CSE 534: Fundamentals of Computer Networks
Project 1 Report

Rajesh Golani and Udit Gupta

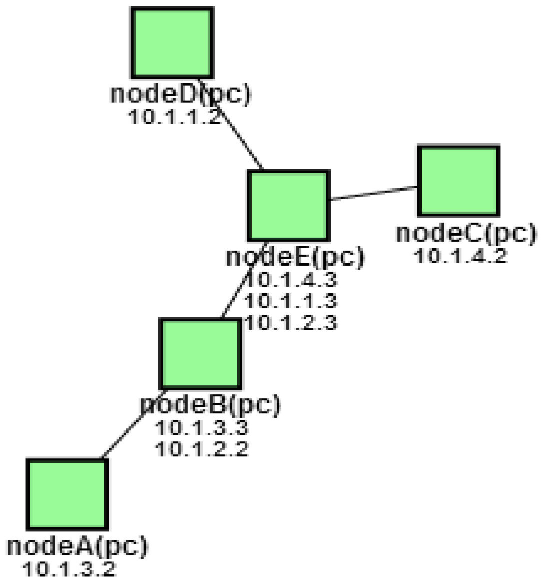
Contents

A. Emulab

A1. Report on how you managed to change the original topology, replacing the hub lan0 by the new physical node E. What were the IP addresses assigned by Emulab to the interfaces on Nodes A, B, C, D & E? Was each node able to successfully ping all the other nodes? Hand in the ns script that defines this new topology.

To use Node E instead of switch, we created following links to connect Nodes B, C, D and E:
1) B -> E
2) C -> E
3) D -> E

Link between Node A and Node B was unchanged. Following Emulab graph displays the network topology:



Following are the IP addresses assigned to interfaces on nodes:

Node	Interface Node	IP
A	B	10.1.3.2
B	A	10.1.3.3
	E	10.1.2.2
C	E	10.1.4.2

D	E	10.1.1.2
E	B	10.1.2.3
	C	10.1.4.3
	D	10.1.1.3

After inducing Node E, we were able to communicate between Nodes i.e Node A was able to connect to other Nodes, B was able to ping other nodes and so on.

The NS script for this part is as follows: **(CSE534_Project1_A1.ns)**

```
set ns [new Simulator]
source tb_compat.tcl

#setting up all the required nodes
set nodeA [$ns node]
set nodeB [$ns node]
set nodeC [$ns node]
set nodeD [$ns node]
set nodeE [$ns node]

set link0 [$ns duplex-link $nodeA $nodeB 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01

set link1 [$ns duplex-link $nodeC $nodeE 100Mb 0ms DropTail]
set link2 [$ns duplex-link $nodeD $nodeE 100Mb 0ms DropTail]
set link3 [$ns duplex-link $nodeB $nodeE 100Mb 0ms DropTail]

# Set the OS.
tb-set-node-os $nodeA UBUNTU10-STD
tb-set-node-os $nodeB UBUNTU10-STD
tb-set-node-os $nodeC UBUNTU10-STD
tb-set-node-os $nodeD UBUNTU10-STD
tb-set-node-os $nodeE UBUNTU10-STD

$ns rtproto Static

$ns run
```

B. The Click Modular Router

B1. Report on what you had to do to install and correctly configure Click as an Ethernet switch on node E. Include the Click configuration scripts you had to develop.

Installation of Click

We used links provided by professor to install click on Node E running Ubuntu-10 STD OS.
Download link: http://www.read.cs.ucla.edu/click/download
Reference link : http://www.pats.ua.ac.be/software/click/click-2.0/installation.pdf

We then used following command to configure click module as provided on the reference link:

```
./configure --disable-linuxmodule --enable-local --enable-etherswitch
```

We had to include `--enable-etherswitch` in configuration command to install EtherSwitch module that we'll need to use to make E as switch.

After this we fired make command and installed click on the Node E.
We used ListenEtherSwitch to configure node E as switch. Our configuration click file is as follows:

```
es  :: ListenEtherSwitch ;

FromDevice(eth2) ->  [0]es[0] -> Queue -> ToDevice(eth2);
FromDevice(eth3) ->  [1]es[1] -> Queue -> ToDevice(eth3);
FromDevice(eth4) ->  [2]es[2] -> Queue -> ToDevice(eth4);
es[3] -> Print("Listen") -> Discard;
```

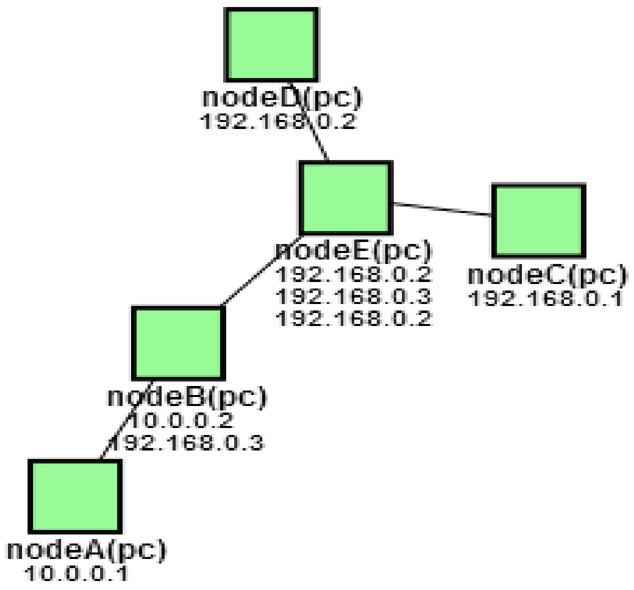
In above configuration, we basically tell Node E from which interface to accept (pull) traffic and to forward(push) it. In the last line we are using, last port to listen to the traffic and printing it on the screen.

