DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

LETI

Sistemas Distribuídos

1º Semestre – 2019/2020

Enunciado do Projeto: SAYF

Search **A**nd **Y**ou will **F**ind

Sumário Executivo

O objetivo do projeto de Sistemas Distribuídos é desenvolver um sistema para a recuperação de objetos perdidos, através de um conjunto de serviços gRPC, implementados na plataforma Java. O sistema chama-se SAYF (*Search And You will Find*). O nome pronuncia-se como a palavra *safe* em inglês.

Este documento apresenta o contexto do projeto, descreve os principais componentes e os requisitos a satisfazer. O projeto está estruturado em duas partes. O documento termina com uma descrição das regras de avaliação.

Introdução

Hoje em dia muitos dos nossos dispositivos preferidos – computadores, *tablets*, telefones, *wearables* – usam tecnologias de comunicação em rádio, sendo as mais usadas o Wi-Fi e o Bluetooth, tal como ilustrado na Figura 1.

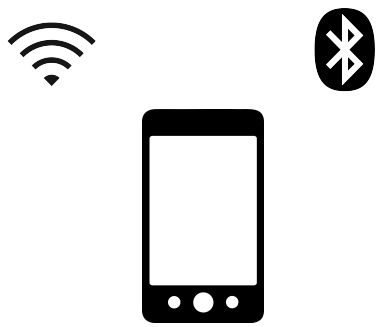

Figura 1: Dispositivo com rádios Wi-Fi e Bluetooth.

O Wi-Fi permite a ligação a redes locais sem fios, através de pontos de acesso (AP - *Access Point*), mas também permite a comunicação direta entre dispositivos. O Bluetooth permite a ligação a redes pessoais que interligam dispositivos como auriculares, teclados, entre muitos outros. Para permitir a comunicação nestes protocolos, cada dispositivo tem um rádio dedicado a essa tarefa que concretiza o nível físico de acesso à rede. No nível da ligação de dados cada dispositivo tem um endereço único que identifica o dispositivo. Este endereço é designado por *MAC address*, tem o comprimento de 48 bits, e é representado da seguinte forma, com dígitos hexadecimais: **14:D6:4D:37:49:22**.

A existência destes identificadores únicos para cada dispositivo é uma ameaça mas é também uma oportunidade. A ameaça é a invasão de privacidade. Como é possível detetar o dispositivo em diversos locais e registar o seu endereço, torna-se possível registar o percurso presumível do dono do dispositivo. Esta informação pode ser usada para concretizar diferentes formas de vigilância. A oportunidade é a capacidade de rastrear os dispositivos e de facilitar a sua recuperação em caso de perda ou roubo, tirando partido da mesma informação recolhida em diversos locais. Neste trabalho vamos focar-nos nesta oportunidade, ou seja, *permitir a recuperação de objetos perdidos*.

1. Primeira parte: sistema SAYF

A ideia do sistema SAYF, a construir, é que existem dispositivos dedicados a vigiar localizações geográficas – chamados *sentries* – que capturam informação nos rádios e reportam o que vêem para um serviço de agregação – chamado *depot*. Os *sentries* podem ser alimentados com dados por um cliente chamado *feeder*. O *depot* armazena as observações, podendo mais tarde responder a pedidos de pesquisa feitos por clientes – chamados *seekers*. A Figura 2 mostra uma visão global dos componentes da solução.

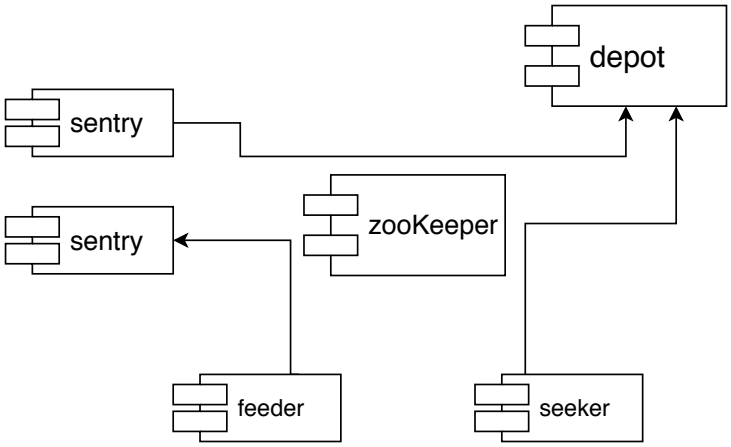

Figura 2: Componentes do sistema SAYF.

