Licenciatura em Engenharia Informática e de Computadores

Aplicação Android 5G QoS

Afonso Nobre, nº 44777

Ricardo Silva, nº 44837

Orientadores

José Simão

Nuno Cota

Relatório beta realizado no âmbito de Projeto e Seminário,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2020/2021

Maio de 2021

Resumo

Os recentes desenvolvimentos na rede 5G oferecem não só uma interface de rádio sofisticada como também uma eficiente arquitetura de sistemas de rede. Estes avanços disponibilizam um leque de oportunidades e novas aplicações, como por exemplo a tecnologia na construção de veículos autónomos. Para determinar a influência das condições da rede em aplicações que dela dependem, é usado um sistema já existente capaz de gerar tráfego em diferentes níveis e protocolos permitindo a recolha de informação para análise da qualidade do serviço.

A solução apresentada integra um sistema de monitorização de redes 5G designado IQ-NPE, o qual é composto essencialmente por três tipos de componentes, nomeadamente:

- *On Board Units (OBU)* - Peça de *hardware* e *software* móvel instalada em veículos de forma a gerar tráfego e recolher os parâmetros de medida de performance em diferentes pontos de controlo e observação.
- *Fixed Side Units (FSU)* - Agente de *software* fixo instalado em localizações estrategicamente pensadas tanto em Portugal como em Espanha. Estes têm a responsabilidade de gerar tráfego e comunicar com as sondas móveis para recolher parâmetros de rede tais como o *downlink* e o *uplink.*
- *Management System* - Plataforma de *software* centralizada responsável pela gestão e configuração de planos de testes. Também é responsável por recolher e processar todos os resultados obtidos pelas sondas durante a execução dos planos de testes.

O desenvolvimento do projeto é motivado pela oportunidade da criação de uma aplicação móvel para simular uma sonda móvel mais simplificada tendo como alvo smartphones comuns, de forma a complementar o sistema já existente oferecendo mais portabilidade e versatilidade.

Palavras-chave: 5G; smartphone; aplicação móvel; Management System

Abstract

The recent developments in 5G technology brings not only a sophisticated radio interface, but also a performant network system architecture. These technical achievements may bring new opportunities and new applications, for example, autonomous vehicles. To determine the influence of radio network conditions on the applications performance, it will be used a system that generates synthetic traffic at different levels and different protocols to collect information, allowing the system to analyze the service quality.

The current solution presents a 5G network monitorization system named IQ-NPE. The system architecture is mainly composed by three components:

- On Board Unit (OBU) – Hardware and software probe to be installed on vehicles to generate traffic and collect performance measurements at different points of control and observation.

- Fixed Side Units (FSU) - Software agent to be installed on both Portugal and Spain. The fixed side unit is used to generate traffic and collect performance measurements on the network side, on both downlink and uplink traffic.

- Management System - Centralized software platform used to manage test plan configuration. It will also be responsible for collecting and processing all performance assessment results obtained during test trials.

The development of this project is motivated by the opportunity of developing a mobile application to simulate a simplified OBU in an ordinary mobile phone to complement the Management System by offering more portability and versatility.

Keywords: 5G; Mobile; Application; Management System

Índice

Lista de Figuras

1. Introdução

Neste capítulo é introduzida a iniciativa para este projeto, a motivação, os objetivos e por último uma breve descrição da estrutura do relatório.

1.1 Motivação

O surgimento de avanços no desenvolvimento da tecnologia 5G abre um novo leque de oportunidades, motivando a criação de novas aplicações capazes de usar esta tecnologia.

O âmbito é criar uma aplicação que permita oferecer uma nova frente ao projeto *Isel Quality of Service Network Performance Evaluation* (*IQ-NPE*)[20], para que este possa criar um plano exaustivo de análise à cobertura à nova tecnologia.

Uma vez que o sistema já desenvolvido tem uma componente portátil bastante limitada surgiu a necessidade de desenvolver uma aplicação móvel de forma a tentar minimizar este problema.

O *smartphone* é um dispositivo cada vez mais comum no quotidiano da sociedade, pelo que se torna no candidato perfeito para complementar o sistema. Com um equilíbrio entre a portabilidade e as necessidades de *hardware*, é criada uma janela de oportunidade para desenvolver uma aplicação que tire partido da flexibilidade e usabilidade do *smartphone* para executar requisitos delegados pelo sistema de gestão.

Figura 1 - Representação de uma sonda *OBU* num veículo

1.2 Objetivo

O objetivo deste projeto é desenvolver uma aplicação móvel para complementar o sistema já existente desempenhando de forma simplificada uma *On Board Unit*, dessa forma oferecendo mais portabilidade e simplicidade na interação local com a sonda. Esta tem que ser capaz de recolher parâmetros de rede de forma passiva, sem criar tráfego ou influência na rede, e realizar planos de testes propostos pelo sistema de gestão.

1.3 Contextualização e Especificações do Projeto

Para garantir que a aplicação seja realmente capaz de executar estas tarefas devem ser cumpridos os seguintes requisitos:

- Fazer a realização de testes passivos, que consistem na recolha de um conjunto de parâmetros de forma a não interferir no tráfego da rede do dispositivo. Estes parâmetros variam entre dados de débito binário, dados de georreferenciação e dados de qualificação do sinal de rede;
- Posteriormente à realização dos testes passivos, a aplicação tem de ser capaz de fazer a disponibilização dos resultados, por exemplo sob forma de gráficos ou tabelas. Tanto a recolha dos parâmetros como a sua ilustração são feitas localmente no dispositivo;
- Fazer a realização de testes ativos que, ao contrário dos testes passivos, vão causar tráfego intencional na rede, de forma a conseguir uma avaliação da qualidade e cobertura da mesma. A execução de uma sequência de *Pings* para um domínio explícito é um exemplo de um teste ativo que a aplicação realiza;
- Fazer a recolha de planos de testes. Estes planos são fornecidos pelo sistema central via *Web API* e consistem numa lista de diferentes tipos de testes que são executados pela aplicação de forma autónoma e sem interação do utilizador. Os testes provenientes de cada plano podem ser de natureza passiva ou ativa;
- Armazenar os resultados de testes realizados fora do contexto de um plano de testes na base de dados local do dispositivo;
- Reportar os resultados da execução de testes provenientes dos planos de testes para o sistema de gestão central.

1.4 Estrutura do relatório

Ao longo do relatório vai ser feita uma descrição geral e detalhada dos aspetos mais relevantes da aplicação implementada. No próximo capítulo vai ser feita uma análise ao sistema já existente e como é que foi feita a integração da nova aplicação no que já existia. Depois é feita uma descrição da arquitetura desenhada para a implementação da aplicação, desenvolvendo na premissa dos desafios que foram propostos e qual a abordagem tomada para os tentar solucionar. O capítulo 4 entra em detalhe sobre cada ponto da implementação da aplicação. O capítulo 5 representa uma avaliação com uma comparação da aplicação implementada com uma aplicação já existente na qual esta foi inspirada. Por fim, no capítulo 6 é feita uma conclusão sobre o trabalho feito e apresentado o trabalho futuro possível.

