

CSIS0234B Computer and Communication Networks (Class B)

Assignment 3

Tutor: vjwmkwan@csis.hku.hk, Deadline Apr 18, 2004, 5:00pm.

In this assignment, you are given a program which simulates a network of **hosts**. Some of them are connected by **links**, allowing the connected hosts to communicate. Your task is to implement routing for these hosts so that they become **routers**—every router should be able to talk to any other router in the network as long as there is a path of links between them.

This is a group assignment allowing at most 2 students in each group.

1. The host simulation

The hosts are simulated by Unix processes running in one computer. Each of them receives links to other hosts from an external **datalink control** program `dlcontrol`, available from the course web page (source file `dlcontrol.cc`, `dlcontrol.hh`). A simple host program is also available. A sample session:

(From first terminal)

```
> ./dlcontrol
```

```
Control: help
```

```
Available: showhost (sh), showlink (sl), addlink (a), dellink (d),  
stoplink (s), resumelink (r), help (h), quit (q)
```

(From second terminal)

```
> ./netclient
```

```
Host 0 started.
```

```
0> help
```

```
Available commands: send, help, quit
```

(From third terminal)

```
> ./netclient
```

```
Host 1 started.
```

```
1>
```

(From first terminal)

```
Control: addlink
```

```
From host: 0
```

```
To host: 1
```

```
Link from 0 to 1 created.
```

(From second terminal)

```
0> send
```

```
Input a line to send: Hello, world!
```

```
Sent!
```

(Terminal of host 1 shows the message "Hello, world!".)

The links are implemented by *streaming pipes*, a type of pipes that allow bidirectional data flow. A host **can be stopped for arbitrarily long** (using Control-Z), and during this time it does not respond to messages and will not send messages to others, until it is resumed (e.g., using the `fg` command). During the period all sent messages will be lost.

The client program contains a datalink layer which sends data to and receive data from such links (`datalink.cc`, `datalink.hh`). By sending and receiving bytes in the streaming pipe, the datalink supports sending and receiving short frames (maximum is around 1000 bytes) over the link. Frames will be dropped if the streaming pipe buffer is full (checksum and framing of the

datalink layer makes sure that the whole frame is either received or dropped completely). You should not modify the code there.

2. Requirements

The network layer (`netclient.cc`) is currently very rudimentary and can only send data to the first data link that it knows about. Your task is to **modify these files** to implement **distance vector** routing into the system, so that it is possible to send a message from any router to any (directly or indirectly) connected router.

Distance vector should be used for route finding. The network layer should recognize the following type of packets:

- **Data** packets. They are used to send data from one router to another. **Forwarding** is done with the routing tables of each router, which should be built automatically using the information obtained by the DV packets. It is possible to have routing loops, so your forwarding code should keep a time-to-live (TTL): a packet going through **more than 60 routers** should be **dropped**, although there needs to be no mechanism like ICMP that allow a routing problem in another router to be reported. Naturally, the destination receiving the packet should know where the packet originate from. You should write a new command, `sendto`, in the network client to replace the original `send` command to generate data packets.
- **DV** (Distance Vector) packets. Each router should send DV packets to all its neighbours every 5 seconds. This achieves three purposes: tell the peer its “host number”, notify the peer that it is still alive, and tell the peer its own routing table. Upon receiving a DV packet, the **routing table** should be **updated** with the received distance vector. You should write a new command `showroutes` to show the current routing table, and `showvectors` to show the currently stored distance vectors from neighbours.

Distance vectors that are older than around 15 seconds should be dropped, and you should assume that the link between the two routers is down. This should also happen if the data link layer explicitly notified the network layer that the link is removed. Unless the datalink layer tells you that the link is removed, you should continue to send your distance vectors, so that if the link goes up again later, the routing table can be updated to use the link.

The routing table should be built according to the distance vector algorithm. All links should have the unit cost; and **15 should be used as the infinity value**. **Split horizon with poisoned reverse** (textbook page 529) should be used to reduce the effect of the count-to-infinity problem.

3. Hints

We suggest that you follow the steps below to complete the assignment.

1. Read the header files `event.hh` and `datalink.hh`, as well as the text file `README.netclient`, to understand the provided programming interfaces.
2. Read the original `netclient.cc` carefully. Understand how they work. You will need to send and receive data in the same way that `netclient.cc` is doing now.
3. Design the basic packet format, so that DV packets and data packets can be differentiated. When doing the design, try to minimize the size of each produced packet, and reduce

routing overhead. Note that since all the “routers” actually runs in the same computer, you don’t need to deal with endians or other representation issues.

4. Let the network layer capture a periodic event of 5 seconds, and send a simplified DV packet to all neighbours at each interval. Use the packet to tell the peer what is your host number (available using the *host_nr* global variable).
5. Add code to receive and record the DV packets. Keep a counter for all recorded DV packets: when a packet is received it should have the value 4. Modify the handler of the 5 seconds periodic event so that all these counters are reduced by 1; and a distance vector is removed if its counter is reduced to 0 (the effect is that the vector expires no earlier than 15 seconds after it is received, and no later than 20 seconds after it is received). Add a `showvectors` command; make sure distance vectors are received and removed correctly. Don’t forget to test the effect of other controls of the datalinks, e.g., from the `dlcontrol` program, or by suspending and resuming some of the routers.
6. Implement a routing table, and write code to put the routing table information to the DV packets. Modify the packet reception code to update the routing table according to the DV packet received. Add a `showroutes` command; make sure the routing tables are built correctly.
7. Implement the `sendto` command for sending data packets. The packet must have at least a type (DATA), source, destination and TTL field. The packet is routed according to the routing table of individual routers.

Note that you don’t need to understand `event.cc`, `datalink.cc`, `streampipe*` and `dlcontrol.cc` for this assignment. But you really feel curious about it, the `README-internals` file might help you understanding their design.

4. Sample program

A sample program is available, which (more or less) implements the requirements here. You can try the program in virtue: run the program by `~c0234b/bin/dlcontrol` and `~c0234b/bin/netclient` in different terminals. You should treat it as clarifying ideas rather than the definitive guide to the features.

There are also two programs in the `testprogs/` directory, which is used to test the event architecture and the streaming pipes. You might treat it as a simple “hello-world” type program for them.