Novos Paradigmas de Rede
Segurança rodoviária suportada em computação de proximidade

PL4
Rui Alves, pg50745
João Mendes, pg50483

Departamento Informático - Universidade do Minho
4710-057, Braga, Portugal
{pg50745,pg50483}@alunos.uminho.pt

Resumo The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introdução

No âmbito da unidade curricular de Novos Paradigmas de Rede, foi-nos proposto a implementação de um protótipo funcional de um serviço de segurança rodoviária, suportado pela comutação de proximidade.

Ao longo do relatório iremos abordar a estrutura elaborada para o desenvolvimento do serviço de segurança, bem como, explicar de forma detalhada a implementação do mesmo. Primeiramente, vamos abordar as apps desenvolvidas para cada componente do sistema (Veiculo, RSU, Servidor e Servidor Remoto). De seguida, abordamos os protocolos desenvolvidos onde falamos sobre as mensagens criadas que serão enviadas e recebidas entre os componentes do sistema. Para além disso, falamos da constituição do Pacote que desenvolvemos para ser transmitido na rede. Por fim, especificamos como é realizado o routing e o fowarding dos diferentes pacotes.

Para finalizar o relatório, apresentamos testes relativos ao funcionamento do serviço implementado, uma breve conclusão e possível trabalho futuro.

2 Implementação

Foram desenvolvidas 4 aplicações para correr nos 4 diferentes componentes do nosso sistema (Veiculo, RSU, Servidor e ServidorRemoto), cada uma destas apps foi desenvolvida utilizando a linguagem de programação JAVA e recorremos a *Threads* de maneira a paralelizar a execução de ações. Foram utilizados *Reentrant Locks* para garantir o sucesso de operações sobre as mesmas variáveis nas diferentes *Threads* em execução.

Relativamente, ao envio de mensagens, enviamos *DatagramPackets* sobre o protocolo UDP através de *DatagramSockets*. De seguida, iremos explicar cada uma das aplicações para entender melhor o seu funcionamento.

2.1 App Veiculo

Relativamente a App do Veiculo, criamos threads relativas a cada ação que o veiculo irá executar, isto é, a receção e envio de mensagens. Relativamente à receção das mensagens, aquando da receção mensagens do tipo 1, primeiramente, verificamos se o vizinho que mandou a mensagem já se encontra na tabela de vizinhos do veiculo em causa, de forma a manter as tabelas de vizinhança devidamente atualizadas.

No que diz respeito às mensagens do tipo 2, estas servem para adicionar o pacote que recebe dos vizinhos à lista de pacotes CAM que possui, *List<Packet> DBlist*, de forma a armazenar cada mensagem para futuro envio ao RSU.

Por fim, as mensagens do tipo *Warning*, que são mensagens numeradas sequencialmente a partir do número 3, estas são referentes às mensagens emitidas pelo Servidor para informar o próprio veiculo relativamente à velocidade em excesso.

- ***Mensagem CAM***: O veiculo envia estas mensagens com o intuito de chegar ao RSU e servidor mais próximo, contendo informação sobre o veiculo de acordo com a velocidade, o piso onde se encontra e a sua posição, valor de x e y.
- ***Mensagem de preenchimento das tabelas de vizinhos***: O veiculo envia constantemente mensagens com o sua posição e identificação de forma a preencher as tabelas de vizinhança dos nodos com que este mantém conetividade.
- ***Mensagem de Warnings***: O veiculo encaminha estas mensagens, através do algoritmo greedyForwarding, até aos veículos que estejam num raio <100 metros do Veiculo que está a cometer a infração.

2.2 App RSU

No que toca a App do RSU, desenvolvemos uma arquitetura igual à app Veiculo, ou seja, elaboramos threads para especificar o funcionamento das ações que o RSU executa. Portanto, a Thread relativa à receção de mensagens guarda os pacotes de acordo como o seu tipo na respetivas listas (base de dados). No nosso sistemas existem 2 RSUs distintos, os veículos irão encaminhar as suas mensagens para o que se encontrar mais próximo da sua posição.

- ***Mensagem CAM***: O RSU recebe mensagens CAM vindas dos Veículos e encaminha-as para o servidor.
- ***Mensagem de preenchimento de tabelas de vizinhança***: O RSU envia e recebe constantemente mensagens com a sua posição e identificação, de forma a preencher as tabelas de vizinhos dos nodos com que este mantém conetividade.

– ***Mensagem de Warnings***: O RSU encaminha estas mensagens, através do algoritmo greedyForwarding, até aos veículos que estejam num raio até 100m do Veiculo que está a cometer a infração.

