PA2

A walkthrough

Overview

- *Emulate a link layer and a network layer protocol*
- 2 Independent Sections
  - –Go Back N (link layer)
  - –Distance Vector Protocol (network layer)
- Will combine the 2 sections such that both algorithms working in tandem

Go Back-N

- Implement the Go-Back-N (GBN) protocol on top of UDP to guarantee that all packets can be successfully delivered to the higher layers in the correct order
- To emulate an unreliable channel, the receiver and the sender need to drop an incoming data packet or an ACK, respectively, with a certain probability

Go Back-N: Details

- One program “gbnnode”
- Two node instances, sender and receiver
- Both node processes will be on the same machine but different port numbers
- Data packet: 1 character

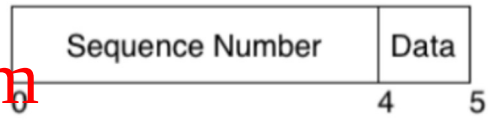


Figure 1: GBN Packet Header Example

- Loss emulation: probabilistic or deterministic

Figure 2: Packet Loss Example

Go Back-N: Setup

Two nodes, sender and receiver

Sender side:

```
$ ./gbnnode 1111 2222 5 -p 0.1
node> send abcdefgh
```

Receiver side:

```
$ ./gbnnode 2222 1111 5 -p 0.1
```

Go Back-N: Loss Rate Calculation

After transmission, sender and receiver each report:

Loss rate = # packets dropped / total # packets

A test: In *probabilistic* mode, actual loss rate should converge to command line drop probability ***p***

Distance Vector Protocol

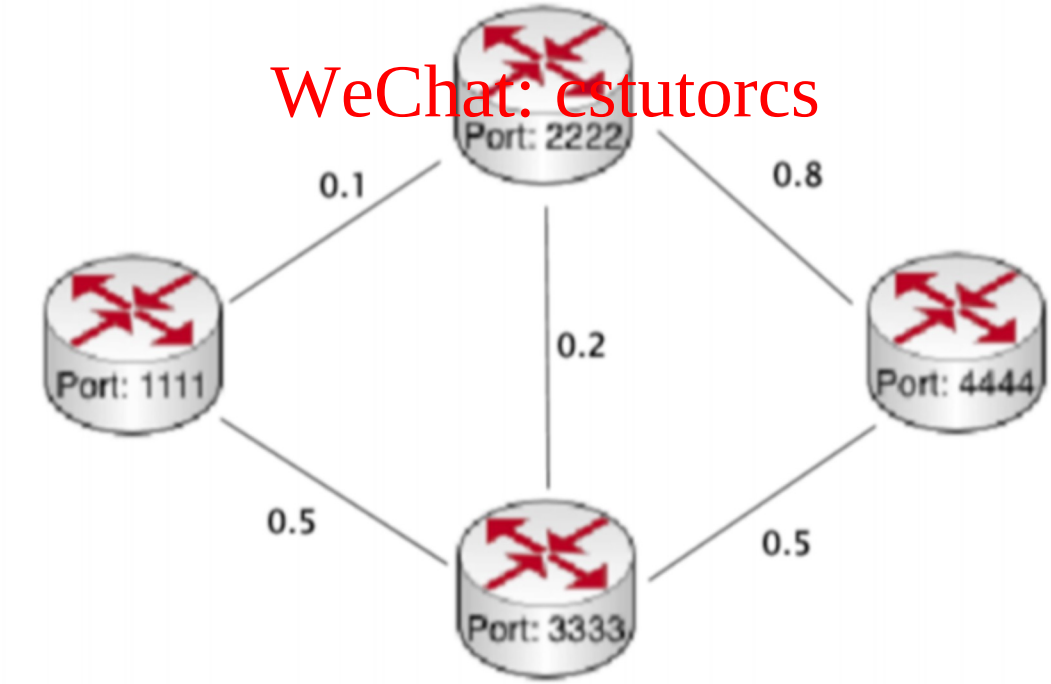
- Objective: Implement a simplified version of a routing protocol in a *static* network.
- Use the **Bellman-Ford algorithm** to build and update the routing tables
- UDP should be used to exchange the routing table information among the nodes
- We assume that that all the nodes run on the same machine and they all have the same IP address
- Each node can be identified uniquely by a (UDP listening) port number, which is specified by the user

Keeping track of Routing Table

- Upon the activation of the program, each node should construct the initial routing table and keep it locally
- The node with the **last** keyword will send out its routing table first
- Using **Bellman-Ford**, each node will keep updating its routing table as long as neighboring nodes send their updated routing tables information
- If there is any change in the routing table, a node should send the updated information to its neighbors.
- NOTE: Each node should send its routing table information to its neighbors at least once.

