Linux-Based CAN-Ethernet Gateway

R. Matějka, M. Sojka
Czech Technical University in Prague

February 6, 2015
Version 6400f84-dirty

Contents

1. Assignment

The goal is to implement CAN-Ethernet gateway based on Linux's AF_CAN subsystem with the following features:

1. Both user- and kernel-space implementations will be developed. User-space one will contain only the most basic functionality.
2. Ethernet side will use UDP datagrams to carry the CAN messages. Later it will be extended to support also TCP.
3. Initial version will route all CAN frames to Ethernet side and all received UDP datagrams to CAN messages. Later, filtering capabilities will be added.
4. UDP frames will contain timestamps of the time when the CAN message was received.
5. For kernel-based gateway, implement a user-space configuration tool, similar to `cangw` tool from `can-utils`.

2. Project overview

2.1. Gateway architecture

The software architecture of canethgw is illustrated on figure 2.1. This section describes what can be seen on this figure.

The can-ethernet gateway (canethgw) is comprised of two kernel threads. One thread is serving the can to ethernet routing (can → eth) and the other thread is serving the oposite direction (eth → can).

The gateway uses berkley sockets as network interaces. There are two sockets in the program. One socket is binded to can network and the other one to ethernet. These two sockets are shared by kernel threads.

To get a better notion about canethgw operation a working cycle description is given. The working cycle of the can → eth kernel thread is as follows. The thread is waiting for incoming data on *can* socket. When data are received, they are put to udp packet and are sent to ethernet. There can be more than one recipient in ethernet network. The program holds list of recipients and sends copy of incoming data to all of them. This working cycle description can be analogically applied to eth → can thread.

The gateway is configured over netlink with cegw tool. More on cegw is in section 2.2

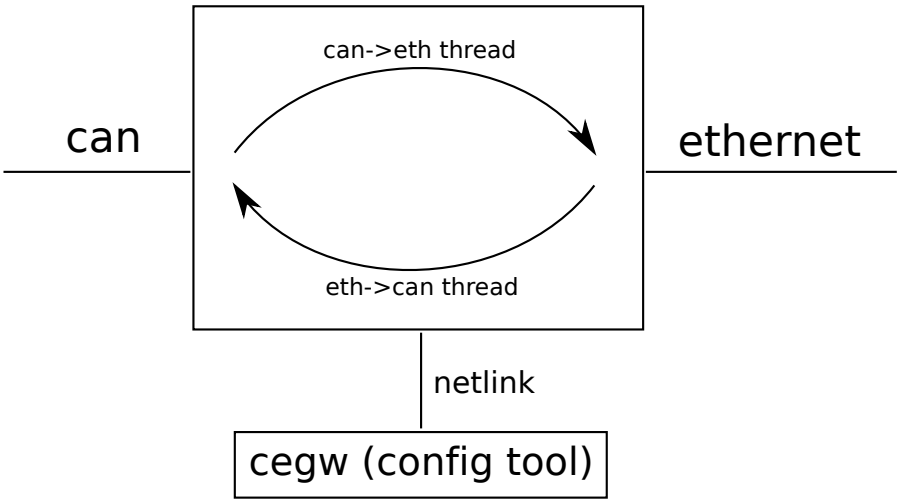

Figure 2.1.: Software architecture of canethgw.

2.2. Associated programs

The project consists not only from canethgw, but also from configuration utility cegw, userspace implementation of the gateway, benchmarking program and other utils. These associated programs are presented below.

cegw Cegw is a configuration utility for canethgw. Its syntax is similar to cangw tool from can-utils. The difference from cangw is that every interface has to be defined with type-prefix. This can be explained well using an example. Figure 2.2 displays commands to do a simple setup of canethgw. This setup sends all can frames received on vcan0 to 127.0.0.1:10502 and all can frames received from 127.0.0.1:10501 to vcan0.

```
# listen for incomping udp at 127.0.0.1 port 10501
cegw --listen udp@127.0.0.1:10501
# all from vcan0 route to 127.0.0.1 port 10502
cegw --add -s can@vcan0 -d udp@127.0.0.1:10502
# all from source address .. route to vcan0
cegw --add -s udp@127.0.0.1:10502 -d can@vcan0
```

Figure 2.2.: Simple setup using cegw.

userspace implementation To measure a performance of canethgw, another implementation of gateway was done in userspace. This implementation resembles the kernel implementation but *poll* is used instead of threads. How to start a simple setup is in figure 2.3.

```
canethgw -s can@vcan0 -d udp@127.0.0.1:10502 -l udp@127.0.0.1:10501
```

Figure 2.3.: Command to run userspace canethgw.

cegwbench This is a benchmarking program. It sends can frames to defined interface and listens on another interface for reception. Then it prints the transmission duration to stdout. How to start simple cegwbench session is in figure 2.4. This program was used for benchmark described in section 4.

```
cegwbench -s udp@127.0.0.1:10501 -d can@vcan0 -n 100 -m oneattime
```

Figure 2.4.: cegwbench example.

cesend Sends a can frame over udp to localhost:10501. This is based on cansend from can-utils and the can frame specification string is the same.

3. Design issues

This chapter encompasses problems which came up during development. This knowledge may be helpful in further research.

The original cangw was designed in modular way, so the first attempt to design canethgw was done on cangw basis. The original cangw works with sk_buff structure and in softirq. However, there are some issues, which complicate applicability of this approach for canethgw. These problems are listed below:

sending udp packet in softirq The original cangw is receiving can frames in softirq. If one wanted to send such a can frame to ethernet with udp/ip packet, he have to avoid sleeping which is not easy. I have not found a way how to send a udp packet using the protocol stack without sleeping. I also think that bypassing protocol stack and reimplementing udp and ip is not a good idea, because it will induce bugs.

receiving udp packet The gateway have to hook somewhere to receive udp or tcp packets. I have not found any convenient method to do this.

