Decodificação de logs veiculares CAN numa rede virtual SocketCAN utilizando o pacote *cantools*

Fábio de L. F. Papais

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)

flfp@cin.ufpe.br

Abstract. *CAN networks represent a vital infrastructure in the automotive domain, facilitating communication among various systems present in vehicles. With the advancement of development in the context of CAN networks, virtual solutions such as SocketCAN (integrated into the Linux kernel) have emerged, enabling the simulation of automotive networks without dedicated hardware. This report presents an approach to decode vehicular logs using the cantools package along with SocketCAN, highlighting its functionalities and illustrating the step-by-step process utilized.*

Resumo. *Redes CAN representam uma infraestrutura vital no âmbito automotivo, facilitando a comunicação entre os diversos sistemas presentes nos veículos. Com o avanço do desenvolvimento no contexto de redes CAN, surgiram soluções virtuais como o SocketCAN, integrado ao kernel Linux, permitindo a simulação de redes automotivas sem hardware dedicado. Este relatório apresenta uma abordagem para decodificar logs veiculares utilizando o pacote cantools em conjunto com o SocketCAN, destacando suas funcionalidades e exemplificando o passo-a-passo utilizado.*

1. Motivação e objetivos

Redes CAN (*Controller Area Network*) são sistemas de comunicação fundamentais no contexto automotivo, permitindo a troca de dados e medições entre os diversos sistemas do veículo (como status do motor, frenagem, aceleração e outros sistemas importantes). Dessa forma, a arquitetura robusta e escalável das redes CAN se tornou a escolha padrão na indústria para realizar transmissão de dados e informação no contexto automotivo. As principais formas de utilizar redes CAN são através de softwares que implementam seu próprio protocolo para se comunicar com algum dispositivo físico. Naturalmente, a necessidade de possuir hardware específico para realizar testes e desenvolvimento pode se tornar um empecilho caso o dispositivo esteja indisponível, o custo seja caro, o dispositivo disponível não seja compatível, entre outros. Em vista destas dificuldades, foram desenvolvidas interfaces CAN puramente virtuais, como a implementação do SocketCAN, disponível como parte do kernel Linux. Neste contexto, é possível ”simular”a comunicação CAN através de mensagens na rede e até mesmo replicar mensagens de outras redes automotivas armazenadas previamente a fim de estudar seu comportamento. O objetivo deste trabalho é simular virtualmente a troca de mensagens em redes automotivas utilizando o ambiente SocketCAN no Linux a partir de arquivos de log automotivos disponibilizados pela comunidade *open source*. Como será explicado adiante, as mensagens CAN necessitam ser decodificadas através de banco de dados chamados arquivos DBC; dessa forma, também será necessário usar o pacote de ferramentas *cantools* para decodificar as mensagens utilizando arquivos DBC também estudados pela comunidade.

2. SocketCAN

A comunicação CAN tradicional utilizando um dispositivo Linux consiste no controlador CAN (que se conecta ao transdutor e barramento CAN), no driver utilizado pelo hardware específico e quaisquer aplicativos utilizados pelo usuário. Esse modelo não é necessariamente versátil, já que cada empresa montadora do hardware precisa fornecer seu próprio driver para que o usuário possa se comunicar efetivamente com a rede, o que se torna ainda mais intricado no contexto da indústria automotiva. Visando criar um modelo portátil e versátil, a Volkswagen Research desenvolveu um conjunto de implementações do protocolo CAN para o kernel do Linux, inicialmente chamado de Low Level CAN Framework (LLCF), mas agora chamado de SocketCAN [Kleine-Budde et al. 2012]. Como ilustrado na Figura 1, o SocketCAN permite a interação de mais de uma aplicação com mais de uma rede CAN virtual ao mesmo tempo; isso é possível pois ele é implementado dentro da camada de rede do Linux, tornando, efetivamente, cada rede CAN uma interface de rede diferente no Linux [The Linux Kernel 2024].