2.3 App Servidor Remoto

A app do Servidor Remoto consiste em armazenar na sua base de dados, *List<Packet> SVdatabase*, os pacotes que o recebe do servidor contendo da informação dos diferentes veículos

2.4 App Servidor

No que diz respeito à app do Servidor, partindo do processo elaborado nas outras apps, criamos também Threads especificas para o tratamento dos diferentes pacotes.

– ***Mensagens CAM***: O servidor recebe mensagens to tipo CAM vindas do RSU e guarda-as, caso vindas de um veiculo na sua área de interesse do servidor, na sua base de dados, *List<Packet> SVdatabase*, as restantes são guardadas em *List<Packet> dadosParaServer* para posterior envio ao Server Remoto .
– ***Mensagem de Warnings***: O servidor limita-se a enviar warnings sobre velocidade máxima permitida (Velocidades definidas por área na topologia) e avisa, também, sobre o estado do piso em que o veiculo se encontra.
– ***Área de Interesse***: O servidor apenas trata mensagens vindas de veículos num raio 500 metros do mesmo, este número de veículos dentro da sua área de interesse é guardado e atualizado para posteriormente mostrar essa informação (*List<Integer> veiculosEmRange*).

3 Protocolos

3.1 Tipo de mensagens

Implementamos 3 tipos de mensagens. Primeiramente cada veiculo está constantemente a enviar mensagens, ***packetType 1*** com a sua *Posição* para o grupo Multicast, de forma a preencher, devidamente, as tabelas de vizinhança, que iremos tratar em baixo.

Relativamente a mensagens com informação sobre o estado do veiculo (mensagens CAM) ***packetType 2*** estas são enviadas direcionadas ao RSU, evitando o *flooding* da rede. Existem ainda mensagens ***Warning packetType 3+*** relativas às mensagens de *Warnings* enviados pelo servidor para os veículos.

O objeto Packet foi inicialmente desenvolvido para tratar as mensagens ***packetType 2*** (CAM), e é adaptado aquando do envio/receção de outro tipo de mensagens, as mensagens de ***packetType 1*** apenas utilizam os campos relativos às coordenadas do veículos e à identificação do mesmo, já o ***Warning packetType 3+*** trata os atributos de maneira diferente, irá ser falado de seguida.

4 Packet

Relativamente à informação que irá ser transmitida nesta rede foi desenvolvido um *Packet* que posteriormente é transformado num array de bytes, os quais irão representar a informação relativa aos veiculos.

- **Int packetType**: Tipo do pacote a ser transmitido;
- **InetAddress ip**: Endereço IPv6 do nodo
- **Int nodeNumber**: Número do nodo na topologia CORE;
- **Double coordX**: Abcissa da localização do veiculo;
- **Double coordY**: Ordenada da localização do veiculo;
- **Int estadoPiso**: Inteiro que representa o estado do piso do veiculo (Atribuído de forma aleatória);
  - **0** : Seco
  - **1** : Chuva
  - **2** : Neve
  - **2** : Gelo
- **Int velocidade**: Inteiro que representa a velocidade do veiculo (Atribuído de forma aleatória entre os valores: 50, 90, 120, 150).

No caso das mensagens ***Warning packetType 3+***, warnings emitidos pelo servidor, o packet a ser transmitido é o mesmo contudo algumas das variáveis tem um significado diferente.

packetType 3:

- **Int packetType**: Número sequencial, a começar em 3, de forma a identificar cada warning emitido pelo servidor.
- **Int nodeNumber**: Número do nodo a quem o *warning* é direcionado;
- **Double coordX**: Abcissa da localização do veiculo em causa;
- **Double coordY**: Ordenada da localização do veiculo em causa;
- **Int estadoPiso**: Inteiro que representa o estado do piso do veiculo (Atribuído de forma aleatória);
- **Int velocidade**: Inteiro que representa a velocidade máxima permitida na zona (explicado na Topologia).

5 Routing e Forwarding

De forma a criar esta rede veicular tolerante a atrasos, foi necessário implementar um paradigma de *store-carry-and-forward* para as mensagens que irão ser transmitidas pelos veículos. Para os veículos enviarem as suas mensagens CAM ***packetType 2*** ao RSU recorremos ao algoritmo ***Greedy Forward***, este algoritmo foi também utilizado no envio das mensagens de *warning* do RSU para os veículos.

