success rate. If we look at the I/O graph of any randomly chosen PCAP file in Wireshark, such as `profile-14-repetition-3.pcap` in figure 27, we can see that that most data is being sent at the beginning of the PCAP.

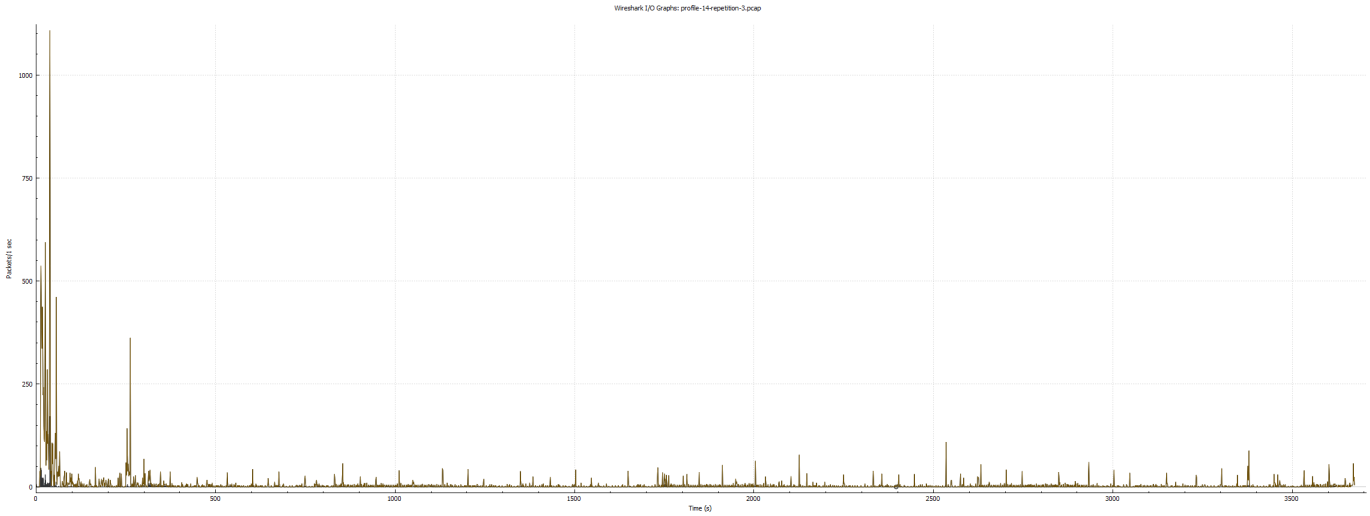


Figure 27: The I/O graph of profile 14, repetition 3

The reason why this happens, is likely due to the fact that the beginning of each PCAP file is also the moment that a profile has just been activated. This offers the apps in that profile the first opportunity to synchronize their data with the servers they communicate with. This observed peak in data at the beginning corresponds with our prediction from section 6.8.4 and could explain why a larger time difference between the moment of profile activation and the start of our recording results in a lower success rate: as more data is being sent at the beginning, we can use more data for our classification when we choose a time interval closer to the moment that the profile was being activated. Since more data leads to more distinctive patterns, we can better distinguish between different profiles.
However, in a real world scenario, it is unlikely that we can take into account when a profile has been activated (e.g. when a user has turned on their phone) while recording LTE data. Therefore, we consider time intervals further away from the moment of the profile activation to be more representative for the results in a real world scenario where an attacker records passive LTE traffic.

Nevertheless, the results also reveal that we can reduce this effect of a declining success rate for shorter time periods by choosing the right distance metric. Kulsinski performs best in all of our experiments, followed by Russellrao. If we look at how Kulsinski performs for 30 minute, 20 minute and 10 minute recordings of LTE data, we notice that there is only one example for each of those durations that result into 198/200 (99%) or 199/200 (99.5%) correct classifications. All of the other examples still result into 200/200 (100%) correct classifications, regardless of which of these time duration was used. If we only use 5 minutes of recorded LTE data for our attack, Kulsinski still manages to correctly classify 196/200 (98%) in the worst case scenario (i.e. time interval 3000-3300 seconds). All of the other 5 minute time intervals have a higher success rate and 50% of them even manage to still correctly classify the profiles in all 200 recordings.
Finally, when using only 1 minute of recorded LTE data, we observe significant lower numbers of correct classifications for most time intervals, even while using Kulsinski. When we look at the amount of correctly classified profiles for each one minute time interval for Kulsinski, we see an average of 171/200 (85%) and a median of 166/200 (83%). In the worst case scenario (time interval 3120-3180 seconds), we can still correctly classify 142/200 (71%). This means that, while the success rate is significantly lower than in our experiments where we used greater time intervals, the attack is still successful in the

majority of cases.

The attack setup that we used closely resembles the attacker scenario of an attacker using a fake base station, as described section 2. Therefore, our achieved results show that it is possible for an attacker to profile users based on the passive LTE traffic that the apps on their phone generate. This means that if an attacker decides to execute this attack at multiple locations, he is able to recognize any user that is connected to his fake base station at any location before. Based on that information, he could compile a list of locations where he encountered this user. This constitutes a major violation of privacy, as it allows an attacker to profile users by engaging in activities such as

- getting information about a persons location
- tracking users by creating logs of their location history
- deriving their interests from their location history

If this attack is being deployed at a wide range of locations, great amounts of data can be collected, further increasing the chances of observing the same users at multiple locations. The hardware required to do so, can be obtained relatively easily. While we used a USRP B205 mini-i with a laptop, an attacker could also use the relatively inexpensive LimeSDR 2.0 mini together with a Raspberry Pi SBC. The affordability of this hardware, coupled with the significant results we achieved in this attack, substantiates the feasibility of this attack. Even though the results for 1 minute time intervals are significantly lower than the results we observed for greater time intervals, this is still a great cause for concern, given the short amount of recording time to achieve these results. Any recording time added to this 1 minute, will already result in an even higher success rate for the attack, quickly approaching the 100%. This unveils the pressing need to take appropriate steps by involved parties, in order to mitigate the risk of falling victim to this attack.

8.1 Detection of our attack

Our attack consists of two phases: capturing the LTE network traffic and performing the analysis on the data. Since the analysis of the intercepted traffic can be done offline, the possibility of detecting the attack is during the recording phase. We eavesdrop using a fake base station to set up a MitM between the connected users and the legitimate eNodeB. Therefore, detection of our attack is similar the detection of fake LTE base stations in general. We refer to the existing body of research on that topic. Examples of the various detection methods that are already available include but are not limited to:

- Monitoring abnormalities in signal strength [18, 21]
- Machine Learning techniques based on Reference Signal Received Power (RSRP) [20]
- Synchronization of network names of legitimate base stations with the UE [19]
- Sample-based higher order noise statistics [22]
- Network-based detection [23]

While these methods are available and some of them (e.g. the last one) are already adopted by the 3GPP standardization organization, detecting a fake base station can still be challenging task and should therefore not be solely relied on as a countermeasure. Therefore, we present additional countermeasures below.

8.2 Countermeasures

Since our attack relies on the ability to extract IP addresses from the intercepted network traffic, a straightforward countermeasure would be to encourage the usage of Virtual Private Networks (VPNs) among mobile network users. Since a VPN reroutes all traffic to a VPN server, an attacker with a malicious eNodeB would only see the IP addresses of the VPN server and the user. As a result, the

conversion of the IP addresses to AS numbers will only result in AS numbers that belong to the VPN, which renders it impossible to distinguish different profiles based on their (dis)similarity of AS numbers. While our attack unveils a substantial threat to privacy, VPNs are a practical and accessible solution, given their wide available and the simplicity of their setup. Therefore, it is important to raise awareness of both the feasibility of this attack and the accessibility of its corresponding solution by promoting the usage of VPNs among mobile network users. However, the usage of a VPN or similar services can also introduce new privacy hazards. While a VPN provider can protect users against attackers using a malicious base station, the VPN server itself could act as a malicious base station as well. Since all traffic is routed throught the VPN server, this enables the VPN provider to perform the same attack as the one it claims to protect the user from. In that case, a VPN is not a protection mechanism, but a means to facilitate the actual attack. Therefore, encouraging users to choose a trusted VPN party is of equal importance. In that light, a user might consider using a VPN from a party with whom they are already sharing their data, such as their phone manufacturer (e.g. Google VPN or Apple Relay). They could also decide to make use of similar solutions, such as proxy servers and Tor, but these solutions should also be approached with the same caution. While the possibility of our attack is dependent on the ability to extract IP addresses from the network traffic, it is important to realize that an attacker might consider implementing this attack in a slightly different way by extracting other data than IP addresses from the network traffic. In that case, the usage of a VPN like service is not an effective remedy for the attack. This makes it important to remain vigilant and cognizant about potential variations of the attack, in the event that they are being deployed. This enables remaining receptive to investigating alternative countermeasures, depending on what data the attacker extracts. Using end-to-end encryption on the IP layer such as IPsec is generally recommended, but only if being be used in tunnel mode in order to hide the IP addresses.

8.3 Other limitations and future research

While the results of this research provide valuable insights, it is important to also acknowledge other limitations besides the earlier mentioned accessible mitigation methods. For example, the limitation of using IP addresses from the PCAP files forces us to assume that the attacker has access to the IP layer of packet and therefore also to the PDCP layer. This implies that the packets are not encrypted, and thus this attack is not performed as a passive sniffer, as that would require to break the encryption first. To make up for this, the attacker acts as a MitM using a rogue base station, which allows him to see the unencrypted packets. However, it is worth noting that this approach requires more effort, as it involves the creation of a custom base station and active meddling with users by tricking them to connect to it. Therefore, it would be a good idea for future research to focus on different parameters than IP addresses, to investigate whether this attack could also be performed as a passive sniffer. This could amplify the impact and dangers of the attack even further.

Another problem could be the fact that we presume the profiles to be static. As profiles consist of a selection of apps, this assumption translates to disregarding the fact that users will keep adding and removing apps through time. In this research, we show that different sets of apps lead to different traffic patterns, allowing us to distinguish profiles. Consequently, if the apps inside a profile itself change, it will also lead to different traffic patterns and therefore become a different profile. Moreover, apps themselves are constantly being updated. This potentially leads to different requests with other AS numbers, which might also influence traffic patterns. As this gives profiles more dynamic and evolving characteristics, it is likely that concept drift will influence our ability to identify users, particularly when comparing new versions of profiles to old profile data. This highlights the importance of regularly updating the dataset in order to mitigate the impact of concept drift and remain the attack's accuracy over time.