But even after doing this, we were not able to send ping to other nodes from one subnet to other. After capturing packet traces from various nodes, we found out that receiver was receiving the
WhoAmI request and replying to it. But Node E was dropping this packet as it was not addressed to Node E. So, we had to enable promiscuous mode on every interface from where E is pulling the traffic. We used to following commands to do this:

```
sudo ifconfig eth4 promisc
sudo ifconfig eth2 promisc
sudo ifconfig eth3 promisc
```

B2. Report on the new IP address assignments in your topology, giving the IP address for each interface (an annotated topology diagram would be a nice, succinct way of doing this). How did you implement these new IP address assignments? Show the forwarding tables at nodes A, B, C & D and explain what you did to configure them correctly with respect to the IP address assignments. Explain what you had to do to the interfaces of Ethernet switch node E, and how, in order to be able to ping across the switch. Explain what you did about the forwarding table of node E. Demonstrate that each of nodes A, B, C & D was able to ping the other nodes. Include

New IP Address Assignment



Emulab topology diagram

IP Address Assignment

Node	Interface Node	IP
A	B	10.0.0.1
B	A	10.0.0.2
	E	192.168.0.3

C	E	192.168.0.1
D	E	192.168.0.2

How did we implement these new IP address assignments?

We selected two private address ranges to assign to two subnets. We decided to keep this as simple as possible. These two ranges were:

10.0.0.0/24 and **192.168.0.0/24**

Forwarding tables

Node A

```
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        *               255.255.255.0   U         0 0          0 eth1
155.98.36.0     *               255.255.252.0   U         0 0          0 eth0
192.168.0.0     nodeB-link0     255.255.0.0     UG        0 0          0 eth1
default         control-router. 0.0.0.0         UG        0 0          0 eth0
```

Node B

```
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        *               255.255.255.0   U         0 0          0 eth1
192.168.0.0     *               255.255.255.0   U         0 0          0 eth2
155.98.36.0     *               255.255.252.0   U         0 0          0 eth0
192.168.0.0     nodeD-link2     255.255.0.0     UG        0 0          0 eth2
default         control-router. 0.0.0.0         UG        0 0          0 eth0
```

Node C

```
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.0.0     *               255.255.255.0   U         0 0          0 eth4
155.98.36.0     *               255.255.252.0   U         0 0          0 eth0
192.168.0.0     nodeD-link2     255.255.0.0     UG        0 0          0 eth4
10.0.0.0        nodeD-link2     255.0.0.0       UG        0 0          0 eth4
default         control-router. 0.0.0.0         UG        0 0          0 eth0
```

Node D

```
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface
192.168.0.0      *                255.255.255.0    U         0 0          0 eth1
155.98.36.0      *                255.255.252.0    U         0 0          0 eth0
192.168.0.0      nodeE-link2      255.255.0.0      UG        0 0          0 eth1
10.0.0.0         nodeE-link2      255.0.0.0        UG        0 0          0 eth1
default          control-router.  0.0.0.0          UG        0 0          0 eth0
```

Node E

```
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface
192.168.0.0      *                255.255.255.0    U         0 0          0 eth3
192.168.0.0      *                255.255.255.0    U         0 0          0 eth4
192.168.0.0      *                255.255.255.0    U         0 0          0 eth2
155.98.36.0      *                255.255.252.0    U         0 0          0 eth0
10.0.0.0         nodeE-link2      255.0.0.0        UG        0 0          0 eth3
default          control-router.  0.0.0.0          UG        0 0          0 eth0
```

Changes to interfaces of Ethernet switch node E to be able to ping across the switch

As explained above, to ping across the switch we had to make Node E to listen to all traffic and not to let it drop packets that are intended for others. For this we set required interfaces to promiscuous mode using following commands :

```
sudo ifconfig eth4 promisc
sudo ifconfig eth2 promisc
sudo ifconfig eth3 promisc
```

Forwarding table of E

We had to turn of the IP routing at forwarding table. We accomplished this using script. Following is the updates NS script:

