

Barrel Shooter: um jogo “peer-to-peer” para “mobile edge clouds”

Rui Pedro Silva

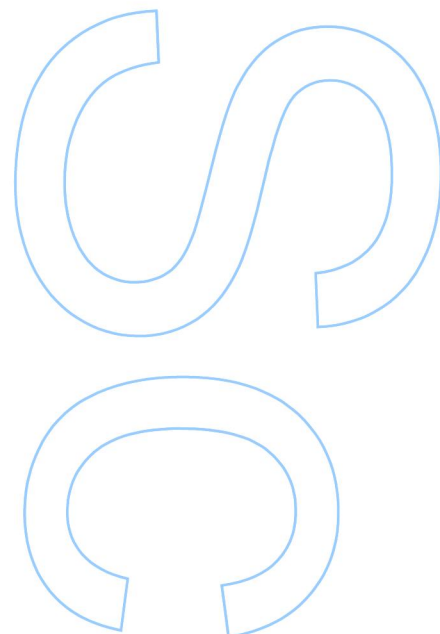
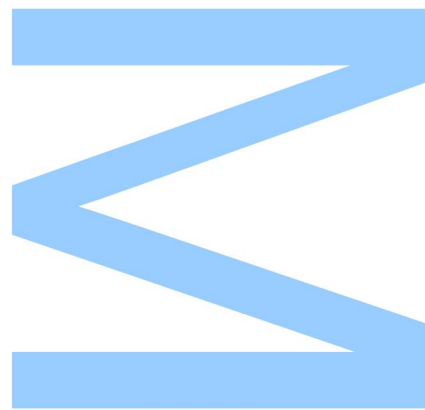
Mestrado Integrado em Engenharia de Redes e
Sistemas Informáticos
Departamento de Ciência de Computadores
2018

Orientador

Eduardo Resende Brandão Marques, Professor Auxiliar, Faculdade
de Ciências da Universidade do Porto

Coorientador

Fernando Manuel Augusto da Silva, Professor Catedrático, Faculdade
de Ciências da Universidade do Porto

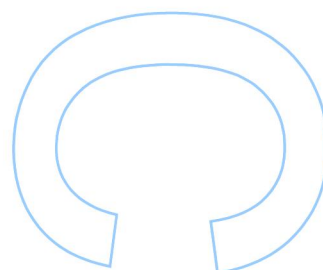
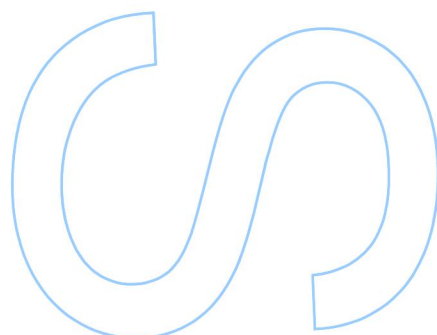
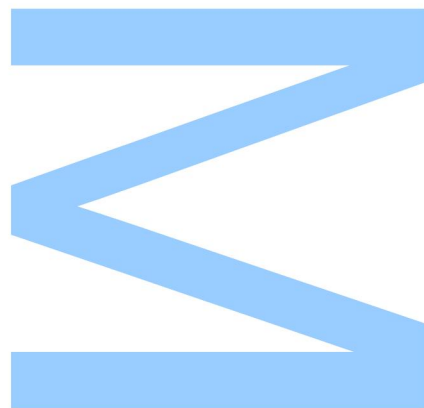




Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, ____/____/____



Agradecimentos

Agradeço ao professor Eduardo Marques e ao professor Fernando Silva pelo seu acompanhamento e dedicação no decorrer desta dissertação. Quero também agradecer ao João Rodrigues e ao Tadeu Freitas que me ajudaram quando tive dificuldades com o projecto.

Agradeço ao meu pai, à minha mãe e aos meus amigos que me apoiaram sempre no meu ciclo de estudos independentemente das minhas escolhas académicas e profissionais.

Por fim gostaria de agradecer à Direção-Geral do Ensino Superior que me garantiu bolsa e condições de estudo dignas durante o meu mestrado integrado.

Obrigado

Resumo

Aplicações para dispositivos móveis usam frequentemente serviços centralizados na “cloud”, não tirando partido das capacidades de processamento e de comunicação ponto-a-ponto de dispositivos. Estas capacidades podem habilitar aplicações sensíveis à proximidade entre utilizadores, que não requeiram obrigatoriamente suporte infra-estrutural, a efectuar computação e comunicação para a extremidade (“edge”) da rede. Fazendo uso de “mobile edge clouds”, os serviços podem correr apenas em dispositivos móveis conectados por tecnologias ponto-a-ponto como WiFi-Direct ou Bluetooth, ou ainda recorrendo a “cloudlets”, servidores de proximidade com capacidades modestas.

Nesta dissertação consideramos como caso de estudo para “mobile edge clouds” um jogo multi-participante para dispositivos móveis como smartphones ou tablets. O jogo, chamado “Barrel Shooter”, tem um espírito “arcade shooter” e funciona em dispositivos Android. Pode ser jogado em “mobile edge clouds” definidas apenas por grupos WiFi-Direct ou que recorram a “cloudlets”, bem como num ambiente tradicional com ligação à Internet e computação na “cloud”. É feita uma avaliação do jogo tendo em conta os vários ambientes de rede e diversas parametrizações, concluindo que o jogo é um bom exemplo das potencialidades oferecidas por “mobile edge clouds”.

Termos — jogos em rede, mobile edge-clouds, WiFi-Direct, cloudlets, peer-to-peer networks, Android

Acrónimos

SDK Software Development Kit

LAN Local Area Network

PC Personal Computer

API Application Programming Interface

UGR User-Generated Replays

JVM Java Virtual Machine

IDE Integrated Development Environment

IP Internet Protocol

UI User Interface

RAM Random Access Memory

NSD Network Service Discovery

Conteúdo

Agradecimentos	1
Resumo	2
1 Introdução	13
1.1 Motivação	14
1.2 Objectivo do trabalho	14
1.3 Contribuições	15
1.4 Estrutura da dissertação	15
2 Trabalho Relacionado	17
2.1 Projecto Hyrax	17
2.1.1 Contexto geral	17
2.1.2 Aplicações e serviços	17
2.1.3 O middleware Hyrax	19
2.2 Jogos multi-participante em “edge clouds”	20
2.2.1 GameOn	20
2.2.2 CrowdMoG	21
2.2.3 Microplay	21
2.2.4 iPokemon with ENORM	22
2.2.5 Jogos comerciais e “open-source”	23

3	O Jogo “Barrel Shooter”	25
3.1	Requisitos	25
3.2	Funcionalidade do jogo	26
3.2.1	Dinâmica geral	26
3.2.2	Regras	27
3.2.3	Ecrãs de setup e de início de jogo	28
3.2.4	Ecrã de fim de jogo	29
3.3	Arquitetura do modelo cliente-servidor	29
3.4	Ambientes de rede	30
4	Implementação	33
4.1	Organização geral	33
4.1.1	Parametrização de jogo	34
4.2	Protocolo de mensagens	35
4.3	Comunicação em rede	37
4.3.1	Suporte de WiFi-Direct	37
4.4	Interface gráfico	38
4.4.1	Classe ExplosionAnimation	40
5	Avaliação	41
5.1	Configurações	41
5.1.1	Ambientes de rede	41
5.1.2	Configurações base de hardware e software	42
5.1.3	Parâmetros de jogo	42
5.1.4	Condução dos testes	43
5.2	Resultados	43
5.2.1	Carga de rede	43
5.2.2	Latência	44

Lista de Figuras

1.1	Top das aplicações mais rentáveis da Google Play Store (11 de Junho de 2018)	13
2.1	Teste de campo da UGR	18
2.2	A aplicação Ramble e esquema do seu uso em teste de campo	18
2.3	Arquitetura do middleware Hyrax	19
2.4	Camada de ligação (“Link Layer”) do middleware Hyrax	20
2.5	Sistema GameOn	20
2.6	Uso do sistema Microplay	21
2.7	Uso e arquitectura do sistema ENORM	22
3.1	Elementos do jogo	26
3.2	Dinâmica de jogo.	27
3.3	Ecrãs de setup e de início de jogo	28
3.4	Ecrã de fim de jogo	29
3.5	Esquema de troca de mensagens cliente-servidor	30
3.6	Ecrã inicial de conexões WiFi-Direct	31
3.7	Ambientes de rede a comparar	32
4.1	Diagrama de componentes da aplicação	33
4.2	Parâmetros de jogo	35
4.3	Mensagens do Protocol Buffers	36
4.4	Código para descoberta de dispositivos próximos	37

4.5	Código para conexão de dispositivos próximos através de WiFi-Direct utilizando o middleware Hyrax	38
4.6	Carregamento das imagens do jogo para a Random Access Memory (RAM)	39
4.7	Desenho do bitmap referente a uma bala	39
4.8	Classe de animação de explosão	40
5.1	Tamanho médio de mensagens GameInfo	44
5.2	Carga de rede.	44
5.3	Latência de todos os modelos de rede nas várias configurações.	45

Lista de Tabelas

5.1 Resultados detalhados. 46

Capítulo 1

Introdução

Os smartphones são uma plataforma ideal para jogos multi-jogador que possam tirar proveito dos jogadores se encontrarem no mesmo espaço físico, graças às suas capacidades computacionais e de rede, como a sua portabilidade e popularidade. Existe bastante potencial neste tipo de jogos, trazendo oportunidades e dificuldades que abrem uma nova dimensão na comunidade científica de modo a transformar uma nova era de jogos em telemóveis e tablets. Nos últimos anos a percentagem de população que possui smartphones aumentou drasticamente, segundo o Consumer Barometer da Google em 2012 18% dos Portugueses possuíam smartphone, em 2017 regista um valor de 67%, tendo aumentado 8% em 2017 [10]. Um dos sectores de maior crescimento nos smartphones são os jogos, que estão sempre no topo dos mercados de aplicações, contando com bastantes jogos multi-jogador nas tabelas das aplicações mais compradas e mais rentáveis, como Minecraft (cerca de 2 milhões de jogadores) pela Mojang e Clash Royale (cerca de 20 milhões de jogadores) pela Supercell que lideram ambas as tabelas [11].

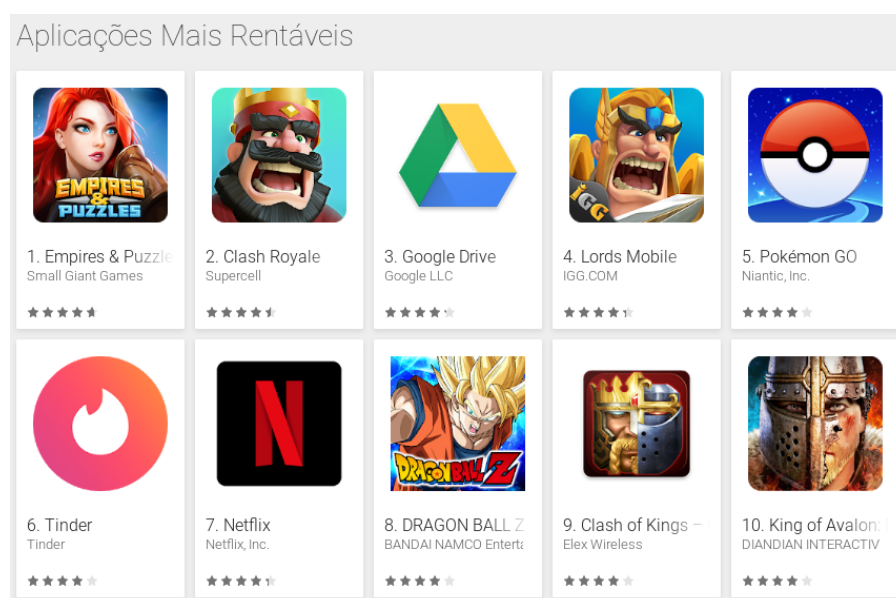


Figura 1.1: Top das aplicações mais rentáveis da Google Play Store (11 de Junho de 2018)

1.1 Motivação

Os smartphones oferecem a possibilidade dos utilizadores interagirem localmente através de jogos multi-jogador, como por exemplo as conhecidas “Local Area Network (LAN) parties”. Estas actividades trazem grandes níveis de satisfação e entretenimento aos participantes devido a estarem todos no mesmo local a jogarem ao mesmo tempo o mesmo jogo. Com as plataformas tradicionais como o uso de Personal Computer (PC), é exigido algum trabalho de preparação para criar condições necessárias para a realização desses jogos. Em contraste os dispositivos móveis por norma estão sempre com o seu utilizador, funcionam com bateria, e não exigem qualquer trabalho de preparação prévio por parte do utilizador para jogos locais espontâneos.