These problems led me to search for a different approach. The found solution is based on berkley sockets. Even this approach has some issues, which cause difficuties. These difficulties are listed below. For more complete information about this solution see section 2.1.

socket creation It is not possible to create a *can* socket in a netlink callback. Such an attempt will cause rtnl_lock deadlock. rtnl_lock is being held when netlink callback is processing, but the same lock is needed by register_netdevice_notifier [4] function when *can* socket is being created. This problem can be solved by postponing the creation process.

socket rebinding Once the socket is binded to a particular address it cannot be changed. Rebinding is not possible. The overcome for this is to release the old socket and create a new one.

netlink feedback This problem is tightly linked to the socket creation issue described above. If some operation cannot be done in netlink handler and have to be postponed, the netlink handler can hardly return any feedback about an operation result. That is why the configuration tool (cegw) doesn't return information about an operation result.

terminating thread The termination of kernel thread is not straightforward in this case, because the needs are slightly different than kernel api. The standard kernel thread is working in a cycle waiting for kthread_should_stop(..) to return true. This approach is not possible because the thread can be sleeping in kernel_sendmsg function and therefore not responding to kthread_stop. Waking using wake_process won't work, because no incoming data will cause sleep again. The socket shutdown is used for waking from receiving.

4. Benchmark

To evaluate canethgw a benchmark was conducted. The objective was to compare performance of the gateway implementated in userspace and in the kernel.

The benchmark was designed as a set of four measurements. There is a separete measurement for each of both directions, that is eth→can and can→eth. These both directions are measured with userspace and kernel implementation. In every of four measurements 100 can frames were transceived.

The benchmark was performed on dedicated computer running very bare, single-purpose distribution to avoid external effects. The results show that kernel implementation is approximately 10% faster. Figure 4.1 shows transmission duration of every can frame sent.

Furher information, including benchmark source code, can be found in *bench* folder in repository[1].

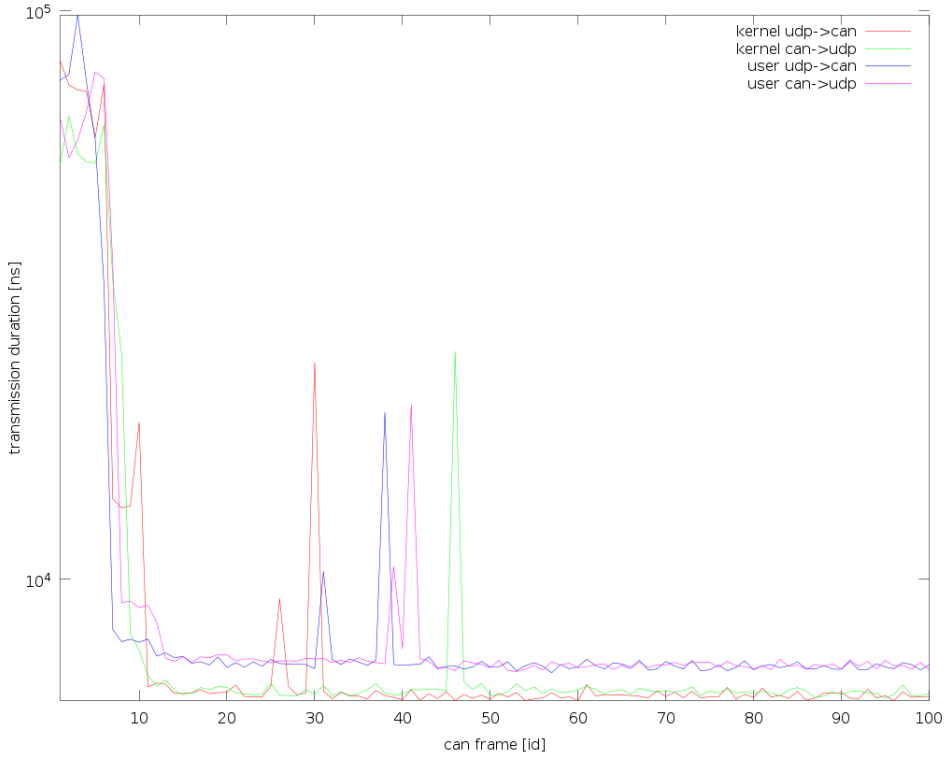

Figure 4.1.: canethgw performance.

A. Repository contents

Repository can be found at [1]. Below is a list of folders and their content.

bench/	Scripts which prepare and perform benchmark on dedicated computer are here.
distro/	Linux distribution used for testing and benchmarking. Busybox and kernel configuration files can be found here.
doc/	Folder with documentation.
can-utils/	CAN utitilities with the userspace `cegw` tool (Git submodule [3]).
linux/	CAN-Ethernet gateway kernel implementation (Git submodule [2]).
test/	Contatins scripts used in debuging to test functionality.
user/	CAN-Ethernet gateway userspace implementation.
utils/cegwbench/	Program used for benchmark.
utils/cesend/	Sends can frame over udp.

Bibliography

[1] Project repository: https://rtime.felk.cvut.cz/gitweb/can-eth-gw.git

[2] Kernel driver: https://rtime.felk.cvut.cz/gitweb/can-eth-gw-linux.git/blob/HEAD:/net/can/canethgw.c

[3] CAN utilities with cegw tool: https://rtime.felk.cvut.cz/gitweb/can-utils.git/shortlog/refs/heads/cegw

[4] register_netdevice_notifier(..): http://lxr.linux.no/linux+v3.5.3/net/core/dev.c#L1351