Updated NS file

```
# This is a simple ns script. Comments start with #.
set ns [new Simulator]
source tb_compat.tcl

set nodeA [$ns node]
set nodeB [$ns node]
set nodeC [$ns node]
set nodeD [$ns node]
set nodeE [$ns node]

#tb-set-ip $nodeA 10.0.0.1

set link0 [$ns duplex-link $nodeA $nodeB 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
tb-set-ip-link $nodeB $link0 10.0.0.2
tb-set-ip-link $nodeA $link0 10.0.0.1

set link1 [$ns duplex-link $nodeC $nodeE 100Mb 0ms DropTail]
tb-set-ip-link $nodeC $link1 192.168.0.1

set link2 [$ns duplex-link $nodeD $nodeE 100Mb 0ms DropTail]
tb-set-ip-link $nodeD $link2 192.168.0.2

set link3 [$ns duplex-link $nodeB $nodeE 100Mb 0ms DropTail]
tb-set-ip-link $nodeB $link3 192.168.0.3

tb-set-node-routable-ip $nodeE 0

# Set the OS on a couple.
tb-set-node-os $nodeA FBSD-STD
tb-set-node-os $nodeB FBSD-STD
tb-set-node-os $nodeC FBSD-STD
tb-set-node-os $nodeD FBSD-STD
tb-set-node-os $nodeE UBUNTU10-STD

$ns rtproto Static
$ns run
```

Results

To demonstrate that each node was able to ping other, we are posting traceroute results here

From Node A to

Node B

`traceroute to nodeb (10.0.0.2), 30 hops max, 60 byte packets`
`1  nodeB-link0 (10.0.0.2)  100.125 ms  100.063 ms  100.033 ms`

Node C

```
traceroute to nodec (192.168.0.1), 30 hops max, 60 byte packets
1  nodeB-link0 (10.0.0.2)  99.988 ms  99.990 ms  99.961 ms
2  nodeC-link1 (192.168.0.1)  400.913 ms  400.894 ms  400.866 ms
```

Node D

```
traceroute to noded (192.168.0.2), 30 hops max, 60 byte packets
1  nodeB-link0 (10.0.0.2)  100.071 ms  100.086 ms  100.056 ms
2  nodeD-link2 (192.168.0.2)  300.973 ms  300.960 ms  300.925 ms
```

From Node B to

Node A

```
traceroute to nodea (10.0.0.1), 30 hops max, 60 byte packets
1  nodeA-link0 (10.0.0.1)  100.097 ms  100.037 ms  100.006 ms
```

Node C

```
traceroute to nodec (192.168.0.1), 30 hops max, 60 byte packets
1  nodeC-link1 (192.168.0.1)  200.416 ms  200.431 ms  200.405 ms
```

Node D

```
traceroute to noded (192.168.0.2), 30 hops max, 60 byte packets
1  nodeD-link2 (192.168.0.2)  200.989 ms  200.923 ms  200.896 ms
```

From Node C to

Node A

```
traceroute to nodea (10.0.0.1), 30 hops max, 60 byte packets
1  nodeD-link2 (192.168.0.2)  200.437 ms  200.441 ms  200.600 ms
2  nodeE-link2 (192.168.0.3)  300.908 ms  300.904 ms  300.917 ms
3  nodeA-link0 (10.0.0.1)  400.823 ms  400.875 ms  400.936 ms
```

Node B

```
traceroute to nodeb (10.0.0.2), 30 hops max, 60 byte packets
1  nodeD-link2 (192.168.0.2)  200.865 ms  200.804 ms  200.812 ms
2  nodeB-link0 (10.0.0.2)  301.507 ms  301.563 ms  301.541 ms
```

Node D

```
traceroute to noded (192.168.0.2), 30 hops max, 60 byte packets
1  nodeD-link2 (192.168.0.2)  200.343 ms  200.432 ms  200.491 ms
```

From Node D to

Node A

```
traceroute to nodea (10.0.0.1), 30 hops max, 60 byte packets
1  nodeE-link2 (192.168.0.3)  200.917 ms  200.857 ms  200.829 ms
2  nodeA-link0 (10.0.0.1)  301.073 ms  301.057 ms  301.029 ms
```

Node B

```
traceroute to nodeb (10.0.0.2), 30 hops max, 60 byte packets
1  nodeB-link0 (10.0.0.2)  201.147 ms  201.095 ms  201.064 ms
```

Node C

```
traceroute to nodec (192.168.0.1), 30 hops max, 60 byte packets
1  nodeC-link1 (192.168.0.1)  200.571 ms  200.517 ms  200.649 ms
```

C. Some 'Rough & Ready' Performance Evaluation

ttcp configurations

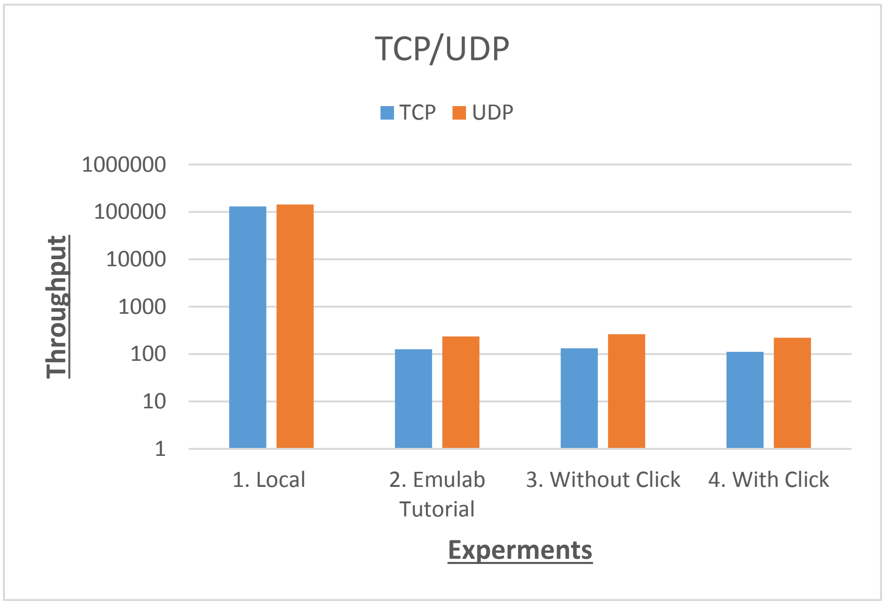
We used ttcp to perform experiments but we also tried with Ipref which was very helpful in understanding result due to different configuration options. We haven't included results for Iperf as ttcp was mentioned in the project handout.