Um exemplo destas actividades é o Pokémon GO Fest, realizado a 22 de Julho de 2017 em Chicago, Illinois, que prometia uma experiência incrível aos jogadores, que se iriam encontrar no mesmo local e desfrutar do evento através dos seus dispositivos e jogar em conjunto. Mas revelou-se um fiasco pois os servidores não conseguiram lidar com tantas pessoas no mesmo sítio. Ninguém conseguia sequer fazer login naquele local e usufruir do jogo [6].

Uma das soluções para este fiasco poderia ser o uso de “mobile edge clouds” (MECs). A ubiquidade de dispositivos pessoais móveis levou à emergência deste paradigma. MECs são redes formadas por dispositivos móveis próximos fisicamente com infraestrutura de suporte inexistente ou limitada, por exemplo quando complementada por servidores na periferia (“edge”) da rede na forma de “cloudlets”, ou aproveitando de forma intermitente e parcial ligações a uma “cloud” tradicional. As MECs podem concentrar computação, comunicação e armazenamento de dados localmente, reduzindo a dependência/fardo de/sobre uma infraestrutura “cloud” padrão, e habilitar aplicações “crowd-sourced” sensíveis à proximidade entre dispositivos.

Uma das principais dificuldades de um jogo multi-jogador num dispositivo móvel, caso seja em tempo-real e não baseado em turnos, é a redução da latência de cada utilizador, ou seja, o tempo que uma acção realizada por um determinado jogador demora a aparecer nos ecrãs dos restantes jogadores. Caso jogos multi-participante tirem partido de “edge clouds”, existe o potencial de reduzir a latência de comunicação entre os dispositivos e assim melhorar a experiência global de jogo. Para tal é possível tirar partido de comunicação ponto-a-ponto via WiFi-Direct ou Bluetooth, ou, em alternativa, através de uma “cloudlet” fisicamente próxima.

1.2 Objectivo do trabalho

O objectivo deste projecto de dissertação foi o desenvolvimento de um jogo multi-participante que funcionasse apenas com comunicação ponto-a-ponto entre dispositivos. O jogo serviu de avaliação a um middleware para “edge clouds” desenvolvido no âmbito do projecto Hyrax. O middleware funciona em dispositivos Android (“tablets” e “smartphones”) e permite de forma automatizada a formação de uma “edge cloud” e posterior comunicação entre dispositivos.

Usufruindo das possibilidades que o middleware Hyrax oferece e do conhecimento obtido através da análise do

estado-da arte relativo a este tema, procurou-se construir um jogo multi-jogador para obter o máximo proveito do middleware Hyrax e que teste o seu desempenho sujeito a uma carga de rede tão intensa quanto possível, bem como a sua escalabilidade face a um número crescente de participantes num jogo.

1.3 Contribuições

Com o trabalho descrito nesta dissertação foram feitas as seguintes contribuições:

- A implementação de um jogo Android multi-jogador, chamado Barrel Shooter. O jogo tem a funcionalidade de um “arcade shooter” e associada uma arquitectura cliente-servidor, fazendo uso de TCP/IP para comunicações sob redes WiFi ou ainda WiFi-Direct recorrendo ao middleware Hyrax. O servidor de jogo pode executar no dispositivo móvel de um dos jogadores, ou alternativamente num dispositivo externo na forma de “cloudlet” ou máquina na “cloud”.
- A avaliação do jogo em três ambientes de rede distintos: (1) recorrendo apenas a grupos WiFi-Direct, (2) recorrendo a uma “cloudlet” para ligação WiFi e possível alojamento do servidor de jogo, e (3) usando a rede Eduroam para configuração WiFi e possível alojamento do servidor de jogo na “cloud”.

1.4 Estrutura da dissertação

A continuação desta dissertação tem a seguinte estrutura. O Capítulo 2 contém o estado da arte relativo ao uso de “edge cloud computing” no contexto de jogos multi-jogador. No Capítulo 3 é descrito e apresentado o jogo criado nesta dissertação denominado de Barrel Shooter. No Capítulo 4 são apresentados detalhes da implementação do jogo. O Capítulo 5 apresenta os resultados obtidos durante o processo de testes da presente dissertação. No Capítulo 6 é feita uma conclusão desta dissertação e apresentado um possível trabalho futuro.

Capítulo 2

Trabalho Relacionado

Neste capítulo apresenta-se o trabalho relacionado relevante ao tema de dissertação, incluindo uma descrição do projecto Hyrax (Secção 2.1) e uma resenha do estado da arte em uso de jogos “peer-to-peer” em “mobile edge clouds” (Secção 2.2).

2.1 Projecto Hyrax

2.1.1 Contexto geral

O projecto Hyrax (<http://hyrax.dcc.fc.up.pt>) considera vários casos de estudo de aplicações para MECs e o desenvolvimento de um “middleware” de suporte a estas. Neste contexto, são fulcrais aspectos como a adaptação dinâmica a “churn”, decorrente da variabilidade dos dispositivos disponíveis na rede e mobilidade dos mesmos, a avaliação de estratégias entre o uso de recursos locais ao nível de um dispositivo ou de uma MEC vs. cloudlets ou clouds disponíveis, ou ainda a validação via simulação de cenários de aplicação, de outra forma onerosos de replicar em escala usando apenas testes de campo. Os cenários de aplicação considerados pelo projecto são diversos, tendo por factores comuns o uso das capacidades dos dispositivos móveis para formar MECs através do uso de comunicações “peer-to-peer”, gerar conteúdo que poderá ser alvo de partilha e/ou processamento distribuído.

2.1.2 Aplicações e serviços

Fazemos a seguir um sumário de algumas das aplicações e serviços para MECS desenvolvidos no contexto do projecto Hyrax.

Na aplicação User-Generated Replays (UGR) considera-se o cenário de partilha de vídeos capturados entre utilizadores durante o evento desportivo [20]. Na Figura 2.1, adaptada de [20], é ilustrado um teste de campo

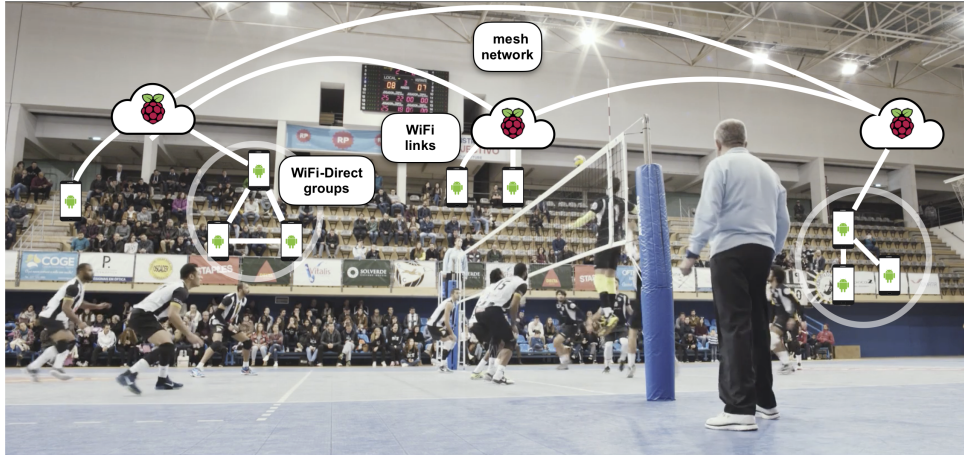


Figura 2.1: Teste de campo da UGR

da UGR que decorreu durante um jogo do campeonato nacional de voleibol que decorreu na Nave Desportiva de Espinho. A UGR possibilitou a captura e disseminação de vídeos em tempo real durante o jogo fazendo uso de grupos WiFi-Direct para partilha directa de vídeos entre utilizadores e de três cloudlets de baixo custo Raspberry Pi que providenciavam um serviço de armazenamento de vídeos e conectividade WiFi a dispositivos. As cloudlets sincronizavam também vídeos através de uma rede mesh possibilitando a disseminação de vídeos entre zonas distintas do recinto. Um teste de campo anterior [23] envolveu a injeção de vídeos seleccionados (ao invés de gerados pelos utilizadores) e sua disseminação por um grupo de utilizadores a assistir numa sala a um jogo da Champions League, empregando também WiFi e WiFi-Direct. A UGR motivou também a análise via simulação de cenários de disseminação de dados envolvendo WiFi, WiFi-Direct e cloudlets usando no ambiente Mininet-WiFi [3, 4].

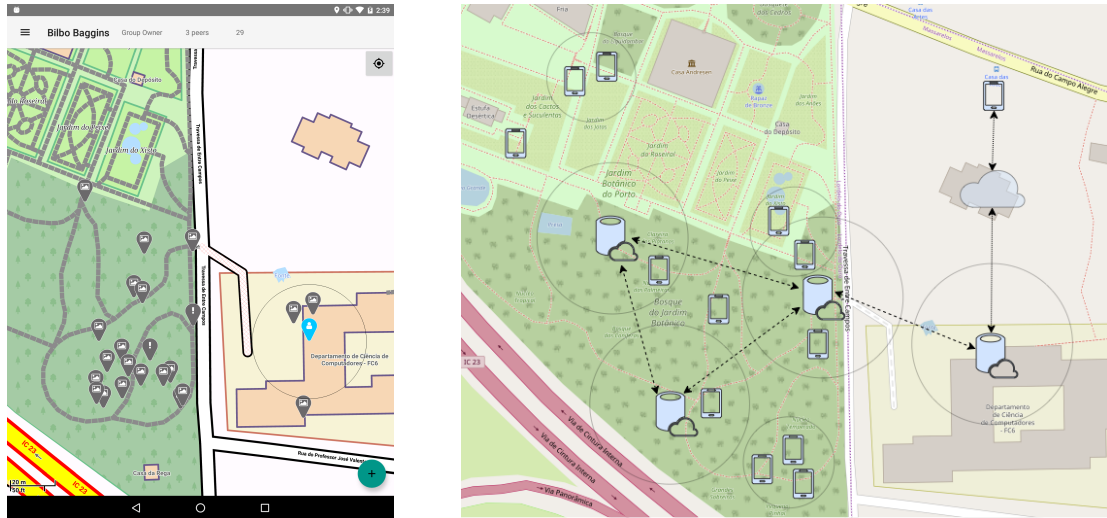


Figura 2.2: A aplicação Ramble e esquema do seu uso em teste de campo

A aplicação Ramble [8, 9] considera também a troca de conteúdos entre utilizadores, mas tendo em conta cenário de comunicações intermitentes e oportunistas, por exemplo atendendo a situações de desastre natural ou colecção de informação por grupos de utilizadores em áreas sem acesso a rede infra-estrutural. A Figura 2.2, com imagens retiradas de [9], ilustra o uso da aplicação em um teste de campo realizado no Jardim Botânico

do Porto, onde um grupo de voluntários tinha por objectivo tirar fotos a um conjunto de pontos de interesse pré-determinados (ex. espécies de árvores específicas) seguindo trajectos diferentes. O conteúdo gerado, na forma de fotografias, vídeos ou mensagens, é geo-referenciado e disseminado pela aplicação à medida que se conseguem estabelecer ligações WiFi-Direct com outros dispositivos ou WiFi a cloudlets de proximidade.

Um outro conjunto de desenvolvimentos no projecto Hyrax relaciona-se com computação ou armazenamento de dados de forma distribuída em MECs. O sistema Panoptic [7] habilita a pesquisa distribuída de fotos de pessoas, motivado por cenários do tipo “Amber alert” (pessoas desaparecidas), procurando balancear a carga entre os vários nós da rede, simultaneamente em combinação com mecanismos para protecção de privacidade. Um outro exemplo é o sistema P3-Mobile [21], que permite a execução paralela de tarefas computacionais em MECs seguindo várias estratégias de organização da rede. No que toca a sistemas de armazenamento, são exemplos o Thyme [5] e Ephesus [22] que permitem o armazenamento de dados segundo modelos distintos: publish-subscribe com dimensão temporal no caso do Thyme, e pares chave-valor no Ephesus.

