

# Programming Project, CS 6390, Dr. Cobb, Spring 05

8th March 2005

## 1 Preface

The language you choose is irrelevant to me.

However, I expect your code to run in the Unix systems here at UTD. Hence, you better make sure there is a compiler for your language in the Unix systems here at UTD.

To turn in your program, you will send us the source code of your program. We will compile it and run it.

Since your source code will likely consist of several files, you will zip them all together (PLEASE DO NOT USE TAR NOR GZIP WE HAVE HAD PROBLEMS IN THE PAST WITH THESE). Among these files there must be a README file that contains your name and ID and DETAILED instructions to compile your code in the Unix system. If we cannot compile it because your instructions are wrong it will receive the same grade as a program that has compilation errors.

## 2 General description

Your task is to have a set of processes communicate with each other via files (no sockets, just plain old files). The purpose of these processes is to implement a weird multicast protocol I came up with.

Each node in our network will be implemented via a Unix process. There will be three different types of processes, hosts, routers, and the controller.

The controller is needed to handle I/O, and would not exist in a real-network. It is needed since we will simulate LANS and I do not have a broadcast mechanism.

Some hosts will send multicast messages, and others will receive multicast messages. For simplicity we assume there exists only a single group.

Routers will learn how to reach each LAN using a protocol that is neither distance vector nor link-state routing (it is a simple form of link-state routing).

Each process should run for 100 SECONDS and should exit on its own after this time elapses.

## 3 LAN and Node ID's

- ID's are a single ASCII digit 0 .. 9
- Each host will have a unique single-digit ID.

- Each router will also have a unique single-digit ID.
- Note that a host and a router can have the same ID. That is fine, since due to the format of the messages below that will not be a problem.
- Each LAN will have a unique single-digit ID.

## 4 Connectivity

- Each host is attached to only a single lan.
- There can be at most 10 lans.
- Each router is attached to at most 10 lans.
- There can be at most 10 hosts and at most 10 routers.

## 5 Process arguments

A host will receive the following arguments in the Unix command line:

```
host-id lan-id type time-to-start period
```

**Host-ID** This is the single-digit ID of the host

**lan-ID** This is the single-digit ID of the lan attached to the host

**type** This is either "sender" or "receiver", i.e. the host is either a sender of the group or a receiver of the group

**time-to-start** time (in seconds, two digits) after which the host may attempt to send its first data message to the group (see below) if it is a sender. Time-to-start is not present if the host is a receiver

**period** is a two-digit integer saying how many seconds should elapse before the host sends the next data message (if it is a sender). The period argument is not present if the host is a receiver.

For example, a host process can be executed from the Unix command line in the following manner (assuming the executable of the host program is simply called host)

```
host 10 2 sender 30 10 &
```

This creates a process from the executable file host and runs it in the background (i.e., batch mode, that is what the & is for). The ID of the host process is 10, its LAN is 2, it is a sender process, and it will begin to send data messages at time 30 and will send the next data message every 10 seconds.

A router process receives the following arguments

```
router-id lan-ID lan-ID lan-ID ...
```

**router-ID** This is a single-digit unique router ID

**lan-ID** Each of lan-ID is the ID of a lan to which the router is attached to

For example, a router process can be called from the Unix command line in the following manner (assuming the executable of the program is simply called router)

```
router 5 1 5 2 9 &
```

This creates a process from the executable file router and runs it in the background. The id of the router is 5, and it is attached to LANS 1, 5, 2, and 9.