For ttcp, we decided to go with 1000 source buffers for TCP and 10000 for UDP. The reason we went this with this is because this transfer was taking time of order of tens of seconds which would enable us to compare results more efficiently.

Analysis

We'll explain these results with the help of bar charts for better understanding.

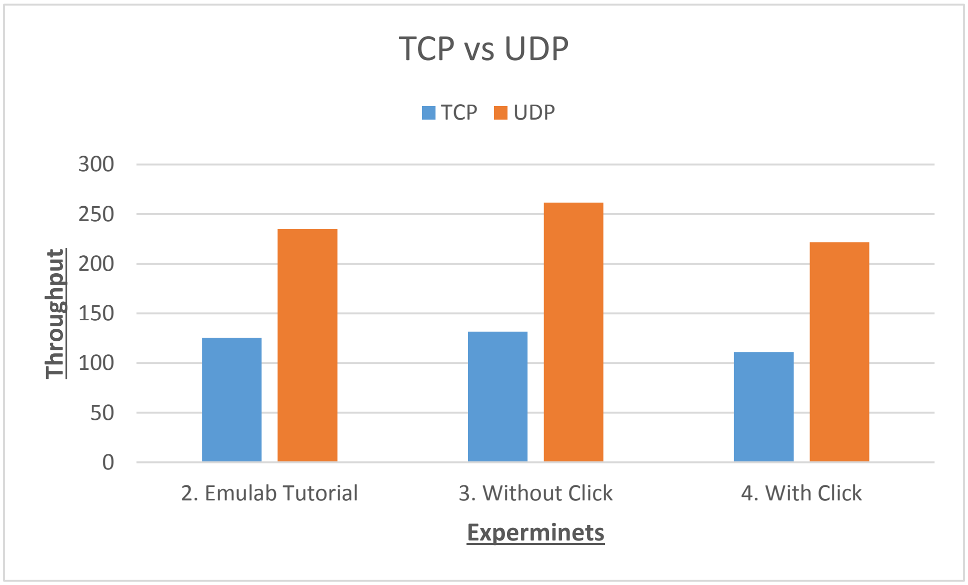
TCP vs UDP



The above graph compare the performance of TCP and UDP on all four experiments. From this we can infer:

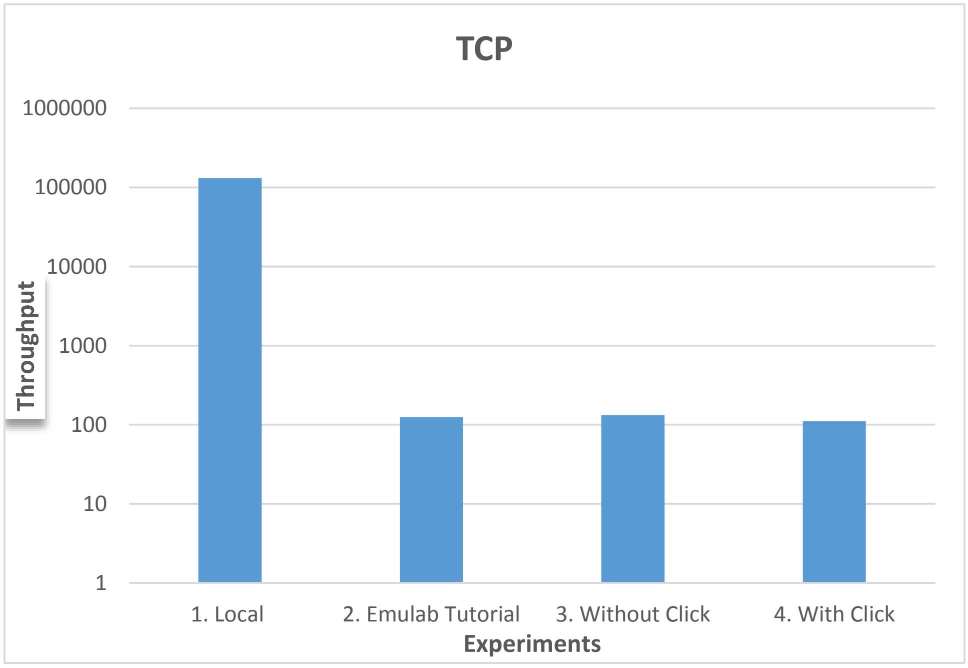
- Throughput of TCP is lower than UDP which might be the result of the nature of these protocols. TCP maintains state which might be the result of its lower performance.