2.1.3 O middleware Hyrax

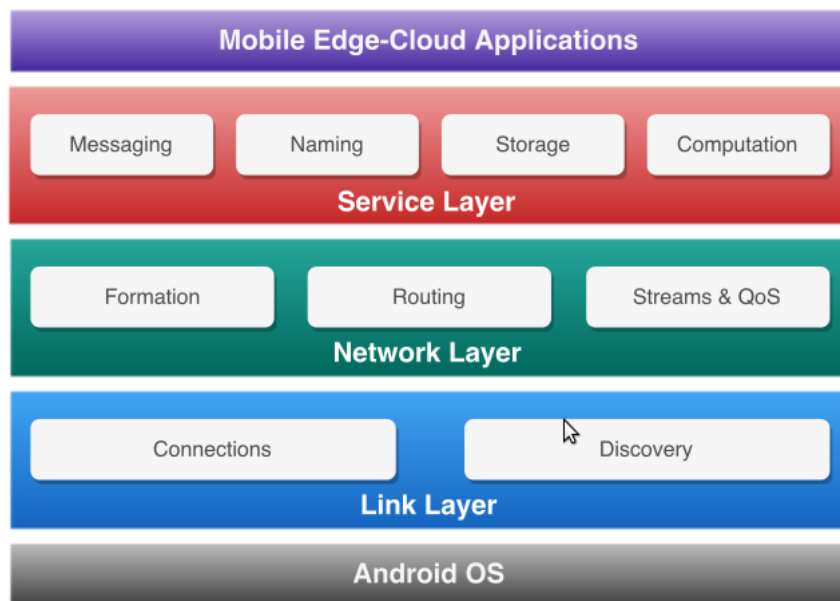


Figura 2.3: Arquitetura do middleware Hyrax

Para suporte a aplicações e serviços do projecto Hyrax, foi desenvolvido um middleware de comunicações para MECs [18, 19]. A arquitectura do middleware é ilustrada no diagrama da Figura 2.3, retirado de [19]. Para serviços e aplicações, o middleware define uma API que abstrai o uso de tecnologias de comunicação heterogéneas (WiFi, WiFi-Direct Bluetooth) e lida com a complexidade de formar uma rede de forma adaptativa face a factores como comunicação intermitente, entrada e saída de dispositivos (“churn”), ou falta de suporte infra-estrutural. O middleware toma forma num conjunto de bibliotecas para dispositivos Android, sem necessidade de adaptações à imagem ou configuração do sistema Android base, ou acesso “root” aos dispositivos.

O middleware está estruturada em duas camadas: a camada de rede (“Network Layer”) e a camada de ligação (“Link Layer”). A camada de rede suporta a formação automática de uma rede, e os aspectos associados de anúncio/descoberta de dispositivos e troca de mensagens (de uma aplicação ou serviço) entre estes. A camada de ligação, ilustrada no diagrama da Figura 2.4 retirado de [19], permite de forma normalizada lidar com as várias tecnologias de comunicação, por forma a habilitar ligações específicas de cada tipo. Nesta dissertação foi empregue a camada de ligação para definir programaticamente o estabelecimento de grupos WiFi-Direct (os detalhes técnicos de uso são descritos no Capítulo 4).

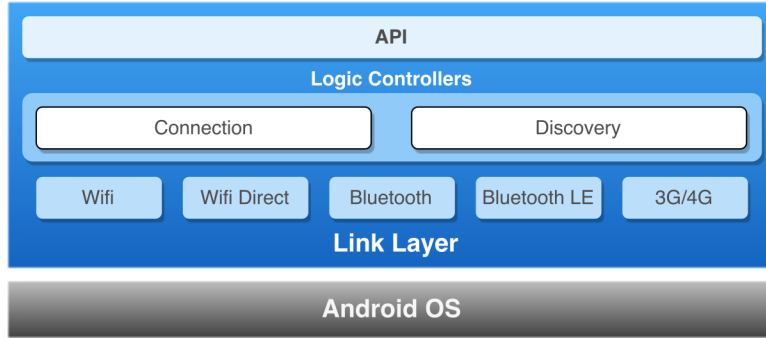


Figura 2.4: Camada de ligação (“Link Layer”) do middleware Hyrax

2.2 Jogos multi-participante em “edge clouds”

2.2.1 GameOn

O GameOn é um sistema para desenvolvimento de jogos “peer-to-peer” que faz uso de comunicações WiFi-Direct [25]. O sistema foi avaliado em situações reais de uso em transportes públicos para três jogos distintos (OpenArena, 2048, e Racer) com arquitectura cliente-servidor, cujo código foi adaptado para usar o GameOn.

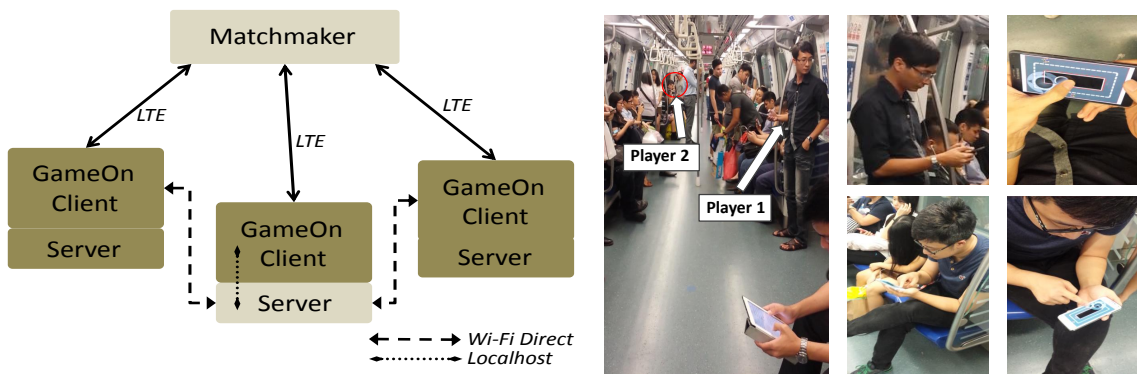


Figura 2.5: Sistema GameOn

A Figura 2.5, com imagens retirada de [25], ilustra o modelo de rede empregue e o uso do sistema em transportes públicos. Em contraste com a implementação usual do modelo cliente-servidor, o GameOn permite que dispositivos desempenhem o papel de servidores de jogo (além de serem clientes), e usa WiFi-Direct para que

as comunicação em rede durante o jogo sem necessidade de suporte infra-estrutural. É no entanto necessário o acesso a um “match-maker” externo para inicializar os jogos. De forma similar ao GameOn, o jogo Barrel Shooter possibilita o estabelecimento de um jogo multi-participante via WiFi-Direct e que um dos dispositivos seja o servidor de jogo.

Um aspecto bastante significativo do trabalho no GameOn é a avaliação conduzida para os vários jogos em diferentes meios de transportes públicos (comboio, autocarro) e horas do dia, observando-se que a latência observada usando comunicações infra-estruturais LTE foi cerca de 10 vezes maior face ao uso do GameOn com Wifi-Direct.

2.2.2 CrowdMoG

De forma análoga ao GameOn, CrowdMog [17] é uma proposta do desenho de uma plataforma de jogos multi-participantes, para jogos multi-participante em transportes públicos. O trabalho propõe apenas a arquitectura do sistema, tendo em conta um estudo empírico da fiabilidade de rede 3G e 4G nos transportes públicos face a aspectos como padrões de mobilidade e intermitência nas condições de rede, e discute o uso de 3G/4G face a outras tecnologias como WiFi-Direct ou Bluetooth.

2.2.3 Microplay

O MicroPlay é um sistema de rede direccionado para jogos multi-jogador cliente/servidor em dispositivos móveis [14]. Tal como possibilita o Barrel Shooter, um dos dispositivos é utilizado como servidor e os outros como clientes. O Microplay funciona apenas sobre WiFi, mas emprega WiFi “over-hearing”, isto é, a capacidade de cada dispositivo escutar os pacotes de dados via WiFi quando configurado no chamado modo promíscuo, para diminuir a latência de jogo.

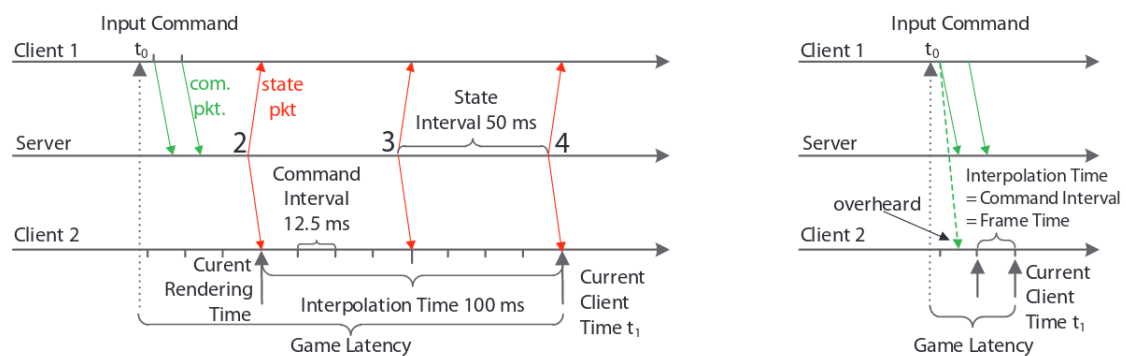


Figura 2.6: Uso do sistema Microplay

A aproximação é ilustrada na Figura 2.6, com imagens retiradas de [14]. A ideia é que todos os dispositivos enviam comandos de jogo ao servidor de jogo, mas podem ao mesmo tempo ouvir os comandos de outros jogadores para prever a evolução de jogo (lado direito da figura) e reduzir a latência. Num teste com um jogo simples de corridas multi-jogador, observou-se que o uso desta técnica permite uma redução bastante significativa

da latência face a cenários que usam comunicações simples client-servidor ou mesmo comunicações com recurso a “broadcast” entre os jogadores. Uma desvantagem da aproximação é a necessidade de ser necessário ter acesso “root” para habilitar comunicações WiFi em modo promíscuo.

2.2.4 iPokemon with ENORM

O ENORM [24] é um sistema para controlo de “edge nodes” dispersos geograficamente, integrando mecanismos para o provisionamento e auto-escalamento de recursos, em linha com o paradigma de “cloudlet-based computing”. O sistema foi avaliado usando o o jogo “open-source” iPokeMon [12], com espírito similar ao popular jogo Pokémon Go, um jogo multi-jogador mundialmente conhecido. Este tipo de jogos representa um excelente desafio ao uso de “edge computing” pois para a utilização massiva da aplicação é necessário não só a replicação física de servidores mas também uma camada extra de processamento que sirva de intermediário entre “edge nodes” e o servidor (na introdução deste dissertação, referimos aliás o exemplo de uma falha desastrosa da infra-estrutura num encontro de jogadores PokeMon Go, por falta de uma infra-estrutura adequada [6]).

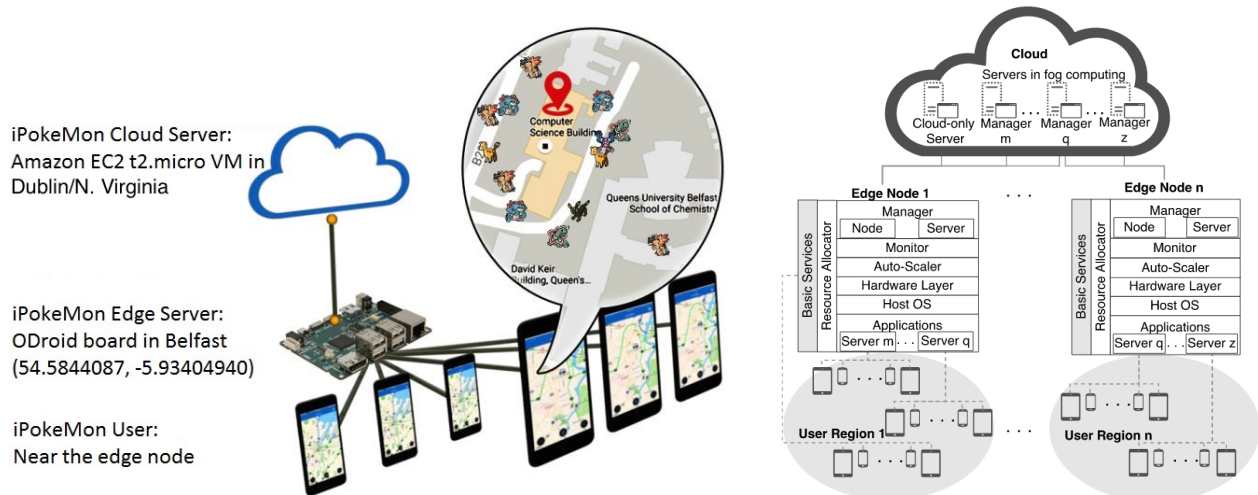


Figura 2.7: Uso e arquitectura do sistema ENORM

A arquitectura do ENORM é ilustrada no diagrama da Figura 2.7, retirado de [24]. Como ilustrado, um “edge node” é responsável por servir utilizadores em determinada localização geográfica, podendo esses utilizadores migrarem entre regiões, e mediar o acesso a um infra-estrutura centralizada na “cloud”. Os “edge nodes” tinham recursos bastante limitados, correndo sobre placas ODROID-XU [16], em intermediação a servidores cloud alojados na Amazon. Os resultados do estudo com o jogo iPokeMon foram bastante positivos, observando-se uma redução da latência da aplicação 20%-80% e do volume de comunicação entre os dispositivos e o servidor central em 95%.