2. Integração com o sistema *IQ-NPE*

2.1 Ligação ao Sistema *IQ-NPE*

O sistema *IQ-NPE* (figura 2) é uma complexa rede de sondas que, ao comunicarem entre si, conseguem criar condições de tráfego virtual e executar testes e análises à cobertura da rede na posição em que se encontram.

As sondas *on-board* (*OBU*) encontram-se instaladas em veículos, e contém vários *modems* para fazer os testes. Contêm uma interface *web* local, que permite que a sua manutenção seja feita por ligação direta ao dispositivo. Estas conseguem também ligar-se ao *CAN BUS* [1] do veículo, de forma a aumentar a eficiência e precisão das análises feitas. A maior parte da análise é feita *offline*, e só apenas depois de todas as medidas estarem completas é que é feita a transmissão dos dados recolhidos para o sistema central.

As sondas fixas (*FSU*) encontram-se posicionadas em certas localizações ao longo da Península Ibérica e correm numa máquina virtual dedicada, que vai ser instala na *Multi-access Edge Computing* (*MEC)* [2] da tecnologia *5G* e/ou no centro *Cooperative Intelligence Transport Systems* (C-*ITS*) [3]. A nível de *hardware*, estas sondas são semelhantes às sondas *on-board*, com a exceção que estas não têm *modems* de comunicação móvel, nem a interface *web* local, porque visto que estas se encontram ligadas à rede central, podem ser geridas pelo sistema de gestão.

O sistema de gestão central é a ponte entre todos estes dispositivos. Este sistema está ligado a todas as sondas e faz a gestão da execução de testes, para evitar interferências no tráfego da rede *5G*. Para evitar interferências o sistema usa conexões fora-de-banda através de redes 3G/4G, usando um *Public Land Mobile* Network (*PLMN*) diferente. É este sistema que constrói e delega os diferentes planos de testes que as sondas vão executar, bem como recebe os resultados e os processa, para depois os entregar a ferramentas especializadas na sua análise.

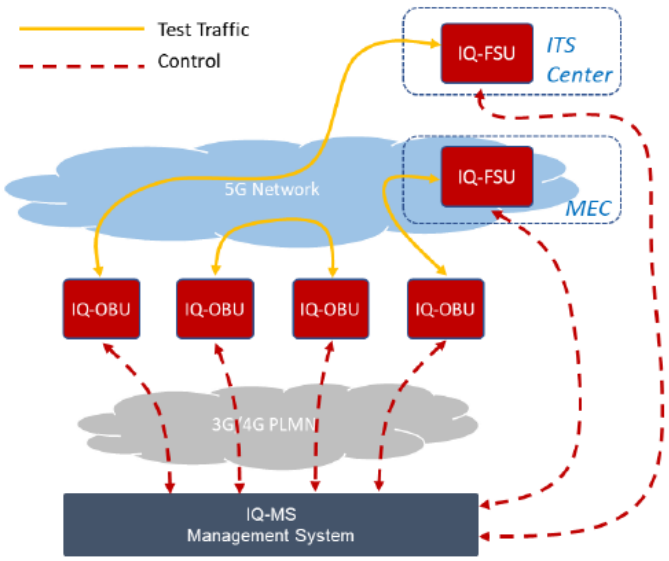

Figura 2 - Arquitetura do sistema *IQ-NPE* [20].

Os planos de testes vêm na forma de um objeto *json* (figura 3) que contém a informação necessária para que cada sonda do sistema seja capaz saber quando e como executar o plano, bem como saber para onde reportar os resultados obtidos da sua realização.


```
{
    "id": "e4f17256-d3a4-11eb-8504-005056840996",
    "name": "New Test Plan",
    "creator": "f28f460e-9ba5-11eb-8e24-005056840996",
    "creatorName": "Ricardo Silva",
    "created": "2021-06-22T21:57:53.270879",
    "modifier": "f28f460e-9ba5-11eb-8e24-005056840996",
    "modifierName": "Ricardo Silva",
    "modified": "2021-06-22T22:02:56",
    "href": "http://192.68.221.41:8080/mobix/probe/43/test-plan/e4f17256-d3a4-11eb-
8504-005056840996",
    "stateDate": "2021-07-02T14:54:37.458",
    "state": "PENDING",
    "startDate": "2021-06-22T21:57:52",
    "stopDate": null,
    "tests": [
        {
            "testType": "PING",
            "id": 1,
            "name": "Ping Test Nobre",
            "delay": "PT30S",
            "timeout": "PT1M",
            "nPings": 200,
            "messageInterval": "PT0.1S",
            "server": {
                "host": "www.google.pt",
                "port": null
            },
            "measurePeriod": "PT1S",
            "href": "http://192.68.221.41:8080/mobix/probe/43/test-plan/e4f17256-
d3a4-11eb-8504-005056840996/setup/1/test-sequence/1/test/1"
        }
    ]
}
```


Figura 3 - Exemplo de um *json* de um plano de testes.

Estes planos de testes fornecidos pelo sistema podem ser criados na aplicação cliente *web*, seja por autoria dos administradores do sistema como dos *developers* da aplicação *mobile.* Foi feita uma interface para possibilitar a criação de planos de testes com facilidade de forma a tirar o máximo de proveito de cada tipo de sonda com testes personalizados (figura 4). Após a criação de um plano de testes com o número desejado de testes a realizar, o criador deve agendar o teste para que este passe a estar disponível para a sonda para o qual o plano foi criado.

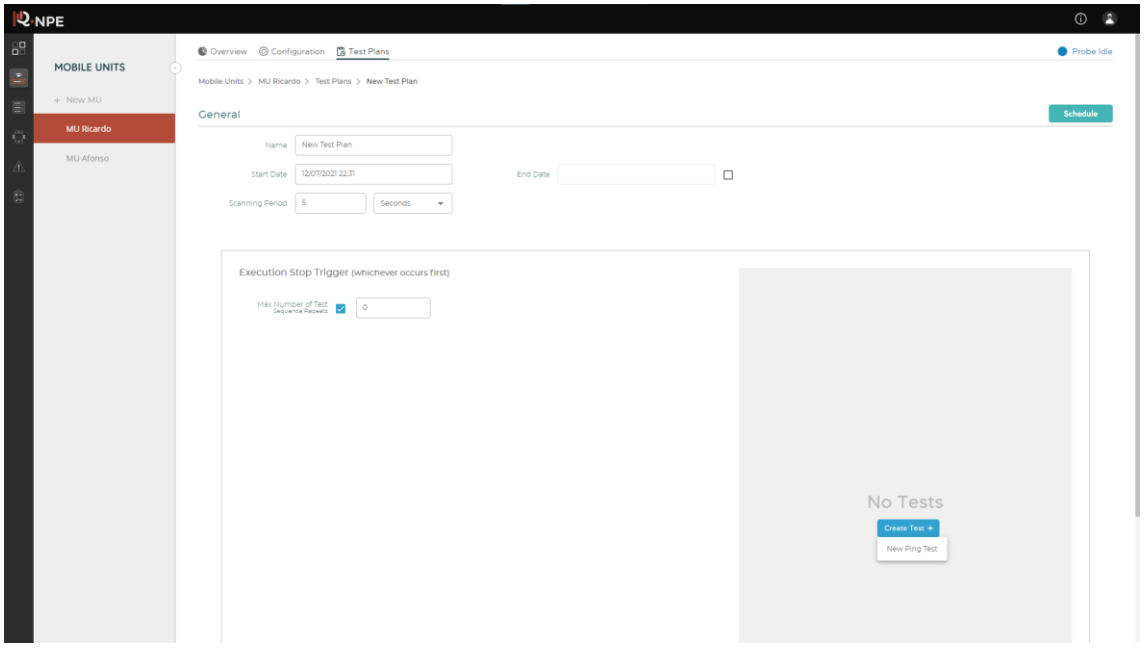

Figura 4 - Criação de um plano de testes para a aplicação móvel.