Another issue to consider is the costs of the SDR. The SDR that we are using is currently being listed for $1,571.00 [33]. A company exploiting our attack (e.g. a mobile service provider) might not be financially prohibited by this, but a typical teenage hacker is unlikely to have sufficient financial means. For an individual falling into this last category, it might be more realistic to purchase the more cost-effective LimeSDR 2.0 mini, currently being listed for $399 [56], but only if the attack is being performed at one location at a time. Deploying this attack at multiple locations simultaneously would require to place such an SDR at each of those locations, all adding up to an impractical total costs. This, while being limited to one location at a time might significantly reduce the feasibility of the attack, depending on its purpose. Performing this attack as a passive sniffer rather than using a fake

base station (as suggested earlier) is not a solution for this unaffordability either, as passive sniffing is often also done using SDRs.
The scalability of the set up in general might also be limited apart from the costs, due to our analysis method. In our analysis, we demonstrate the feasibility of recognizing the profile in a PCAP file, based on its similarity with every other PCAP files. This method is feasible within our lab environment, due to the fact that our setup is a simplification of a real-world scenario. In our a closed world, 20-class problem, we only use 200 PCAP files. However, in a real-world scenario where the attack is being applied on a large scale, there will be significantly greater amounts of data, rendering pairwise comparison with all other recordings impracticable. Moreover, the attacker would need to cope with millions of profiles. In that case, different methods should be applied to determine the ASN-based similarity with other PCAP files. For example, Machine Learning (ML) classification techniques could be used to train the ASN vectors for each profile and then predict the profile of any new recording based on its ASN vector. We leave the implementation of such alternative comparison techniques to future work.

8.4 5G

While our attack focuses on LTE, it is important to note that its applicability is likely to extend to 5G as well, due to both the architectural similarities between both protocols and the principle of our attack. In our attack, we demonstrate that the traffic generated by apps is very specific for the apps that are used. Based on the 5G specifications [57] and its similarity with LTE, we do not have reasons to assume that this concept will be affected by any differences between the protocols. However, it is not unlikely that changes in the 5G protocol impact the attacker's ability to intercept the mobile network traffic. This is because of the enhanced protection mechanisms against IMSI catchers in 5G, which may introduce challenges in setting up a fake base station. Examples of such mechanisms include the Elliptic Curve Integrated Encryption Scheme (ECIES) and SUPI concealment, guaranteed GUTI refreshment, protected redirections, a general informative detection framework, authentication between UE and network, integrity protected signalling and secure algorithm negotiations [58]. Nonetheless, it is not impossible [59, 60, 61]. Alternatively, as suggested earlier, an attacker could explore the possibilities of extracting different data from the network traffic, which does not rely on the fact that the traffic is not encrypted. This would allow an attacker to perform the attack as a passive sniffer, eliminating the need to set up a fake base station and thereby circumventing the enhanced security mechanisms. For example, an attacker could focus on metadata of encrypted mobile network traffic, similar to e.g. [1] or [5].
While our findings together with these suggestions strongly indicate that 5G is not immune to our attack, we leave the exact implementation of a 5G compatible version of our attack to future work.
It is also important to note that, even if an attacker seems unsuccessful in using this attack in 5G, he might still be able to deploy a downgrade attack [62] in spite of 5G's lack of backwards compatibility. This will force the user to connect to the fake base station using LTE, making them vulnerable to the attack despite their 5G subscription.

9 Conclusion

In this thesis, we explored the possibility of using passive LTE network traffic as a way to fingerprint mobile users in order to track them. We described the attack in a real world scenario and then demonstrated how we created an experimental set up. We generated 20 profiles that each simulated mobile users with a different set of apps installed, who are not actively using their phone. We then recorded the resulting network traffic of each of those profiles 10 times, for 1 hour per recording, resulting in a total of 200 hours of recorded network traffic. The objective of this study was to evaluate the effectiveness of an attack in which we attempt to derive the profile that was being used in each of those recordings by comparing the similarity with the other recordings. In order to compare the recordings, we created "ASN vectors" for each of the recordings, based on the IP addresses that we found. We evaluated the attacks for different distance metrics and different durations. The distance metric Kulsinski performed best for all durations, followed by Russellrao. We discovered that we could correctly derive the profile that was being used in all 200 recordings when using the entire hour of recorded data in each PCAP file. To examine the success rate of our attack with a shorter recording

time, we used part of the 1 hour recordings we already had. We noticed that the success rate depends on which part we used. It decreased as we picked a later start time for the time interval that we extracted from each PCAP file. However, by using Kulsinski we could still derive the correct profile for at least 98% of the profiles while using only 5 minutes of data. This was in the worst performing example, while all other examples have higher scores, especially for the longer time intervals. For each time intervals longer than 5 minutes, we could only find one example that did not result in correctly classifying 100% of the profiles. For 1 minute time intervals, we saw a success rate of 85% on average and 71% in the worst case scenario. In a real world scenario, this translates to an attacker using a malicious eNodeB that only requires users to be connected for 60 seconds in order to achieve those results. Any additional time that users will stay connected, will result into significantly higher success rates, allowing the attacker to track users at multiple locations. This is a major privacy concern, especially if being applied at large scale. We proposed possible countermeasures, such as increasing awareness, encouraging VPN usage and the detection of fake base stations. While these countermeasures provide valuable mitigation tactics, it is crucial to understand that these tactics are not immune to circumvention due to their inherent limitations. They should therefore be approached with caution. This also applies to 5G, whose susceptibility to our attack is strongly indicated by the architectural similarities it shares with LTE and the potential for bypassing any enhanced security measures. It is important to stay vigilant of potential variants of the attack that do not rely on the ability to extract IP addresses, as this would allow an attacker to perform the attack as passive sniffer, making it even more feasible to deploy. Future research is needed to examine the possibility of such variants, as well as to explore additional countermeasures for the attack.

References

[1] Kohls, Katharina, et al. "Lost traffic encryption: fingerprinting lte/4g traffic on layer two." Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks. 2019.

[2] Gijón, Carolina, et al. "Encrypted Traffic Classification Based on Unsupervised Learning in Cellular Radio Access Networks." IEEE Access 8 (2020): 167252-167263.

[3] Espinal, Albert, Rebeca Estrada, and Carlos Monsalve. "Traffic analysis of internet applications on mobile devices over LTE and wireless networks." Revista Ibérica de Sistemas e Tecnologias de Informação E22 (2019): 81-94.

[4] Trinh, Hoang Duy, et al. "Mobile Traffic Classification through Physical Control Channel Fingerprinting: a Deep Learning Approach." IEEE Transactions on Network and Service Management (2020).

[5] L. Zhai, Z. Qiao, Z. Wang and D. Wei, "Identify What You are Doing: Smartphone Apps Fingerprinting on Cellular Network Traffic," 2021 IEEE Symposium on Computers and Communications (ISCC), 2021, pp. 1-7, doi: 10.1109/ISCC53001.2021.9631415.

[6] APP ANNIE. "*State of Mobile 2020.*" Data.ai, accessed 14 March 2023. www.data.ai/download/en/insights/market-data/state-of-mobile-2020/2001_State_of_Mobile_2020_Main_EN.pdf

[7] Lexi Sydow and Sam Cheney. "*2017 Retrospective: A Monumental Year for the App Economy.*" Data.ai, accessed 14 March 2023. www.data.ai/en/insights/market-data/app-annie-2017-retrospective/

[8] Holtmanns, Silke, Siddharth Prakash Rao, and Ian Oliver. "User location tracking attacks for LTE networks using the interworking functionality." 2016 IFIP Networking conference (IFIP Networking) and workshops. IEEE, 2016.

[9] Trogh, Jens, et al. "Outdoor location tracking of mobile devices in cellular networks." EURASIP Journal on Wireless Communications and Networking 2019.1 (2019): 1-18.

[10] Margolies, Robert, et al. "Can you find me now? Evaluation of network-based localization in a 4G LTE network." IEEE INFOCOM 2017-IEEE Conference on Computer Communications. IEEE, 2017.

[11] Ni, Lei, et al. "Accurate localization using LTE signaling data." 2017 IEEE International Conference on Computer and Information Technology (CIT). IEEE, 2017.

[12] Ye, Xiaokang, et al. "Neural-network-assisted UE localization using radio-channel fingerprints in LTE networks." Ieee Access 5 (2017): 12071-12087.

[13] Shaik, Altaf, et al. "Practical attacks against privacy and availability in 4G/LTE mobile communication systems." arXiv preprint arXiv:1510.07563 (2015).

[14] Yu, Chuan, Shuhui Chen, and Zhiping Cai. "Lte phone number catcher: A practical attack against mobile privacy." Security and Communication Networks 2019 (2019): 1-10.

[15] Mjølsnes, Stig F., and Ruxandra F. Olimid. "Easy 4G/LTE IMSI catchers for non-programmers." Computer Network Security: 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings 7. Springer International Publishing, 2017.

[16] Van Rijsbergen, Kenneth. "The effectiveness of a homemade IMSI catcher build with YateBTS and a BladeRF." University of Amsterdam 28 (2016).

[17] Redactie Prisewise. "*Nederlander hamstert massaal mobiele telefoons.*" Pricewise.nl, 1 july 2021. https://www.pricewise.nl/blog/nederlander-hamstert-massaal-mobieltjes/

[18] Quintin, Cooper. "Detecting Fake 4G LTE Base Stations in Real Time." (2021).

[19] Al Mazroa, Alanoud, and Mohammed Arozullah. "Securing the user equipment (UE) in LTE networks by detecting fake base stations." (2015).

[20] Nakarmi, Prajwol Kumar, Jakob Sternby, and Ikram Ullah. "Applying Machine Learning on RSRP-based Features for False Base Station Detection." Proceedings of the 17th International Conference on Availability, Reliability and Security. 2022.

[21] Ali, Arslan, and Georg Fischer. "Symbol based statistical RF fingerprinting for fake base station identification." 2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA). IEEE, 2019.

[22] Ali, Arslan, and Georg Fischer. "Enabling fake base station detection through sample-based higher order noise statistics." 2019 42nd International Conference on Telecommunications and Signal Processing (TSP). IEEE, 2019.

[23] Nakarmi, Prajwol Kumar, et al. "Murat: Multi-RAT false base station detector." arXiv preprint arXiv:2102.08780 (2021).

