CS 435: Cloud Networking

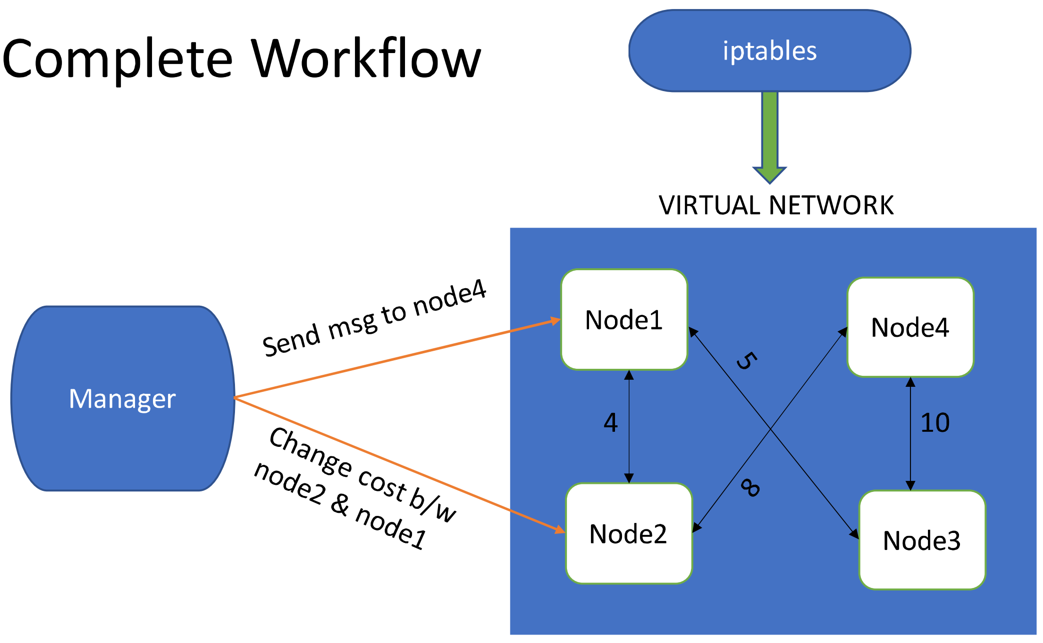
Programming Assignment 2

Unicast Routing

Agenda

- What should be done?
- How it should be done?
  - Environment Setup
  - Implementation
  - Testing
  - Grading
  - Tips