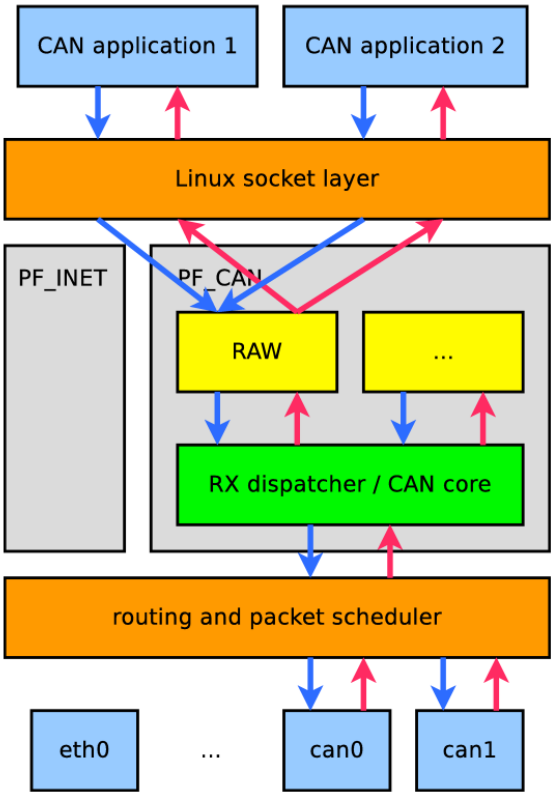

Figura 1. Ilustração do funcionamento do SocketCAN [Kleine-Budde et al. 2012]. Perceba as diferentes redes ”can0”e ”can1”utilizadas dentro da camada de rede do Linux.

Para configurar sua primeira rede SocketCAN no Linux, são necessárias duas etapas condensadas nos três comandos iniciais ilustrados na Figura 2. Inicialmente, é necessário usar o comando *modprobe*, que permite carregar módulos no kernel, para configurar o módulo *vcan* caso ele ainda não esteja pronto para uso. Com o módulo do SocketCAN carregado, podemos criar uma nova interface de rede chamada ”vcan0”no Linux utilizando o comando *ip link*. Ilustrativamente, também é possível utilizar o co-

mando *ip link ls* para visualizar a nova interface do tipo *vcan* criada, em último lugar na lista.

```
❯ modprobe vcan |
  sudo ip link add dev vcan0 type vcan |
  sudo ip link set up vcan0 |
  ip link ls
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp3s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN mode DEFAULT group default qlen 1000
    link/ether b0:25:aa:3c:cd:a0 brd ff:ff:ff:ff:ff:ff
3: wlp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode DORMANT group default qlen 1000
    link/ether e4:5e:37:fa:7f:85 brd ff:ff:ff:ff:ff:ff
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN mode DEFAULT group default
    link/ether 02:42:b3:60:07:66 brd ff:ff:ff:ff:ff:ff
6: vcan0: <NOARP,UP,LOWER_UP> mtu 72 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000
    link/can
```

Figura 2. Comandos utilizados para configurar sua rede SocketCAN e visualizá-la após criada.

Dessa forma, é possível ter uma nova rede CAN chamada ”vcan0”configurada no Linux; com ela, será possível enviar e visualizar frames CAN trafegando. Uma ferramenta muito comum utilizada no contexto do SocketCAN é o pacote *can-utils*, com diversos utilitários que facilitam não somente o envio de mensagens e monitoramento da rede, mas também conversão de arquivos de log, comparação de mensagens, entre outras funcionalidades [Linux-CAN 2024]. Na Figura 3 está um exemplo com o comando *cangen*, que gera mensagens CAN aleatórias, e o comando *candump*, que permite visualizar as mensagens trafegando na rede.

```
fabio in Documentos/Projetos e Cursos/dbc-convert
❯ cangen vcan0

fabio in Documentos/Projetos e Cursos/dbc-convert
❯ candump vcan0
  vcan0  355   [8]  68 CB 2C 1B 2E E4 D1 35
  vcan0  0D2   [8]  7A 26 D0 7E FA 3D 19 62
  vcan0  7C9   [4]  28 29 52 44
  vcan0  47E   [7]  A9 52 70 36 26 1B D3
  vcan0  4C3   [8]  4D 7E CF 37 47 9F 83 38
  vcan0  427   [8]  4E BF 4C 7D E2 F3 91 25
  vcan0  396   [1]  EE
  vcan0  599   [4]  59 49 31 17
  vcan0  400   [2]  1C 15
  vcan0  25F   [6]  CC E2 3F 00 28 62
  vcan0  58B   [4]  C5 37 FB 5C
  vcan0  11B   [8]  30 F5 D6 6C DE 75 B0 05
  vcan0  519   [8]  26 15 34 3E 41 A9 1D 56
  vcan0  42B   [4]  23 9D AF 7B
  vcan0  285   [1]  2C
  vcan0  636   [6]  0C 79 4F 5B 36 22
```