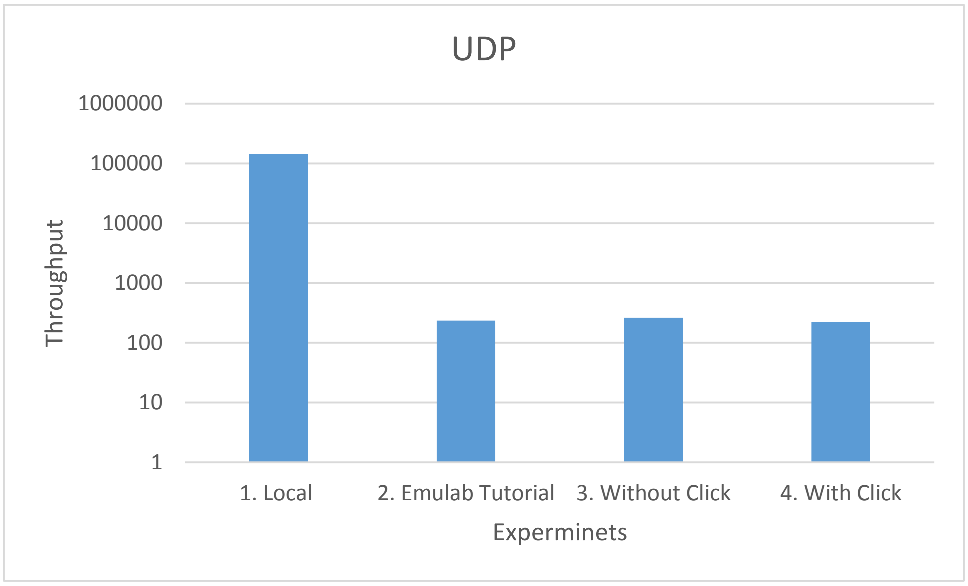
- Here the performance difference does not look that significant but it is. To make it more clearly, following is the graph comparing results of last three experiments.



Comparison of results of four experiments

Following are the two bar graphs comparing the results of four experiments for both TCP and UDP.





- From above diagrams, we can see that for both TCP and UDP introduction of click configured switch does not affect throughput significantly. We can see little degradation in throughput but it does not affect the performance marginally.
- This small difference in throughput might be because we have very small number of users and we might see significant performance degradation if we introduce more users to our topology.

<u>Why from node A to C?</u>

We think this is because if we do this from A to C, we can figure out about the bottleneck in the network. If we send it through A, performance degradation happens between link A-B and we can have more accurate estimate about throughput at node C as we can figure out whether E was the performance bottleneck.

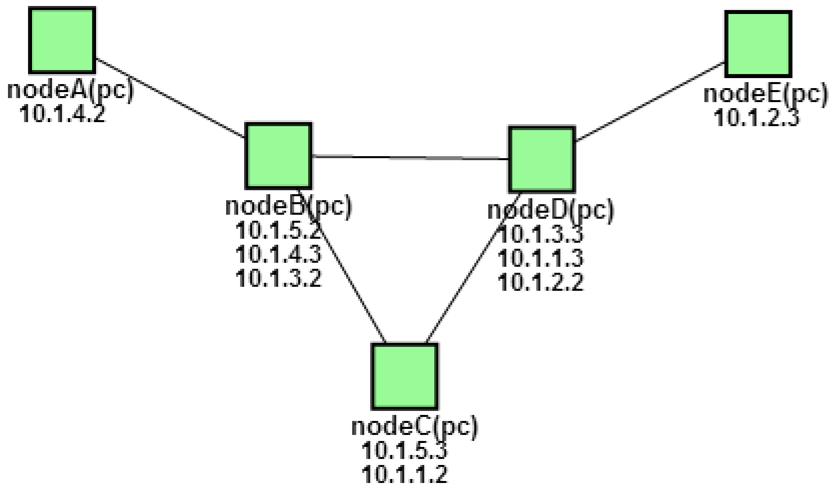
If we try this in reverse order i.e from C -> A, we cannot be sure whether bottleneck was node E or link A-B.

This is particularly useful if we want to find out whether click configured switch is bottleneck which in our case is node E.

D. Experiments with RIP

D1. Report on the IP address assignment in your topology (again, an annotated topology diagram would be a nice, succinct way of doing this), and how you activated and configured RIP on your nodes. Show the forwarding tables. Was each node able to successfully ping all the other nodes? Provide the traceroute output that gives the path between nodes A & E. Hand in the ns script that defines the topology.