[24] Roger Piqueras Jover. 2016. LTE Security, Protocol Exploits and Location Tracking Experimentation with Low-Cost Software Radio. arXiv (1607.05171) (2016). arXiv:1607.05171 http://arxiv.org/abs/1607.05171

[25] Altaf Shaik, Ravishankar Borgaonkar, N. Asokan, Valtteri Niemi, and Jean-Pierre Seifert. 2016. Practical Attacks Against Privacy and Availability in 4G/LTE Mobile Communication Systems. In Network and Distributed System Security Symposium (NDSS '16). Internet Society, San Diego, CA, USA

[26] FarhanF M. Aziz, Jeff S. Shamma, and Gordon L. Stüber. 2015. Resilience of LTE Networks Against Smart Jamming Attacks: Wideband Model. In Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '15). IEEE, Hong Kong, China, 1344–1348

[27] Roger Piqueras Jover. 2013. Security Attacks Against the Availability of LTE Mobility Networks: Overview and Research Directions. In International Symposium on Wireless Personal Multimedia Communications (WPMC '13). IEEE, Atlantic City, NJ, USA

[28] Marc Lichtman, Roger Piqueras Jover, Mina Labib, Raghunandan Rao, Vuk Marojevic, and Jeffrey H. Reed. 2016. LTE/LTE-A Jamming, Spoofing, and Sniffing: Threat Assessment and Mitigation. IEEE Communications Magazine 54, 4 (April 2016), 54–61.

[29] Marc Lichtman, Jeffrey H. Reed, T. Charles Clancy, and Mark Norton. 2013. Vulnerability of LTE to Hostile Interference. In IEEE Global Conference on Signal and Information Processing (GlobalSIP '13). IEEE, Austin, TX, USA, 285–288.

[30] R. Ghannam, F. Sharevski and A. Chung, "User-targeted Denial-of-Service Attacks in LTE Mobile Networks," 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Limassol, Cyprus, 2018, pp. 1-8, doi: 10.1109/WiMOB.2018.8589140.

[31] Fei, Teng, and Wenye Wang. "LTE is vulnerable: Implementing identity spoofing and denial-of-service attacks in LTE networks." 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019.

[32] Jermyn, Jill, Gabriel Salles-Loustau, and Saman Zonouz. "An analysis of dos attack strategies against the lte ran." Journal of Cyber Security and Mobility (2014): 159-180.

[33] Ettus Research. "*USRP B205mini-i.*" Ettus.com, accessed 9 February 2023. https://www.ettus.com/all-products/usrp-b205mini-i/

[34] Xiaomi. "*POCO F2 Pro.*" Mi.com, accessed 9 February 2023. https://www.mi.com/nl/poco-f2-pro/

[35] Sysmocom. "*sysmoISIM-SJA2 SIM + USIM + ISIM Card (10-pack) with ADM keys.*" Sysmocom.de, accessed 10 February 2023. https://shop.sysmocom.de/sysmoISIM-SJA2-SIM-USIM-ISIM-Card-10-pack-with-ADM-keys/sysmoISIM-SJA2-10p-adm

[36] HID. "*HID OMNIKEY 3121.*" Hidglobal.com, accessed 10 February 2023. https://www.hidglobal.com/products/omnikey-3121

[37] Osmocom. "*Osmocom/pysim: A python tool to program SIMs / USIMs / ISIMs.*" Github, accessed 10 February 2023. https://github.com/osmocom/pysim

[38] openLTE Wiki, bwojtowi and dchard. "*Programming you own USIM card.*" Sourceforge, accessed 29 February 2023. https://sourceforge.net/p/openlte/wiki/Programming%20you%20own%20USIM%20card/

[39] srsRAN Project. "*Open Source RAN.*" srsRAN.com, accessed 9 February 2023. https://www.srsran.com/

[40] Ettus Research. "*UHD (USRP Hardware Driver™).*" Ettus.com, accessed 10 February 2023. https://www.ettus.com/sdr-software/uhd-usrp-hardware-driver/

[41] Ettus Research. "*USRP Hardware Driver and USRP Manual.*" Ettus.com, accessed 10 February 2023. https://files.ettus.com/manual/page_install.html

[42] srsRAN Project. "*Installation Guide - srsRAN 4G 23.04 documentation.*" srsRAN.com, accessed 12 February 2023. https://docs.srsran.com/projects/4g/en/latest/general/source/1_installation.html

[43] srsRAN Project. "*COTS UE - srsRAN 4G 23.04 documentation.*" srsRAN.com, accessed 12 February 2023. https://docs.srsran.com/projects/4g/en/latest/app_notes/source/cots_ue/source/index.html

[44] AndroidRank. "*Most installed android apps.*" Androidrank.org, accessed 13 September 2022. https://www.androidrank.org/applist.csv

[45] Google. "*Android Debug Bridge (adb).*" developers.android.com, accessed 11 June 2023. https://developer.android.com/tools/adb

[46] Tim Wijkman-van Aalst. "*We moeten nog minimaal een jaar wachten op het échte 5G.*" NU.nl, 26 May 2023. https://www.nu.nl/tech/6264962/we-moeten-nog-minimaal-een-jaar-wachten-op-het-echte-5g.html

[47] Vadim Mikhailov. "*GitHub - mvp/uhubctl: uhubctl - USB hub per-port power control.*" Github, accessed 16 March 2023. https://github.com/mvp/uhubctl

[48] Google. "*Manifest.permission.*" developers.android.com, accessed 14 March 2023. https://developer.android.com/reference/android/Manifest.permission

[49] Longsworth, Ralph C., and WILLIAM A. Steyert. "Technology for liquid-nitrogen-cooled computers." IEEE transactions on electron devices 34.1 (1987): 4-7.

[50] Scapy. "*Welcome to Scapy's documentation!.*" readthedocs.io, accessed 27 October 2021. https://scapy.readthedocs.io/en/latest/

[51] Wireshark. "*tshark(1) Manual Page.*" Wireshark.org, accessed 5 November 2021. https://www.wireshark.org/docs/man-pages/tshark.html

[52] Piotr (pgawlowicz). "*srsRAN_4G/enb.conf.example at master · srsran/srsRAN_4G · GitHub.*" Github, 6 April 2023. https://github.com/srsran/srsRAN_4G/blob/master/srsenb/enb.conf.example

[53] Pypi. "*MaxMind GeoIP2 API.*" Pypi.org, accessed 26 April 2023. https://pypi.org/project/geoip2/

[54] Maxmind. "*GeoLite2 ASN Database.*" Maxmind.com, accessed 26 April 2023. https://dev.maxmind.com/geoip/docs/databases/asn

[55] Scikit Learn. "*sklearn.neighbors.DistanceMetric.*" Scikit-learn.org, accessed 5 January 2023. https://scikit-learn.org/0.24/modules/generated/sklearn.neighbors.DistanceMetric.html

[56] Lime Microsystems. "*LimeSDR Mini 2.0.*" CrowdSupply, accessed 2 June 2023. https://www.crowdsupply.com/lime-micro/limesdr-mini-2#products

[57] 3GPP. "*5G System Overview.*" 3GPP.org, accessed 5 June 2023. https://www.3gpp.org/technologies/5g-system-overview

[58] 3GPP. Specification # 33.809: Study on 5G security enhancements against False Base Stations (FBS) (2018)

[59] Chlosta, Merlin, et al. "5G SUCI-Catchers: Still catching them all?." Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks. 2021.

[60] Lee, Meng Huan, I-Hsien Liu, and Jung Shian Li. "Fake Base Station Threats in 5G Non-Public Networks." 28th International Conference on Artificial Life and Robotics, ICAROB 2023. ALife Robotics Corporation Ltd, 2023.

[61] Ali, Hamza, et al. "Protecting IMSI from Fake Base Stations Exploitation and Spoofers Impersonation in 5G and Beyond Cellular Networks." (2022).

[62] Khan, Mohsin, et al. "Defeating the downgrade attack on identity privacy in 5G." Security Standardisation Research: 4th International Conference, SSR 2018, Darmstadt, Germany, November 26-27, 2018, Proceedings 4. Springer International Publishing, 2018.

[63] Scikit. "*Metrics: _dist_metrics.pyx.tp.*" Github, accessed 18 June 2023. https://github.com/scikit-learn/scikit-learn/blob/main/sklearn/metrics/_dist_metrics.pyx.tp

[64] Harris, David and Harris, Sarah (2012-08-07). Digital design and computer architecture (2nd ed.). San Francisco, Calif.: Morgan Kaufmann. p. 129. ISBN 978-0-12-394424-5.

A Code samples for the research setup

A.1 Simplified apps installation script

We present a simplified example of an installation script below. This script offers a way to automate the installation of the 500 apps through the Google Play store.

```
#Function that can execute system calls
def systemCmd(commandstr):
    output = subprocess.check_output(commandstr, shell=True, stderr=subprocess.STDOUT)
    return output

#Emulates a tap of the user at the provided coordinates
def tap(coordinates):
    x, y = coordinates
    tapstring = "adb shell input tap " + str(x) + " " + str(y)
    systemCmd(tapstring)

#Opens a specific app in the Google Play store, using the provided packagename
def openAppInPlayStoreByPackageName(packagename):
    commandstr = "adb shell am start -a android.intent.action.VIEW -d 'market://
    details?id=" + packagename + "'"
    systemCmd(commandstr)

#Gets the coordinates of a button, based on a string
def getButtonPosition(buttontext):
    #Based on https://stackoverflow.com/questions/18924968/using-adb-to-access-a-
    particular-ui-control-on-the-screen

    #Create a view
    errorhider = " 2>/dev/null" if not displayerrors else ""
    cmd1 = "adb pull \$(adb shell uiautomator dump" + errorhider + " | grep -oP '[^
    ]+.xml') /tmp/view.xml"

    #Get position of button based on text of the button
    cmd2 = "perl -ne 'printf \"%d %d\n\", ($1+$3)/2, ($2+$4)/2 if /text=\"" +
    buttontext + "\"[^>]*bounds=\"\[(\d+),(\d+)\]\[(\d+),(\d+)\]\"/' /tmp/view.xml"

    systemCmd(cmd1)
    time.sleep(1)
    position = systemCmd(cmd2)

    return position.decode()

#Installs the app that is provided by the packagename
def installapp(packagename):
    openAppInPlayStoreByPackageName(packagename)
    time.sleep(1)

    #Find the coordinates of the install button on the Play Store page
    buttonpos = getButtonPosition("Install")

    #Hit that button
    tap(buttonpos.split(' '))
```