The controller process receives the following arguments

```
"host" id id id . . . id "router" id id . . . id "lan" id id . . . id
```

**"host"** is simply the string "host"

**id id . . .** These ID's that follow is a list of all the ID's of all the hosts in the network

**"router"** is simply the string "router"

**the** ID's that follow is a list of all the ID's of all the routers in the network

**"lan"** is simply the string "lan"

**id id . . .** These ID's that follow is a list of all the ID's of all the LANS in the network

For example, the (because there is only one) controller process can be called from the Unix command line in the following manner (assuming the executable of the controller program is simply called controller)

```
controller host 9 7 1 router 5 8 11 lan 1 5 2 9 &
```

## 6 Emulation of LANS using files

Each host will open a file for writing, called houtX, where X is the ID of the host. Here, the host will write the messages that it wants to send over its LAN (note that the host has only one LAN). Each message will be written in a separate line of the file.

Each router will also open a file called routX, where X is the ID of the router. Here, the host will write the messages that it wants to send over any of the LANS that it is connected to. Each message will be written in a separate line of the file. Furthermore, if the router sends the "same" message over more than one LAN, the message has to be written MULTIPLE times into the file, one per LAN over which the message is to be sent.

Note that routers and hosts only APPEND to their out files, thus, messages are NOT over-written. (This leaves a nice record of the network activity for you to see when your program runs)

The controller will have a list of all the hosts, routers, and LANS (see above).

Each LAN will have a file, lanX, where X is the ID of the lan. Every message that a host/router sends to lan X is copied (APPENDED) by the controller into file lanX.

ONCE EVERY SECOND, the controller will check the out file of each host/router, and if there are new messages in this file, it will copy (APPEND) the message to the corresponding file of the LAN

Note that the controller can have up to 30 files open (the hout of every host, the rout of every router and the lanX of every LAN). Since this may be too much (see if you can open 30 files at once I think you can but check it out) the controller may have to open some files, manipulate them, REMEMBER where it left off in the file, and then close the file.

## 7 Host behavior

A host sends only two types of messages. One of them reports to the routers in its lan that it is a receiver. The other one is a multicast message if the host is a sender.

### 7.1 Sender

If a host is a sender, it must wait until its time-to-start. Then it begins to send messages of the following form every "period" seconds

```
smcast host-lan-id host-lan-id host-id seq-no
```

**smcast** This is simply the string "smdata" (which stands for multicast data).

**host-lan-id** the lan-id of the host (again).

**host-id** The one-digit host id of the sender

**seq-no** This is a two digit number that grows by one everytime the sender sends a new mcast message

Note that the host-lan-id appears twice. There is a reason for this. Routers can also send smcast messages (see below) but they will not have the host-lan-id appear twice (see below on routers).

The host will send this message when it starts up, and after that it will send the message again every "period" seconds.

In order for things to work in a sensible manner, the time-to-start of each host should be big enough to ensure that the routing protocol of the routers has stabilized, i.e., routes to each lan are stable. I will make sure this is the case in the test cases I check.

## 8 Receiver

A host advertises that it is a receiver by sending the following message

```
receiver lan-id
```

**lan-id** is the lan of the host

**receiver** is just the string "receiver".

The receiving host will open an output file whose name is of the following form received-X, where X is the single digit id of the receiver.

ONCE PER SECOND, if the host is a receiver, the host will check for new messages in file lanX, where X is the lan of the host.

If a host sees an "smcast" message over the lan to which it is attached to, then it will copy the entire message into the output file mentioned above.

## 9 Router Behavior

### 9.1 Learning the topology

The protocol via which each router learns the topology of the system is a mixture of a modified version of a link-state routing protocol (it has a “flavor” of a distance-vector protocol, sort of like a mixture of both).

Each router will send to its neighbors a “tree”. This tree is a spanning tree of the entire network, and basically indicates the path that the router believes is the shortest path to each destination. This information is NOT broadcast to the entire network, it is just exchanged between neighbors (which is why it has a flavor of a “distance-vector” protocol which does not do broadcast, just exchange between neighbors, however here we exchange a “tree” and not a “vector”).