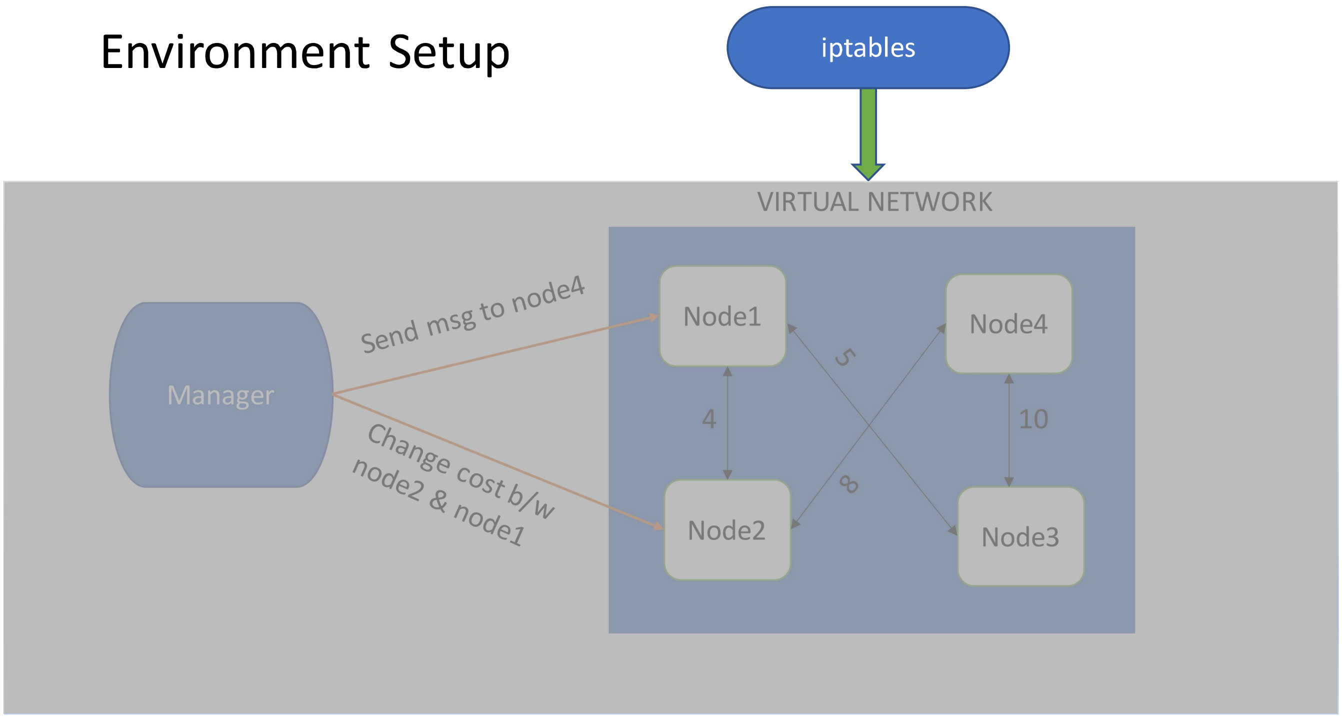
Complete Workflow



What should be done?

- Implement a shortest path routing algorithm
  - Either Distance Vector/Path Vector **OR** Link State Routing
  - Route packets (messages) from source to destination
  - Update routes based on configuration changes – link cost changes, failures
- What will be the routers/nodes?
  - Individual processes bound to a virtual interface & assigned a Virtual IP address
- How links are set up between routers/nodes
  - Virtual network interfaces using Linux iptables
- Where will be doing this?
  - Should be done inside VM from MP1 (Ubuntu 20.04 LTS)

Environment Setup

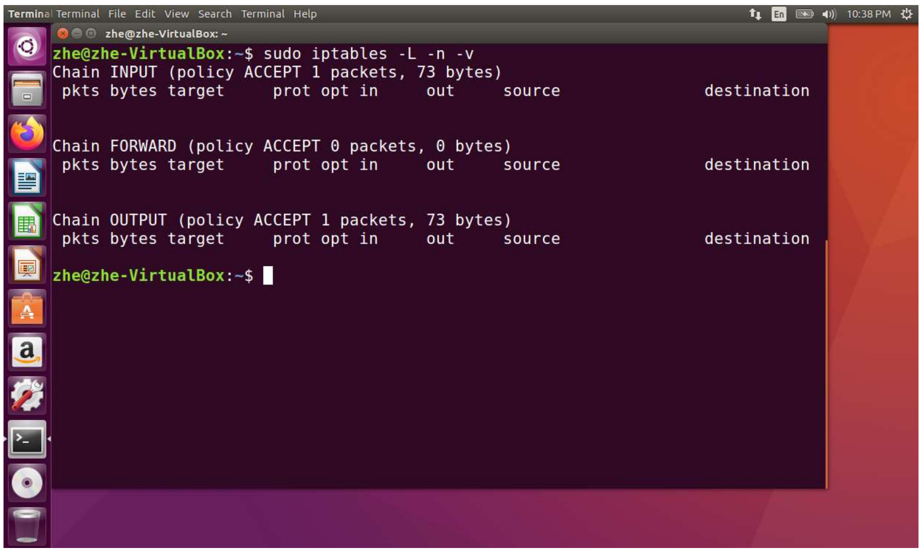


Environment Setup - iptables

- Wikipedia: “iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall.”
- References:
  - https://linux.die.net/man/8/iptables
  - https://www.thegeekstuff.com/2011/06/iptables-rules-examples/