Lógica do algoritmo (***Greedy Forward***):

- **Veiculo para RSU**: Cada veiculo recorre a uma tabela de vizinhos, da qual iremos falar de seguida, e caso este tenha conetividade direta com um dos RSUs envia a informação para o mesmo, caso contrario envia para o vizinho mais perto da posição do RSU (a posição é conhecida).
- **Servidor para Veículos**: O Servidor envia mensagens de *warning* direcionadas ao veiculo que se encontra a realizar a infração e envia também mensagem a veículos que se encontrem a um determinado raio deste veiculo alvo.

5.1 Tabela de vizinhança

Esta tabela presente nos veículos e no RSU contem todos os nodos (Veículos) que se encontram em range de comunicação. Para manter esta tabela devidamente atualizada foi criada uma Thread especifica, que se encarrega de efetuar uma limpeza desta lista caso o veiculo esteja um determinado tempo sem enviar mensagem, este *timeout* é definido com o valor de 10s. De maneira a implementar esta tabela foi criada um lista com objetos do tipo VehicleInfo (List<VehicleInfo>):

- **int NodeNumber**: Número do nodo da topologia CORE;
- **double x**: Abcissa da localização do veiculo;
- **double y**: Ordenada da localização do veiculo;
- **long updateTime**: Identificador da hora em que foi adicionado à tabela;
- **InetAddress carIp**: Endereço IPv6 do veiculo.

5.2 Armazenamento de mensagens

Uma vez que estamos a tratar redes tolerantes a atrasos, todas as mensagens devem ser guardadas até existir oportunidade de envio das mesmas, para tal são criadas listas ***List<Packet>*** que posteriormente ao envio das mensagens são esvaziadas, fazendo com que não existam réplicas das mensagens na rede. Ambas as listas existem nos veículos e no RSU.

- **List<Packet> database** : Lista que contem todas as mensagens ***packetType 2*** (CAM) que devem ser encaminhadas pelos nodos;
- **List<Packet> warnings** : Lista que contem todos os warnings ***packetType 3*** emitidos pelo servidor, para que estes sejam encaminhados até ao seu destino.

6 Resultados e Testes

Para ambiente de testes foi utilizado o emulator CORE, onde foi criada uma topologia que permite a mobilidade de nodos, representando os diferentes veículos em circulação, para este efeito foi criado um script de mobilidade (.scen) capaz de automatizar este movimento. Foram também definidas áreas de velocidade máxima, para o servidor conseguir identificar os veículos em excesso de velocidade.

- **Abcissa menor que 220**: Velocidade máxima : 50 Km/h
- **Abcissa entre 220 e 310**: Velocidade máxima : 90 Km/h
- **Abcissa maior que 310**: Velocidade máxima : 120 Km/h

7 Topologia

Na nossa topologia existem Veículos (Nodos pertencentes à wlan 14), 2 RSUs como os respetivos servidores e existe também a rede backbone com o respetivo Servidor Remoto. Em toda a topologia foram, apenas, utilizados endereços IPv6.