Figura 3. Mensagens CAN trafegando na rede utilizando o pacote *can-utils*

3. Arquivos DBC e o pacote *cantools*

Como visto na Figura 3, mensagens CAN brutas não são necessariamente legíveis. Para ”traduzir”estas mensagens e entender o que cada mensagem significa, são utilizados arquivos Database CAN. Arquivos DBC são arquivos de definição que descrevem a estrutura dos dados que são trocados numa rede CAN, incluindo sinais, mensagens e nós conectados. Esses arquivos DBC funcionam como um ”dicionário”que permite a interpretação e o processamento das informações transmitidas na rede CAN, além de fornecer uma interface padronizada para qualquer software que interaja com ela. Cada regra de decodificação num arquivo descreve uma mensagem CAN, com seu respectivo ID, que

é esperada trafegar na rede. Como é visualizado na Figura 4, cada sinal descrito numa mensagem possui informações detalhadas de como extrair o valor real do dado que está trafegando, com parâmetros como nome, tamanho, tipo (*unsigned* ou *signed*), escala e desvio (para conversão em valor real), unidade, entre outros [CSS Electronics 2024].

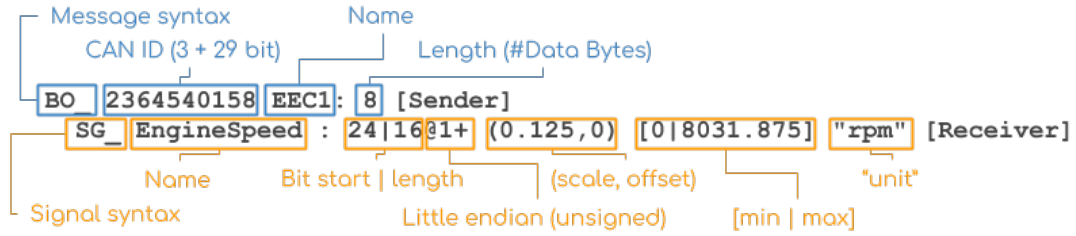

Figura 4. Estrutura geral de uma regra para decodificação de uma mensagem CAN [CSS Electronics 2024]

Considerando a quantidade de mensagens numa rede e a complexidade de decodificar estas mensagens manualmente, existem ferramentas que permitem fazer o processo de decodificação automaticamente, dado um arquivo DBC existente. Uma delas é chamada *cantools*, que não somente decodifica mensagens brutas CAN, mas também interpreta arquivos DBC em diferentes formatos, mostra de forma legível arquivos DBC, entre outras funcionalidades ilustradas na Figura 5 [cantools 2024].