<u>Topology diagram and IP address assignment</u>:



Following are the IP addresses assigned to interfaces on nodes:

Node	Interface Node	IP
A	B	10.1.4.2
B	A	10.1.4.3
	C	10.1.5.2
	D	10.1.3.2
C	B	10.1.5.3
	D	10.1.1.2
D	B	10.1.3.3
	C	10.1.1.3
	E	10.1.2.2
E	D	10.1.2.3

RIP activation and configuration:

We ran the following two commands on all the nodes provided on the link mentioned by professor in Project handout document:

1. To enable routing, following command was used:
```
      sysctl -w net.inet.ip.forwarding=1
```
2. For route discover, following command was executed:
```
      routed -s -P rdisc_interval=15
            -s : force routed to supply routing information
            -P : add parameter
Rdisc_interval : route discovery interval time
```

Forwarding tables at each node:

Node A

```
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            155.98.36.1        UGSc        7       42   fxp0
10.1.1/24          10.1.4.3           UGc         0        0   fxp3
10.1.2/23          10.1.4.3           UGc         1       17   fxp3
10.1.4/24          link#4             UC          2        0   fxp3
10.1.4.2           00:90:27:de:c3:6a  UHLW        1        4    lo0
10.1.4.3           00:02:b3:65:cf:df  UHLW        5       11   fxp3    515
10.1.5/24          10.1.4.3           UGc         1        2   fxp3
127.0.0.1          127.0.0.1          UH         11       77    lo0
```

Node B

```
Destination        Gateway            Flags    Refs      Use  Netif Expire
default            155.98.36.1        UGSc        7       16   fxp0
10.1.1/24          10.1.3.3           UGc         0        0   fxp2
10.1.2/24          10.1.3.3           UGc         0       20   fxp2
10.1.3/24          link#3             UC          1        0   fxp2
10.1.3.3           00:03:47:73:91:35  UHLW        3        3   fxp2    499
10.1.4/24          link#2             UC          1        0   fxp1
10.1.4.2           00:90:27:de:c3:6a  UHLW        1       30   fxp1    459
10.1.5/24          link#5             UC          1        0   fxp4
10.1.5.3           00:02:b3:65:bc:31  UHLW        1        4   fxp4    495
127.0.0.1          127.0.0.1          UH         11       77    lo0
```

Node C

```
Destination        Gateway            Flags     Refs      Use  Netif Expire
default            155.98.36.1        UGSc         6       20   fxp0
10.1.1/24          link#5             UC           1        0   fxp4
10.1.1.3           00:02:b3:65:b8:c9  UHLW         1        0   fxp4    684
10.1.2/24          10.1.1.3           UGc          0        0   fxp4
10.1.3/24          10.1.5.2           UGc          0        0   fxp1
10.1.4/24          10.1.5.2           UGc          0        2   fxp1
10.1.5/24          link#2             UC           1        0   fxp1
10.1.5.2           00:03:47:94:bc:58  UHLW         2        2   fxp1    393
127.0.0.1          127.0.0.1          UH          11       78    lo0
```

Node D

```
Destination        Gateway            Flags     Refs      Use  Netif Expire
default            155.98.36.1        UGSc         6       34   fxp0
10.1.1/24          link#2             UC           1        0   fxp1
10.1.1.2           00:03:47:94:c5:ae  UHLW         0        0   fxp1    649
10.1.2/24          link#5             UC           1        0   fxp4
10.1.2.3           00:02:b3:65:d1:2f  UHLW         1       27   fxp4    442
10.1.3/24          link#3             UC           1        0   fxp2
10.1.3.2           00:03:47:73:90:7d  UHLW         1        3   fxp2    362
10.1.4/23          10.1.3.2           UGc          0       20   fxp2
127.0.0.1          127.0.0.1          UH          11       77    lo0
```

Node E

```
Destination        Gateway            Flags     Refs      Use  Netif Expire
default            155.98.36.1        UGSc         7       25   fxp0
10.1.1/24          10.1.2.2           UGc          0        0   fxp1
10.1.2/24          link#2             UC           2        0   fxp1
10.1.2.2           00:03:47:94:c1:ad  UHLW         4        4   fxp1    421
10.1.2.3           00:02:b3:65:d1:2f  UHLW         1       12    lo0
10.1.3/24          10.1.2.2           UGc          0        0   fxp1
10.1.4/23          10.1.2.2           UGc          0       23   fxp1
127.0.0.1          127.0.0.1          UH          11       77    lo0
```

Was each node able to successfully ping all the other nodes?

Before configuring RIP, only directly connected nodes were able to ping each other. i.e A-B, B-C etc. But, A was not able to ping E.

After RIP configuration and routing activation on each node using above mentioned two commands, communication between all of the nodes in the topology was established i.e Yes, nodes were able to successfully ping each other.

Traceroute results between A to E

Traceroute from A to E :

```
traceroute to nodeE-link3(10.1.2.3), 64 hops max, 44 byte packets
1  nodeB-link0 (10.1.4.3)  0.278 ms  0.177 ms  0.154 ms
2  nodeD-link4 (10.1.3.3)  0.288 ms  0.285 ms  0.258 ms
3  nodeE-link3 (10.1.2.3)  0.401 ms  0.364 ms  0.355 ms
```

Traceroute from E to A :

```
traceroute to nodeA-link0(10.1.4.2), 64 hops max, 44 byte packets
1  nodeD-link3 (10.1.2.2)  0.265 ms  0.159 ms  0.143 ms
2  nodeB-link4 (10.1.3.2)  0.280 ms  0.259 ms  0.244 ms
3  nodeA-link0 (10.1.4.2)  0.383 ms  0.354 ms  0.344 ms
```