Environment Setup - iptables

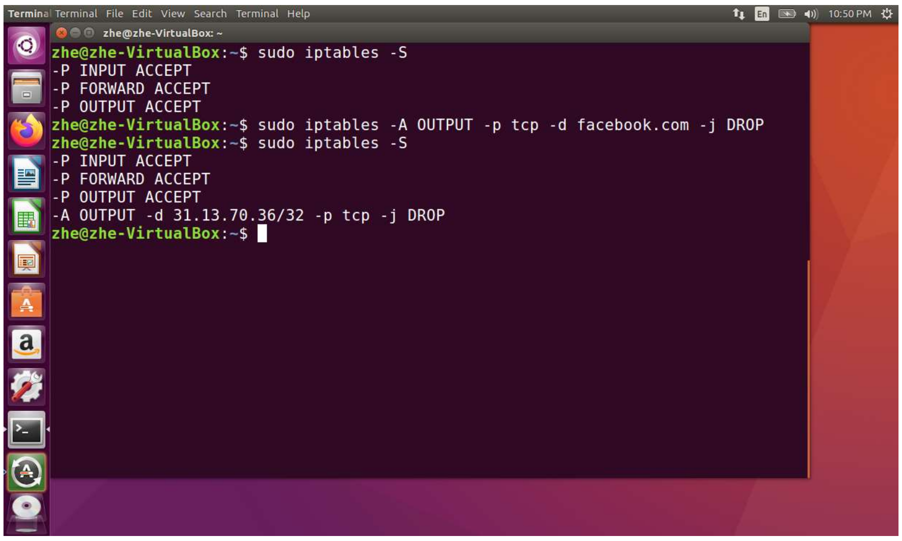
- Example: list incoming/outgoing traffic



```
zhe@zhe-VirtualBox:~$ sudo iptables -L -n -v
Chain INPUT (policy ACCEPT 1 packets, 73 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 1 packets, 73 bytes)
 pkts bytes target     prot opt in     out     source               destination

zhe@zhe-VirtualBox:~$
```

Environment Setup - iptables

- Example: show rules – block outgoing traffic to FB – show rules



```
zhe@zhe-VirtualBox:~$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
zhe@zhe-VirtualBox:~$ sudo iptables -A OUTPUT -p tcp -d facebook.com -j DROP
zhe@zhe-VirtualBox:~$ sudo iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A OUTPUT -d 31.13.70.36/32 -p tcp -j DROP
zhe@zhe-VirtualBox:~$
```

Environment Setup - iptables

- We will use Linux iptables to setup & enforce communication rules in the virtual network
  - A virtual interface (e.g. eth0:1, eth0:2, etc.) & a virtual IP address will be created & assigned to each node/process in the system
  - A node with ID n gets address 10.1.1.n. IDs from 0 through and including 255 are valid.
  - iptables rules will be applied to restrict which of these addresses can talk to which others. 10.1.1.30 and 10.1.1.0 can talk to each other if and only if they are neighbors in the emulated topology.
- To create a link

```
sudo iptables -I OUTPUT -s <node1_ip_addr> -d <node2_ip_addr> -j ACCEPT &&
sudo iptables -I OUTPUT -s <node2_ip_addr> -d <node1_ip_addr> -j ACCEPT
```