2.2 Integração no sistema

A aplicação móvel foi desenhada à imagem de uma sonda *on-board*. No entanto, visto que o *smartphone* não é tão preciso e tão capaz como uma dessas sondas, pequenas adaptações foram feitas na modelação da aplicação. Apesar dos testes passivos à cobertura serem semelhantes (excetuando a interface de ligação ao veículo), os planos foram simplificados para que o telemóvel se comporte com o desempenho desejado. Face a estes requisitos, tiveram de ser criadas novas rotas na *API* do sistema de gestão *IQ-MS*, desenvolvidas exclusivamente para a nova sonda móvel (*Mobile Unit – MU*). Assim sendo, a interação entre sondas e *MUs* é feita de forma semelhante às comunicações de sondas de outros tipos entre si, e dessa forma é possível manter coerência na nova entrada no sistema (figura 5). A comunicação com o sistema é realizada através de uma *Web API* usando como autenticação *Basic Authentication.*

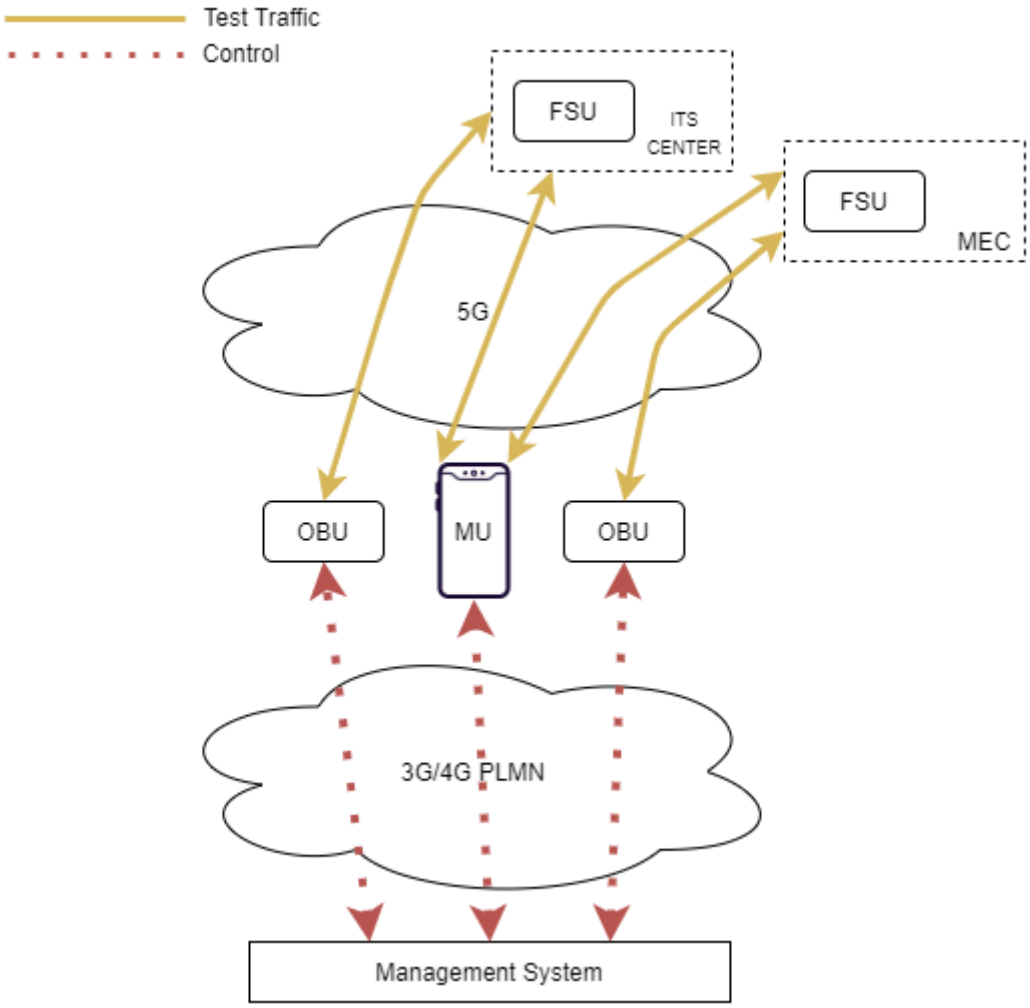

Figura 5 - Arquitetura do novo Sistema com a *MU.*

3. Arquitetura

Neste capítulo é realizada uma descrição detalhada da arquitetura adotada para a construção da aplicação android e a sua integração no sistema previamente desenvolvido.

3.1 Desafios Propostos

Desenvolver uma aplicação de testes à cobertura da rede não é tão linear quanto soa.

O desafio inicial foi fazer a execução de testes passivos na aplicação. Estes testes consistem na recolha de dados de naturezas diferentes e, visto que diferentes tipos de dados requerem diferentes processos de acesso ao sistema A*ndroid*, os testes são retirados sobre contextos diferentes. Por exemplo, para retirar dados sobre o débito binário é necessário aceder a uma framework especializada no tráfego, e para recolher a localização é necessário aceder a uma *framework* especializada em dados de localização.

Além da simultaneidade na execução de testes, é necessário garantir que a análise de rede seja feita em segundo plano, bem como garantir que a análise seja feita quando a aplicação se encontra tanto em *foreground* como no *backstack* do *Android*. Quando a aplicação se encontra em *foreground*, as recolhas feitas em segundo plano são lidas e apresentadas ao utilizador, sob forma de gráfico, tabela, ou mesmo um conjunto de campos nomeados.

Existe um conceito de sessões de teste controladas no contexto da aplicação. Estas são sessões cujo propósito é fazer a realização de testes passivos com o intuito de armazenar os seus resultados no dispositivo de forma a possibilitar ao utilizador a visualização e a extração, para ficheiro *.xls*, de todos os resultados recolhidos. O objetivo desta funcionalidade é permitir ao utilizador fazer uma consulta posterior dos resultados que foram obtidos no momento da recolha.

Além do conceito de sessões controladas, existe também um conceito de planos de testes. Estes planos contêm listas de testes ativos ou passivos, que permitem à aplicação executar uma bateria de avaliações à rede em determinado momento. Os testes são disponibilizados pelo sistema de gestão e a estes vem acoplada uma data de início do plano. O objetivo é que seja possível programar a execução de vários testes para um futuro breve. Com este objetivo surgem dois problemas. O primeiro é a possível dificuldade de comunicação com o sistema de gestão, uma vez que o *token* de autenticação necessário poderá ficar inválido, e o segundo é a natureza dos diferentes testes. Alguns dos testes do plano podem ser de natureza assíncrona, o que derrota o princípio da execução de testes isolada de forma a minimizar as interferências na rede.