2.2.5 Jogos comerciais e “open-source”

Os jogos comerciais multi-participante recorrem tipicamente a ligações Internet e servidores na “cloud”. Existem no entanto uma quantidade considerável de jogos que podem ser jogados sem esse tipo de suporte infra-estrutural. Por exemplo, são listados em [1] mais de 800 jogos da Android Play Store que podem ser jogados “offline”, em maior parte dos casos recorrendo a uma LAN, mas em menor número usando também WiFi-Direct ou Bluetooth. Na lista encontramos por exemplo o popular jogo Minecraft [15] que pode ser jogado através de uma LAN, mas que poderia ser alvo de utilização WiFi-Direct empregando servidores de jogo alternativos “open-source” como o Cuberite [2]. O mesmo tipo de adaptação poderia ser considerado para jogos disponíveis “open-source” como o Cubes [13] (um jogo no espírito do Minecraft) ou o iPokeMon [12] (no espírito do PokeMon, usado como caso de estudo do sistema ENORM, referido anteriormente).

Capítulo 3

O Jogo “Barrel Shooter”

Neste capítulo descrevemos o jogo “Barrel Shooter” em termos dos seus requisitos (Secção 3.1), funcionalidade (3.2), arquitectura de operação em rede (3.3), e ambientes de rede para o qual o jogo foi concebido (3.4).

3.1 Requisitos

O desenvolvimento do jogo foi guiado pelos seguintes requisitos:

- Uso de TCP/IP para comunicações, de forma a poder funcionar em dispositivos móveis independentemente da rede de suporte, tais como ligação a uma LAN ou à Internet via WiFi ou a “edge clouds” via WiFi-Direct.
- Funcionar em dispositivos Android, por forma a fazer uso middleware Hyrax para criação de grupos WiFi-Direct.
- Poder funcionar apenas com dispositivos móveis usando WiFi ou WiFi-Direct, ou, alternativamente, fazendo uso de um servidor de jogo alojado numa LAN ou Internet.
- Ter uma carga de rede variável em função de parâmetros do jogo. Em particular, o jogo deverá permitir gerar tráfico relativamente elevado.

3.2 Funcionalidade do jogo



Figura 3.1: Elementos do jogo

3.2.1 Dinâmica geral

O jogo “Barrel Shooter” foi criado de raíz para o contexto desta dissertação. Com um espírito de jogos antigos “arcade shooter”, os vários jogadores são munidos de um canhão móvel que dispara balas, com o objectivo de acertar em alvos que tomam a forma de barris móveis.

Esta dinâmica é ilustrada na Figura 3.1, onde estão anotados de (1) a (6) os vários elementos de jogo:

1. O canhão do jogador, que se move na parte de baixo do ecrã. O movimento do canhão é automático em função da posição de mira do jogador (descrita abaixo).
2. Um alvo em movimento (barril). Os barris são móveis com direcção arbitrária e existem em número fixo de acordo com a parametrização do jogo.
3. Posição do jogador (mira), nome do jogador e sua pontuação. O canhão do jogador pára quando alinhado.
4. Explosão após colisão de uma bala disparada por um canhão com um alvo.
5. Bala do tipo normal.

6. Bala do tipo “round”, maior e mais rápida.

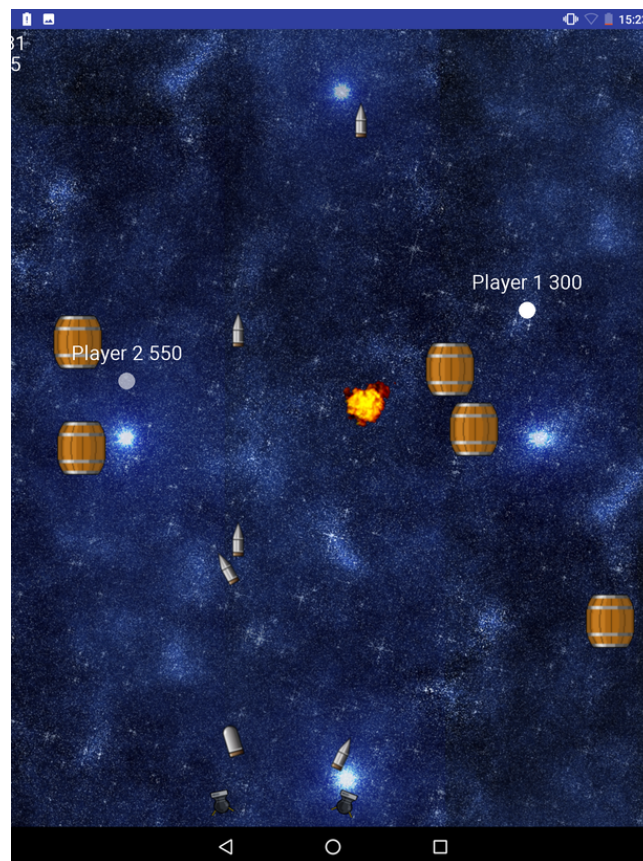


Figura 3.2: Dinâmica de jogo.

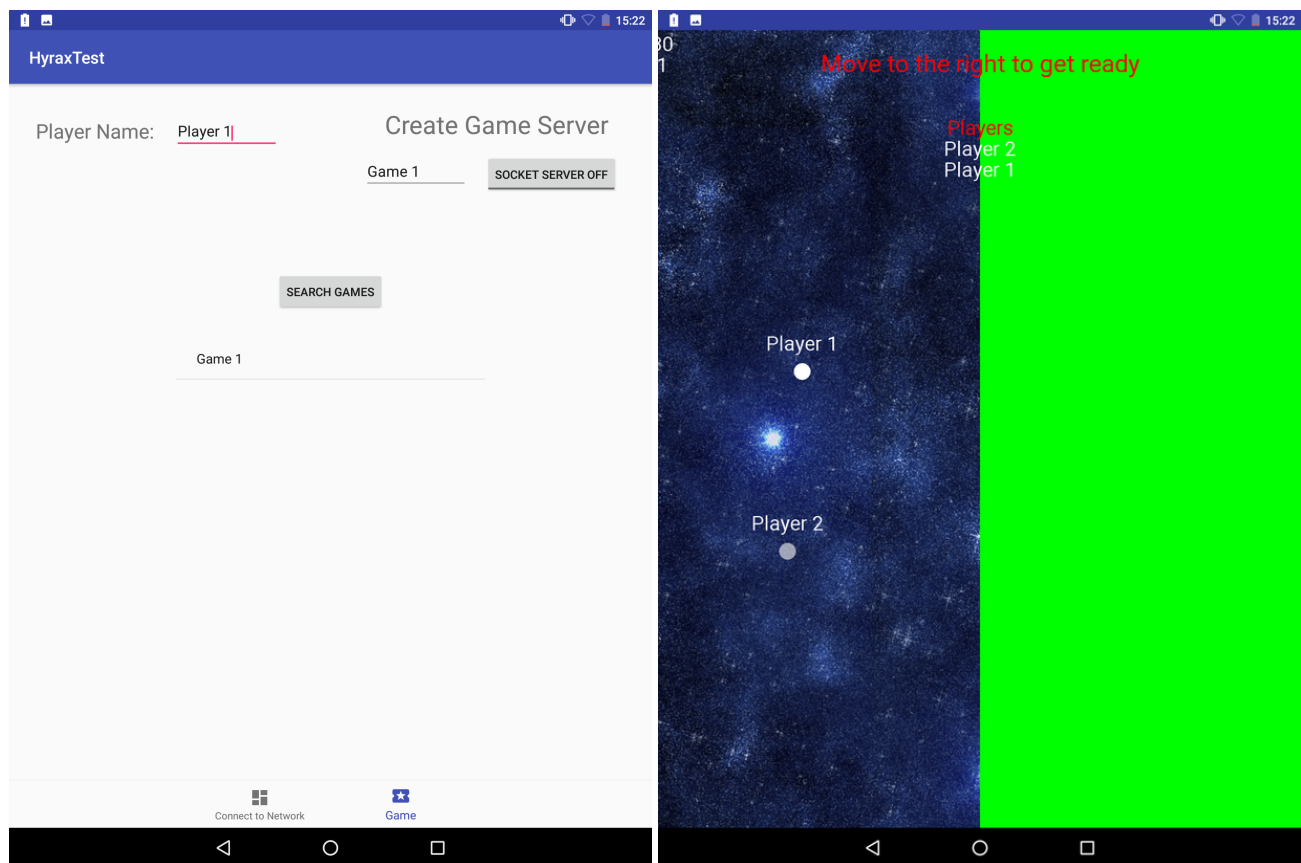
Na Figura 3.2 podemos observar o decorrer de um jogo com dois jogadores. Por cima da mira de cada jogador encontra-se o seu nome e a sua pontuação. Quando um alvo é eliminado uma animação de explosão dá início e de seguida aparece outro alvo numa localização arbitrária, de modo a que o número de alvos seja sempre igual, neste caso são 5. O outro aspecto ilustrado concerne o movimento de canhões e de balas. O canhão não é estático e vai acompanhando os movimentos da mira do jogador com deslize no eixo horizontal, parando quando alinhado verticalmente com a mira. Por sua vez, as balas são sempre disparadas a partir de um canhão na direção da mira do jogador associado.

3.2.2 Regras

As regras de jogo são bastante simples:

- Cada alvo eliminado dá 50 pontos ao jogador responsável pelo abate.
- A cada 500 pontos o jogador é premiado com uma bala diferente, maior e mais rápida. A frequência de disparo mantém-se no entanto constante.
- No fim da duração parametrizada para o jogo, o jogador com mais pontos é o vencedor.

3.2.3 Ecrãs de setup e de início de jogo



(a) Ecrã de setup e conexão ao servidor de jogo

(b) Ecrã inicial de jogo

Figura 3.3: Ecrãs de setup e de início de jogo

Na Figura 3.3a temos o ecrã de setup e conexão ao servidor de jogo, que tanto pode estar a correr num dos dispositivos, como noutra máquina com um Internet Protocol (IP) address definido previamente. Aqui dá-se a hipótese de o jogador escolher o seu nome durante o jogo, ser um game owner e correr um servidor de jogo no seu dispositivo com um determinado nome ou conectar-se a um dos servidores visíveis na lista antes ou após da procura (podem já estar alguns servidores na lista predefinidos, no caso do servidor de jogo que está a correr em `hyrax.dcc.fc.up.pt`).

Na Figura 3.3b encontra-se o ecrã inicial de jogo, em que já se pode observar a posição dos outros jogadores que se encontram em jogo no mesmo servidor. Para dar início ao jogo basta todos os jogadores se deslocarem para a direita, tocando em qualquer parte verde com o dedo. A posição de cada jogador adapta-se com o toque no ecrã e está a ser comunicada via socket.

3.2.4 Ecrã de fim de jogo

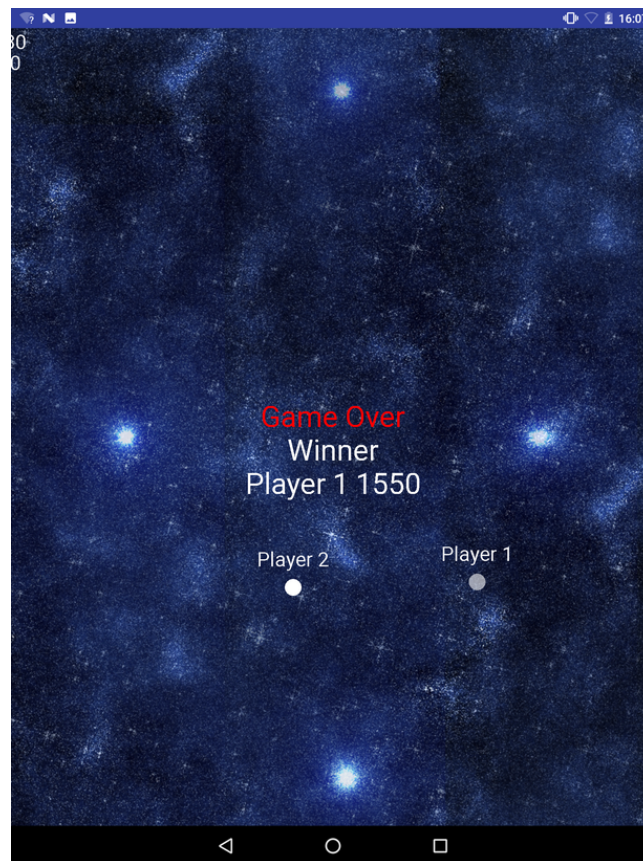


Figura 3.4: Ecrã de fim de jogo

Quando o jogo acaba o jogador com mais pontos é declarado vencedor (Figura 3.4) e todos os jogadores voltam ao ecrã inicial de jogo ilustrado na Figura 3.3b.

3.3 Arquitetura do modelo cliente-servidor

O servidor é responsável pela gestão de todos os aspectos do jogo e de manter os clientes actualizados. Este tem como requisito conseguir correr nos dispositivos cliente, de modo a um deles poder fazer de servidor, e correr numa “cloudlet” ou num servidor remoto. O servidor estará sempre à escuta de novas conexões por parte de clientes.