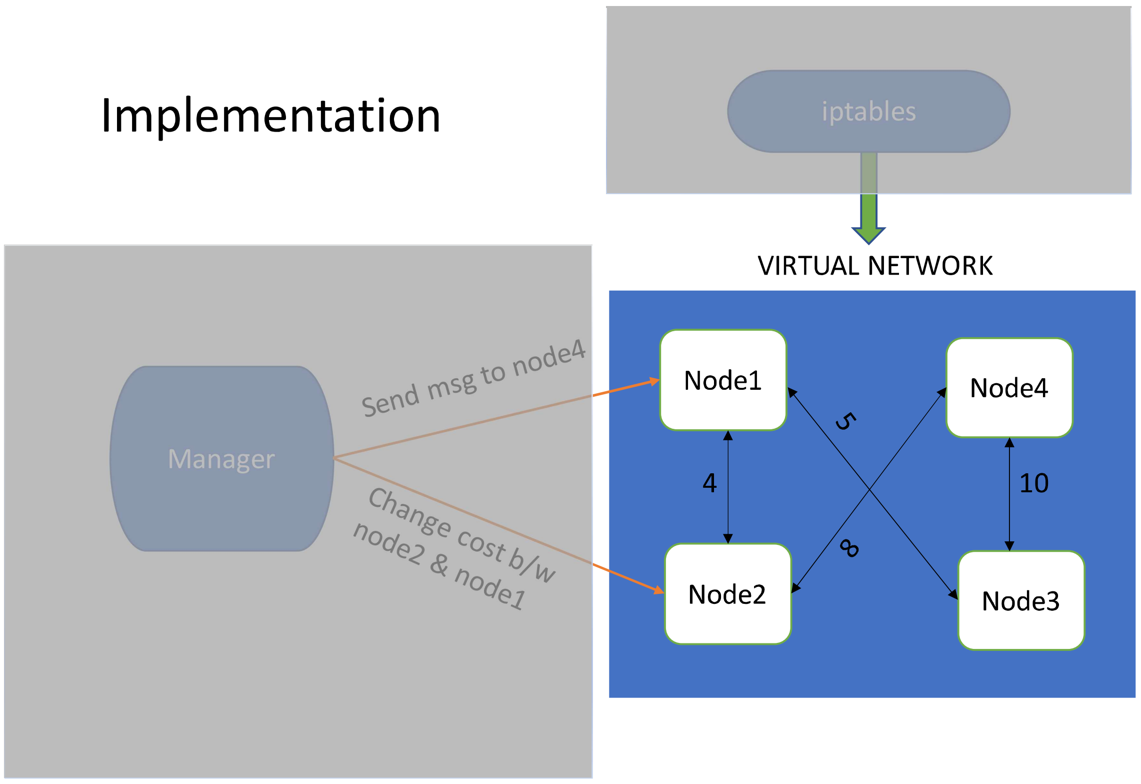
- To take down a link

```
sudo iptables -D OUTPUT -s <node1_ip_addr> -d <node2_ip_addr> -j ACCEPT &&
sudo iptables -D OUTPUT -s <node2_ip_addr> -d <node1_ip_addr> -j ACCEPT
```

Environment Setup (cont.)

- Use make_topology.pl file provided in the assignment files
    - run "`perl make_topology.pl test_topology.txt`"
    - The topology file contains a pair of node ids on each line, which will become neighbors in the virtual topology.
- In make_topology.pl, you must replace **eth0** with the name of the VM's network interface.
    - Accessed by running ifconfig in the terminal (e.g., enp0s3 or a variant of it)

Implementation



Implementation - Your Router’s Key Tasks

- Control Plane
  - Detect the state of links using heartbeat.
  - Run the link state (LS) or distance/path vector (DV/PV) protocols to communicate between routers, find shortest paths and construct forwarding table
- Data Plane
  - Receive the “send” command from Manager and follows its instructions.
  - Transmit/Relay the message in “send” command from source to destination using the forwarding table at each hop, and write the activities to a log file.

Implementation - Your code

- Starter code in the assignment files (main.c & monitor_neighbors.h).
- Running the code:

```
./<router_binary> nodeid initial_costs_file logfile
```

  - *nodeid* is the virtual interface number for this node.
  - *initial_costs_file* is the cost file.
    - On each line, the node id and cost is provided.
    - Missing entry means cost is 1.
    - Neighbor link might not exist at that time.
  - *logfile* is where your program writes the events it was a part of. All the nodes must make a log file even if no event happens.

Implementation – Neighbor discovery

- Every node periodically sends heartbeats to all the nodes (ping every node). This code is given.
- If node a receives a heartbeat from node b, node a knows node b is its neighbor, puts node b into its neighbor list.
- If node a doesn’t receive HBs from node b for some time, node a learns the link is down, removes node b from neighbor list.