3.2 Solução para os objetivos propostos

Para o primeiro desafio foi implementado um padrão que contenha diferentes implementações de recolha para cada conjunto de parâmetros de rede, e um algoritmo que invoque em simultâneo cada implementação. Assim, é possível recolher todos os tipos de parâmetros de rede abrangidos pelos testes passivos e ilustrar os seus resultados ao utilizador em tempo real.

Na tarefa da execução de testes passivos a paralelização é conseguida através do uso de uma *thread* da *Thread Pool* do sistema para fazer a recolha e armazenamento, em base de dados, de cada tipo de parâmetro. Deste modo, atráves da base de dados é possível para a *UI* obter os resultados das recolhas.

Na paragem da aplicação, quando esta passa para o *backstack* do *Android*, a representação de resultados por parte da *UI* deixa de ser necessária, mas a recolha de parâmetros continua a ser executada pela *thread* de *background* a quem as tarefas foram delegadas, resolvendo assim o primeiro desafio.

Para resolver o problema do *token* de autenticação, é feita uma revalidação do *token* de 15 em 15 minutos, para que este seja válido no momento da execução dos testes.

Para resolver o problema do assincronismo dos testes ativos, é necessário sequencializar a sua execução, garantindo que o teste seguinte apenas é executado quando o anterior terminar e devolver os resultados.

3.3 Arquitetura da aplicação

Para todos os desafios e soluções apontados nas subsecções anteriores foi desenhada uma arquitetura semelhante à arquitetura típica de uma aplicação *Android* (figura 6). É feito uso de *Activities* e *Fragments* para a camada de apresentação, ou seja, estes apenas têm a responsabilidade de gerir a *UI*, as interações entre o utilizador e a aplicação e as interações com o sistema operativo *Android*. Comunicam também com o *ViewModel*, elemento pertencente à camada de acesso a dados cujo tempo de vida está restringido ao tempo de vida da *Activity* ou *Fragment* que o usar. Para fazer o acesso a dados o *ViewModel* acede a um *Repository*, que é o responsável por fazer as recolhas de dados. Para os recolher, existem duas rotas possíveis: fazer um pedido à base de dados local *Room* ou fazer um pedido *HTTP* à *Web Api* disponibilizada pelo sistema *IQ-NPE* fazendo uso da biblioteca *Volley*.

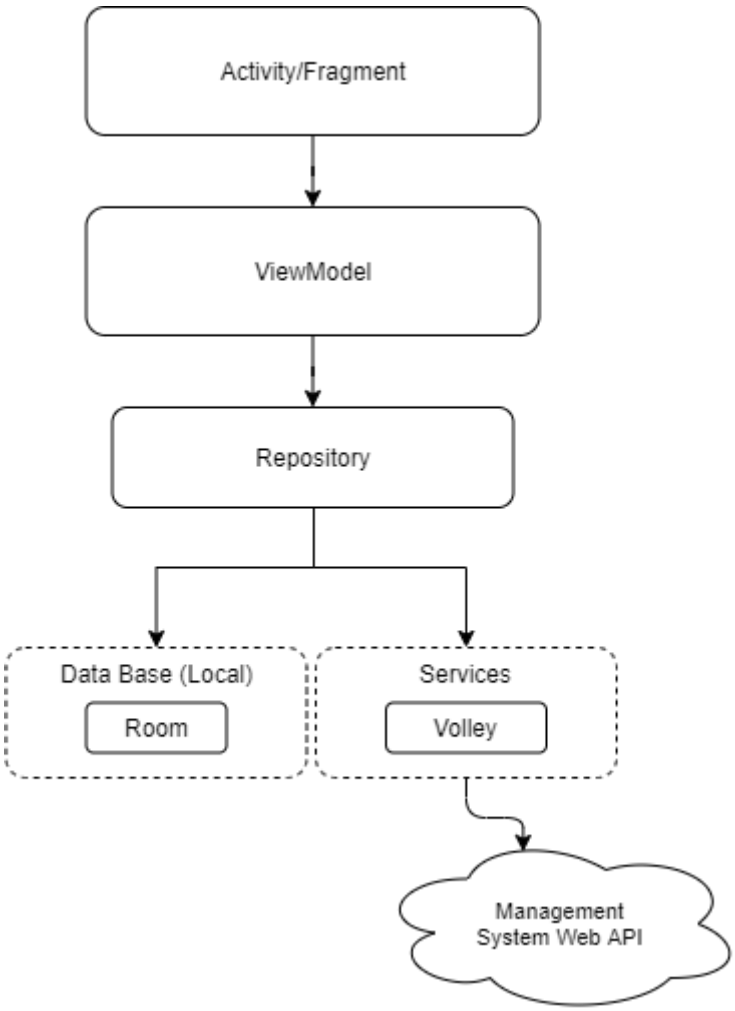

Figura 6 - Arquitetura geral da aplicação.

3.4 Visão geral das tecnologias utilizadas

À luz do facto de que a aplicação implementada não encaixar no estereótipo habitual de uma aplicação mobile, e fugir um pouco à modelação conceptual que é costume encontrar noutras aplicações, foi feita a escolha de implementar a mesma em nativo, usando a linguagem de *Kotlin*.

Foram usadas várias *Frameworks* para implementar a lógica da aplicação, designadamente:

- *Volley* [4] – Utilizado para fazer os pedidos *HTTP* à *API* disponibilizada. É possível redefinir o modo como os pedidos e respostas são processados para personalizar cada pedido à forma da aplicação.
- *MPAndroidChart* [5] – Esta é a *Framework* utilizada para o desenho dos vários gráficos de informação que se encontram ao longo da aplicação. Com uma estrutura clara, é possível definir o número e aspeto de varáveis presentes em cada gráfico e, através da afetação de *flags*, personalizar a navegação dentro de cada gráfico também.
- *Room* [6] – Implementado por base em *SQLite*, esta é a *Framework* que permite fazer a criação de uma base de dados local ao processo da aplicação. Através de um esquema de Entidades e classes *DAOs* (*Data Access Object*), é possível fazer pedidos de base de dados com facilidade.
- *Material UI* [7] – *Framework* de componentes visuais. É a fonte de vários componentes relativos à *UI* da aplicação, tanto estruturais como meramente visuais.

4. Detalhes de implementação

Nas próximas secções vai ser explicado a forma como a aplicação foi desenhada, dividindo a apresentação de cada um dos principais componentes da solução.

4.1 Autenticação

O primeiro componente da aplicação é o componente que faz a gestão da autenticação do utilizador. Para uma autenticação bem-sucedida é necessário que o utilizador faça o *login* via *endpoint* com uso de *Basic* Authentication na aplicação, que regista o dispositivo na rede, para poder usar a aplicação.