Listing 3: Python example to automate app installation

A.2 Code to verify phone connection status

The Python function below returns 0 if the phone is connected and 1 if the phone is not connected.

```
def getServiceState():
  #Execute the adb command to get the status and save its output into the temp
    variable
    temp = systemCmd("adb shell dumpsys telephony.registry | grep -i -o '{\
    textbackslash}smDataRegState=.'")

  '''
```

```
  Temp now contains multiple lines of "mDataRegState=..." strings and also one empty
    line (the last line, line [-1])
  The line above that (line [-2]) is the only relevant line, as it gives us the most
    recent status.
  So now we extract that line:'''
  lateststatus = temp.decode('utf-8').split('\n')[-2]
  #Now we only need the last digit:
  return int(lateststatus[-1])
```

Listing 4: Python example to retrieve the phone's connection state

A.3 Code to end srsRAN

The code below gets the right process identifier (PID) for an ongoing process. This can be used to end that process.

```
def getPid(programkeyword):
    #Execute the system command to list processes that match the programkeyword
    commandstr = "sudo ps -aux | grep " + programkeyword
    output = systemCmd(commandstr).decode()

    #Print the output, i.e. the list of programs matching the programkeyword
    print("Overview of processes matching '", programkeyword, "':")
    print(output)
    print("======================")

    #Turn the output into a list
    processlist = output.split('\n')

    pid = -1

    #For all items in that processlist
    for i in range(len(processlist)):
        #Split that item into another list
        splitline = processlist[i].split(" ")
        #Remove empty strings from list
        splitline = list(filter(None, splitline))

        #If the item itself is not empty
        if(len(splitline) != 0):
            #If this is the item satisfying our criteria, print its list
            if(splitline[0] == 'root' and splitline[-5] == 'S+' and splitline[-1] ==
    programkeyword and splitline[-2] != "grep" and splitline[-2] != "--color=auto"):
                #Print it
                print(splitline)
                #The pid is the second element in the list of this item
                pid = int(splitline[1])
    return pid


print("The pid that we need is:", getPid("srsepc"))
```

Listing 5: Retrieving the pid of the process that we need to kill

Using this code, we can create the Python code to kill srsRAN:

```
def killprogram(programkeyword):
    pid = getPid(programkeyword)
    commandstr = "kill " + str(pid)
    print("Killing process ", str(pid))
    print(systemCmd(commandstr))

def killsrsRAN():
    print("Killing srsenb...")
    killprogram("srsenb")
    time.sleep(5)
    print("Killing srsepc...")
```

```
12      killprogram("srsepc")
13
14  killsrsRAN()
```

Listing 6: Killing srsRAN

B Setting the app permissions

Granting an app all of its permissions can be done during the installation of an app using the following command in `adb`:

```
# adb install -g [apk file]
```

Here, [apk file] is the filename of an apk file of an Android app, located on a computer.
In our case, we have already installed the apps, which we wanted to do through the Play Store (see Appendix A.1) rather than through apk files. Therefore, we need another approach to still be able to grant each app the permissions it wants. Here is how we proceed:

1. Download all of the installed apps to our computer
   We iterate over our list of package names of the 500 apps that we already installed. For each package name, we do the following:

   (a) Find the paths of apk files that belong to the app on the phone:
   ```
   # adb shell pm path [package name]
   ```

   (b) For each path to an apk that we found, download that apk from the phone to the computer:
   ```
   # adb pull [path on the phone to that apk] [path to some folder on a computer]
   ```
   Now all of the apk files that belong to the app are downloaded to a folder on the computer. To organize the apk files into folders that represent the different apps, it is advisable to execute these commands in a script. A Python script, for example, could create subfolders on the computer with the right names, based on the path of the apk files.

2. Create a list with permissions for each app
   First, we create a list with all android permissions that an app could ask for. We can use [48] as a reference. Now, for all of the apk files that we downloaded on the computer, we do the following:

   (a) Execute this command to view the permissions that it wants:
   ```
   output = systemCmd("aapt d permissions \"" + apkfile + "\"")
   ```

   (b) Extract the permissions from the output of the previous command using a regex:

   ```
   regex_perm = "name='android.permission.([^']*)'"
   permissions = re.findall(regex_perm, output.decode())
   ```

   We now have a Python list for the apk file with the permissions that it wants.

   (c) For each permission in this list, we now discover the index of that same permission in the list of all of the android permissions (that we created earlier). We then have a list of indexes, that represent the permissions that this apk file wants.
   This means that we can now create the following two-element list:
   [package name of the apk file], [indexes of permissions]

   (d) We then add this entry to another list called *packagenames_and_permissions*. However, it is possible that there is already an entry in this list with the same package name, as the same app can have multiple apk files. In that case, we append any additional indexes to that already existing entry instead.

   Once we have iterated over all apk files and the permissions that they desire, we end up with a list that looks like this:

```
packagenames_and_permissions = [['com.yodo1.crossyroad', [0, 1, 2, 3, 4, 4, 5,
 6, 7]], ['in.swiggy.android', [0, 3, 4, 4, 5, 6, 7, 8, 9, 10, 11, 12, 2, 13,
14, 15, 16, 17, 18, 19, 20, 21, 1]], ['com.weather.Weather', [0, 20, 8, 22, 3,
 4, 6, 17, 23, 5, 7]], ['com.ForgeGames.SpecialForcesGroup2', [0, 1, 3, 5, 4,
15, 17, 2, 14, 20, 8]], ['com.tencent.mm', [24, 25, 3, 20, 8, 19, 26, 0, 15,
6, 14, 18, 17, 5, 1, 27, 28, 29, 13, 30, 31, 16, 32, 33, 34, 35, 4, 7, 2, 36,
37, 38, 39, 40]], ['com.supercell.brawlstars', [0, 3, 5, 30, 4, 17, 41, 1, 2,
19, 7]], ['com.ss.android.ugc.trill', [20, 13, 42, 0, 3, 2, 1, 4, 19, 14, 43,
5, 26, 18, 6, 17, 27, 44, 45, 46, 7, 47, 48, 49, 25, 50, 15, 51, 52, 24]], ['
```

```
com.netflix.mediaclient', [3, 4, 27, 24, 0, 5, 14, 15, 1, 7, 50, 7]], ['com.
matchington.mansion', [0, 4, 3, 2, 1, 5, 6, 7, 53]], ['com.microsoft.office.
outlook', [0, 1, 3, 17, 10, 54, 18, 44, 45, 55, 46, 56, 49, 57, 58, 8, 20, 5,
48, 19, 42, 2, 59, 14, 38, 37, 6, 7, 15, 51]], ['com.FDGEntertainment.redball4
.gp', [0, 3, 5, 4, 1]], ..., ['com.google.android.apps.youtube.music', [3, 4,
27, 28, 0, 32, 130, 2, 1, 8, 13, 17, 5, 10, 55, 23, 7, 53, 6, 14]], ['com.
nianticlabs.pokemongo', [17, 27, 28, 8, 7, 2, 1, 0, 10, 23, 3, 5, 4, 19, 38,
37, 6, 22, 34, 21, 18, 20]], ['com.amazon.kindle', [21, 5, 3, 4, 0, 1, 46, 55,
 23, 7, 26, 29, 2, 27, 107, 50, 106, 8, 20, 28, 6, 83, 16, 30]], ['com.google.
android.music', [0]]]
```

This is just an example showing the permissions for 15 apps. But given the fact that we have installed 500 apps, the actual list is considerably more extensive.

3. We now grant each app the permissions it wants
   Now that we know what permissions each app wants, we can grant them to the apps on the phone. Granting an app permissions can be done as follows:

```
1  def grantPermission(packagename, permission):
2      commandstr = "adb shell pm grant " + packagename + " android.permission."
   + permission
3      systemCmd(commandstr)
4
5  def grantPermissions(packagename, permissions):
6      for permission in permissions:
7          grantPermission(packagename, permission)
8
```

   We can now iterate over the *packagenames_and_permissions* list. For every entry in that list:

   (a) We can convert the indexes in this entry to strings. For each index, we look at the string at that index in the list of all android permissions.

   (b) With the package name and a list of the permissions in strings, we can now call the *grantPermissions(packagename, permissions)* function, to grant the permissions to the app specified by the package name.

Every app now has the permissions that it requires.

C The 500 installed apps

Top 500 most downloaded apps worldwide (on 13 September 2022 [44])		
Rank	**App title**	**Package name**
1	Candy Crush Jelly Saga	com.king.candycrushjellysaga
2	Sticker.ly - Sticker Maker	com.snowcorp.stickerly.android
3	Club Factory - Online Shopping App	club.fromfactory
4	BYJU'S – The Learning App	com.byjus.thelearningapp
5	Chess	com.jetstartgames.chess
6	Asphalt Nitro	com.gameloft.android.ANMP.GloftAGHM
7	Modern Strike Online: PvP FPS	com.gamedevltd.modernstrike
8	Fast Racing 3D	com.julian.fastracing
9	Car Parking Multiplayer	com.olzhas.carparking.multyplayer
10	Dumb Ways to Die 2: The Games	air.au.com.metro.DumbWaysToDie2
11	Alibaba.com - B2B marketplace	com.alibaba.intl.android.apps.poseidon
12	Zombie Catchers-Undead Shooter	fi.twomenandadog.zombiecatchers
13	Geometry Dash Meltdown	com.robtopx.geometrydashmeltdown
14	Paper.io 2	io.voodoo.paper2
15	Photoshop Express Photo Editor	com.adobe.psmobile
16	Kaspersky Security & VPN	com.kms.free
17	Tomb of the Mask	com.playgendary.tom
18	Disney+	com.disney.disneyplus
19	Earn to Die 2	com.notdoppler.earntodie2
20	One Booster: Antivirus&Cleaner	com.cleanteam.oneboost
21	Glow Hockey	com.natenai.glowhockey
22	Azar - video chat & livestream	com.azarlive.android
23	Muslim Pro: Quran Athan Prayer	com.bitsmedia.android.muslimpro
24	Mail.ru - Email App	ru.mail.mailapp
25	QR & Barcode Scanner	com.gamma.scan
26	Talking Ginger	com.outfit7.talkinggingerfree
27	Talking Ben the Dog	com.outfit7.talkingben
28	Six Pack in 30 Days	sixpack.sixpackabs.absworkout
29	Mario Kart Tour	com.nintendo.zaka
30	Tank Stars	com.playgendary.tanks
31	MP3 Video Converter	com.springwalk.mediaconverter
32	Signal Private Messenger	org.thoughtcrime.securesms
33	Bubble Shooter	bubbleshooter.orig
34	Words of Wonders: Crossword	com.fugo.wow
35	"Emoji keyboard-Themes,Fonts"	com.emoji.coolkeyboard
36	YouCam Perfect - Photo Editor	com.cyberlink.youperfect
37	Collage Maker | Photo Editor	photoeditor.layout.collagemaker
38	LINE Let's Get Rich	com.linecorp.LGGRTHN
39	Parchisi STAR Online	com.superking.parchisi.star
40	ibis Paint X	jp.ne.ibis.ibispaintx.app
41	VN Video Editor Maker VlogNow	com.frontrow.vlog
42	Photo Editor Picsa: Collage	com.lyrebirdstudio.montagenscolagem
43	"Nubank: conta, cartão e mais"	com.nu.production
44	Hungry Shark World	com.ubisoft.hungrysharkworld
45	Talking Angela	com.outfit7.talkingangelafree
46	Higgs Domino Island	com.neptune.domino
47	"Adobe Scan: PDF Scanner, OCR"	com.adobe.scan.android
48	Brainly – Get Homework Answers	co.brainly

Continued on next page...