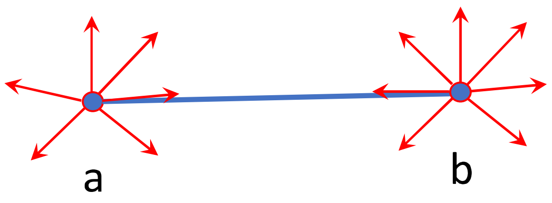
Implementation - Heartbeat

- Use UDP heartbeat to detect link changes.
- Each node keeps periodically broadcasting heartbeat to its neighbors.
- Separate thread in starter code.

```
pthread_t announcerThread;
pthread_create(&announcerThread, 0, announceToNeighbors,
(void*)0);

…
void* announceToNeighbors(void* unusedParam){
          …
          while(1)  {
                    …
          }
}
```

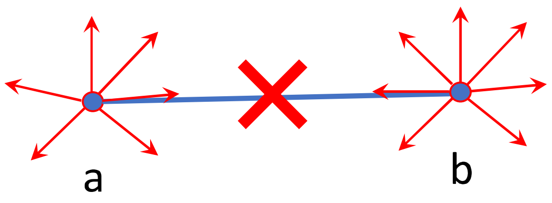
Implementation - Heartbeat

- Link a<->b is up:
- Node a and b receives each other’s heartbeats.



Implementation - Heartbeat

- Now, suppose link a<->b is down:
- Node a and b will detect this event from heartbeat times out.

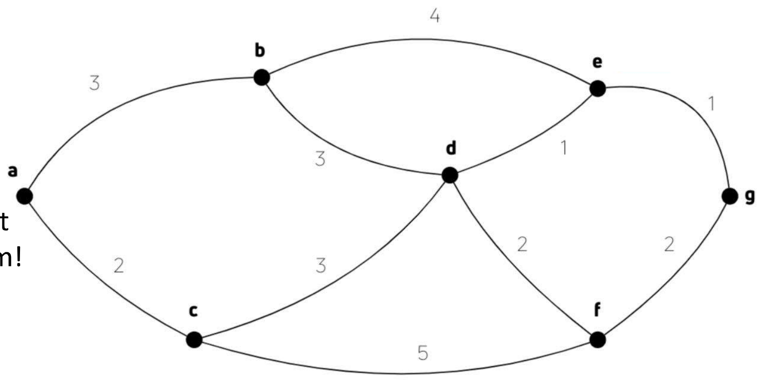


Implementation – Link Cost

- First, a node reads values from the initial cost file
  - Can have costs between nodes where the link is currently down!!
  - Cost is 1 for a neighbor which is not specified

E.g. the cost file input to node a:
b 3
c 2
d 100
e 40
…

Node a and d,e are not neighbors. Ignore them!



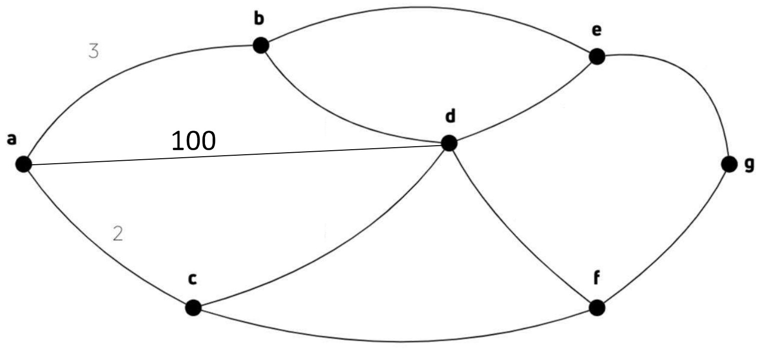
Implementation – Link Cost

- Now, suppose the link b/w a<->d is up.
  - Cost will be 100 which is last recorded

…

E.g. the cost file input to node A:
b 3
c 2
d 100
e 40
…

Node A and E are not neighbors. Ignore it!



Link a<->d is up

Implementation - Logging

- Send a packet (source node)
  - sending packet dest [nodeid] nexthop [nodeid] message [text text]

  e.g. sending packet dest 11 message hello there!
- Forwarded a packet (intermediate nodes)
  - forward packet dest [nodeid] nexthop [nodeid] message [text text]

  e.g. forward packet dest 56 nexthop 11 message Message1
- Received a packet (destination node)
  - receive packet message [text text text]

  e.g. receive packet message Message2!
- Unreachable Node (source node)
  - unreachable dest [nodeid]

  e.g. unreachable dest 12

  Note: It must be exact as described since it is used by autograder. Be careful!!

Testing

How do you go about testing the code you have developed?