Para o utilizador se autenticar no sistema é necessário que este tenha sido registado previamente no sistema de gestão. Essa responsabilidade faz parte dos administradores do sistema de gestão, pelo que apenas por pedido aos mesmos é possível fazer o registo. Após registado, um utilizador recebe o *username* e a *password* dos administradores. Numa primeira instância têm de ser preenchidos 3 campos: *username*, *password* e *serial number* (figura 7). Este último campo tem de ser recolhido por parte do utilizador a partir das definições do dispositivo, é só é inserido na primeira vez que este faz *login*. Este campo não é obtido de forma automática visto que a partir da versão de *Android* 10 em diante, por motivos de segurança e privacidade, não é possível obter o identificador programaticamente.

Assim que o processo de *login* do utilizador estiver concluído, a aplicação vai passar a registar o dispositivo na rede. Após esse processo, o *serial number* fica gravado na biblioteca de *settings* da aplicação. Desse modo, ao usar o número de série do dispositivo, pode-se garantir a identidade de cada *smartphone* no registo da rede do sistema, e torna-se possível *logins* de vários utilizadores num só dispositivo.

Figura 7 - *Screenshot* de *login*.

4.1.1 Revalidação do *token*

O *token* tem formas de ficar inválido, sendo o *timeout* uma delas. A aplicação tem como objetivo realizar a execução de planos de testes a qualquer momento. Para isso é necessário que o utilizador tenha o *login* feito até que este decida o contrário. Para evitar que o utilizador tenha de inserir as credenciais sempre que usar a aplicação, a partir do momento em que o *login* está feito, é lançado um *Worker* [9] que, de 15 em 15 minutos, irá executar a tarefa de fazer um controlo de ligação que realiza a revalidação do *token*. Esta revalidação é feita através de um *endpoint* da *API*, para que a aplicação se possa manter em comunicação com a *API*.

4.2 Recolha de parâmetros de rede

Uma vez dentro da aplicação, é apresentado um ecrã com uma grande quantidade de informação. A aplicação é capaz de recolher parâmetros de várias naturezas de forma autónoma. Esta subsecção explica como é feita cada uma dessas recolhas, e o porquê da abordagem implementada.

Em primeiro lugar é necessário destacar que são feitas duas recolhas em simultâneo: a recolha de parâmetros de rede móvel, de rádio e de localização (denominados daqui em diante de *radio parameters*) e a recolha dos valores instantâneos de débito binário, ou seja, a velocidade de *upload* e *download* (denominados *throughput*). Feita a distinção, assim que o utilizador passa do *login*, é lançada uma entidade que é responsável pela recolha desses parâmetros em segundo plano. A fim de cumprir a execução de recolha, foram contempladas duas propostas de solução.

A primeira era fazer uso de uma instância de *WorkManager* [10]. O objetivo é que esta entre num *loop* que sirva de entidade responsável pela recolha dos parâmetros, e fica dentro do ciclo durante tempo indeterminado. O problema é que ao iniciar uma sessão de testes passivos, é necessário criar uma nova instância com novos parâmetros, mas visto que não é garantido fazer a paragem de execução da instância quando esta tiver iniciado, esta proposta mostrou-se inadequada.

Como segunda proposta a ideia foi fazer uma implementação genérica de *JobService* [12]. A maior diferença entre a implementação anterior é que esta *framework* permite fazer o cancelamento de um trabalho pelo seu *id*. O objetivo é fazer com que a redefinição da função *onStartJob* proveniente de *JobService* atue como um delegado que executa as tarefas de recolha de parâmetros. Estas duas tarefas de recolha são executadas com uso do padrão de desenho *Strategy*[13]. Foi definida uma abstração *IJob* (figura 8) e, por cada recolha a ser feita, uma concretização dessa abstração que redefine a função *job* da abstração. Através de uma identificação por enumerado, a instância de *JobService* criada consegue identificar todas as concretizações desejadas e executar cada uma delas.

```
interface IJob {

    fun job (params: Map<JobParametersEnum, Any?>)

    fun getJobTimeout () : Long

    fun getJobParameters():Array<JobParametersEnum>
}
```

Figura 8 - Implementação da abstração *IJob*.

```
val workInstance = JobsMap.worksMap[jobType]

if (System.currentTimeMillis() > (lastRuns[index] +
workInstance!!.getJobTimeout())) {
    lastRuns[index] = System.currentTimeMillis()

    asyncTask({

        workInstance.job(createWorkerParams(workInstance))

    })

}
```

Figura 9 - Inicialização de um trabalho por parte do *Scheduler*.

Dessa forma é possível ter o delegado genérico *JobService* que injeta as dependências e executa cada instância da função *job* de cada concretização de *IJob*. Faz parte da responsabilidade deste *scheduler* saber quando executar cada um dos trabalhos de recolha. Dentro de cada concretização há duas outras importantes funções chave para o correto funcionamento de cada trabalho: uma função que devolve o intervalo de frequência, em milissegundos, em que este trabalho deve ser executado, e uma que devolve uma lista de identificação de parâmetros que o trabalho precisa. O *scheduler* requer apenas de fazer a criação dos parâmetros necessários e chamar a instância de *job* sempre que o momento atual exceda o momento da última execução adicionado ao tempo de *timeout* (figura 9).

De cada vez que o trabalho de cada uma das concretizações é executado, as informações recolhidas são gravadas na base de dados local do processo. Gravar estas informações na base de dados permite que a *UI* seja notificada sempre que houver uma alteração aos registos, ou seja, um novo bloco de informação relevante para apresentar ao utilizador.

Com esta implementação torna-se possível fazer facilmente a criação de novas concretizações de *IJob*.

4.3 Recolha e representação de dados

Uma vez feita a recolha de todos os parâmetros necessários, é altura de notificar a *UI* sobre as alterações que decorreram, para proporcionar ao utilizador a observação de como as alterações estão a fluir.

4.3.1 Recolha de dados

Para a *UI* poder ser notificada, tem de ser criada uma ponte entre a atividade e a base de dados. Essa ponte é feita através de uma observação em tempo real a uma *query* à base de dados. Por paradigma do *Room*, uma pesquisa na base de dados devolve uma instância de *LiveData* [14]. Essa instância contém uma função denominada *observe()* que recebe dois parâmetros:

- O contexto da observação (que é a instância do *lifecycleOwner* [15]);
- Uma instância de *Observer* [16];

Este segundo parâmetro de entrada atua como uma função de *callback* que recebe como parâmetro os dados que se pretendem observar. Dessa forma, ao registar um O*bserver* a uma *query* à base de dados, é possível executar o *callback* sempre que o trabalho faça uma recolha com sucesso.

4.3.2 Representação de dados

Sempre que o *callback* de *Observer* executa, vai ser feita uma atualização dos gráficos que são apresentados no ecrã. Em cada execução, ao analisar o parâmetro de entrada do *callback*, é criada uma nova entrada na informação a apresentar. Esta nova informação vai ser apresentada sob várias formas, sejam elas uma tabela de detalhes sobre cada célula móvel disponível no dispositivo, ou num gráfico de linhas com um eixo de abcissas temporal, que revela a evolução dos valores de cada parâmetro de rede.

No primeiro caso, o utilizador depara-se com um separador "*Main*" que oferece uma tabela com dados e outros detalhes em relação às diferentes células de rede. Estes valores são atualizados sempre que o c*allback* dispara, ou seja, quando há uma alteração proveniente da camada de acesso a dados (figura 10).