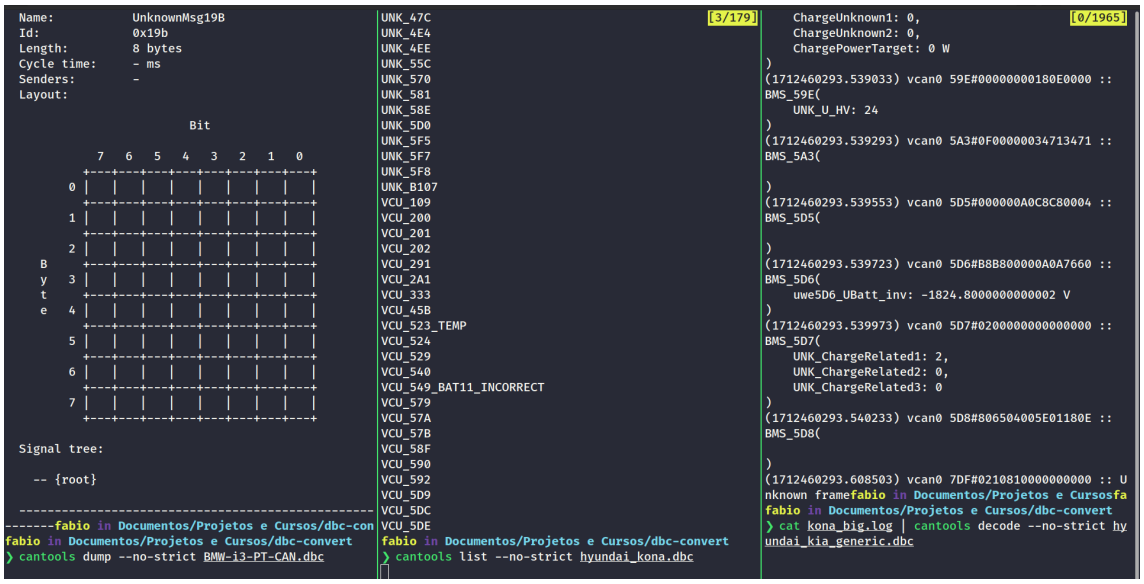

Figura 5. Demonstração de três comandos do pacote *cantools* para visualização legível de arquivos DBC (primeiro e segundo terminal) e decodificação de arquivos de log (terceiro terminal)

4. Decodificando logs automotivos

Com as ferramentas descritas ao longo do relatório, é possível atingir o objetivo do trabalho de executar uma simulação de uma rede automotiva realizando a decodificação das mensagens trafegadas na rede. Para realizar a simulação com dados reais, são necessários logs de redes CAN de redes automotivas reais, muitas vezes disponíveis em repositórios

no *Github*, por exemplo, ou através de outros portais na internet. Mesmo assim, a principal dificuldade está em obter arquivos DBC, pois são guardados sob forte sigilo pelas montadoras e dificilmente disponíveis integralmente. Alguns pesquisadores e entusiastas, porém, se empenham na engenharia reversa das mensagens CAN de redes automotivas específicas na tentativa de identificar o significado de cada mensagem CAN circulando na rede [Buscemi et al. 2023]. Dessa forma, neste trabalho, alguns arquivos DBC disponibilizados pela comunidade foram utilizados para testes, mas não possuem todas as informações que circulam na rede específica. Para testes, foram encontrados arquivos de logs de cinco veículos (BMW i3, Hyundai Kona, Honda Civic, Kia EV6 e logs J1939, utilizados em veículos pesados), porém, por dificuldades na conversão dos diferentes formatos encontrados e incompatibilidades com as ferramentas utilizadas, apenas dois foram de fato testados: Hyundai Kona e Kia EV6. A simulação está ilustrada na Figura 6 e também disponível em formato de vídeo no link `https://drive.google.com/file/d/15q8I5Efn03skwkllH2h7PWEXJ-5zltlh/view?usp=sharing`.