1st option, use the test topology in mp2_code/ files

- Setup the test topology
- Run the code binary for multiple nodes
  - Open multiple terminals.
    - One terminal per node/router
    - Command to start router process:

    ```
    ./<router_binary> mynodeid initial_costs_file logfile
    ```
- Use manager from another terminal to send various commands
  - Command to execute a manager command:

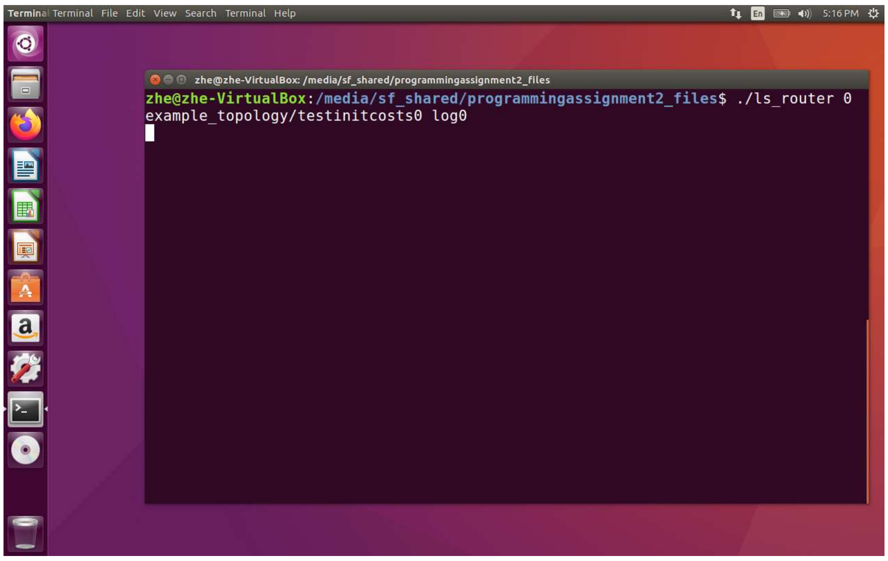
  ```
  ./<manager_binary> destnode command [args]
  ```

2nd option, use the autograder

- When you've done testing with 1st option, use autograder to test your code automatically under more complicated network topologies

Testing – ScreenShot1

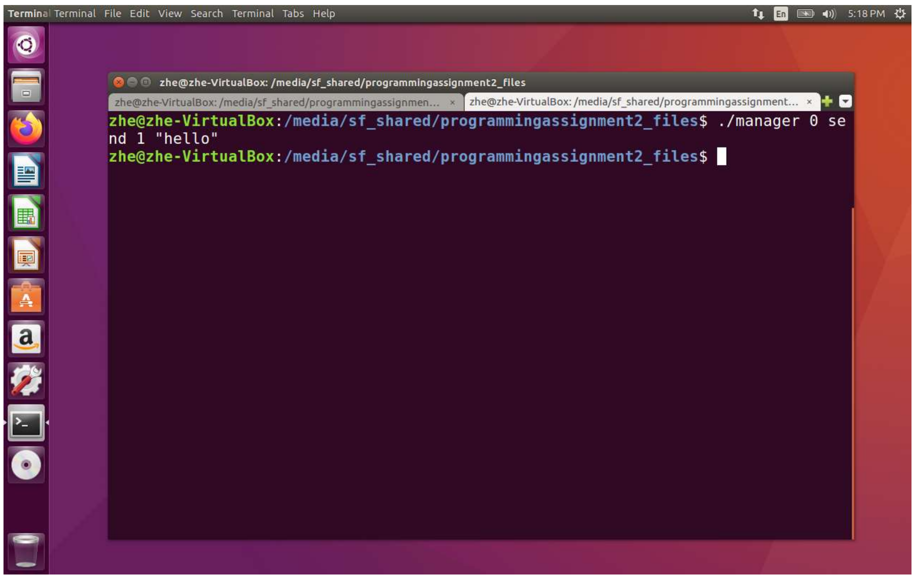
- To start a router process


Terminal Terminal File Edit View Search Terminal Help
5:16 PM
zhe@zhe-VirtualBox: /media/sf_shared/programmingassignment2_files

```
zhe@zhe-VirtualBox:/media/sf_shared/programmingassignment2_files$ ./ls_router 0
example_topology/testinitcosts0 log0
```