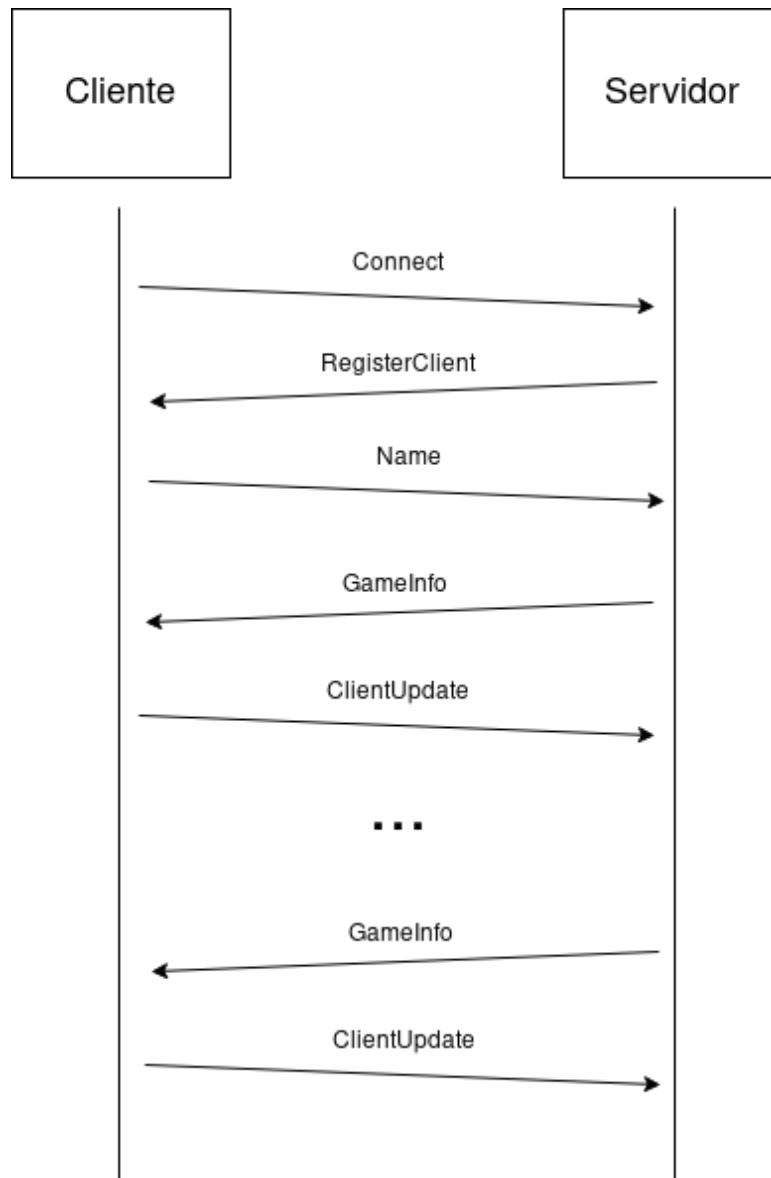
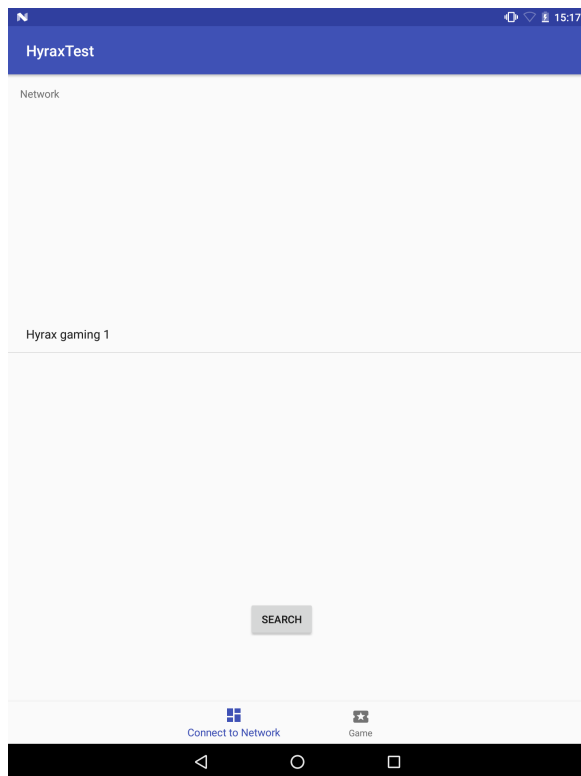


Figura 3.5: Esquema de troca de mensagens cliente-servidor

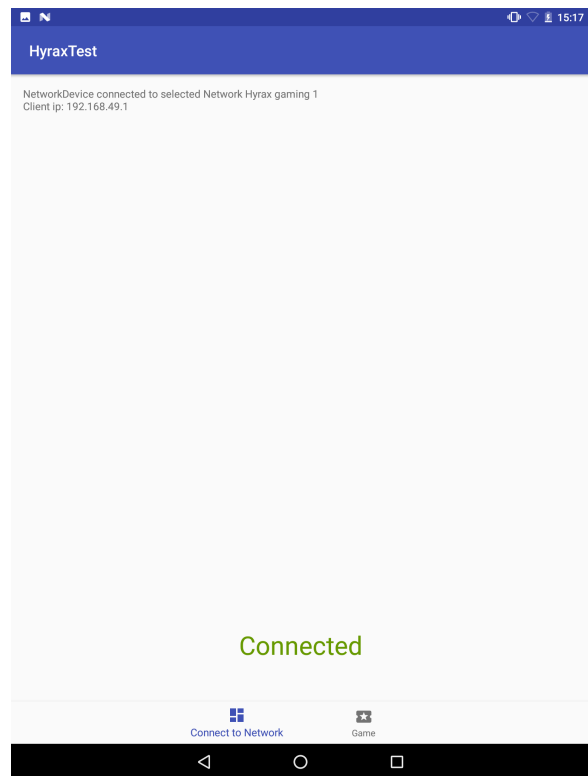
No esquema da Figura 3.5 podemos observar que existe uma comunicação inicial entre o cliente e o servidor que permite o registo do cliente no jogo e a configuração do seu nome. Após o início de jogo, que é feito depois de todos os jogadores estarem a postos, a troca de mensagens entre o cliente e o servidor é um ciclo, em que o servidor envia o estado do jogo ao cliente e o cliente envia ao servidor a sua nova posição de jogo.

3.4 Ambientes de rede

Neste jogo a rede pode ser constituída por WiFi-Direct ou WiFi, podendo ser um dos dispositivos o servidor de jogo ou este residir num host na rede, recorrendo ao uso de “cloud” ou “cloudlets”. Deste modo conseguimos testar todas as soluções apresentadas na introdução.



(a) Descoberta de dispositivos próximos através de WiFi-Direct

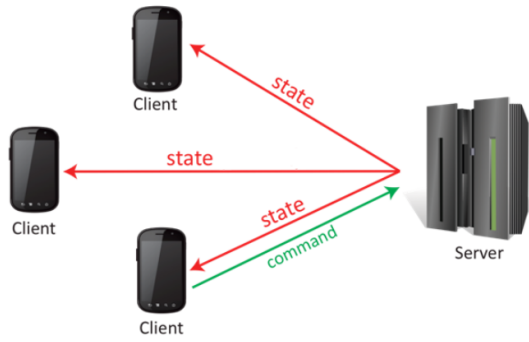


(b) Conexão a dispositivos próximos através de WiFi-Direct

Figura 3.6: Ecrã inicial de conexões WiFi-Direct

Na Figura 3.6a podemos ver o ecrã inicial da aplicação, que permite procurar por dispositivos próximos através de WiFi-Direct utilizando a camada de ligação do “middleware” Hyrax. Os dispositivos encontrados ficam visíveis numa lista e seleccionando um deles é feita uma tentativa de ligação. Uma vez a ligação efectuada o ecrã muda e verificamos uma mensagem similar à Figura3.6b, que significa que a conexão WiFi-Direct foi bem sucedida e podemos passar à próxima fase de setup de jogo.

Regular Networking Model

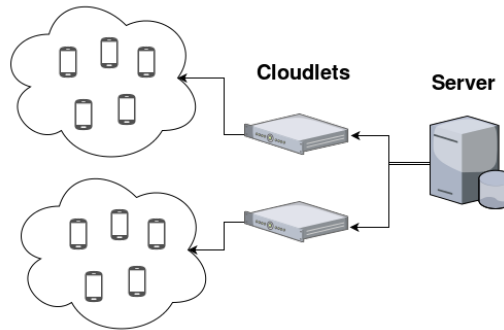


(a) Ambiente de rede regular



(b) Ambiente de rede utilizando o middleware Hyrax

Mobile Edge-cloud



(c) Ambiente de rede utilizando cloudlets

Figura 3.7: Ambientes de rede a comparar

No modelo (a) a latência de jogo pode ser obtida através da soma do tempo que um cliente demora a comunicar a sua jogada ou estado ao servidor, o processamento dessa mesma jogada, e posterior comunicação a todos os outros clientes o novo estado de jogo. Enquanto que no modelo (b) a comunicação é feita entre os clientes e os mesmos processam a informação obtida de modo a eliminar uma grande parte da latência de jogo que é a comunicação com o servidor.

Também foi testado o uso de cloudlets de forma a obter mais um caso de teste válido para este tipo de aplicações. Uma cloudlet é um nó de processamento de pequena escala, colocado próximo dos dispositivos, que tem como objectivo ajudar as aplicações móveis em tarefas exaustivas computacionalmente, provendo os dispositivos com poder computacional com uma menor latência do que um servidor comum devido a estar mais próxima dos mesmos fisicamente.

Capítulo 4

Implementação

Neste capítulo é descrita a implementação do nosso jogo peer-to-peer e detalhes sobre todos os seus componentes. Na secção 4.1 descreve-se a organização geral do projecto. Na secção 4.2 descreve-se a arquitectura do projecto e a sua estrutura. Na secção 4.3 descreve-se a comunicação em rede da aplicação.

4.1 Organização geral

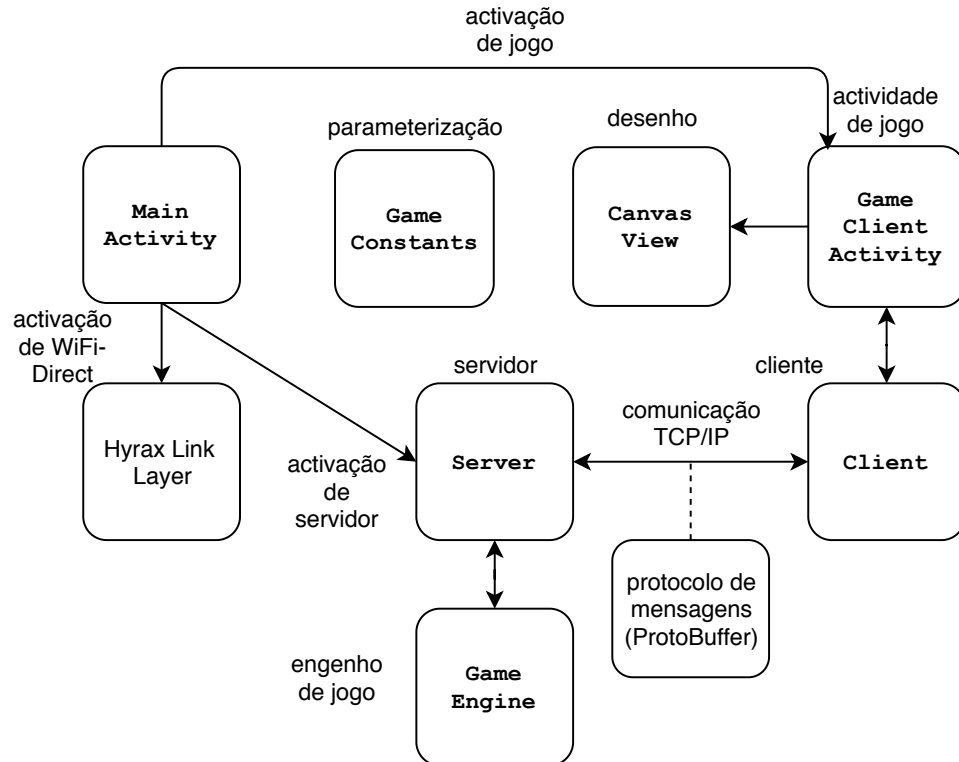


Figura 4.1: Diagrama de componentes da aplicação

O Barrel Shooter foi programado em Java usando o Android Studio IDE, tendo como alvo dispositivos que suportem versões da Android API superior à 21. Em suporte ao jogo foram usadas duas bibliotecas: a Hyrax Link Layer para habilitar comunicações WiFi-Direct e a Google Protocol Buffers (<https://developers.google.com/protocol-buffers>) para definição do protocolo de mensagens cliente-servidor. As imagens usadas no jogo foram obtidas de OpenGameArt (<https://opengameart.org>), um serviço de partilha de conteúdos gráficos de uso livre para jogos.