Na primeira parte do projeto vai-se construir o *depot*, o *sentry*, o *feeder* e o *seeker*. O *ZooKeeper* é um servidor de nomes bem conhecido, e é usado para registar e localizar os servidores. Nas subsecções seguintes especificam-se detalhes sobre cada componente.

1.1. Servidor *sentry*

O servidor *sentry* recebe observações feitas pelo rádio e depois envia um conjunto de observações acumuladas para o armazenamento. Cada sentinela tem associado um nome e uma geolocalização

em coordenadas: latitude, longitude. Por exemplo, uma sentinela no IST Taguspark tem as coordenadas 38.737613, -9.303164; uma sentinela à porta do Pavilhão Central do IST tem as coordenadas 38.736748, -9.138908.

As operações a disponibilizar são as seguintes:

- `feed` – recebe uma observação de um *MAC address*.
- `push` – envia as observações recebidas desde a última invocação para o *depot*.

1.2. Servidor *depot*

O servidor *depot* regista sentinelas, recebe e armazena observações de dispositivos e depois responde a pesquisas.

Uma sentinela só pode enviar observações depois de se registar com sucesso. As sentinelas têm um nome, que deve ser descritivo do local onde se encontram. Este nome apenas pode conter letras e números, e tem comprimento mínimo de 3 e máximo de 12 caracteres.

As operações a disponibilizar por este servidor são as seguintes:

- `join` – recebe um nome de uma nova sentinela e as suas coordenadas de localização. O nome indicado para a sentinela tem que ser único i.e. não pode ser duplicado de um já existente.
- `report` – recebe um conjunto de observações, com nome da sentinela, e com os endereços observados. O servidor *depot* regista as observações com a sua data e hora no momento da receção;
- `search` – recebe um pedido de pesquisa, com um endereço MAC completo e um número máximo de resultados a devolver. Devolve os registos de observação, em mensagens estruturadas com campos individuais para cada atributo, arrumados numa coleção, ordenada por data e hora decrescente (mais recente primeiro).
- `searchMatch` – recebe um pedido de pesquisa, com um fragmento de endereço MAC, indicação de primeiros ou últimos bits, e um número máximo de resultados a devolver para cada endereço MAC encontrado. Devolve os registos de observação, também estruturados em campos individuais e arrumados numa coleção, ordenados por MAC crescente e depois por data e hora decrescente (mais recente primeiro).

1.3. Testes automáticos dos servidores

Espera-se que cada servidor inclua um cliente de testes – *sentry-client* e *depot-client* – que permita verificar o funcionamento correto das operações descritas neste enunciado.

Os *testes de integração* (IT) verificam o cumprimento do contrato de um serviço através de invocações remotas. Devem usar `JUnit` para invocar todos os servidores remotos necessários (que se assume terem sido previamente lançados de forma correta).

Dado que os `protobuf` serão definidos por cada grupo, tendo por base a especificação deste documento, não existirão baterias de testes automáticos pré-determinados pelos docentes.

1.4. Operações auxiliares dos servidores

Cada servidor deve dispor também de um conjunto de operações de controlo que se destinam apenas a facilitar a realização de testes. Por convenção, o seu nome deve começar por `ctrl_`.

- `ctrl_ping` – recebe um pedido de sinal de vida e responde com uma mensagem indicando o estado do servidor;
- `ctrl_clear` – deve limpar totalmente o estado do servidor;
- `ctrl_init...` – opcional – permitem definir parâmetros de configuração inicial do servidor.

1.5. Cliente *feeder*

O *feeder* permite fornecer observações de dispositivos ao *sentry*.