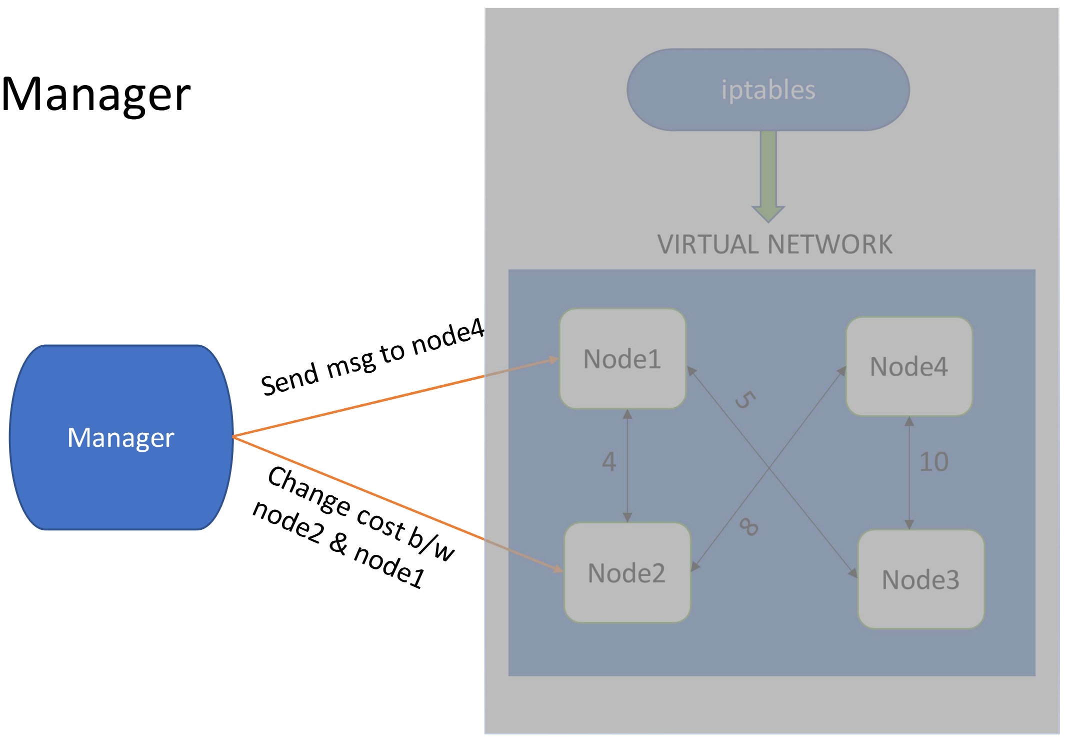
Testing – ScreenShot2

- To send a manager command


Terminal Terminal File Edit View Search Terminal Tabs Help
5:18 PM
zhe@zhe-VirtualBox: /media/sf_shared/programmingassignment2_files
zhe@zhe-VirtualBox: /media/sf_shared/programmingassignmen...
zhe@zhe-VirtualBox: /media/sf_shared/programmingassignment...

```
zhe@zhe-VirtualBox:/media/sf_shared/programmingassignment2_files$ ./manager 0 se
nd 1 "hello"
zhe@zhe-VirtualBox:/media/sf_shared/programmingassignment2_files$ 
```

Manager


iptables
VIRTUAL NETWORK
Node1
Node4
Node2
Node3
5
4
10
8
Manager
Send msg to node4
Change cost b/w node2 & node1

Testing - Manager

- Used to send commands to nodes in the virtual network.
- Two commands sent in UDP packets:
  - Send a packet

```
./<manager_binary> <source_node> send <dest_node> <message>
```

  - Change costs

```
./<manager_binary> <node1> cost <node2> <new_cost>
./<manager_binary> <node2> cost <node1> <new_cost>
```

- Source code & binary for the manager is provided. You are welcome to read it to have a better understanding of what it does.