Hence, each router receives a spanning tree from each of its neighbors. The router keeps in memory the latest spanning tree received from each of its neighbors. To create its own spanning tree, the router combines the edges of all the spanning trees received from its neighbors (thus obtaining a graph, which may have cycles), and then applies Dijkstra’s algorithm to obtain a shortest spanning tree that reaches all LANs (at least that reaches all LANs reachable via its neighbors)

Periodically (once every 5 seconds), the router sends the following message over each of its outgoing LANs

```
route router-lan-id router-id SPANNING-TREE
```

where

**route** is just the string “route”

**router-lan-id** is the id of the lan attached to the router over which this message is being sent

**router-id** is the id of the router sending this message

**SPANNING-TREE** is a sequence of edges described in more detail below.

Edges on the spanning tree are either of the form (router:router-id, lan:lan-id) or of the form (lan:lan-id, router:router-id), where “router:” and “lan:” are just ascii strings, and lan-id is the id of a LAN that is attached to the router whose id is router-id.

For example, if router 5 generates a “route” message whose SPANNING-TREE is of the following form

```
(router:5, lan:2) (lan:2, router:3) (router:3, lan:8) (lan:8, router:1) (router:1, lan:9)
```

That means that there exist the following path

```
router 5 -> lan 2 -> router 3 -> lan 8 -> router 1 -> lan 9
```

which router 5 can use to reach lans 2, 8, and 9. In this case the tree is just a simple path (but a tree nonetheless).

In addition, a router must inform its neighbors of any receivers in any of the lans in its tree. Therefore, if the router knows there are receivers in a lan, instead of using an edge (router:router-id, lan:lan-id) it uses an edge of the form [router:router-id, lan:lan-id]. Thus, if above instead of (router:3, lan:8) we use instead [router:3,lan:8] we are indicating that there is a receiver in lan 8.

## 9.2 Routing Multicast Messages

Assume a LAN has only a single router (this is relaxed later below). It then sees that a sender host sent an smcast message. In order for this message to be received by the receivers, we will use a form of source routing. The router will pick up the message, then compute the spanning tree that reaches all receivers, and include this spanning tree in a new multicast message that will reach all lans with receivers.

How is this spanning tree computed? The router knows a spanning tree of the entire network (as described above), the spanning tree that reaches all receivers is a subset of this spanning tree, with only those edges necessary to reach all LANS that have a receiver.

### 9.2.1 Generating the embedded tree

Upon receiving an smcast message of the following form,

```
smcast host-lan-id host-lan-id host-id seq-no
```

the router will generate a message of the following form for each one of its neighboring lans that lead to a receiver

```
rmcast router-lan-id router-id sender-host-id seq-no SPANNING-TREE
```

where

**rmcast** is just the string “rmcast”

**router-lan-id** is the id of the lan attached to the router over which this message is being sent

**router-id** is the id of the router sending this message

**sender-host-id** is the id of the original sender host of the message

**seq-no** is the seq number of the sender (from the smcast message)

**SPANNING-TREE** is a sequence of edges (only edges with () delimiters, no need to use [] delimiters) that reaches all lans with receivers.

### 9.2.2 Routing with the embedded tree

A router that sees an rmcast over its lan will check the SPANNING tree in the message. If itself is located in this spanning tree, it will forward the message over the next-hop lans indicated in the spanning tree.

Note, however, that the “router-lan-id” field will change on a per-hop basis. Other than this, no other field in the message needs to change as it goes from one hop to the next.

Receivers do not care about “rmcast” messages, they care only about “smcast” messages. Hence, whenever a router forwards an rmcast message over a LAN where receivers are present, it also forwards an smcast message. The difference between an smcast generated by the router and one by the source is that the router has the following format.

```
smcast router-lan-id host-lan-id host-id seq-no
```

where router-lan-id is the lan-id over which this message is currently being sent. Note that since router-lan-id is different from host-lan-id, then the routers in this lan will ignore the message, and only the receivers will care about it.

### 9.2.3 Choosing the parent router

In the event that there are multiple routers in the same LAN where the source node is located, then the router with the least id will respond to the smcast message and generate rmcast messages,

THE END :-)