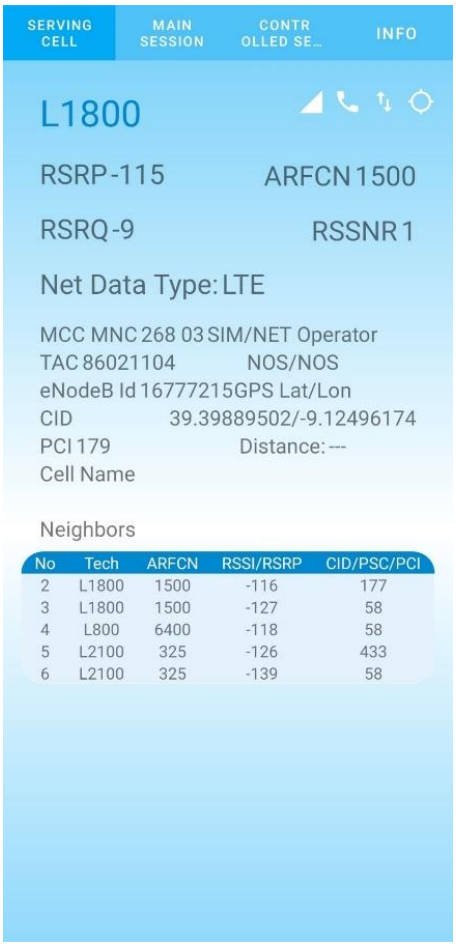

Figura 10 - Exemplo dos detalhes de cada uma das células de rede móvel.

O segundo separador "*Graphs*" oferece uma variedade de gráficos que apresentam partes de um todo a nível de parâmetros de rádio. Estes apresentam a evolução a nível temporal de todos os parâmetros de rede da célula de serviço atual do dispositivo e do débito binário. Em cada execução há uma atualização da informação por parte de cada gráfico, é criada uma nova entrada e cada instância de gráfico é notificada de que houve uma alteração. Posteriormente a aplicação avança o gráfico sobre o eixo temporal de forma que o utilizador esteja sempre a ver a informação atualizada (figura 11).

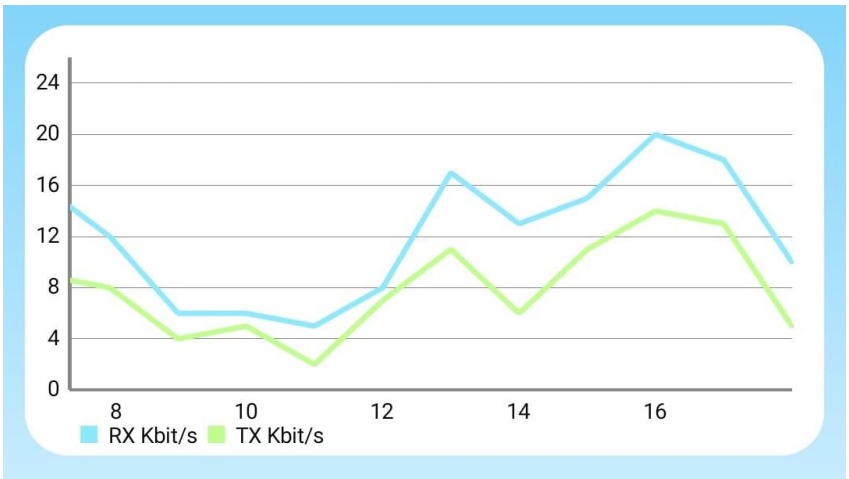

Figura 11 - Gráfico do *Throughput*.

Como foi referido anteriormente, existe um conceito de sessões de teste. Mas para que a aplicação possa disponibilizar ao utilizador a visualização da informação recolhida sem que este tenha de gravar uma sessão de teste, existe também uma sessão principal, denominada sessão *default*.

Quando o utilizador inicia uma nova sessão de testes tem de se fazer a paragem da sessão *default* para que a aplicação se possa concentrar inteiramente na nova sessão. Nesta transição é necessário que todas as instâncias de *observer* registadas para a sessão anterior sejam canceladas, para se poderem registar novas instâncias que apontem para a nova sessão. Este processo é necessário no sentido em que se houver mais do que um *observe* para o mesmo gráfico, dá-se um conflito de informação retornada, pelo que o utilizador iria ver informação errada.

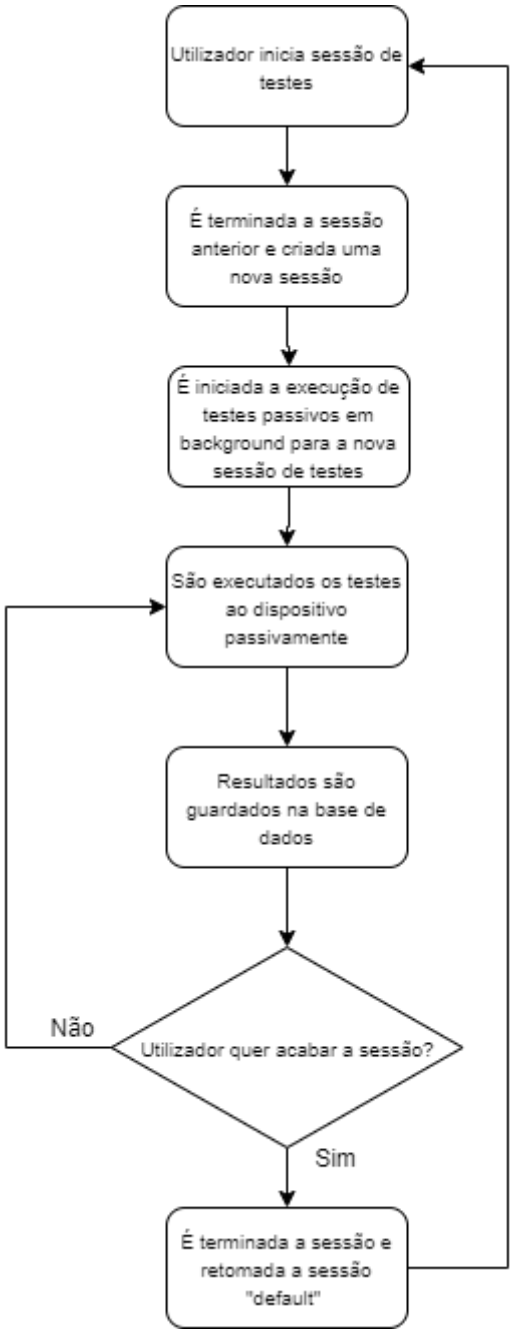

Figura 12 - Fluxograma de início e término de uma sessão de teste.

4.3.3 Comunicação entre fragmentos na criação de novas sessões de teste

Sendo que os fragmentos utilizam informação da mesma fonte, é necessário que estes estejam sincronizados entre si para que a informação apresentada entre fragmentos seja coesa. Dessa forma, houve uma necessidade de implementar um mecanismo que permita que os fragmentos comuniquem entre si. A solução foi introduzir a *framework EventBus*[8]. Esta *framework* fornece ao sistema *Android* a possibilidade de fazer a comunicação de eventos. Dessa forma é possível criar uma estrutura de *Publisher/Subscriber*, que serve de ponte de comunicação entre fragmentos.