Testing - Manager (continued)

Examples:

- **Example1**: Request Node 12 to send a message “*this_message*” to Node 23:
```
./manager_send 12 send 23 this_message
```
- **Example2**: Set the new cost between Node 12 and 30 as 2:
```
./manager_send 12 cost 30 2
./manager_send 30 cost 12 2
```

Aside – Manager Msg Handling in Node

- Two types of messages are received by nodes from manager

4 ASCII bytes

- Send a packet: **senddestIDmessage** ASCII message (<= 100 bytes)

Net order 2 bytes

- Change costs: **costdestIDnewCost**

Net order 4 bytes

Aside – Manager Msg Handling in Node

Examples:

- **Example1**: destination Id is 4 and message is “hello”. Message from manger will be
  - “send4hello”
  - where 4 occupies 2 bytes and is in the **network order**. You need to convert it to the **host order** to get the correct destID. Note that there is no space delimiter among "send", destID and the message body.

- **Example2**: the manager informs node 2 that the cost to node 5 is set to 33. The command received by node 2 will be:
  - “cost533”
  - where 5 occupies 2 bytes and 33 occupies 4 bytes. Both of them are in the **network order**. You need to convert them to the host order to get the correct destID and newCost. Note that there is no space delimiter among "cost", destID and newCost.

Notes:

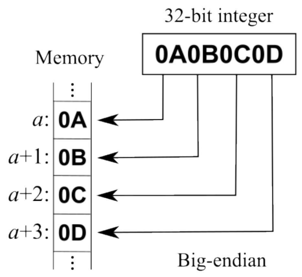
- The source code of the manager is included in the attachment. You can read it to gain a better understanding.

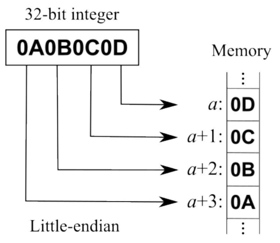
Aside – note about autograder

- Autograder only tests sending packets (”send” msg) and bringing a link up/down. It does not include changing costs
- However, we recommend handling “cost” msgs in your code
    - Ease of testing for bringing link up/down – can just change the cost to some special value, like -1 to signal infinite link cost
    - Additional code change to handle these msgs is minimal as one must handle link state changes

Aside – Network Order

- Network order – big endianness
  - the most significant byte at the smallest memory address
- Host order – Depends on the host
  - Common case: little endianness
    the least significant byte at the smallest address
- Need to convert when sending ints to network!
  - Look up htons(), htonl(), ntohs(), ntohl() for information.





Testing

- Can also modify the given topology dynamically
- How – Remember iptables!
  - To create a link

```
sudo iptables -I OUTPUT -s <node1_ip_addr> -d <node2_ip_addr> -j ACCEPT &&
sudo iptables -I OUTPUT -s <node2_ip_addr> -d <node1_ip_addr> -j ACCEPT
```

  - To take down a link

```
sudo iptables -D OUTPUT -s <node1_ip_addr> -d <node2_ip_addr> -j ACCEPT &&
sudo iptables -D OUTPUT -s <node2_ip_addr> -d <node1_ip_addr> -j ACCEPT
```

Grading

`autograde.py` is in MP2.zip, provided on Coursera.

1. Copy your code and its compiled version into the directory where you unzip.
2. Run "*`python autograde.py <program_name> <VM_network_interface>`*"
   Example: *`python autograde.py vec_router enp0s3`*
3. If successful, an `out.zip` file will be generated. Unzip the `out.zip` file "*`unzip -r out.zip`*"
4. Create a new directory called "`code/`". Copy your .c files, .h files and Makefile to the folder.
5. Zip both folders "`zip -r out.zip results/ code/`"

Grading

- Notes
  - Eight progressively hard test cases.
  - Use testcases as a guideline to add features to your program incrementally.

Tips

- When sending a message, your node should send it via the path with the lowest cost.
  - Pay attention to cost ties, choose lowest ID next hop
- Once link state changes, the routing should converge quickly.
- You need to design the message format between your routers.
- Be careful! Avoid flooding too many messages to the network.
  - Adjust the interval between periodic messages.
  - Avoid the loop: A sends update to B, B sends back to A, A sends back to B …
- Flush the file buffer after fwrite