Rank	App title	Package name
49	Matchington Mansion	com.matchington.mansion
50	Palco MP3: Listen and download	com.studiosol.palcomp3
51	Yandex Browser with Protect	com.yandex.browser
52	Train Status Ticket Book PNR	com.ixigo.train.ixitrain
53	Where's My Water? 2	com.disney.wheresmywater2_goo
54	Bus Simulator : Ultimate	com.zuuks.bus.simulator.ultimate
55	Stick War: Legacy	com.maxgames.stickwarlegacy
56	MyFitnessPal: Calorie Counter	com.myfitnesspal.android
57	Snapdeal: Online Shopping App	com.snapdeal.main
58	Remini - AI Photo Enhancer	com.bigwinepot.nwdn.international
59	DEAD TARGET: Zombie Games 3D	com.vng.g6.a.zombie
60	YouTube Kids	com.google.android.apps.youtube.kids
61	Head Ball 2 - Online Soccer	com.masomo.headball2
62	BOOYAH!	com.mambet.tv
63	"GuitarTuna: Guitar,Tuner,Chord"	com.ovelin.guitartuna
64	"TuneIn Radio: News, Music & FM"	tunein.player
65	Tiles Hop: EDM Rush!	com.amanotes.beathopper
66	"Dailyhunt: News, Video,Cricket"	com.eterno
67	Join Clash 3D	com.freeplay.runandfight
68	GO Keyboard Lite - Many Theme	com.jb.gokeyboard
69	Amazon Kindle	com.amazon.kindle
70	YouCam Makeup - Selfie Editor	com.cyberlink.youcammakeup
71	MARVEL Future Fight	com.netmarble.mherosgb
72	Sweet Selfie: Camera & Editor	com.cam001.selfie
73	Meesho: Online Shopping App	com.meesho.supply
74	GUNSHIP BATTLE: Helicopter 3D	com.theonegames.gunshipbattle
75	Meta Business Suite	com.facebook.pages.app
76	MORTAL KOMBAT: A Fighting Game	com.wb.goog.mkx
77	JioTV	com.jio.jioplay.tv
78	AJIO - House Of Brands	com.ril.ajio
79	Banco do Brasil	br.com.bb.android
80	World Soccer League	com.touchtao.soccerkinggoogle
81	Galaxy Attack: Alien Shooting	com.alien.shooter.galaxy.attack
82	Getcontact	app.source.getcontact
83	Cooking Madness -A Chef's Game	com.biglime.cookingmadness
84	Cut the Rope 2	com.zeptolab.ctr2.f2p.google
85	Beach Buggy Racing	com.vectorunit.purple.googleplay
86	"Voot, Bigg Boss, Colors TV"	com.tv.v18.viola
87	CCleaner – Phone Cleaner	com.piriform.ccleaner
88	Pixel Art - color by number	com.europosit.pixelcoloring
89	Music Player & MP3 Player	musicplayer.musicapps.music.mp3player
90	Cut the Rope	com.zeptolab.ctr.ads
91	AccuWeather: Weather Radar	com.accuweather.android
92	WiFi Passwords: Instabridge	com.instabridge.android
93	"ZEPETO: 3D avatar, chat & meet"	me.zepeto.main
94	Hike News & Content	com.bsb.hike
95	My Cafe — Restaurant Game	com.melesta.coffeeshop
96	Tinder: Dating app. Meet. Chat	com.tinder
97	Stumble Guys	com.kitkagames.fallbuddies
98	Fun Race 3D	com.slippy.linerusher
99	Amazon Prime Video	com.amazon.avod.thirdpartyclient
100	Reddit	com.reddit.frontpage

Continued on next page...

Rank	App title	Package name
101	SuperVPN Fast VPN Client	com.jrzheng.supervpnfree
102	Bike Race: Motorcycle Games	com.topfreegames.bikeracefreeworld
103	OLX: Buy & Sell Near You with Online Classifieds	com.olx.southasia
104	": , , "	com.avito.android
105	Google Earth	com.google.earth
106	Happy Color® – Color by Number	com.pixel.art.coloring.color.number
107	Deezer: Music & Podcast Player	deezer.android.app
108	Craftsman: Building Craft	com.craftsman.go
109	Kiloblocks Lite	com.explorationbase.ExplorationLite
110	Marvel Contest of Champions	com.kabam.marvelbattle
111	ColorNote Notepad Notes	com.socialnmobile.dictapps.notepad.color.note
112	Hunter Assassin	com.rubygames.assassin
113	Automatic Call Recorder	com.appstar.callrecorder
114	Block Craft 3D: Building Game	com.fungames.blockcraft
115	Kik — Messaging & Chat App	kik.android
116	Zepp Life(MiFit)	com.xiaomi.hm.health
117	Flight Pilot: 3D Simulator	com.fungames.flightpilot
118	NBA LIVE Mobile Basketball	com.ea.gp.nbamobile
119	The Weather Channel - Radar	com.weather.Weather
120	Battlegrounds Mobile India	com.pubg.imobile
121	My Talking Tom Friends	com.outfit7.mytalkingtomfriends
122	"Ola, Safe and affordable rides"	com.olacabs.customer
123	foodpanda: Food & Groceries	com.global.foodpanda.android
124	Cover Fire: Offline Shooting	com.generagames.resistance
125	OK: Social Network	ru.ok.android
126	Bubble Witch 3 Saga	com.king.bubblewitch3
127	Pandora - Music & Podcasts	com.pandora.android
128	Music Player &MP3- Lark Player	com.dywx.larkplayer
129	Modern Combat 5: mobile FPS	com.gameloft.android.ANMP.GloftM5HM
130	Joom. Shopping for every day	com.joom
131	Angry Birds Go!	com.rovio.angrybirdsgo
132	Live Transcribe & Notification	com.google.audio.hearing. ... ...visualization.accessibility.scribe
133	SimCity BuildIt	com.ea.game.simcitymobile_row
134	Zalo	com.zing.zalo
135	Parallel Space - Multiple accounts & Two face	com.lbe.parallel.intl
136	Shopee: Compre de Tudo Online	com.shopee.br
137	YouCut - Video Editor & Maker	com.camerasideas.trimmer
138	Bubble Witch 2 Saga	com.midasplayer.apps.bubblewitchsaga2
139	Pet Rescue Saga	com.king.petrescuesaga
140	APUS Launcher: Theme Launcher	com.apusapps.launcher
141	Plants vs. Zombies™	com.ea.game.pvzfree_row
142	Tiny Flashlight + LED	com.devuni.flashlight
143	Shadow Fight 3 - RPG fighting	com.nekki.shadowfight3
144	Flipboard: The Social Magazine	flipboard.app
145	Photo Lab Picture Editor & Art	vsin.t16_funny_photo
146	Top Eleven Be a Soccer Manager	eu.nordeus.topeleven.android
147	YouVersion Bible App + Audio	com.sirma.mobile.bible.android
148	"WPS Office-PDF,Word,Excel,PPT"	cn.wps.moffice_eng
149	Microsoft Teams	com.microsoft.teams
150	Dragon Mania Legends	com.gameloft.android.ANMP.GloftDOHM
151	Criminal Case	com.prettysimple.criminalcaseandroid

Continued on next page...

Rank	App title	Package name
152	Mini World: CREATA	com.playmini.miniworld
153	Wattpad - Read & Write Stories	wp.wattpad
154	Zombie Tsunami	net.mobigame.zombietsunami
155	Secure Folder	com.samsung.knox.securefolder
156	Hill Climb Racing 2	com.fingersoft.hcr2
157	Where is my Train	com.whereismytrain.android
158	Akinator	com.digidust.elokence.akinator.freemium
159	Toca Life World	com.tocaboca.tocalifeworld
160	Plague Inc.	com.miniclip.plagueinc
161	Storage Saver	com.samsung.memorysaver
162	"PayPal - Send, Shop, Manage"	com.paypal.android.p2pmobile
163	Mini Militia - Doodle Army 2	com.appsomniacs.da2
164	"Discord: Talk, Chat & Hang Out"	com.discord
165	"Nox Cleaner - Booster, Master"	com.noxgroup.app.cleaner
166	PUBG MOBILE LITE	com.tencent.iglite
167	StarMaker: Sing and Play	com.starmakerinteractive.starmaker
168	Uber - Driver: Drive & Deliver	com.ubercab.driver
169	FastVid: Video Downloader for Facebook	fb.video.downloader
170	WEBTOON	com.naver.linewebtoon
171	Sonic Dash - Endless Running	com.sega.sonicdash
172	Garena Liên Quân Mobile	com.garena.game.kgvn
173	Xender - Share Music Transfer	cn.xender
174	Football Strike: Online Soccer	com.miniclip.footballstrike
175	Magic Tiles 3	com.youmusic.magictiles
176	HP Print Service Plugin	com.hp.android.printservice
177	Cymera - Photo Editor Collage	com.cyworld.camera
178	App Vault	com.mi.android.globalminusscreen
179	JOOX Music	com.tencent.ibg.joox
180	Mobizen Screen Recorder	com.rsupport.mvagent
181	ShareChat - Made in India	in.mohalla.sharechat
182	inDriver — Offer your fare	sinet.startup.inDriver
183	Red Ball 4	com.FDGEntertainment.redball4.gp
184	Worms Zone .io - Hungry Snake	com.wildspike.wormszone
185	Backgrounds HD (Wallpapers)	com.ogqcorp.bgh
186	Avakin Life - 3D Virtual World	com.lockwoodpublishing.avakinlife
187	Google Classroom	com.google.android.apps.classroom
188	ZEDGE™ Wallpapers & Ringtones	net.zedge.android
189	eBay: The shopping marketplace	com.ebay.mobile
190	Themes	com.huawei.android.thememanager
191	"GO SMS Pro - Messenger, Free Themes, Emoji"	com.jb.gosms
192	Samsung Cloud	com.samsung.android.scloud
193	Twitch: Live Game Streaming	tv.twitch.android.app
194	Gangstar Vegas: World of Crime	com.gameloft.android.ANMP.GloftGGHM
195	War Robots Multiplayer Battles	com.pixonic.wwr
196	1945 Air Force: Airplane games	com.os.airforce
197	Among Us	com.innersloth.spacemafia
198	Fishdom	com.playrix.fishdomdd.gplay
199	DEER HUNTER CLASSIC	com.glu.deerhunt2
200	Crossy Road	com.yodo1.crossyroad
201	WhatsApp Wallpaper	com.whatsapp.wallpaper
202	PLAYit-All in One Video Player	com.playit.videoplayer
203	"Google Go: A lighter, faster way to search"	com.google.android.apps.searchlite