O *feeder* tem uma interface-utilizador de linha de comando. O programa deve ler do *standard input* endereços MAC de observações, um por linha. As observações devem ser memorizadas à medida que são lidas. Ao receber uma linha vazia, ou ao detetar o fecho do *input*, o cliente deve enviar as observações acumuladas para o servidor.

Argumentos

O *feeder* deverá receber como argumento na linha de comando o nome do *sentry* a contactar para fazer o envio das observações. O nome deve ser traduzido para endereço com a ajuda do servidor de nomes. Este argumento é obrigatório.

1.6. Cliente *seeker*

O *seeker* é uma interface-utilizador de linha de comando que permite lançar interrogações ao *depot*. Para um dado dispositivo, deve ser possível: obter a sua observação mais recente; obter o seu rasto de observações.

Argumentos

O *seeker* deverá receber como argumento na linha de comando o nome do *depot* a contactar. Este argumento é opcional. O nome deve ser traduzido para endereço com a ajuda do servidor de nomes. Se não for indicado, o *seeker* deverá escolher o único *depot* disponível, ou havendo vários, um deles de forma aleatória.

Endereços para pesquisa

Um dispositivo é identificado por um endereço MAC com o formato `XX:XX:XX:XX:XX:XX` onde *X* é um dígito hexadecimal, em maiúsculas ou minúsculas.

O endereço a procurar pode ser especificado totalmente, com indicação de todos os 48 bits, ou parcialmente, através de um fragmento, com indicação dos primeiros ou últimos dígitos. A notação `XX:` permite pesquisar por endereços que comecem pelos bits especificados por XX. A notação `:XX` permite pesquisar endereços que terminem com os bits indicados. Os fragmentos são especificados com múltiplos de 8 bits, até um total de 40 bits.

Exemplos:

Fragmento de 8 bits iniciais (tamanho mínimo):

```
14:
```

Fragmento de 16 bits iniciais:

```
14:D6:
```

Fragmento de 16 bits iniciais (também se podem usar minúsculas):

```
14:d6:
```

Fragmento de 40 bits iniciais (tamanho máximo):

```
14:D6:4D:37:49:
```

Fragmento de 24 bits finais:

```
:37:49:22
```

Comando *track*

O comando `track` procura a observação mais recente do dispositivo com o endereço ou fragmento de endereço. O resultado devem ser linhas com o formato: Endereço MAC,Data-hora,Nome-Sentinela,Latitude,Longitude

O resultado deve ser ordenado por endereço MAC crescente e data-hora decrescente (mais recente primeiro). A data-hora deve seguir o formato ISO 8601. As coordenadas são apresentadas em notação decimal.

Exemplos:

Procura o dispositivo com o endereço exatamente igual ao indicado, e nada é devolvido (linha vazia):

```
$ track 14:D6:4D:37:50:23
```

Procura o dispositivo com o endereço exatamente igual ao indicado, e é devolvido um resultado:

```
$ track 14:D6:4D:37:49:22
14:D6:4D:37:49:22,2019-10-04T10:02:07,Taguspark,38.737613,-9.303164
```

Procura dispositivos cujo endereço comece nos valores indicados. Neste caso, apenas foi encontrado um resultado:

```
$ track 14:D6:
14:D6:4D:37:49:22,2019-10-04T10:02:07,Taguspark,38.737613,-9.303164
```

Procura dispositivos cujo endereço termine nos valores indicados. Foram encontrados dois resultados:

```
$ track :49:22
10:FF:EA:37:49:22,2019-10-04T11:02:07,Taguspark,38.737613,-9.303164
14:D6:4D:37:49:22,2019-10-04T10:02:15,Taguspark,38.737613,-9.303164
```

Comando *trace*

O comando `trace` procura o caminho percorrido pelo dispositivo, ordenado da observação mais recente para a mais antiga. O resultado são linhas com o mesmo formato e ordenação do comando *track*.