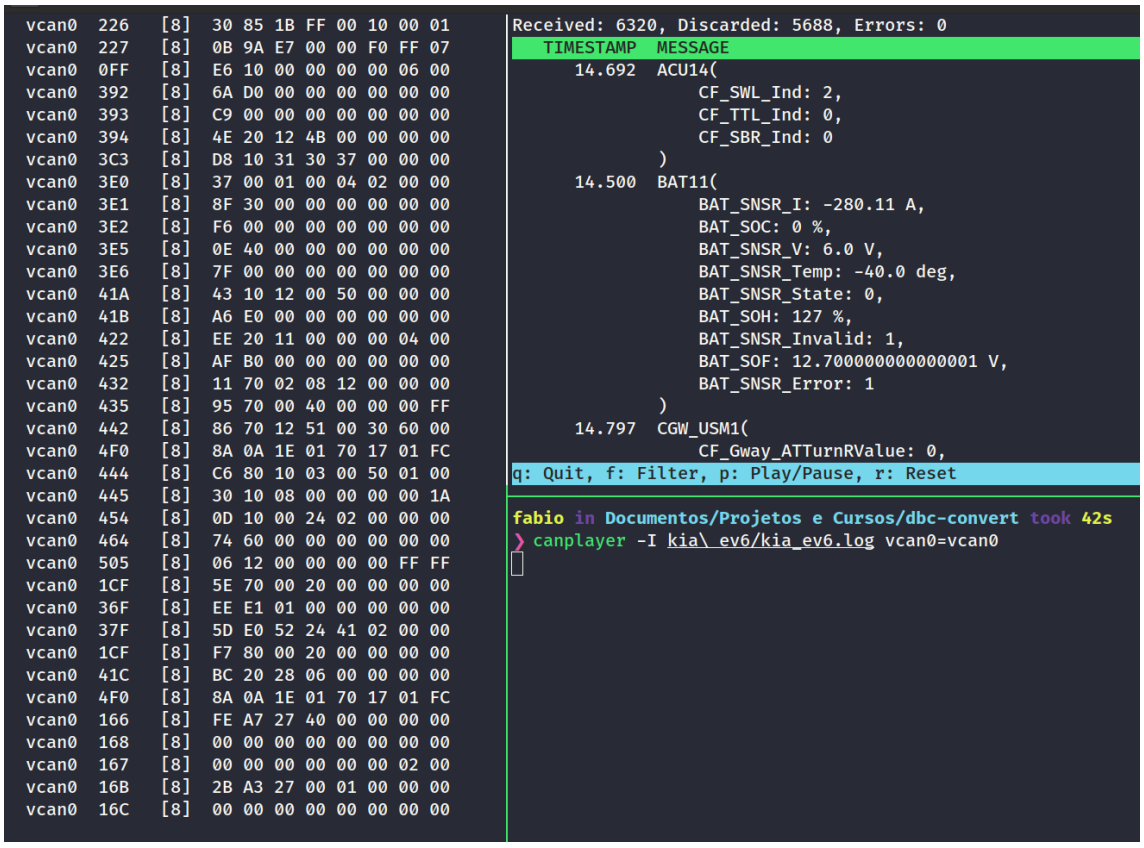

Figura 6. Demonstração de simulação de rede automotiva e decodificação de mensagens CAN utilizando as ferramentas SocketCAN e *cantools*.

A Figura 6 mostra 3 terminais que representam a simulação da rede. O terminal a esquerda está executando o comando ***candump vcan0*** visto anteriormente, para visualização das mensagens CAN trafegando na rede. Já o terminal do canto inferior direito está executando o comando ***canplayer***, que permite introduzir mensagens CAN de um determinado arquivo de log na rede especificada (no caso, a ”vcan0”), respeitando o tempo determinado nas timestamps do arquivo de log. Por fim, o terminal no canto superior direito está executando o comando ***cantools monitor***, que realiza a decodificação

em tempo real dos frames CAN na rede, mostrando a tradução de cada mensagem e sinal. Um aspecto interessante desta simulação (que é possível observar na demonstração em formato de vídeo) é o fato de arquivos DBC mais detalhados darem mais informações sobre as mensagens, enquanto arquivos DBC genéricos fornecerem pouquíssimas traduções de mensagens enviadas na rede, ressaltando a importância da engenharia reversa no estudo de redes CAN.

5. Conclusão

Neste trabalho foi possível conhecer, testar e entender a aplicabilidade de dois conjuntos de ferramentas no contexto de pesquisa e desenvolvimento para redes CAN: a rede virtual SocketCAN e o pacote *cantools*. O objetivo proposto foi concluído, embora ainda existam oportunidades de investigação, como a interpretação dos dados decodificados (para descrever o comportamento dos carros), a interpretação dos dados não decodificados (”quais informações do veículo estão faltando?”), conversão mais precisa de diferentes formatos de logs disponibilizados pela comunidade, entre outras iniciativas. Com certeza, a preparação do relatório e deste trabalho tornou possível colocar em prática o conhecimento teórico obtido sobre redes automotivas, além de conhecer uma nova área de pesquisa e desenvolvimento de perto.

Referências

Buscemi, A., Turcanu, I., Castignani, G., Panchenko, A., Engel, T., and Shin, K. G. (2023). A survey on controller area network reverse engineering. *IEEE Communications Surveys & Tutorials*.

cantools (2024). cantools. Disponível em `https://github.com/cantools/cantools`. Acesso em 08 de abr. de 2024.

CSS Electronics (2024). CAN DBC File Explained - A Simple Intro. Disponível em `https://www.csselectronics.com/pages/can-dbc-file-database-intro`. Acesso em 07 de abr. de 2024.

Kleine-Budde, M. et al. (2012). SocketCAN - the official CAN API of the Linux kernel. In *Proceedings of the 13th International CAN Conference (iCC 2012), Hambach Castle, Germany CiA*, pages 05–17.

Linux-CAN (2024). can-utils. Disponível em `https://github.com/linux-can/can-utils`. Acesso em 07 de abr. de 2024.

The Linux Kernel (2024). SocketCAN - Controller Area Network. Disponível em `https://www.kernel.org/doc/html/latest/networking/can.html`. Acesso em 07 de abr. de 2024.