Continued on next page...

Rank	App title	Package name
204	"Amazon India Shop, Pay, miniTV"	in.amazon.mShop.android.shopping
205	Traffic Racer	com.skgames.trafficracer
206	Uber Eats: Food Delivery	com.ubercab.eats
207	Camera360 :Photo Editor&Selfie	vStudio.Android.Camera360
208	Indeed Job Search	com.indeed.android.jobsearch
209	Home Workout - No Equipment	homeworkout.homeworkouts.noequipment
210	Talking Tom Cat 2	com.outfit7.talkingtom2free
211	Lazada 9.9!	com.lazada.android
212	Video Editor & Maker VideoShow	com.xvideostudio.videoeditor
213	Swiggy Food & Grocery Delivery	in.swiggy.android
214	Smart diagnosis	com.huawei.hwdetectrepair
215	Mercado Libre: Compras Online	com.mercadolibre
216	Dream League Soccer 2022	com.firsttouchgames.dls7
217	Photomath	com.microblink.photomath
218	Drag Racing	com.creativemobile.DragRacing
219	Swamp Attack	com.outfit7.movingeye.swampattack
220	SHEIN-Fashion Shopping Online	com.zzkko
221	tango-Live Stream & Video Chat	com.sgiggle.production
222	Group Sharing	com.samsung.android.mobileservice
223	Opera Browser: Fast & Private	com.opera.browser
224	CSR 2 - Drag Racing Car Games	com.naturalmotion.customstreetracer2
225	Uber - Request a ride	com.ubercab
226	Amazon Music: Discover Songs	com.amazon.mp3
227	Resso Music - Songs & Lyrics	com.moonvideo.android.resso
228	Kick the Buddy	com.playgendary.kickthebuddy
229	Phoenix Browser - Fast & Safe	com.transsion.phoenix
230	Pokémon GO	com.nianticlabs.pokemongo
231	eFootball™ 2023	jp.konami.pesam
232	Episode - Choose Your Story	com.episodeinteractive.android.catalog
233	Bitmoji	com.bitstrips.imoji
234	Minecraft Trial	com.mojang.minecrafttrialpe
235	Pooking - Billiards City	com.billiards.city.pool.nation.club
236	Gardenscapes	com.playrix.gardenscapes
237	Helix Jump	com.h8games.helixjump
238	SimSimi	com.ismaker.android.simsimi
239	CAIXA	br.com.gabba.Caixa
240	Brain Out: Can you pass it?	com.mind.quiz.brain.out
241	JioSaavn - Music & Podcasts	com.jio.media.jiobeats
242	Trivia Crack	com.etermax.preguntados.lite
243	Dr. Driving	com.ansangha.drdriving
244	"Vi: Recharge, Music, Games, TV"	com.mventus.selfcare.activity
245	Photo Editor Pro - Polish	photo.editor.photoeditor.photoeditorpro
246	Samsung Link (Terminated)	com.sec.pcw
247	Last Day on Earth: Survival	zombie.survival.craft.z
248	CapCut - Video Editor	com.lemon.lvoverseas
249	The Sims™ FreePlay	com.ea.games.simsfreeplay_row
250	Samsung Music	com.sec.android.app.music
251	Coin Master	com.moonactive.coinmaster
252	Google Sheets	com.google.android.apps.docs.editors.sheets
253	Samsung Members	com.samsung.android.voc
254	Google Keep - Notes and Lists	com.google.android.keep
255	Brawl Stars	com.supercell.brawlstars

Continued on next page...

Rank	App title	Package name
256	The Simpsons™: Tapped Out	com.ea.game.simpsons4_row
257	MX Player	com.mxtech.videoplayer.ad
258	CAIXA Tem	br.gov.caixa.tem
259	Google Translate	com.google.android.apps.translate
260	Adobe Acrobat Reader: Edit PDF	com.adobe.reader
261	Temple Run	com.imangi.templerun
262	Microsoft Word: Edit Documents	com.microsoft.office.word
263	Clock	com.google.android.deskclock
264	Google Calendar	com.google.android.calendar
265	Samsung Security Policy Update	com.policydm
266	Bigo Live - Live Streaming App	sg.bigo.live
267	Gmail Go	com.google.android.gm.lite
268	myIM3: Data Plan & Buy Package	com.pure.indosat.care
269	Airtel Thanks – Recharge & UPI	com.myairtelapp
270	Microsoft SwiftKey Keyboard	com.touchtype.swiftkey
271	"Canva: Design, Photo & Video"	com.canva.editor
272	Google Assistant Go	com.google.android.apps.assistant
273	Samsung One UI Home	com.sec.android.app.launcher
274	Microsoft Outlook	com.microsoft.office.outlook
275	B612 Camera&Photo/Video Editor	com.linecorp.b612.android
276	Yandex Go — taxi and delivery	ru.yandex.taxi
277	WiFi Master: WiFi Auto Connect	com.halo.wifikey.wifilocating
278	Navigation for Google Maps Go	com.google.android.apps.navlite
279	Banana Kong	com.fdgentertainment.bananakong
280	Avast Antivirus & Security	com.avast.android.mobilesecurity
281	Samsung Calculator	com.sec.android.app.popupcalculator
282	Paytm: Secure UPI Payments	net.one97.paytm
283	AVG AntiVirus & Security	com.antivirus
284	Hungry Shark Evolution	com.fgol.HungrySharkEvolution
285	Brain Test: Tricky Puzzles	com.unicostudio.braintest
286	Google Meet (original)	com.google.android.apps.meetings
287	Google Play Games	com.google.android.play.games
288	Twitter	com.twitter.android
289	Google Docs	com.google.android.apps.docs.editors.docs
290	UC Mini-Download Video Status & Movies	com.uc.browser.en
291	SnackVideo	com.kwai.bulldog
292	Mi Calendar	com.xiaomi.calendar
293	MyTelkomsel - Buy Package	com.telkomsel.telkomselcm
294	PK XD - Explore Universes!	com.movile.playkids.pkxd
295	Samsung Push Service	com.sec.spp.push
296	FIFA Soccer	com.ea.gp.fifamobile
297	Shadow Fight 2	com.nekki.shadowfight
298	Google Maps Go	com.google.android.apps.mapslite
299	Clash of Clans	com.supercell.clashofclans
300	Briefing	flipboard.boxer.app
301	"Tumblr—Fandom, Art, Chaos"	com.tumblr
302	Weather - By Xiaomi	com.miui.weather2
303	Dropbox: Secure Cloud Storage	com.dropbox.android
304	World of Tanks Blitz - PVP MMO	net.wargaming.wot.blitz
305	Google Play Services for AR	com.google.ar.core
306	slither.io	air.com.hypah.io.slither
307	Google One	com.google.android.apps.subscriptions.red

Continued on next page...

Rank	App title	Package name
308	Farm Heroes Saga	com.king.farmheroessaga
309	Telegram	org.telegram.messenger
310	Speech Services by Google	com.google.android.tts
311	Google Play Books & Audiobooks	com.google.android.apps.books
312	Screen Recorder - XRecorder	videoeditor.videorecorder.screenrecorder
313	Waze Navigation & Live Traffic	com.waze
314	Microsoft PowerPoint	com.microsoft.office.powerpoint
315	Badoo - Dating. Chat. Meet.	com.badoo.mobile
316	CamScanner - PDF Scanner App	com.intsig.camscanner
317	FaceApp: Face Editor	io.faceapp
318	WhatsApp Messenger	com.whatsapp
319	Tokopedia	com.tokopedia.tkpd
320	Grab Superapp	com.grabtaxi.passenger
321	Google Play services	com.google.android.gms
322	Spotify: Music and Podcasts	com.spotify.music
323	Pou	me.pou.app
324	LINE: Calls & Messages	jp.naver.line.android
325	Real Racing 3	com.ea.games.r3_row
326	Myntra - Fashion Shopping App	com.myntra.android
327	Standoff 2	com.axlebolt.standoff2
328	ShareMe: File sharing	com.xiaomi.midrop
329	Need for Speed™ No Limits	com.ea.game.nfs14_row
330	SmartThings	com.samsung.android.oneconnect
331	TikTok	com.ss.android.ugc.trill
332	Wish: Shop And Save	com.contextlogic.wish
333	My Talking Tom	com.outfit7.mytalkingtomfree
334	"VK: music, video, messenger"	com.vkontakte.android
335	Candy Camera - photo editor	com.joeware.android.gpulumera
336	Talking Tom Gold Run	com.outfit7.talkingtomgoldrun
337	Calculator	com.google.android.calculator
338	Wynk Music -Songs & HelloTunes	com.bsbportal.music
339	Turbo VPN - Secure VPN Proxy	free.vpn.unblock.proxy.turbovpn
340	"Solo Launcher-Clean,Smooth,DIY"	home.solo.launcher.free
341	Samsung Notes	com.samsung.android.app.notes
342	Shopee 9.9 Super Shopping Day	com.shopee.id
343	Files by Google	com.google.android.apps.nbu.files
344	Special Forces Group 2	com.ForgeGames.SpecialForcesGroup2
345	Google Assistant	com.google.android.apps.googleassistant
346	Cooking Fever: Restaurant Game	com.nordcurrent.canteenhd
347	Retrica - The Original Filter Camera	com.venticake.retrica
348	GO Launcher -Themes&Wallpapers	com.gau.go.launcherex
349	Gaana Hindi Song Music App	com.gaana
350	Samsung Health	com.sec.android.app.shealth
351	Candy Crush Saga	com.king.candycrushsaga
352	My Talking Angela	com.outfit7.mytalkingangelafree
353	YouTube Music	com.google.android.apps.youtube.music
354	My Talking Tom 2	com.outfit7.mytalkingtom2
355	Dragon City Mobile	es.socialpoint.DragonCity
356	Temple Run 2	com.imangi.templerun2
357	Link to Windows	com.microsoft.appmanager
358	imo-International Calls & Chat	com.imo.android.imoim
359	Duolingo: language lessons	com.duolingo

Continued on next page...