A organização e funcionalidade geral do código do Barrel Shooter é ilustrada no diagrama da Figura 4.1. Os aspectos principais são os seguintes:

- A classe `MainActivity` é a actividade executada de início pela aplicação Android, tendo por papel: interação com o Hyrax Link Layer quando se pretende configurar uma rede WiFi-Direct, activação de um servidor de jogo no dispositivo local, e lançamento de um jogo em si quando estabelecida a ligação a um servidor (que pode executar local ou remotamente).
- A classe `GameClientActivity` é a actividade despoletada pelo início de um jogo em ligação a um servidor. Tem associada instâncias da classe `Client`, responsável por comunicar com o servidor, e da classe `CanvasView`, responsável pelo desenho dos elementos do jogo no ecrã.
- O servidor de rede é implementado pela classe `Server` e tem associada uma instância de `GameEngine`, que implementa o engenho de jogo. Como referido antes, o servidor pode ser activado pela aplicação Android mas também numa ambiente externo de um servidor remoto, já que o código de `Server` e `GameEngine` usa a API Java standard sem dependências de Android.
- A parametrização do jogo é definida por `GameConstants` nos mais variados aspectos, tais como o número de alvos, a frequência de actualização de jogo, etc.

4.1.1 Parametrização de jogo

O projecto tem um ficheiro de configuração em Java denominado por `GameConstants.java`, onde se pode modificar bastantes configurações do jogo, que é utilizado tanto pelo cliente como pelo servidor:

```

public interface GameConstants {
    int NUMBER_OF_TARGETS = 5;
    int BULLET_BASE_SPEED = 20;
    int BULLET_RATE = 25;
    int TARGET_MS_PER_FRAME = 33;
    int TARGET_MS_PER_MESSAGE = 33;
    int TARGET_MS_PER_SERVER_CYCLE = 33;
    int TARGET_MOVING_SPEED = 3;
    int GAME_DURATION = 60000;
    int FINISHED_GAME_TIMEOUT = 10000;

    int SCREEN_WIDTH = 1536;
    int SCREEN_HEIGHT = 1952;
    int TARGET_WIDTH = 128;
    int TARGET_HEIGHT = 116;
}

```

Figura 4.2: Parâmetros de jogo

- NUMBER_OF_TARGETS - alvos presentes em jogo
- BULLET_BASE_SPEED - píxeis que a bala avança por frame
- BULLET_RATE - frames entre lançamento de balas
- TARGET_MS_PER_FRAME - tempo de duração de um frame
- TARGET_MS_PER_MESSAGE - cadência de envio de mensagens cliente-servidor
- TARGET_MS_PER_SERVER_CYCLE - duração de ciclo do engenho de jogo
- TARGET_MOVING_SPEED - píxeis que os alvos avançam por frame
- GAME_DURATION - duração do jogo em milisegundos
- FINISHED_GAME_TIMEOUT - duração do ecrã de fim de jogo em milisegundos
- SCREEN_WIDTH - largura do ecrã de jogo em píxeis
- SCREEN_HEIGHT - altura do ecrã de jogo em píxeis
- TARGET_WIDTH - largura dos alvos em píxeis
- TARGET_HEIGHT - altura dos alvos em píxeis

4.2 Protocolo de mensagens

Para a troca de mensagens entre o servidor e os clientes foi elaborado um protocolo de mensagens com Protocol Buffers da Google, sintaxe 3. Esse protocolo contém as seguintes mensagens:

```

message RegisterClient{ int32 clientID = 1; }
message ClientUpdate{
    int32 clientID = 1;
    Position position = 2;
}
message GameInfo {
    enum GameState {
        WAITING = 0;
        PREPARING = 1;
        PLAYING = 2;
        FINISHED = 3;
    }
    enum CanonType{
        NORMAL = 0;
        ROUND = 1;
        BIG = 2;
        OP = 3;
    }
    message Canon{
        CanonType canonType = 1;
        Position canonPosition = 2;
        int32 direction = 3;
    }
    message Bullet{
        CanonType bulletType = 1;
        Position bulletPosition = 2;
        int32 direction = 3;
    }
    message Explosion{
        int32 explosionID = 1;
        Position position = 2;
        int32 round = 3;
    }
    message Player{
        int32 playerID = 1;
        Position playerPosition = 2;
        Canon canon = 3;
        repeated Bullet bullets = 4;
        int32 points = 5;
        string name = 6;
    }
    GameState gameState = 1;
    repeated Player players = 2;
    repeated Position targetsPositions = 3;
    repeated Explosion explosions = 4;
}
message Position{
    int32 x = 1;
    int32 y = 2;
}
message Name{ string name = 1; }

```

Figura 4.3: Mensagens do Protocol Buffers

- Mensagem `RegisterClient` serve para registar um cliente com um determinado ID.
- Mensagem `ClientUpdate` serve para o cliente enviar a sua posição actual ao servidor.

- Mensagem `GameInfo` serve para enviar o estado de jogo do servidor para os clientes, contendo todas as informações relativas ao jogo actual.
- Mensagem `Position` serve para enviar posições bi-dimensionais.
- Mensagem `Name` serve para enviar uma string com um nome.

4.3 Comunicação em rede

Para por em prática as soluções propostas neste projecto foi necessário garantir a conectividade através de WiFi-Direct, WiFi e ethernet de todos os dispositivos e servidores envolvidos.

4.3.1 Suporte de WiFi-Direct

```
public void discover(){
    for(String s : wifi.getScanners())
        wifi.cancelDiscovery(s);

    wifi.discover(new ActionListener<Collection<Device>>() {
        @Override
        public void onActionDone(@NotNull Outcome<Collection<Device>> outcome) {
            //Receives the result of discovery (all devices found).
        }

        @Override
        public void onActionResult(@NotNull Outcome<Collection<Device>> outcome) {
            Collection<Device> devices = outcome.getOutcome();
            dev = devices.toArray(new Device[devices.size()]);
            mListener.updateDevices(devices);
        }
    });
}
```

Figura 4.4: Código para descoberta de dispositivos próximos

No caso da comunicação através de WiFi-Direct foi utilizado o middleware Hyrax para descoberta de dispositivos próximos com o código presente na Figura 4.4. Neste trecho de código podemos observar que, uma vez feita a procura, é retornado um array de dispositivos aos quais é possível efectuar uma ligação WiFi-Direct.

```

public void connect(final int device) {
    if(dev.length == 0)
        return;

    // connects to a remote device
    wifi.connect(dev[device],
        ConnectionProperties.newBuilder(dev[device].getName()).build(), new
        ActionListener<Device>() {
            @Override
            public void onActionResult(@NotNull Outcome<Device> outcome) {
                if (outcome.isSuccessful()) {
                    mListener.deviceConnected(dev[device].getName(), getIpAddress());
                } else {
                    // Connection error
                    mListener.connectError(outcome.getError().toString());
                }
            }
        });
}

```

Figura 4.5: Código para conexão de dispositivos próximos através de WiFi-Direct utilizando o middleware Hyrax

No trecho de código da Figura 4.5 é feita uma conexão WiFi-Direct a um dispositivo. No caso da conexão ser bem sucedida é lançado um evento de “deviceConnected” com o nome do dispositivo ao qual foi feita a ligação e o seu endereço IP.

4.4 Interface gráfico

Para o desenho dos objectos de jogo foi utilizado uma classe CanvasView que estende a classe View da Application Programming Interface (API) Android, redefinindo as implementações dos métodos onTouchEvent e onDraw providos pela classe View. Esta classe é responsável pela captura dos eventos de toque no ecrã e sua posição, e também pelo desenho do estado actual do jogo em cada cliente consoante o GameInfo. Corre numa thread própria, diferente da thread de comunicação com o servidor, o que lhe permite ser independente e ter o seu próprio frame rate. Para uma experiência média de jogo foi escolhido um frame rate de 30 frames por segundo, um dos objectivos desta experiência é que a latência de comunicação seja igual ou inferior a este frame rate de modo a não se notar atrasos e quebras visuais.

```

barrelBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.barrel);
backgroundBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.background_space);
canonBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.canon);
normalBulletBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.normal);
roundBulletBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.round);
bigBulletBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.big);
opBulletBit = BitmapFactory.decodeResource(getResources(),
    R.drawable.op);

ExplosionAnimation.addFrame(BitmapFactory.decodeResource(getResources(),
    R.drawable.e1));

```

Figura 4.6: Carregamento das imagens do jogo para a RAM

Na Figura 4.6 estão presentes todos os assets gráficos do jogo, sendo que a última linha, referente ao primeiro frame da animação da explosão, se repete 16 vezes pois a explosão tem 16 frames.

```

canvas.drawBitmap(normalBulletBit, b.getBulletPosition().getX(),
    b.getBulletPosition().getY(), paint);

```

Figura 4.7: Desenho do bitmap referente a uma bala

Na Figura 4.7 é demonstrado como é efectuado o desenho de todos os bitmaps durante o jogo. Neste exemplo específico está a ser desenhada uma bala do tipo normal na sua posição, proveniente da mensagem **GameInfo**.

4.4.1 Classe ExplosionAnimation

```
public class ExplosionAnimation {
    private Position position;
    private long startTime;
    private double duration, fps;
    public static ArrayList<Bitmap> frames = new ArrayList<>();

    ExplosionAnimation(Position p, double fps){
        this.position = p;
        this.duration = (1.0 / fps) * 1000 * frames.size();
        this.fps = fps;
        this.startTime = SystemClock.elapsedRealtime();
    }
    public Bitmap getCurrentBitmap(){
        long ts = (SystemClock.elapsedRealtime() - this.startTime);
        int current = (int) Math.round(ts / ((1.0 / fps) * 1000));
        return frames.get(Math.min(current, frames.size() - 1));
    }
    public boolean isAnimating(){
        long ts = SystemClock.elapsedRealtime();
        if(this.startTime + this.duration < ts)
            return false;
        return true;
    }
    public static void addFrame(Bitmap b){
        frames.add(b);
    }
    public Position getPosition() {
        return position;
    }
}
```

Figura 4.8: Classe de animação de explosão

Na Figura 4.8 é apresentada a classe que permite correr a animação da explosão. Podemos observar que o objecto desta classe guarda sempre a informação relativa a uma explosão num determinado momento, como a sua posição, tempo de início, duração e os frames por segundo pretendidos. Esta classe permite uma animação a um rate independente do ciclo de desenho, controlada pelas suas propriedades, através do cálculo presente na função `getCurrentBitmap`, que devolve o frame actual da animação consoante o seu frame rate alvo.

Capítulo 5

Avaliação

Este capítulo apresenta uma avaliação feita do Barrel Shooter. A avaliação compreende testes efectuados em ambientes distintos de rede e diversas parametrizações do jogo, apreciados depois em termos de tráfego e latência de comunicação. Começamos por apresentar as configurações dos ambientes de rede, hardware correspondente, e configurações dos parâmetros de jogo (Secção 5.1), seguidos dos resultados dos testes e da sua apreciação (Secção 5.2).

5.1 Configurações

5.1.1 Ambientes de rede

Foram conduzidos testes em três ambientes distintos, de acordo com a discussão feita no Capítulo 3 e correspondente ilustração na Figura 3.7 (página 32):

- **Rede Wifi-Direct:** Neste caso o jogo funciona apenas recorrendo a dispositivos Android que fazem parte do mesmo grupo WiFi-Direct. O servidor de jogo pode correr no responsável pelo grupo de WiFi-Direct ou num outro membro do grupo.
- **Ambiente cloudlet:** uma cloudlet próxima aos dispositivos serve de ponto de acesso WiFi. O servidor de jogo pode correr na cloudlet ou em um dos dispositivos.
- **Ambiente cloud:** é feito uso de uma rede pública WiFi, no caso a rede Eduroam. O servidor de jogo pode correr numa máquina arbitrária sem proximidade aos dispositivos, ou em um dos dispositivos.

No total temos 6 configurações de rede distintas, pois cada um dos ambientes engloba duas variantes no alojamento do servidor de jogo.

5.1.2 Configurações base de hardware e software

Para materialização dos ambientes de rede descritos anteriormente foram utilizados:

- 5 tablets Google Nexus 9 equipados com CPU dual-core a 2.3 GHz, 2 GB de RAM, 16 GB de armazenamento interno, placa WiFi 802.11 a/b/g/n/ac, com sistema operativo Android 7.1.1.
- Um computador Intel NUC N2830 usado como cloudlet, com CPU Intel Celeron dual-core a 2.4 GHz, 8 GB de RAM, e placa WiFi, correndo o sistema operativo Ubuntu Linux 14.04.
- A máquina virtual `hyrax.dcc.fc.up.pt` alojada na cloud do Departamento de Ciência de Computadores da Faculdade de Ciências da Universidade do Porto, com 4GB de RAM, CPU dual-core a 2.3GHz, correndo o sistema operativo CentOS Linux 3.19, acessível através da rede pública Eduroam.