Exemplos:

Procura o rasto dispositivo com o endereço exatamente igual ao indicado, e nada é devolvido (linha vazia):

```
$ trace 14:D6:4D:37:50:23
```

Procura o rasto do dispositivo com o endereço exatamente igual ao indicado, e são devolvidos três resultados:

```
$ trace 14:D6:4D:37:49:22
14:D6:4D:37:49:22,2019-10-04T10:02:05,Taguspark,38.737613,-9.303164
14:D6:4D:37:49:22,2019-10-03T08:10:20,Alameda,38.736748,-9.138908
14:D6:4D:37:49:22,2019-10-02T22:33:01,Taguspark,38.737613,-9.303164
```

Procura o rasto de dispositivos cujo endereço comece nos valores indicados. Neste caso, apenas foi encontrado um resultado:

```
$ trace 14:D6:
14:D6:4D:37:49:22,2019-10-04T10:02:05,Taguspark,38.737613,-9.303164
```

Procura o rasto de dispositivos cujo endereço termine nos valores indicados. Foram encontrados rastos de dois dispositivos:

```
$ track :49:22
10:FF:EA:37:49:22,2019-10-01T10:07:22,Taguspark,38.737613,-9.303164
10:FF:EA:37:49:22,2019-10-01T09:02:10,Alameda,38.736748,-9.138908
14:D6:4D:37:49:22,2019-10-04T10:02:05,Taguspark,38.737613,-9.303164
14:D6:4D:37:49:22,2019-10-03T08:10:20,Alameda,38.736748,-9.138908
14:D6:4D:37:49:22,2019-10-02T22:33:01,Taguspark,38.737613,-9.303164
```

1.7. Tecnologia

Todos os componentes do projeto serão implementados na linguagem de programação Java. A invocação remota de serviços deve ser suportada por serviços gRPC.

Cabe ao grupo definir os *protocol buffers* que julguem necessários para concretizar o projeto. Não existem contratos pré-definidos.

Cada servidor – *sentry*, *depot* – é lançado de forma autónoma. Todos os servidores devem ser localizados dinamicamente pelos respetivos clientes, por intermédio de um servidor de nomes bem conhecido. Mais precisamente, cabe a cada servidor, quando lançado, registar-se a si próprio no servidor de nomes, indicando o seu endereço. Os nomes a usar para os serviços são[1]: `/grpc/sayf/depot/1`, `/grpc/sayf/sentry/1`, `/grpc/sayf/sentry/2`, etc. Os números de instância podem ser substituídos por nomes textuais. Por exemplo, o nome de uma instância da sentinela pode ser `Taguspark` e a outra `Alameda`.

O servidor de nomes a usar é o ZooKeeper, a correr na máquina e porto por omissão.

1.8. Sumário

Em resumo, na primeira parte do trabalho, é necessário implementar dois servidores: *sentry* e *depot*, com registo de endereços no servidor de nomes e com testes de integração; e dois clientes: *feeder* e *seeker* com interface-utilizador na linha de comandos.

2. Segunda Parte: Tolerância a Faltas

Na segunda parte vai-se replicar o servidor *depot* para permitir uma tolerância a faltas no armazenamento de observações e nas pesquisas.

Vai-se implementar uma variante do protocolo *gossip architecture* com consistência fraca.

Recomenda-se que cada grupo considere variantes do protocolo *gossip* tendo em conta o cenário específico deste projeto. O objetivo será adequar o modelo de consistência aos requisitos da aplicação. As opções devem ser discutidas com os docentes nas aulas de apoio e depois descritas no relatório de projeto.

2.1. Múltiplos servidores

Em vez de existir um único *depot*, passam a existir vários servidores, chamados 'gestores de réplica' ou simplesmente 'réplicas'. Todos são registados no servidor de nomes.

Os clientes enviam pedidos, de leitura ou modificação, a uma réplica indicada pelo utilizador. Caso o utilizador não indique uma réplica, deve ser escolhida uma de forma iterativa (uma de cada vez) ou de forma aleatória (sorteada).

Neste protocolo, uma réplica pode aceitar uma observação sem contactar as outras réplicas.