O fragmento "*Controlled Session*" apresenta um botão de criação de uma sessão de testes. Quando o utilizador clica nesse botão, é criada uma notificação sob forma de evento, que irá ser comunicada em *Broadcast*. Assim, todos os elementos que estejam dependentes da alteração estão registados na *framework* do *EventBus* conseguem estar subscritos ao *Broadcast* do evento propagado pelo fragmento que contém o botão. Assim estes podem ser informados de quando uma nova sessão vai iniciar e podem reiniciar a observação de dados, para esta passar a referir o *id* da nova sessão.

Quando o utilizador decide terminar a sessão de testes, clica no botão de fecho de sessão que se encontra também no terceiro fragmento, e um processo igual ao de criação vai decorrer, terminando assim a sessão. Após o seu término, vai aparecer uma nova entrada no fragmento que contém os botões, que corresponde à sessão gravada. Esta nova entrada, ao ser clicada, demonstra num *Dialog* as informações, sob forma de gráficos iguais aos do fragmento "*Charts*", que foram gravadas durante o tempo em que o utilizador manteve a sessão de testes a correr. É também disponibilizada uma forma ao utilizador de exportar os dados para um ficheiro *.xls*, para que este possa ter uma forma de extrair os dados para fora do contexto aplicacional.

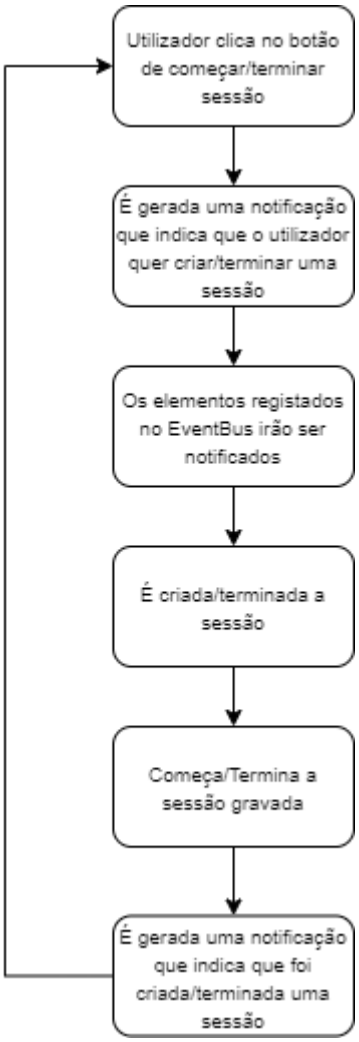

Figura 13 - Fluxograma de comunicação entre fragmentos.

4.4 Implementação de testes autónomos

Esta subsecção vai explicar o funcionamento e processamento necessários para a execução de testes autonomamente. Estes testes podem ser ativos ou passivos, e vêm em conjuntos denominados planos de testes.

4.4.1 Obtenção de planos de testes

Estes planos de testes vêm no corpo da resposta ao *endpoint* de controlo de conexão. Nesta resposta vem uma lista de identificadores que correspondem aos *ids* dos planos de testes pendentes de execução pela sonda em questão. Posteriormente ao fazer o pedido de controlo, é feita uma iteração pela lista para obter os detalhes de cada plano de testes. Cada plano de testes contém uma lista de testes, que serão executados de forma diferente mediante o seu tipo. De momento, por convenção, existe apenas um tipo de testes, que é o teste da execução de uma sequência de *Pings* para um determinado domínio, que será daqui em diante denominado por teste *Ping*.

Após o *login*, é lançado um *Worker* chamado *ManagementSystemMaintenanceWorker* que vai periodicamente realizar o pedido de controlo, de forma a estar sempre a par com os planos de testes a executar. No resultado do controlo, quando se está na posse da lista de identificadores, em cada resultado de detalhe do plano é feito o *schedule* de um novo *Worker* denominado *AutonomousTestWorker* que, ao receber as informações do plano, o vai executar e, no fim, reportar os resultados obtidos para o sistema de gestão.

4.4.2 Execução dos planos de testes

Na implementação destes testes foi estabelecido que estes teriam de ser executados em qualquer momento. Para implementar esse requisito foi feita uma implementação semelhante à da execução de testes passivos. Foi criada uma abstração *IWork* e uma concretização da mesma para a execução de testes *Ping*.

Os planos de testes, por terem a possibilidade de conter vários tipos de testes no mesmo plano, originam uma dificuldade de sequenciar a sua execução. Esta necessidade de sequenciação tem como objetivo minimizar as interferências indesejadas na rede durante a execução de cada teste. Então, visto que estes testes podem ser de natureza síncrona ou assíncrona, surge o desafio de apenas executar um teste quando o anterior estiver concluído. Para isso é feito um algoritmo de natureza recursiva que faz uso de uma lista e implementa uma função de *callback* que executa após a execução do teste. Este *callback* retira o elemento à cabeça da lista e chama de novo a função que invocou inicialmente o teste com a lista atualizada. Por cada teste executado a informação do resultado é armazenada na base de dados para disponibilizar ao utilizador os planos de testes e os respetivos testes executados. Se a execução de cada teste não gerar qualquer tipo de erro, os resultados obtidos na execução do teste são enviados ao sistema de gestão para que este consiga observar os avanços do plano de testes em tempo real. Os resultados são todos persistidos na base de dados para cobrir a eventualidade de haver uma falha na comunicação dos mesmos ao sistema de gestão. Nesse caso é disponibilizado um botão para que o utilizador, quando voltar à aplicação, consiga enviar para o sistema os resultados que não foram comunicados previamente.

5. Avaliação Experimental

Em traços gerais, a estrutura e apresentação, à exceção do esquema de cores, foi baseada e inspirada numa aplicação já existente de objetivo semelhante, denominada *NetMonitor* [19]. Com esta base é possível averiguar a veracidade das recolhas efetuadas por parte da aplicação *QoS*. De outra forma seria complicado verificar se os parâmetros recolhidos apresentavam resultados com algum nível de verdade. Não é fácil fazer a verificação com toda a certeza, visto que no momento de trocar uma aplicação para plano de fundo e trazer a outra para primeiro plano algo pode mudar. Ainda assim com dois *screenshots* (figura 14), (figura 15) é possível compreender a coerência dos dados recolhidos.

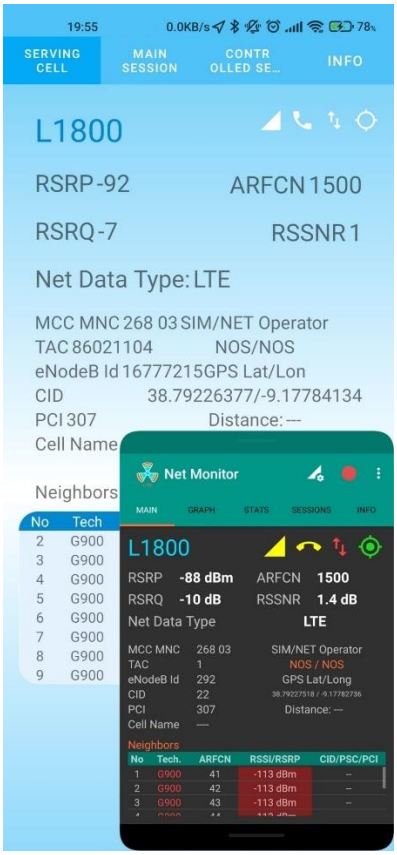

Figura 14 - Visualização da comparação de dados entre as duas aplicações.