NS Script

```
# Generated by NetlabClient
set ns [new Simulator]
source tb_compat.tcl

# Nodes
set nodeA [$ns node]
tb-set-node-os $nodeA FBSD-STD
set nodeB [$ns node]
tb-set-node-os $nodeB FBSD-STD
set nodeC [$ns node]
tb-set-node-os $nodeC FBSD-STD
set nodeD [$ns node]
tb-set-node-os $nodeD FBSD-STD
set nodeE [$ns node]
tb-set-node-os $nodeE FBSD-STD

# Links
set link0 [$ns duplex-link $nodeA $nodeB 100Mb 0.0ms DropTail]
set link1 [$ns duplex-link $nodeB $nodeC 100Mb 0.0ms DropTail]
set link2 [$ns duplex-link $nodeC $nodeD 100Mb 0.0ms DropTail]
set link3 [$ns duplex-link $nodeD $nodeE 100Mb 0.0ms DropTail]
set link4 [$ns duplex-link $nodeB $nodeD 100Mb 0.0ms DropTail]

$ns rtproto Manual
$ns run

# NetlabClient generated file ends here.
# Finished at: 3/19/14 9:44 PM
```

D2. Report on how you made the link B-D go down, and at what time. How did you determine that time? If you made the link B-D go down by means of your ns script, then hand in this modified script. What was the evolution of RIP readjusting to the new situation and how quickly was connectivity re-established to node E via the B-C-D path? Show the forwarding tables. Provide the traceroute output that gives the new path between nodes A & E.

Bringing down link B-D

When?

We decided to bring down link after carrying out following steps:

- Configuring RIP and enabling routing on every node in topology
- Allowing RIP to discover paths to every other node in topology
- Making sure every node is able to ping each other
- Confirming that path between A to E is 3-hop path using traceroute command

How?

Instead of using NS Script to bring down B-D link, we ran following commands to bring down the link:

On Node B

```
sudo ifconfig fxp2 down
```

On Node D

```
sudo ifconfig fxp2 down
```

Connectivity Reestablishment to Node E

In our case, we noticed that connectivity was getting reestablished pretty quickly (order of few seconds) for Node E via 4-hop path A-B-C-D-E.

We tested this using following command:

```
ping nodee | while read pong; do echo "$(date): $pong"; done
```

This command will ping to Node E and will display results to the screen. We are also displaying time on the screen to measure time for RIP to establish new path.

Forwarding tables:

To explain this evolution, routing table for every node is displayed.

Node A

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             155.98.36.1         UGSc         8      282   fxp0
10.1.1/24           10.1.4.3            UGc          0        0   fxp3
10.1.2/24           10.1.4.3            UGc          1       24   fxp3
10.1.4/24           link#4              UC           1        0   fxp3
10.1.4.3            00:02:b3:65:cf:df   UHLW         3        4   fxp3    404
10.1.5/24           10.1.4.3            UGc          0        2   fxp3
127.0.0.1           127.0.0.1           UH          11       77    lo0
```

Node B

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             155.98.36.1         UGSc         9       82   fxp0
10.1.1/24           10.1.5.3            UGc          0        0   fxp4
10.1.2/24           10.1.5.3            UGc          0       23   fxp4
10.1.4/24           link#2              UC           1        0   fxp1
10.1.4.2            00:90:27:de:c3:6a   UHLW         0      311   fxp1    385
10.1.5/24           link#5              UC           1        0   fxp4
10.1.5.3            00:02:b3:65:bc:31   UHLW         3        4   fxp4    390
127.0.0.1           127.0.0.1           UH          11       77    lo0
```

Node C

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             155.98.36.1         UGSc         8      118   fxp0
10.1.1/24           link#5              UC           1        0   fxp4
10.1.1.3            00:02:b3:65:b8:c9   UHLW         1        0   fxp4    360
10.1.2/24           10.1.1.3            UGc          0      113   fxp4
10.1.4/24           10.1.5.2            UGc          0       94   fxp1
10.1.5/24           link#2              UC           1        0   fxp1
10.1.5.2            00:03:47:94:bc:58   UHLW         1        2   fxp1    354
127.0.0.1           127.0.0.1           UH          11       77    lo0
```

Node D

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             155.98.36.1         UGSc         8       82   fxp0
10.1.1/24           link#2              UC           1        0   fxp1
10.1.1.2            00:03:47:94:c5:ae   UHLW         1        0   fxp1    339
10.1.2/24           link#5              UC           1        0   fxp4
10.1.2.3            00:02:b3:65:d1:2f   UHLW         0      203   fxp4    339
10.1.4/23           10.1.1.2            UGc          0       20   fxp1
127.0.0.1           127.0.0.1           UH          11       77    lo0
```

Node E

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             155.98.36.1         UGSc         8       90   fxp0
```

```
10.1.1/24          10.1.2.2           UGc          0        0   fxp1
10.1.2/24          link#2             UC           1        0   fxp1
10.1.2.2           00:03:47:94:c1:ad  UHLW         2        0   fxp1    321
10.1.4/23          10.1.2.2           UGc          2      203   fxp1
127.0.0.1          127.0.0.1          UH          11       78    lo0
```

<u>**Traceroute output that gives the new path between nodes A & E**</u>

From Node A to E