Rank	App title	Package name
360	HiCare	com.huawei.phoneservice
361	Zomato: Food Delivery & Dining	com.application.zomato
362	Samsung My Files	com.sec.android.app.myfiles
363	Wallpapers	com.google.android.apps.wallpaper
364	Hay Day	com.supercell.hayday
365	Microsoft OneDrive	com.microsoft.skydrive
366	TikTok Lite	com.zhiliaoapp.musically.go
367	Video Editor & Maker - InShot	com.camerasideas.instashot
368	Smule: Karaoke Songs & Videos	com.smule.singandroid
369	Motorola Notifications	com.motorola.ccc.notification
370	Skype	com.skype.raider
371	Android Accessibility Suite	com.google.android.marvin.talkback
372	Android Auto	com.google.android.projection.gearhead
373	"SHAREit: Transfer, Share Files"	com.lenovo.anyshare.gps
374	Garena Free Fire: 5th Anniv.	com.dts.freefireth
375	Lords Mobile: Tower Defense	com.igg.android.lordsmobile
376	Samsung Voice Recorder	com.sec.android.app.voicenote
377	Gmail	com.google.android.gm
378	Smash Hit	com.mediocre.smashhit
379	8 Ball Pool	com.miniclip.eightballpool
380	YouTube Music	com.google.android.music
381	Samsung Internet Browser	com.sec.android.app.sbrowser
382	Clash Royale	com.supercell.clashroyale
383	Picsart Photo & Video Editor	com.picsart.studio
384	Google Photos	com.google.android.apps.photos
385	Microsoft Excel: Spreadsheets	com.microsoft.office.excel
386	Hotstar	in.startv.hotstar
387	Phone Master–Junk Clean Master	com.transsion.phonemaster
388	Viber - Safe Chats And Calls	com.viber.voip
389	Google Drive	com.google.android.apps.docs
390	Minion Rush: Running Game	com.gameloft.android.ANMP.GloftDMHM
391	Mobile Legends: Bang Bang	com.mobile.legends
392	MyJio: For Everything Jio	com.jio.myjio
393	WeChat	com.tencent.mm
394	Google Duo	com.google.android.apps.tachyon
395	"UC Browser-Safe, Fast, Private"	com.UCMobile.intl
396	Mi Video - Video player	com.miui.videoplayer
397	Ultra data saving	com.samsung.android.uds
398	dfndr security: antivirus	com.psafe.msuite
399	Castle Clash: World Ruler	com.igg.castleclash
400	Pixel Gun 3D - FPS Shooter	com.pixel.gun3d
401	"Safe Security - Antivirus, Booster, Phone Cleaner"	com.qihoo.security
402	Subway Surfers	com.kiloo.subwaysurf
403	"PhonePe UPI, Payment, Recharge"	com.phonepe.app
404	Plants vs Zombies™ 2	com.ea.game.pvz2_row
405	Netflix	com.netflix.mediaclient
406	TikTok	com.zhiliaoapp.musically
407	WhatsApp Business	com.whatsapp.w4b
408	LinkedIn: Jobs & Business News	com.linkedin.android
409	Google News - Daily Headlines	com.google.android.apps.magazines
410	ZOOM Cloud Meetings	us.zoom.videomeetings
411	Google	com.google.android.googlequicksearchbox

Continued on next page...

Rank	App title	Package name
412	Candy Crush Soda Saga	com.king.candycrushsodasaga
413	Jetpack Joyride	com.halfbrick.jetpackjoyride
414	Galaxy Wearable (Samsung Gear)	com.samsung.android.app.watchmanager
415	Facebook Lite	com.facebook.lite
416	Likee - Community of Interests	video.like
417	Firefox Fast & Private Browser	org.mozilla.firefox
418	HUAWEI Video	com.huawei.himovie.overseas
419	KineMaster - Video Editor	com.nexstreaming.app.kinemasterfree
420	Agar.io	com.miniclip.agar.io
421	Vector: Parkour Run	com.nekki.vector
422	Calculator	com.miui.calculator
423	Booking.com: Hotels and more	com.booking
424	Currents	com.google.android.apps.plus
425	Samsung Email	com.samsung.android.email.provider
426	Township	com.playrix.township
427	Sniper 3D: Gun Shooting Games	com.fungames.sniper3d
428	Google Chrome: Fast & Secure	com.android.chrome
429	ANT Radio Service	com.dsi.ant.service.socket
430	Huawei Health	com.huawei.health
431	Amazon Shopping	com.amazon.mShop.android.shopping
432	Opera Mini: Fast Web Browser	com.opera.mini.native
433	Asphalt 8 - Car Racing Game	com.gameloft.android.ANMP.GloftA8HM
434	Roblox	com.roblox.client
435	Granny	com.dvloper.granny
436	Google Wallet	com.google.android.apps.walletnfcrel
437	Carrom Pool: Disc Game	com.miniclip.carrom
438	Boom Beach	com.supercell.boombeach
439	"BeautyPlus - Retouch, Filters"	com.commsource.beautyplus
440	Traffic Rider	com.skgames.trafficrider
441	Mercado Pago: cuenta digital	com.mercadopago.wallet
442	Device Help	com.motorola.genie
443	Phone Clone	com.hicloud.android.clone
444	File Manager : free and easily	com.mi.android.globalFileexplorer
445	Facebook	com.facebook.katana
446	Instagram	com.instagram.android
447	Carrier Services	com.google.android.ims
448	Gojek	com.gojek.app
449	Microsoft OneNote: Save Notes	com.microsoft.office.onenote
450	ANT+ Plugins Service	com.dsi.ant.plugins.antplus
451	Truecaller: Caller ID & Block	com.truecaller
452	Yahoo Mail – Organized Email	com.yahoo.mobile.client.android.mail
453	"Google Pay: Save, Pay, Manage"	com.google.android.apps.nbu.paisa.user
454	Messenger Lite	com.facebook.mlite
455	Garena Free Fire MAX	com.dts.freefiremax
456	iFood comida e mercado em casa	br.com.brainweb.ifood
457	Contacts	com.google.android.contacts
458	Idle Miner Tycoon: Gold & Cash	com.fluffyfairygames.idleminertycoon
459	Huawei Backup	com.huawei.KoBackup
460	Android System WebView	com.google.android.webview
461	Google Slides	com.google.android.apps.docs.editors.slides
462	Microsoft Office: Edit & Share	com.microsoft.office.officehubrow
463	Google TV	com.google.android.videos

Continued on next page...

Rank	App title	Package name
464	Extreme Car Driving Simulator	com.aim.racing
465	Homescapes	com.playrix.homescapes
466	YouTube	com.google.android.youtube
467	Snapchat	com.snapchat.android
468	Shazam: Music Discovery	com.shazam.android
469	KakaoTalk : Messenger	com.kakao.talk
470	Kwai - Watch cool&funny videos	com.kwai.video
471	Themes	com.bbk.theme
472	Google Lens	com.google.ar.lens
473	Google Maps	com.google.android.apps.maps
474	Call of Duty Mobile Season 7	com.activision.callofduty.shooter
475	µTorrent®- Torrent Downloader	com.utorrent.client
476	Pinterest	com.pinterest
477	YouTube Go	com.google.android.apps.youtube.mango
478	Period Calendar Period Tracker	com.popularapp.periodcalendar
479	Geometry Dash Lite	com.robtopx.geometryjumplite
480	Hill Climb Racing	com.fingersoft.hillclimb
481	Samsung Print Service Plugin	com.sec.app.samsungprintservice
482	Ludo King™	com.ludo.king
483	SoundCloud: Play Music & Songs	com.soundcloud.android
484	Talking Tom Cat	com.outfit7.talkingtom
485	Flipkart Online Shopping App	com.flipkart.android
486	Fruit Ninja®	com.halfbrick.fruitninjafree
487	AliExpress	com.alibaba.aliexpresshd
488	Mi Music	com.miui.player
489	Messenger	com.facebook.orca
490	VivaVideo - Video Editor&Maker	com.quvideo.xiaoying
491	Messages	com.google.android.apps.messaging
492	Phone by Google	com.google.android.dialer
493	Gboard - the Google Keyboard	com.google.android.inputmethod.latin
494	Angry Birds 2	com.rovio.baba
495	Huawei Mobile Services	com.huawei.hwid
496	PUBG MOBILE	com.tencent.ig
497	Angry Birds Classic	com.rovio.angrybirds
498	"Hago- Party, Chat & Games"	com.yy.hiyo
499	Google Street View	com.google.android.street
500	Device Care	com.samsung.android.lool

D The profiles

The purpose of this appendix is to disclose profiles that we used. Table 3 shows the apps that are used in each profile. Each app is represented by a number, which equals its rank in Appendix C minus 1. For example, the app with index 320 (i.e. rank 321) is the app with package name "com.google.android.gms" (i.e. "Google Play services"). This app has been added to every profile to ensure that Google Play services is always enabled. Without Google Play services, some apps do not work. Apart from Google Play services, every profile contains 50 other, randomly selected apps. These are the apps that will be enabled when a profile is activated. All of the other apps will then be disabled.