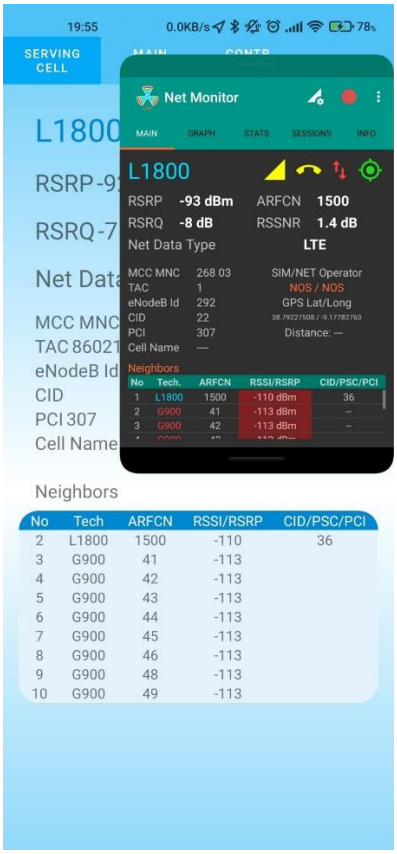

Figura 15 - Continuação da comparação.

É de notar que no curto espaço de tempo em foram tirados os *screenshots* que desapareceu a primeira linha da tabela, o que prova a dificuldade de fazer a avaliação com toda a certeza de correção.

6. Conclusão e trabalho futuro

6.1 Conclusão

O *smartphone* é cada vez mais um marco de extrema importância na vida da sociedade atual, pelo que desenvolver uma aplicação compatível com esse *gadget* se torna numa experiência inabalável e de grande relevância para os projetistas. De notar que a aplicação não está marcada ainda como terminada, nem no fim da sua linha evolutiva. Ainda há espaço para crescimento, e há certos pontos em que é certamente possível resolver alguns dos *bugs* que ou sejam produto da implementação ou produto das limitações do *Android* para a decisão de modelagem tomada. Todavia, foi possível que o grupo implementasse uma aplicação com potencial de ter bastante utilidade enquanto ferramenta de análise de rede móvel, tornando no geral este projeto uma experiência positiva.

6.2 Trabalho futuro

Face ao estado atual do projeto, é possível inferir que a aplicação ainda se encontra numa posição de possível evolução. Apesar da recolha de parâmetros estar otimizada a ser feita em paralelo com o resto das funcionalidades, e em *background*, existe espaço para aumentar o leque de abrangência de parâmetros recolhidos. É também perfeitamente possível que no futuro existam mais testes que a aplicação possa realizar, além das sequências de *ping*, visto que a aplicação se encontra preparada para fazer recolhas de novos tipos de dados, e realização de novos tipos de testes, precisando apenas da implementação da lógica desses mesmos testes, e da integração no resto da interface. Desse modo, há várias possibilidades de se poder contribuir para o melhor complemento do projeto *IQ-NPE*.

7. Referências

[1] Autoditex, "CAN BUS," [Online]. Available: https://autoditex.com/page/can-bus--controller-area-network-34-1.html. [Acedido em 5 4 2021].

[2] ETSI, "ETSI - Multi Access Edge Computing (MEC)," ETSI, [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing. [Acedido em 5 4 2021].

[3] V. Trucks, "What are Cooperative Intelligent Transport Systems?," Volvo Trucks, [Online]. Available: https://knowledgehub.volvotrucks.com/technology-and-innovation/what-are-cooperative-intelligent-transport-systems. [Acedido em 5 4 2021].

[4] A. Developers, "Volley Overview," Android Developers, [Online]. Available: https://developer.android.com/training/volley. [Acedido em 13 4 2021].

[5] PhilJay, "PhilJay/MPAndroidChart," [Online]. Available: https://github.com/PhilJay/MPAndroidChart. [Acedido em 15 4 2021].

[6] A. Developers, "Room," Android Developers, [Online]. Available: https://developer.android.com/reference/androidx/room/package-summary. [Acedido em 17 4 2021].

[7] M. UI, "Develop - Android - Material Design," Material UI, [Online]. Available: https://material.io/develop/android. [Acedido em 19 4 2021].

[8] greenrobot, "EventBus - Events for Android," greenrobot, [Online]. Available: https://greenrobot.org/eventbus/. [Acedido em 10 5 2021].

[9] A. Developers, "Worker," Android Developers, [Online]. Available: https://developer.android.com/reference/androidx/work/Worker. [Acedido em 15 4 2021].

[10] A. Developers, "WorkManager," Android Developers, [Online]. Available: https://developer.android.com/reference/androidx/work/WorkManager. [Acedido em 15 4 2021].

[11] A. Developers, "Managing Work," Android Developers, [Online]. Available: https://developer.android.com/topic/libraries/architecture/workmanager/how-to/managing-work. [Acedido em 15 4 2021].

[12] A. Developers, "JobService," Android Developers, [Online]. Available: https://developer.android.com/reference/android/app/job/JobService. [Acedido em 2 5 2021].

[13] R. Guru, "Strategy," Refactoring Guru, [Online]. Available: https://refactoring.guru/design-patterns/strategy. [Acedido em 20 5 2021].

[14] A. Developers, “LiveData,” Android Deveolpers, [Online]. Available: https://developer.android.com/reference/android/arch/lifecycle/LiveData. [Acedido em 15 4 2021].

[15] A. Developers, “LifecycleOwner,” Android Developers, [Online]. Available: https://developer.android.com/reference/android/arch/lifecycle/LifecycleOwner. [Acedido em 15 4 2021].

[16] A. Developers, “Observer,” Android Developers, [Online]. Available: https://developer.android.com/reference/androidx/lifecycle/Observer. [Acedido em 15 4 2021].

[17] A. Developers, “ViewModel,” Android Developers, [Online]. Available: https://developer.android.com/reference/android/arch/lifecycle/ViewModel. [Acedido em 17 4 2021].

[18] A. Developers, “ViewModelProvider.Factory,” Android Developers, [Online]. Available: https://developer.android.com/reference/android/arch/lifecycle/ViewModelProvider.Factory. [Acedido em 21 4 2021].

[19] V. V, “NetMonitor,” Vitaly V, [Online]. Available: https://play.google.com/store/apps/details?id=ru.v_a_v.netmonitor. [Acedido em 4 4 2021].

[20] António Serrador, Carlos Mendes, Nuno Datia, Nuno Cota, Nuno Cruz, Ana R. Beire, “A Performance Measurement Platform for C-ITS over 5G“, EUCNC, junho 2021, Porto Portugal.