5.1.3 Parâmetros de jogo

Conduzimos uma série de testes preliminares para perceber que parâmeterização do jogo seria aconselhável ter para realizar um conjunto de testes representativo, tendo feito a seguinte escolha:

- **Duração de jogo e teste:** fixada a 60 segundos, sendo recolhidos resultados de 5 jogos sucessivos para cada execução de um teste.
- **Frequência da comunicação em rede:** fixada a 30 Hz. Isto significa então que, 30 vezes por segundo e por cada cliente, o servidor envia uma mensagem `GameInfo` (representando o estado do jogo) e recebe de volta uma mensagem `ClientUpdate` (representando o estado do jogador).
- **Número de alvos:** variável em número de 5, 25, ou 125. O número de alvos tem impacto no tamanho das mensagens `GameInfo`, como ilustramos depois neste capítulo. Isto acontece não só porque cada alvo tem de ser representado no estado de jogo, mas também porque ao existir mais alvos a ocorrência de elementos de explosão é mais elevada.
- **Jogadores (clientes):** em número de 2 ou 5 de jogadores (clientes), um por dispositivo. A ideia foi avaliar o caso elementar de jogo ponto-a-ponto com 2 dispositivos, e outro potencialmente mais intenso em termos de carga de rede com 5 dispositivos. De forma similar ao número de alvos, o número de jogadores também impacta no tamanho de mensagens `GameInfo`, dada a necessidade de representar a posição de cada jogador e das suas balas. Dada a combinatória razoável de configurações de rede e parâmetros de jogo, não recolhemos dados para as configurações intermédias de 3 e 4 jogadores. Alguns testes efectuados com 3/4 jogadores não pareciam também apresentar diferenças significativas de desempenho face às configurações de 2/5 jogadores.

5.1.4 Condução dos testes

Os testes foram conduzidos ao longo de vários dias numa sala fechada do Departamento de Ciência de Computadores, no qual não se encontravam outros dispositivos móveis ou pontos de acesso WiFi, em uma altura fora de aulas e em que a densidade de pontos de acesso WiFi para além da Eduroam era relativamente baixa.

Apesar destas cautelas, observou-se em alguns casos grande variabilidade e valores bastante anormais nos resultados de alguns testes, cujos resultados foram descartados. Em particular, notou-se em dadas alturas e de forma aparentemente não-determinista, uma instabilidade significativa de comportamento de grupos WiFi com 5 dispositivos. Note-se ao mesmo tempo que ter alguma variabilidade e pouco controlo sobre as condições de rede foi precisamente o ponto de considerar as configurações fazendo uso da rede pública Eduroam.

Para cada teste, o jogo foi deixado a correr sozinho com mira fixa para cada jogador, mas tal não tem impacto na comunicação cliente-servidor visto que o cliente reporta sempre a posição da mira a cada passo de jogo. Não seria então significativo por exemplo simular o comportamento de um jogador via movimentos induzidos aleatoriamente para a mira.

5.2 Resultados

5.2.1 Carga de rede

Na Figura 5.1 é apresentado um gráfico com a média do tamanho de uma mensagem **GameInfo** por cada uma das combinações das parametrizações em termos de clientes (C) e alvos (A), tomando as medidas do tamanho de mensagem observadas para as 6 configurações de rede possíveis. Face à explicação da parametrização do jogo feita anteriormente, a carga de rede depende destes parâmetros, sendo irrelevante o ambiente de rede em si. Note-se ainda que a mensagem **ClientUpdate**, enviada do cliente para o servidor, tem um tamanho limitado a alguns bytes, não tendo por isso peso significativo na carga de rede. No gráfico observamos uma gama de valores entre menos de 1 KB a um pouco mais de 4 KB, que crescem como esperado em função do número de clientes e alvos.

A partir do tamanho médio de mensagens, e levando também em conta a frequência de jogo (30 Hz) e o número de clientes (2 ou 5), podemos inferir por sua vez a carga induzida na rede em termos de dados de jogo contabilizados para todas as ligações clientes-servidor. Os valores médios são apresentados no gráfico da Figura 5.2. Podemos observar uma carga que atinge cerca de 700 KB/s na configuração mais pesada.

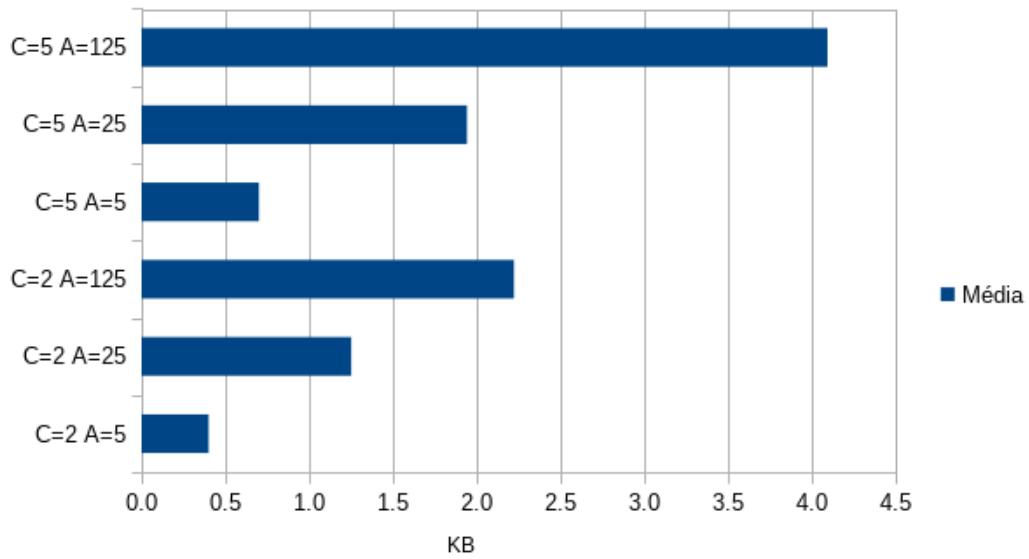


Figura 5.1: Tamanho médio de mensagens **GameInfo**.

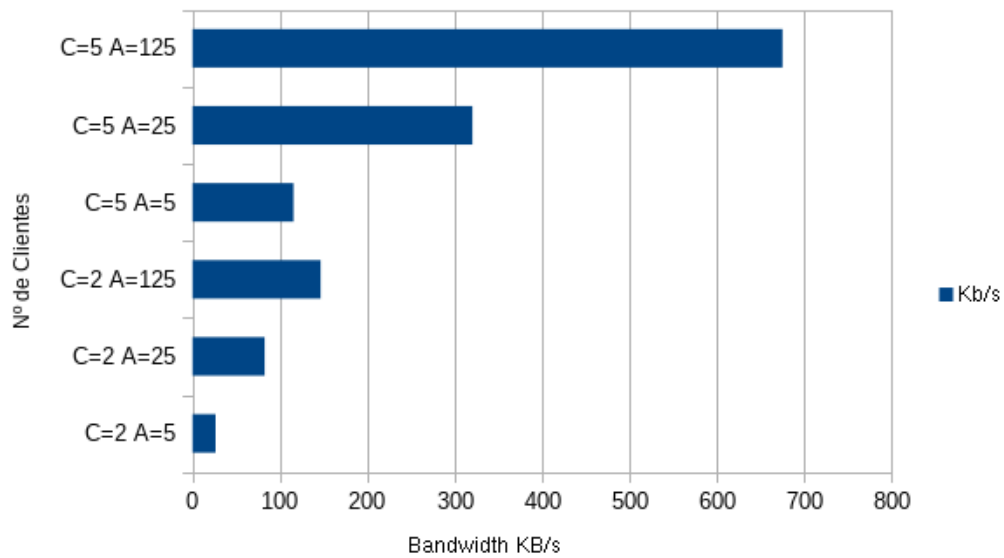


Figura 5.2: Carga de rede.

5.2.2 Latência

Para cada configuração foi medida a latência de jogo em termos de comunicação realizada em cada passo de jogo, isto é o tempo de “round-trip time” medido no servidor compreendido entre o envio de **GameInfo** pelo servidor seguido pela recepção de **ClientUpdate**. Com esta aproximação, os tempos puderam ser recolhidos apenas no servidor de jogo, e evitou-se também lidar com alguma eventual dessincronização de relógios entre dispositivos.

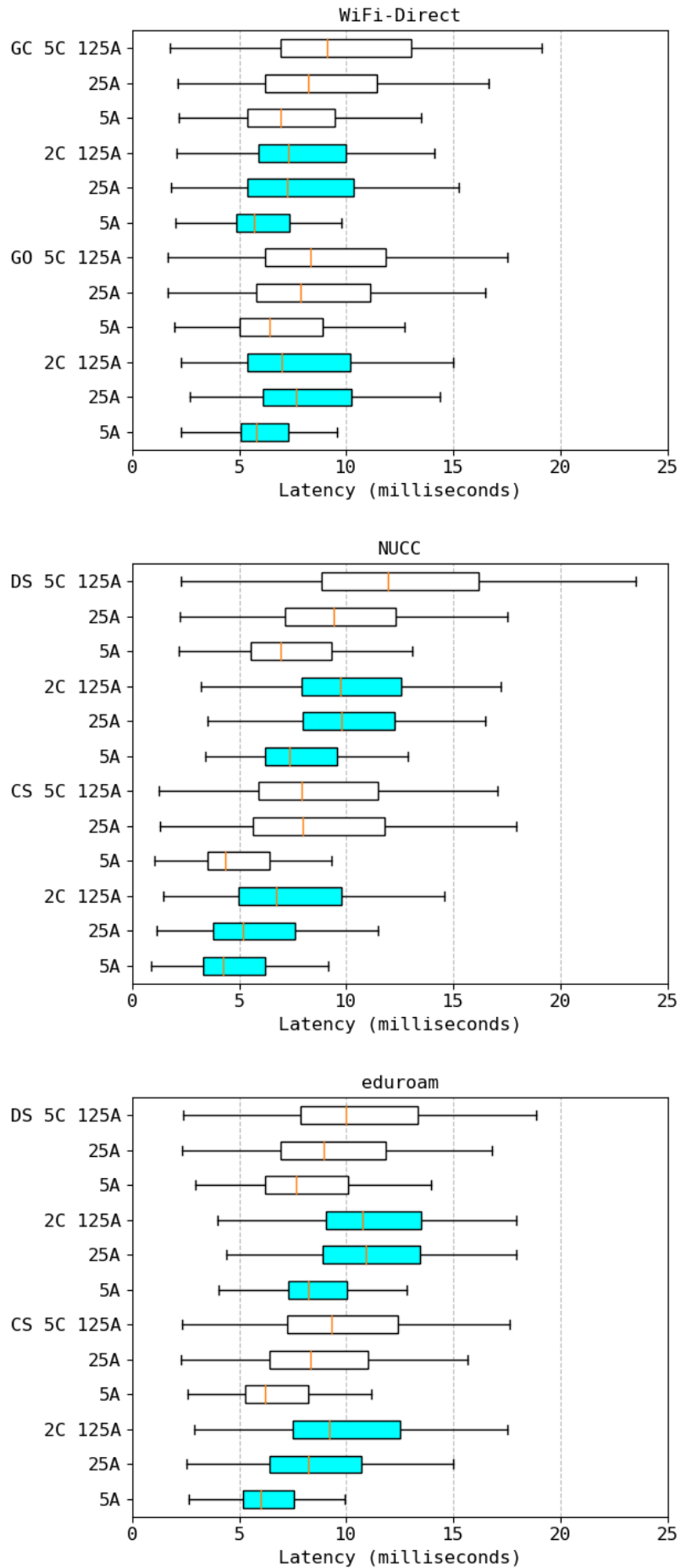


Figura 5.3: Latência de todos os modelos de rede nas várias configurações.