2.2. Modelo de interação e faltas

Como modelos de interação e faltas, deve assumir-se que:

- O sistema é assíncrono e a comunicação pode omitir mensagens (apesar do projeto usar HTTP como transporte, deve assumir-se que outros protocolos de menor fiabilidade podem ser usados);
- Os gestores de réplica podem falhar silenciosamente mas não arbitrariamente, i.e., não há falhas bizantinas;
- Embora o conjunto de gestores de réplica seja estático, os seus endereços não são conhecidos *a priori* e podem variar ao longo do tempo;
- Existe sempre, pelo menos, uma réplica ativa para atender os clientes;
- As falhas das réplicas são transientes e não definitivas.

2.3. Atualização de réplicas entre si

As réplicas podem ter vistas divergentes sobre as observações. Para reduzir a divergência, as modificações são propagadas entre réplicas, de forma relaxada, quando possível.

O processo de atualização de réplicas deve usar um *timestamp* vetorial para representar uma versão resultante da execução cumulativa de um conjunto de atualizações.

Assim sendo, cada réplica deve ter um temporizador, que executa a atualização (*gossip*) periodicamente, com vista à partilha de observações com todas as outras réplicas.

2.4. Leituras consistentes por cliente

A solução a construir deve garantir que, um mesmo cliente nunca deve fazer leituras inconsistentes.

O caso que deve ser evitado é o seguinte:

- Um cliente C1 faz uma leitura a partir da réplica R1 e o valor lido reflete uma atualização u1. Por outras palavras, o cliente "vê" a atualização u1;
- posteriormente, o mesmo cliente C1 lê, de uma outra réplica R2, e o valor lido não reflete a atualização u1. Por outras palavras, a atualização u1 "desapareceu";

Este caso é uma anomalia de consistência, porque uma observação, já vista pelo cliente, pode desaparecer. A resolução desta anomalia necessitará de modificações no *front-end* do lado do cliente, tal como será estudado nas aulas teóricas.

2.5. Sumário

Em resumo, na segunda parte do trabalho, é necessário criar várias réplicas do *depot*, em que cada uma pode responder autonomamente a clientes, em que existe partilha periódica de atualizações entre réplicas (*gossip*), e em que são evitadas leituras inconsistentes por um mesmo cliente.

Compete também aos grupos, para melhorar o seu trabalho, identificar outras anomalias de consistência a evitar, de modo a que o modelo de consistência fraca da aplicação seja adequada aos seus objetivos funcionais.

3. Avaliação

3.1. Identificador de grupo

O identificador do grupo tem o formato CXX, onde: C representa o campus (A para Alameda e T para Taguspark); XX representa o número do grupo de SD atribuído pelo Fénix. Por exemplo, o grupo A22 corresponde ao grupo 22 sediado no campus Alameda; já o grupo T07 corresponde ao grupo 7 sediado no Taguspark.

3.2. Colaboração e lideranças

O Git é um sistema de controlo de versões do código fonte que é uma grande ajuda para o trabalho em equipa. Toda a partilha de código para trabalho deve ser feita através do GitHub. Cada membro da equipa deve atualizar a sua foto no GitHub para facilitar a identificação e comunicação.

A atualização do repositório deve ser feita com regularidade, correspondendo à distribuição de trabalho entre os estudantes e às várias etapas de desenvolvimento. Cada elemento do grupo deve atualizar o repositório do seu grupo à medida que vai concluindo as várias tarefas que lhe foram atribuídas.

Cada componente do projeto (módulo Maven) tem que ter um **líder** de desenvolvimento, identificado no ficheiro `README.md` dentro da pasta desse módulo, juntamente com os outros contribuidores. Cada membro da equipa deve ser responsável, pelo menos, por um dos componentes do trabalho. O responsável por um módulo de testes de servidor deve ser diferente do responsável pelo respetivo servidor, ou seja, os servidores devem ser testados por diferentes pessoas.

3.3. Entregas

As entregas do projeto serão feitas também através do repositório GitHub[2]. A cada parte do projeto a entregar estará associada uma *tag*. Cada grupo tem que marcar o código que representa cada entrega a realizar com uma *tag* específica – `SD_P1` e `SD_P2` – antes da hora limite de entrega.