```
traceroute to nodeE-link3 (10.1.2.3), 64 hops max, 44 byte packets
1  nodeB-link0 (10.1.4.3)  0.374 ms  0.196 ms  0.284 ms
2  nodeC-link1 (10.1.5.3)  0.423 ms  0.363 ms  0.360 ms
3  nodeD-link2 (10.1.1.3)  0.596 ms  0.549 ms  0.528 ms
4  nodeE-link3 (10.1.2.3)  0.684 ms  0.633 ms  0.636 ms
```

D3. Report on how you made the link D-E go down, and at what time. How did you determine that time? What was the evolution of RIP readjusting to the new situation, and how long did it take nodes A, B, C & D to finally realize that node E was unreachable? Show this count-to-infinity evolution by taking one node of your choice as an example and giving the evolution of its forwarding table.

<u>**Bringing down link D-E**</u>

<u>**When?**</u>

For this, we used the same environment as D2. We made sure following steps are taking place before trying this experiment.

- Configuring RIP and enabling routing on every node in topology
- Allowing RIP to discover paths to every other node in topology
- Making sure every node is able to ping each other
- Confirming that path between A to E is 3-hop path using traceroute command

<u>**How?**</u>

Instead of using NS Script to bring down D-E link, we ran following commands to bring down the link:

On Node D

```
sudo ifconfig fxp4 down
```

On Node E

```
sudo ifconfig fxp1 down
```

Evolution of RIP

After bringing down the link, we observed the count to infinity problem. Here, Node D was forwarding packets to other nodes (B or C) as its direct link to Node E was down.

While these other nodes were thinking that they have path to E through D and they were advertising this path. Due to this, packet was in loop resulting in “Count-to-infinity” situation.

After some time(order of minutes), we noticed that link entry to E from D’s forwarding table was removed and other nodes were also able to update this information and realizing that Node E is down.

Evolution of D’s forwarding table

To explain above mentioned evolution, we’ll explain evolution of D’s forward table. For this, we’ll describe D’s forwarding table at three points of time.

1. Before bringing down link between D and E
2. After bringing down the link but before all nodes realize that link to E is down.
3. After every node has updated forwarding table, realizing node E is unreachable.

Here, D and E are in subnet with prefix 10.1.2/24.

1. Before bringing down link between D and E

There is direct path between Node D and Node E. Node D is connected to E on interface fxp4 as highlighted in routing table.

```
Destination         Gateway             Flags     Refs      Use  Netif Expire
default             control-router.emu  UGSc         9      998   fxp0
10.1.1/24           link#2              UC           1        0   fxp1
nodeC-link2         link#2              UHLW         1        0   fxp1
10.1.2/24           link#5              UC           1        0   fxp4
nodeE-link3         00:02:b3:65:d1:2f   UHLW         1        2   fxp4   1191
10.1.3/24           link#3              UCc          1        0   fxp2
10.1.4/24           nodeB-link4         UGc          0      127   fxp2
10.1.5/24           nodeC-link2         UGc          0        0   fxp1
localhost           localhost           UH          11       85    lo0
control-net/22      link#1              UC           1        0   fxp0
control-router.emu  00:d0:bc:f4:14:f8   UHLW         1        0   fxp0   1191
pc166.emulab.net    localhost           UGHS         0        0    lo0
```

2. After bringing down the link but before all nodes realize that link to E is down.

Bringing down this interface results change in D’s routing table. Now as there is no direct path to E, D sees that C has path to E and accordingly updates its routing table.

D is connected to C on interface fxp2 and forwards packets intended for E which results in count-to-infinity problem.

```
Destination         Gateway             Flags     Refs       Use  Netif Expire
default             control-router.emu  UGSc         8       700   fxp0
10.1.1/24           link#2              UC           1         0   fxp1
nodeC-link2         00:03:47:94:c5:ae   UHLW         1         0   fxp1    652
10.1.2/24           nodeB-link4         UGc          0         0   fxp2
10.1.3/24           link#3              UCc          1         0   fxp2
10.1.4/24           nodeB-link4         UGc          0       127   fxp2
10.1.5/24           nodeC-link2         UGc          0         0   fxp1
localhost           localhost           UH          11        77    lo0
control-net/22      link#1              UC           1         0   fxp0
control-router.emu  00:d0:bc:f4:14:f8   UHLW         1         0   fxp0   1198
pc166.emulab.net    localhost           UGHS         0         0    lo0
```

3. After every node has updated forwarding table, realizing node E is unreachable.

After some time when every node learns that node E is down, forwarding tables are updated. In D’s forwarding table, there is no route to E now as it is unreachable.

```
Destination         Gateway             Flags     Refs       Use  Netif Expire
default             control-router.emu  UGSc         8       773   fxp0
10.1.1/24           link#2              UC           1         0   fxp1
nodeC-link2         00:03:47:94:c5:ae   UHLW         1         0   fxp1    371
10.1.3/24           link#3              UCc          1         0   fxp2
10.1.4/24           nodeB-link4         UGc          0       127   fxp2
10.1.5/24           nodeC-link2         UGc          0         0   fxp1
localhost           localhost           UH          11        77    lo0
control-net/22      link#1              UC           1         0   fxp0
control-router.emu  00:d0:bc:f4:14:f8   UHLW         1         0   fxp0   1199
pc166.emulab.net    localhost           UGHS         0         0    lo0
```