HOME EXAM 1

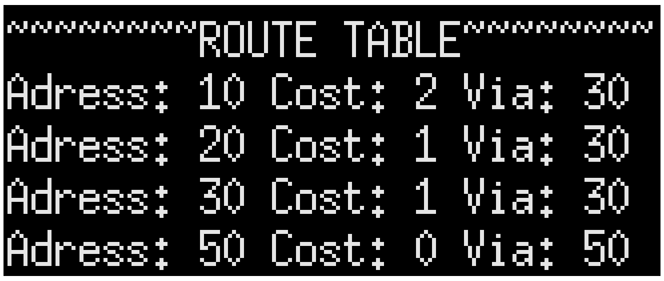
INF3190 – SPRING 2018

15130

The implementation of DVR with Split Horizon:

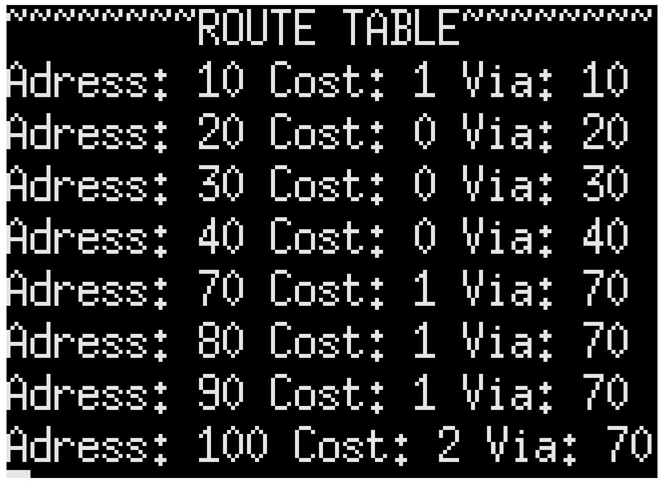
The Routing Server uses DVR with Split Horizon to calculate it's routing table. The way I have implemented DVR is that when the router connects to the daemon, the daemon advertises its local mips with a cost of 0. Then the router broadcast these local mips into the network and the neighbouring daemon sends them to its own routers. And when a router receives information about a route that is either new or better than it previous it adds the new route to its routing table (and increases the given cost by 1). Then it starts a unicast to all its neighbours (a direct neighbour is a route where the cost is 1, and address == via) with all it available routes except those routes who travels through that neighbour. (if you look at figure 1, then when 50 unicast to 30, it only includes the routing info to address 50, as the other routes travel too/through 30).

And that is the gist of my implementation of DVR /w Split Horizon, although I implemented a little bit of poison reverse, when a route gets invalidated due to inactivity from that route, the router then unicast that invalidated route is now outdated to its neighbours, hindering them of spreading an invalid route. I only implemented this to avoid a "count to infinity" loop between B-C-D, which could occur if A or E disconnected. Eliminating a lot of time waiting for the B-C-D loop to reach 16.



```
~~~~~~~~~ROUTE TABLE~~~~~~~~~
Adress: 10 Cost: 2 Via: 30
Adress: 20 Cost: 1 Via: 30
Adress: 30 Cost: 1 Via: 30
Adress: 50 Cost: 0 Via: 50
```

Figur 1Routing Table for node 50 using the topology from assignment 1



```
~~~~~~~~~ROUTE TABLE~~~~~~~~~
Adress: 10 Cost: 1 Via: 10
Adress: 20 Cost: 0 Via: 20
Adress: 30 Cost: 0 Via: 30
Adress: 40 Cost: 0 Via: 40
Adress: 70 Cost: 1 Via: 70
Adress: 80 Cost: 1 Via: 70
Adress: 90 Cost: 1 Via: 70
Adress: 100 Cost: 2 Via: 70
```

Figur 2 Routing Table for B in the home Exam topology

The Design of the Program:

The overall program is designed with the daemon being the foundation of the network layer, while the clients being the application layer and the routing server adding the routing capabilities enabling the daemon to forward messages across the MIP-network. The other programs don’t communicate directly but sends messages via the daemon which sends the message to other daemons.

The daemon starts by creating 3 listen sockets 1 for a client and two for the routing server (routing and forwarding). Both the client and routing socket sends messages to the daemon in the same matter, an address of 1 char, then the message (which is dividable by 4). The forwarding socket receives a destination of 1 char, then returns the next jump destination of 1 char (or 255 if the destination is not in the routing table). It then creates ethernet sockets, then ties the ethernet socket to a given mip address and an interface. (it creates up to number of interfaces or local mips, whichever is smallest). The ehternet socket sends data in form of the ehternet_frame, which has the mac addresses of src and dst, and also has a mip_frame with a header and a message from either a client or router.

It then creates and put all of the created sockets into to an epoll, and then enters a loop, waiting for the epoll_wait() to return an event. When epoll_wait catches an event, it calls epoll_event() to figure out which socket just received data. Then it sends the date to the corresponding socket_handler (eht, routing, forward, and client). After it has handled the data received, it returns to the epoll_wait() loop to wait for the next reception of data.

How to run:

You might have to attempt starting the more than one time due to unbinding some of the sockpaths. It shouldn’t happen, but it might.

The easiest way to compile is to use the supplied makefile.

```
make daemon

make pingc

make pings

make ruter
```

and to run the program you can use the following commands:

```
./daemon [-h] [-d] <socket_application>  <routing_socket> <forward_socket> [mip_adresses]

./pings [-h] <socket_application>

./pingc [-h] <mip_destination> <message>  <socket_application>
```

./ruter [-h] <routing_socket> <forward_socket>

NB: mip_addresses must be in the range of 0-254.

NB: the message length INCLUDING the terminating NULL byte must be a multiple of 4, I.E “HEL” and not “hell”

NB: the help command does not execute the program, but only prints the help text.

How to stop:

Just Ctrl-c inside the daemon, it will stop the client and routing server.

You can also stop both the client and routing server separately, and connect again with another routing server or client (either server or ping)

Program files:

- colours.h
- daemon.c
- makefile
- ping_client.c
- ping_server.c
- ruter.c

Assumptions, peculiarities and other comments:

Valid MIP_Adresses range from 0 – 254, 255 is not a valid mip_adress due to it being both the broadcast mip_adress and padding for the router.

The daemon does not work without the router. I.E you cannot run two daemons on two neighbouring nodes and ping each other. Both needs a router connected.

When testing for invalidating and flushing of routes, I recommend turning the INVALID and FLUSH timeouts in the top of “ruter.c” to something more reasonable than 180 and 240 respectively.