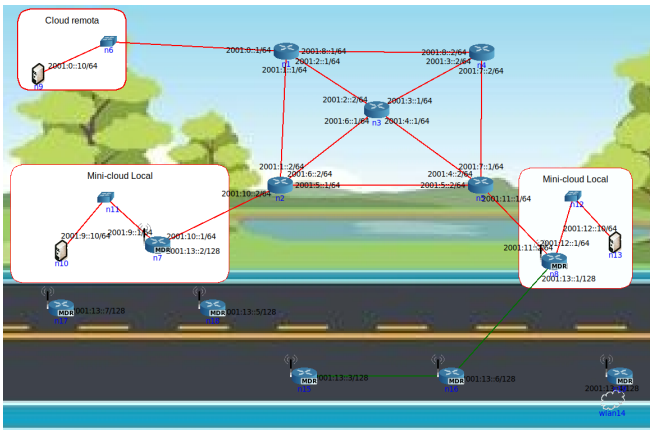

Figura 1. Topologia CORE

8 Testes

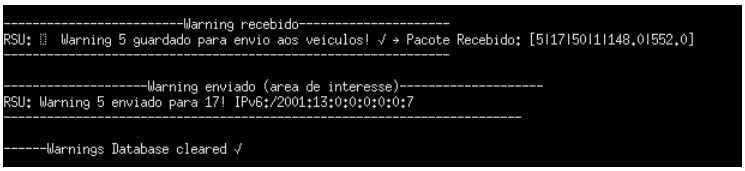

Figura 2. RSU: encaminhamento de warning

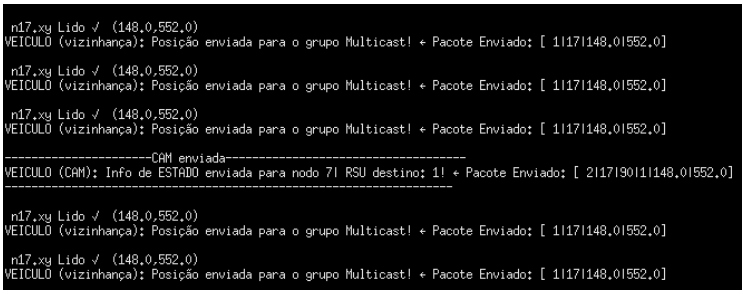

Figura 3. Veiculo:envio de mensagens CAM e Posição

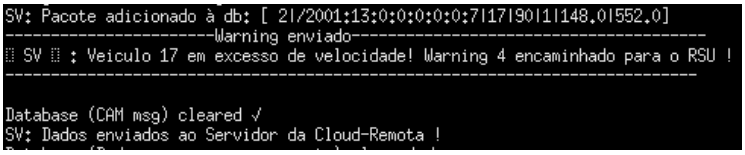

Figura 4. Servidor: Tratamento de mensagens

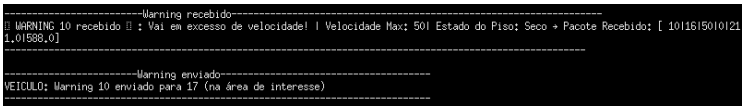

Figura 5. Veiculo 16: Exemplo

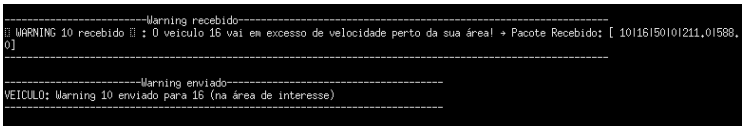

Figura 6. Veiculo 17: Exemplo

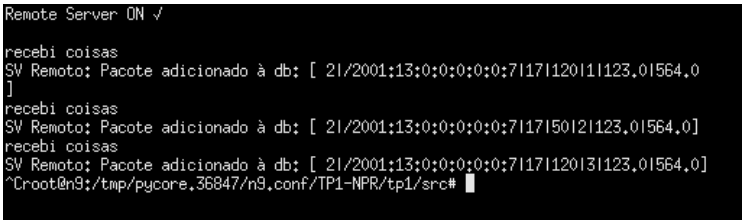

Figura 7. Servidor Remoto (cloud remota)

9 Conclusão e trabalho Futuro

A elaboração deste trabalho prático teve o intuito de nos ajudar a perceber o funcionamento de uma rede veicular e as suas necessidades de segurança rodoviária, desenvolvemos uma rede onde a comunicação entre veículos e as unidades de comunicação fixas permitem uma troca eficiente de informações sobre mobilidade, riscos e situações de perigo.

No entanto, deparamos-nos com algumas dificuldades para perceber o funcionamento em si e como implementar os protocolos e tipos de pacotes necessários para as diferentes situações de teste.

Relativamente às mensagens de warnings e informações sobre o estado dos veículos implementados, estas são apenas um exemplo criados para a nossa topologia e ambiente de teste que poderiam ser expandidas e desenvolvidas em mais detalhe, no caso dos warnings emitidos pelo servidor apenas abordamos o caso relativo a veículos em excesso de velocidade, sendo que outros tipos de informações poderiam ser implementadas no futuro

Para além disso, apesar de já termos implementadas mensagens DENM a notificar os veículos que se encontram numa área em redor de um veiculo que se encontra em excesso de velocidade, podaríamos implementar, por exemplo, o caso de situações de acidente, onde um veiculo enviaria mensagens para os restantes veículos vizinhos que se encontram em seu redor de forma a notifica-los do acontecido. Para além disso, num trabalho futuro poderíamos, também, implementar uma forma mais eficiente para definir o estado do piso, que consistia em dividir a área em certas zonas onde estariam presentes os diferentes estados possíveis, em vez de ser um estado random como implementamos, o valor da velocidade é também um valor random que nunca é alterado de forma "inteligente". Não implementamos a etapa 3 relativa ao serviço alargado em rede veicular de dados nomeados, que consiste em substituir os endereços por nomes para a comunicação da rede veicular.

Contudo, conseguimos cumprir com a implementação de todas as funcionalidades, pondo em pratica tudo o que foi lecionado nas aulas teóricas.

Em conclusão, o desenvolvimento deste protótipo funcional destaca a importância das redes veiculares e da computação de proximidade na melhoria da segurança rodoviária.