3.4. Qualidade do código

A qualidade da estrutura base engloba os seguintes aspetos de avaliação (em todas as partes):

- Configuração (POMs e *Handlers*)
- Código legível (incluindo comentários relevantes)
- Tratamento de exceções adequado
- Sincronização correta

3.5. Primeira parte (P1)

A primeira parte vale 10 valores em 20, distribuídos da seguinte forma:

- *depot* (35%)
  - Implementação das operações (50%)
  - Testes de integração desenvolvidos pelo grupo (30%)
  - Qualidade do código (20%)
- *sentry* (25%)
  - Implementação das operações (50%)
  - Testes de integração desenvolvidos pelo grupo (30%)
  - Qualidade do código (20%)
- utilização correta do servidor de nomes (10%)
- *feeder* (10%)
- *seeker* (20%)

A data limite de entrega é: *sexta-feira, 8 de novembro de 2019, 19:00*.

[2]Os grupos que não usam GitHub para trabalhar podem recorrer a uma entrega via Fénix. Embora não seja suposto um grupo entregar o projeto por ambas as vias (Fénix e GitHub), nessas situações o corpo docente considerará **apenas** o projeto submetido via Fénix.

3.6. Segunda parte (P2)

A segunda parte vale 10 valores em 20, distribuídos da seguinte forma:

- Qualidade do código (20%)
- Replicação (20%)
- Atualização (*gossip*) entre réplicas (20%)
- Leituras consistentes pelo mesmo cliente (20%)
- Relatório e demonstração (20%)

A data limite de entrega é: *sexta-feira, 29 de novembro de 2019, 19:00*.

3.7. Relatório

Na segunda parte, além do código-fonte, deve também ser entregue o relatório. O documento, em formato PDF, tem que ser colocado na pasta `doc/` na raiz do projeto. O ficheiro deve chamar-se `CXX-relatorio-tolfaltas.pdf`.

O documento deve conter:

- 1 folha de rosto:
  - Identificador do grupo em formato CXX;
  - URL do repositório no GitHub;
  - Fotos, números e nomes dos membros do grupo (ordenados por número de estudante crescente, da esquerda para a direita) – pede-se que o grupo tire uma foto em conjunto para reforçar o espírito de equipa!
- 3 páginas de conteúdo:
  - Definição do modelo de faltas: que faltas são toleradas;
  - Figura da solução de tolerância a faltas;
  - Breve explicação da solução, suportada pela figura anterior;
  - Descrição de opções de otimização/melhoria;
  - Detalhe do protocolo (troca de mensagens).

3.8. Demonstração

Cada grupo deve preparar um *guião de demonstração*, com casos de utilização, passo a passo, que demonstram as melhores funcionalidades do trabalho.

Tolerância a faltas – Duração inferior a *5* minutos

- Caso *F1*: passos para demonstrar o funcionamento normal da replicação
- Caso *F2*: passos para demonstrar tolerância a falta

O guião deve também incluir instruções de instalação e configuração, que devem começar com a obtenção do código no repositório Git.

O guião deve ser incluído na pasta `demo/` na raiz do projeto em formato PDF. O documento deve chamar-se `CXX-guiao-tolfaltas.pdf`.

A demonstração do trabalho será realizada, ao vivo, na discussão do trabalho.

3.9. Discussão

Todos os estudantes têm discussão final do projeto. As notas das várias partes, 1 e 2, são indicativas e sujeitas a confirmação na discussão final, na qual todo o trabalho desenvolvido durante o semestre será tido em conta. As notas atribuídas são individuais. É muito importante que a divisão de tarefas ao longo do semestre seja equilibrada pelos membros do grupo.

Todas as discussões e revisões de nota do trabalho devem contar com a presença obrigatória de todos os membros do grupo.

3.10. Atualizações

Para as últimas novidades sobre o projeto, consultar a página Web regularmente:

`http://disciplinas.tecnico.ulisboa.pt/leic-sod/2019-2020-sem1/labs/`

O esclarecimento de dúvidas será realizado através do Piazza:

`https://piazza.com/tecnico.ulisboa.pt/fall2019/sd20_1`

Bom trabalho!