The randomly generated profiles	
Profile	App indexes (= $rank\_number - 1$)
0	320,315,147,404,355,286,243,358,77,144,304,161,447,329,410,345,208,287,412,276,461,328,297,472, 16,250,13,128,188,145,73,386,85,248,393,52,278,5,93,359,379,467,468,101,442,446,409,429,136,110,344
1	320,378,250,315,352,369,484,4,307,167,146,134,302,120,177,182,398,423,491,7,197,339,351,57,155, 40,417,427,145,388,455,450,49,449,498,442,135,92,274,207,323,87,411,344,299,164,336,221,111,129,173
2	320,197,355,243,246,3,245,335,300,473,88,328,397,132,29,46,0,406,75,133,73,37,36,383,79,399,278,460, 225,388,407,209,240,344,446,54,292,465,393,85,26,277,374,351,430,19,491,485,331,375,181
3	320,122,324,364,388,62,182,132,468,335,334,48,194,450,5,295,344,38,401,375,74,358,438,86,360,232, 230,134,338,157,186,359,139,425,328,277,101,106,350,57,67,275,486,150,60,70,387,239,406,156,278
4	320,434,425,138,45,220,468,296,476,482,480,439,355,47,433,364,10,313,310,75,42,281,209,240,211, 305,124,122,262,354,40,412,304,390,445,186,311,216,469,185,479,474,66,317,333,427,444,92,401,488,191
5	320,487,479,423,258,451,183,195,475,445,319,141,196,260,221,176,313,211,35,222,40,375,251,97,342, 357,488,394,125,419,421,410,25,85,177,89,135,398,150,483,58,26,477,148,31,397,307,407,367,215,273
6	320, 320,444,195,56,473,445,487,220,133,468,222,146,307,64,332,17,70,347,136,391,84,105,10,50,477, 192,49,103,231,66,326,169,134,380,204,283,262,117,365,180,88,407,28,394,318,335,342,111,427,18
7	320,395,290,200,98,469,308,61,302,251,448,0,65,172,429,38,240,432,31,399,369,418,355,259,338,397, 371,305,27,455,81,273,268,234,382,269,267,387,182,111,53,425,20,45,315,344,143,256,201,191,69
8	320,333,137,430,473,171,485,496,413,220,199,3,85,25,33,170,90,447,412,258,406,182,108,379,443,205, 444,173,308,60,38,215,359,69,465,274,121,310,356,470,46,290,47,222,98,303,489,219,318,70,35
9	320,446,490,106,397,111,491,183,333,456,137,81,180,45,98,136,205,187,144,95,27,470,495,315,388,287, 454,380,228,358,156,238,341,16,186,57,128,419,296,486,23,67,254,46,371,383,360,438,37,24,412
10	320,156,118,348,84,461,35,225,121,209,386,429,120,242,309,398,232,206,87,436,89,301,393,9,303,484, 311,426,308,294,473,298,390,12,438,291,247,191,279,205,119,33,11,14,81,478,281,109,403,15,302
11	320,243,333,53,459,37,280,69,274,409,429,161,22,75,63,95,416,183,155,141,422,41,11,176,129,68,403, 376,286,283,207,453,32,39,317,138,214,472,187,2,445,397,119,400,5,251,257,281,288,230,185
12	320,138,287,186,72,275,418,465,425,417,241,426,61,286,69,246,362,311,420,398,68,198,448,36,45,227, 442,41,232,129,12,114,349,379,207,278,32,429,144,365,28,487,480,279,222,452,297,310,329,50,188
13	320,101,10,45,128,365,364,42,257,325,380,20,262,465,162,43,158,297,184,65,104,293,12,452,390,206, 423,443,118,412,153,313,70,445,15,360,106,407,3,449,419,122,69,31,198,167,202,141,362,367,345
14	320,88,427,147,311,83,286,47,463,292,269,295,491,351,314,187,460,11,138,196,337,418,285,181,278, 62,166,32,152,324,168,274,139,157,232,357,28,4,261,266,459,235,43,474,184,50,10,159,116,52,382
15	320,212,341,426,202,258,115,388,402,239,45,93,375,185,251,379,268,43,47,75,316,378,100,170,248, 4,468,116,333,3,397,149,452,176,315,465,60,334,451,90,76,56,483,178,429,61, 320,159,196,238,329
16	320,188,253,78,214,191,108,438,451,174,20,239,121,99,439,413,444,35,456,30,198,419,373,11,16,466, 259,288,449, 320,268,443,465,445,147,195,9,182,197,51,66,100,448,193,127,498,109,85,53,70,482
17	320,242,470,279,114,431,275,209,179,300,317,210,90,442,388,253,78,307,162,24,350,310,27,474,182,193, 161,211,224,270,163,445,70,105,246,216,397,251,2,359,77,496,404,436,168,378,437,232,121,145,41
18	320,227,38,387,164,421,437,373,77,298,296,1,267,5,280,426,269,234,461,479,86,110,48,31,378,449,423, 304,16,315,300,389,124,410,55,13,420,51,40,143,220,250,106,455,282,299,273,182,73,161,99
19	320,1,448,288,473,202,338,192,47,479,38,486,308,399,241,232,163,247,351,10,286,81,487,319,250,37, 118,471,16,303,164,184,242,229,251,263,282,289,236,480,76,70,238,231,181,198,477,239,330,78,31

Table 3: The list of activated apps in each profile

E Results in table format

The tables below show the exact number of correct classifications for each time interval, for the different distance metrics that we used. In other words, each number denotes for how many out of the 200 pcap files we were able to correctly classify the profile that was activated when the data was recorded. A score of 200 correct classifications means a success rate of 100%, since we have a total of 200 PCAP files. The first column of each table denotes a time interval in seconds. The other columns are the different distance metrics.

Performance comparison of distance metrics with 60 minutes of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-3600	200	200	200	197	197	200	197	200

Performance comparison of distance metrics with 30 minutes of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-1800	200	200	200	198	198	200	198	200
1800-3600	155	167	199	168	168	196	168	176

Performance comparison of distance metrics with 20 minutes of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-1200	200	200	200	199	200	200	200	200
1200-2400	173	174	200	166	167	200	167	180
2400-3600	159	169	198	164	164	196	164	174

Performance comparison of distance metrics with 10 minutes of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-600	200	200	200	199	199	200	199	200
600-1200	178	181	200	175	175	200	175	187
1200-1800	167	169	200	160	161	200	161	173
1800-2400	149	157	200	160	162	198	162	167
2400-3000	148	154	199	150	150	197	150	164
3000-3600	162	171	200	163	164	198	164	172

Performance comparison of distance metrics with 5 minutes of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-300	200	200	200	198	199	200	199	200
300-600	191	193	200	192	192	200	192	198
600-900	188	190	200	179	179	200	179	193
900-1200	178	184	200	171	173	200	173	188
1200-1500	165	167	200	155	160	200	160	171
1500-1800	138	152	197	149	149	196	149	163
1800-2100	153	161	199	161	162	198	162	173
2100-2400	140	150	198	145	148	197	148	160
2400-2700	149	157	198	148	150	198	150	164
2700-3000	144	153	200	141	143	200	143	159
3000-3300	153	162	196	150	153	196	153	166
3300-3600	156	165	198	146	150	198	150	171

Performance comparison of distance metrics with 1 minute of data								
Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
0-60	179	186	192	181	182	192	182	187
60-120	183	191	197	183	185	197	185	193
120-180	180	189	200	179	180	200	180	193
180-240	175	182	199	171	172	199	172	187
240-300	146	153	192	139	141	192	141	161
300-360	186	190	198	185	185	198	185	193
360-420	162	165	197	144	148	197	148	175
420-480	146	154	191	140	145	191	145	160
480-540	152	160	192	143	144	192	144	166
540-600	137	144	178	144	145	178	145	154
600-660	156	163	190	148	149	190	149	171
660-720	144	152	188	132	133	188	133	161
720-780	133	139	181	129	133	181	133	149
780-840	157	164	184	154	155	184	155	168
840-900	144	147	195	137	138	195	138	155
900-960	158	165	190	148	151	190	151	171
960-1020	148	148	183	141	143	183	143	154
1020-1080	138	145	183	138	142	183	142	153
1080-1140	154	160	180	145	147	180	147	162
1140-1200	118	125	172	116	121	172	121	135
1200-1260	137	142	181	127	130	181	130	157
1260-1320	128	133	174	118	119	174	119	145
1320-1380	103	113	158	100	103	158	103	127
1380-1440	115	120	156	108	109	156	109	132
1440-1500	96	102	153	106	109	153	109	111
1500-1560	93	113	162	103	113	162	113	119
1560-1620	99	104	156	102	103	156	103	113
1620-1680	100	104	149	90	93	149	93	114
1680-1740	123	136	175	106	107	175	107	140
1740-1800	108	121	165	117	120	164	120	133
1800-1860	132	140	185	135	135	184	135	150
1860-1920	126	132	172	117	122	172	122	142
1920-1980	114	129	160	100	104	160	104	134
1980-2040	114	117	158	106	107	158	107	127
2040-2100	119	120	171	112	113	171	113	127
2100-2160	111	113	163	106	108	163	108	123
2160-2220	109	118	167	104	111	167	111	124
2220-2280	100	115	157	100	107	157	107	122
2280-2340	103	107	147	102	104	147	104	113
2340-2400	100	104	156	114	114	154	114	119
2400-2460	119	127	164	125	130	164	130	137
2460-2520	112	120	166	117	117	166	117	129
2520-2580	104	115	164	101	101	164	101	124
2580-2640	119	123	155	105	107	155	107	128
2640-2700	119	125	165	114	117	165	117	131
2700-2760	108	109	159	102	103	159	103	122
2760-2820	106	114	162	108	114	162	114	126
2820-2880	112	119	162	105	108	162	108	125
2880-2940	112	115	163	102	103	163	103	122
2940-3000	107	112	160	113	116	160	116	124
3000-3060	120	136	168	125	125	168	125	141

Continued on next page...

Time (s)	dice	jaccard	kulsinski	matching	rogerstanimoto	russellrao	sokalmichener	sokalsneath
3060-3120	110	117	165	102	110	165	110	129
3120-3180	98	103	142	101	101	142	101	112
3180-3240	108	113	147	107	111	147	111	120
3240-3300	106	114	151	106	109	151	109	117
3300-3360	103	108	146	102	104	146	104	112
3360-3420	98	104	159	97	98	159	98	114
3420-3480	88	98	149	93	99	149	99	115
3480-3540	120	127	172	110	115	172	115	136
3540-3600	109	121	165	118	119	165	119	133