Configuração			min	Q_1	Q_2	Q_3	max	μ	σ
WiFi-Direct									
GC	2C	5A	2.04	4.88	5.67	7.32	711.84	8.25	19.09
		25A	1.84	5.38	7.22	10.32	720.02	9.67	19.55
		125A	2.10	5.89	7.28	9.99	361.38	9.96	14.79
	5C	5A	2.17	5.39	6.92	9.45	2528.50	9.88	29.15
		25A	2.11	6.20	8.21	11.43	3201.80	11.60	29.98
		125A	1.74	6.92	9.10	13.00	2761.30	13.19	33.41
GO	2C	5A	2.29	5.06	5.80	7.32	747.20	8.38	16.46
		25A	2.69	6.10	7.66	10.25	553.26	10.27	15.56
		125A	2.31	5.39	6.97	10.20	866.67	10.13	18.23
	5C	5A	1.96	5.01	6.39	8.87	1292.93	8.89	14.87
		25A	1.67	5.82	7.87	11.14	904.87	10.41	14.90
		125A	1.67	6.20	8.33	11.85	880.31	11.90	18.81
Intel NUC									
DS	2C	5A	3.41	6.23	7.33	9.56	6959.27	24.57	230.42
		25A	3.50	7.98	9.79	12.23	6958.40	26.18	222.59
		125A	3.22	7.94	9.70	12.56	7084.48	26.55	231.63
	5C	5A	2.16	5.53	6.95	9.31	5522.83	19.92	173.30
		25A	2.25	7.13	9.39	12.32	4111.10	19.80	136.40
		125A	2.31	8.84	11.93	16.16	3937.31	24.62	133.77
CS	2C	5A	0.91	3.30	4.25	6.23	1589.25	10.70	37.60
		25A	1.17	3.77	5.18	7.63	2715.81	12.66	54.27
		125A	1.47	4.99	6.74	9.78	1441.70	12.77	37.41
	5C	5A	1.06	3.53	4.33	6.41	2718.95	10.53	45.35
		25A	1.32	5.65	7.98	11.80	2721.45	15.57	54.09
		125A	1.26	5.88	7.90	11.46	4236.49	15.85	62.43
Eduroam									
DS	2C	5A	4.02	7.27	8.23	10.05	251.40	9.34	7.07
		25A	4.42	8.90	10.92	13.42	322.21	11.93	9.00
		125A	3.99	9.06	10.74	13.49	359.70	12.43	11.92
	5C	5A	2.96	6.19	7.64	10.08	373.67	10.41	16.86
		25A	2.36	6.91	8.93	11.84	499.41	10.29	8.88
		125A	2.40	7.84	9.99	13.35	978.60	11.27	14.07
CS	2C	5A	2.65	5.19	5.99	7.56	1297.39	7.78	15.67
		25A	2.54	6.43	8.22	10.72	335.34	9.64	10.88
		125A	2.91	7.50	9.22	12.52	7725.59	12.76	65.86
	5C	5A	2.59	5.26	6.23	8.20	497.91	7.74	10.20
		25A	2.27	6.39	8.32	11.02	347.70	9.91	12.47
		125A	2.31	7.25	9.31	12.43	681.32	11.27	14.07

Tabela 5.1: Resultados detalhados.

A Figura 5.3 apresenta os resultados na forma de “box-plots” para os três ambientes de rede (WiFi-Direct, Intel NUC, e Eduroam) e respectivas configurações em termos do alojamento do servidor de jogo, número de clientes (C) e número de alvos (A): Para o alojamento do servidor de jogo, no caso de Wifi-Direct GO denota que o servidor executou no “group owner”, nó responsável pela rede, e GD se num outro membro do grupo; para as configurações Intel NUC e Eduroam CS e DS denotam respectivamente que o servidor correu na cloudlet/cloud (CS) ou num dispositivo móvel (DS). Os “box plots” apresentam os tradicionais quartis e a extremidade à

direita tem também em conta o tradicional “inter-quartile range” ($IQR = Q_3 - Q_1$). A extremidade à esquerda é no entanto o valor mínimo da latência, visto que alguns medidas $Q_1 - IQR$ tomariam valores negativos. Os valores “outlier” (discutidos mais à frente) são omitidos, porque são bastante altos comparativamente e não ajudariam a uma boa visualização gráfica. Os números concretos de latência são apresentados na Tabela 5.1, incluindo além dos quartis também os valores máximos de latência, bem como os de média e de desvio padrão.

Assinalamos agora os principais aspectos dos resultados:

- **Comportamento geral** — Globalmente, o comportamento regular do jogo é perfeitamente compatível com a frequência de 30 Hz para a comunicação. Os valores de latência são inferiores a 25 ms em todos os casos, e no limite 25 ms permitiria uma frequência de 40 Hz. Isto é coerente com a experiência visual suave que se observa tipicamente para o ecrã de jogo durante o seu desenrolar. Claro que poderíamos aumentar a carga de rede arbitrariamente (ex. via maior número de alvos), acabando por comprometer a experiência de jogo.
- **Comparação de ambientes** — As latências no ambiente WiFi-Direct tendem a ser melhores do que as do ambiente Intel NUC, especialmente no caso de 5 clientes, e mais aproximadas no geral às do ambiente Eduroam. Este aspecto é significativo na medida em que demonstra a exequibilidade de WiFi-Direct para jogos sem suporte de redes infra-estrutural, e em particular do uso do middleware Hyrax.
- **Sensibilidade ao número de clientes e alvos** — Como esperado, de forma geral a latência aumenta com o número de jogadores e alvos em todos os ambientes de rede. Apenas os resultados do ambiente Eduroam para configurações em que o servidor corre num dispositivo (DS) contradizem um pouco esta tendência: os resultados são melhores para 5 clientes do que para 2 pelo menos claramente no que toca ao valor da mediana. Supomos que esta anomalia se devem a variabilidade na rede quando os testes foram feitos.
- **Alojamento do servidor** — Para cada ambiente de rede, os resultados demonstram diferença significativa para o modo de alojamento de servidor. Para WiFi-Direct, os resultados são melhores para o servidor de jogo alojado no “group owner” (GO) face a outro membro do grupo (GD) o que era para nós expectável dado nessa configuração existir apenas um “hop” de rede entre cada cliente e o servidor. Algo similar é observado nos ambientes Intel NUC e Eduroam, onde as latências são menores para o servidor na cloudlet ou cloud (CS) ao invés de um dos dispositivos (DS). Para a configuração cloudlet, temos novamente a vantagem de um “hop” de rede. Foi suposto que para o ambiente Eduroam o facto de o servidor estar presente no Departamento se traduza também em menos “hops”, sendo que não fizemos qualquer avaliação em concreto do trajecto dos pacotes.
- **Valores “outlier”**: como evidenciado na Tabela 5.1, os valores máximos de latência são altos face ao comportamento regular, superando 1 segundo em vários casos e indo até perto de 8 segundos em alguns poucos (estes últimos surpreendentes pela magnitude). Estes valores máximos são “outliers” necessariamente, como se depreende dos “box plots” que enquadram os resultados apropriadamente. Não descartando algum possível erro lógico de programação ou de medição (que não encontramos), os valores mais altos de latência são no entanto coerentes com os eventos de variabilidade imprevista nas condições

de rede anteriormente referidos na Secção 5.1.4 que em certas ocasiões levavam a um bloqueio do jogo momentaneamente.

Capítulo 6

Conclusão

Nesta dissertação foi implementado e avaliado o jogo Android multi-jogador “Barrel Shooter”, usando o “middleware” Hyrax, em vários ambientes de rede, com ênfase em “mobile edge clouds”. Foi descrita a arquitetura do jogo e os seus aspectos de implementação que cumpriram os requisitos para a avaliação proposta. A avaliação e os resultados demonstrados foram positivos, favorecendo a utilização de “mobile edge clouds” com recurso a redes WiFi-Direct em tarefas intensas como jogos multi-jogador, conseguindo obter performance pelo menos comparável à utilização de servidores dedicados na “cloud”.

Identificámos como trabalho futuro algumas possíveis direções e desafios:

- Para além de WiFi-Direct utilizar outras tecnologias de comunicação tais como WiFi-TDLS, Bluetooth ou WiFi-Aware e averiguar se são adequadas ao problema descrito nesta dissertação. Em particular o middleware Hyrax suporta WiFi-TDLS e Bluetooth ao nível do “link layer”, e seria possível adaptar o jogo para executar sobre este tipo de comunicações. No caso de Bluetooth, seria necessário no entanto passar a usar canais de comunicação distintos dos tradicionais sockets TCP/IP.
- Ao nível do middleware Hyrax, seria também interessante o uso do “network layer”, que oferece uma API de “messaging” sobre “mobile edge clouds” e abstrai aspectos como o estabelecimento de ligações e formação de rede.
- O tipo de avaliação conduzida neste trabalho, considerando diferentes ambientes de rede, parece-nos relevante para jogos cliente-servidor já estabelecidos no mercado que podem executar num ambiente LAN. Por exemplo, existem servidores “open-source” para o popular jogo Minecraft, alguns dos quais podem inclusivamente executar em dispositivos móveis.
- Em contraste com uma arquitectura cliente-servidor, jogos com lógica distribuída e eventualmente até geo-referenciada, por exemplo no espírito do conhecido jogo Pokemon Go, poderão constituir casos de estudo ricos para o uso de “mobile edge clouds”.

Bibliografia

- [1] Android Multiplayer Games Offline – LAN, Bluetooth, WiFi Direct. <https://sites.google.com/site/androidmultiplayergames/>.
- [2] K. Bansberg. Cuberite – A lightweight, fast and extensible game server for Minecraft. <https://cuberite.org/>.
- [3] T. Castanheira. Simulation of mobile edge-cloud applications using mininet-wifi. Master’s thesis, Dep. Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, 2018.
- [4] T. Castanheira, J. Silva, E. R. B. Marques, and L. Lopes. Simulation of mobile edge-cloud applications using mininet-wifi. In *INFORUM’18*, 2018.
- [5] F. Cerqueira, J. A. Silva, J. M. Lourenço, and H. Paulino. Towards a persistent publish/subscribe system for networks of mobile devices. In *Proc. MECC’17*. ACM, 2017.
- [6] M. Farokhmanesh. I went to Pokémon Go Fest, and it was a disaster. The Verge, <https://www.theverge.com/2017/7/25/16019404/pokemon-go-fest-refunds-disaster-review>, 2017.
- [7] T. Freitas, J. Rodrigues, D. Bogas, M. Coimbra, and R. Martins. Panoptic, privacy over edge-clouds. In *Proc. FiCloud’18*, pages 325–332. IEEE Computer Society, 2018.
- [8] M. Garcia. Ramble: opportunistic content dissemination for infrastructure-deprived environments. Master’s thesis, Dep. Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, 2018.
- [9] M. Garcia, J. Rodrigues, E. R. B. Marques, and L. Lopes. Ramble: opportunistic content dissemination for infrastructure-deprived environments. In *INFORUM’18*, 2018.
- [10] Google Consumer Barometer. <https://www.consumerbarometer.com>.
- [11] Google Play Store – Top Apps. <https://play.google.com/store/apps/top>.
- [12] IPokeMon – Pokémon like game on iOS with Location Based Service. <http://dev.kjuly.com/iPokeMon>, 2012.
- [13] E. Jones. Cubes – A simple voxel game featuring single player, multiplayer and endless procedurally generated terrain. <https://cubes.ethanjones.me/>.
- [14] A. Le, L. Keller, C. Fragouli, and A. Markopoulou. Microplay: A networking framework for local multiplayer games. In *Proc. MobileGames’12*, pages 13–18. ACM, 2012.
- [15] Minecraft. <https://minecraft.net/>.

- [16] ODROID-XU4 board. <https://wiki.odroid.com/odroid-xu4/odroid-xu4>.
- [17] S. Pushp, C. H. Liu, F. Liu, and J. Song. Multi-player gaming in public transport crowd: Opportunities and challenges. In *Proc. WF-IoT'14*, pages 331–336. IEEE Computer Society, 2014.
- [18] J. Rodrigues. *A Middleware for Mobile Edge-Cloud Applications*. PhD thesis, Dep. Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, 2018. A aguardar defesa.
- [19] J. Rodrigues, E. R. B. Marques, L. Lopes, and F. Silva. Towards a middleware for mobile edge-cloud applications. In *Proc. MECC'17*, pages 1:1–1:6. ACM, 2017.
- [20] J. Rodrigues, E. R. B. Marques, J. Silva, L. Lopes, and F. Silva. Video dissemination in untethered edge-clouds: a case study. In *Proc. DAIS'18*, pages 137–152. Springer, 2018.
- [21] J. Silva, D. Silva, E. R. B. Marques, L. Lopes, and F. Silva. P3-Mobile: Parallel computing for mobile edge-clouds. In *Proc. CrossCloud'17*, pages 5:1–5:7. ACM, 2017.
- [22] J. A. Silva, R. Monteiro, H. Paulino, and J. M. Lourenço. Ephemeral data storage for networks of hand-held devices. In *Proc. Trustcom/BigDataSE/ISPA'16*, pages 1106–1113. IEEE, 2016.
- [23] P. M. P. Silva, J. Rodrigues, J. Silva, R. Martins, L. Lopes, and F. Silva. Using Edge-Clouds to Reduce Load on Traditional WiFi Infrastructure and Improve Quality of Experience. In *Proc. ICFEC'17*, pages 61–67. IEEE, 2017.
- [24] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos. ENORM: A Framework For Edge NOde Resource Management. *IEEE Transactions on Services Computing*, 2018.
- [25] N. Zhang, Y. Lee, M. Radhakrishnan, and R. K. Balan. GameOn: P2p gaming on public transport. In *Proc. MobiSys'15*, pages 105–119. ACM, 2015.