Initializing Network Topology

```
$ ./dvnode 1111 2222 .1 3333 .5
$ ./dvnode 2222 1111 .1 3333 .2 4444 .8
$ ./dvnode 3333 1111 .5 2222 .2 4444 .5
$ ./dvnode 4444 2222 .8 3333 .5 last
```

Status Messages!

- Make sure to follow the status messages specified (Don't do anything fancy please)


```
[1353035852.173] Node 1111 Routing Table
- (.1) -> Node 2222
- (.3) -> Node 3333; Next hop -> Node 2222
- (.8) -> Node 4444; Next hop -> Node 2222

[1353035852.192] Node 2222 Routing Table
...

[1353035852.239] Node 3333 Routing Table
...
```

Please note all the assumptions in this section

- Max Nodes = 16
- Links and distances (costs) specified at start and stay *static* throughout test
- Distance is same in both directions
- Use UDP (if you did PA1 then you already know how!)

Combination

- GBN part + DV routing part to emulate a computer network with *dynamic link state*
- Goal is to integrate code you've already written.

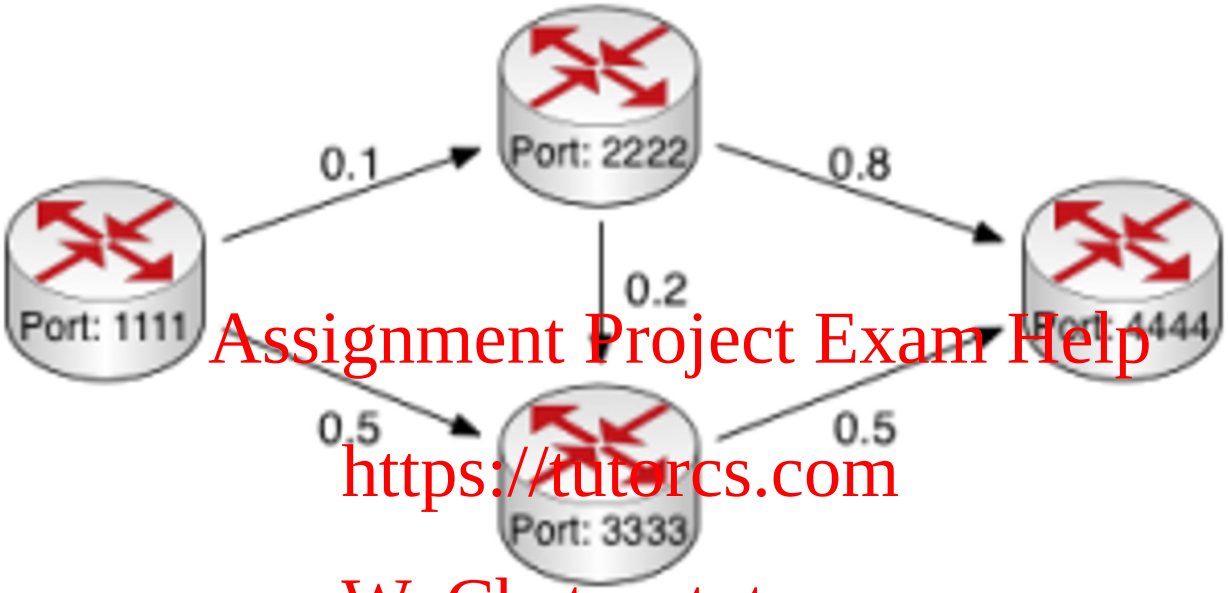
Dynamic Links

- Use code from GBN section
- *Probe packets* instead of char messages
- Loss rate calc: simplified
  - Only probabilistic packet drop
  - ACKs never dropped
  - Window size == 5

$$Link\ cost = \begin{cases} 0, \text{(initial value)} & \text{if no probe packets have been sent} \\ \frac{Total\ number\ of\ dropped\ packets}{Total\ number\ of\ sent\ packets}, & \text{otherwise} \end{cases}$$

Probe packets

- Probe packets sent continuously, loss rate updated
- Each link connects two nodes, specify *sender* node and *receiver* node
- Probe packets sent *in one direction*
  - Senders only get Link loss rate when receiver sends over Routing Table update



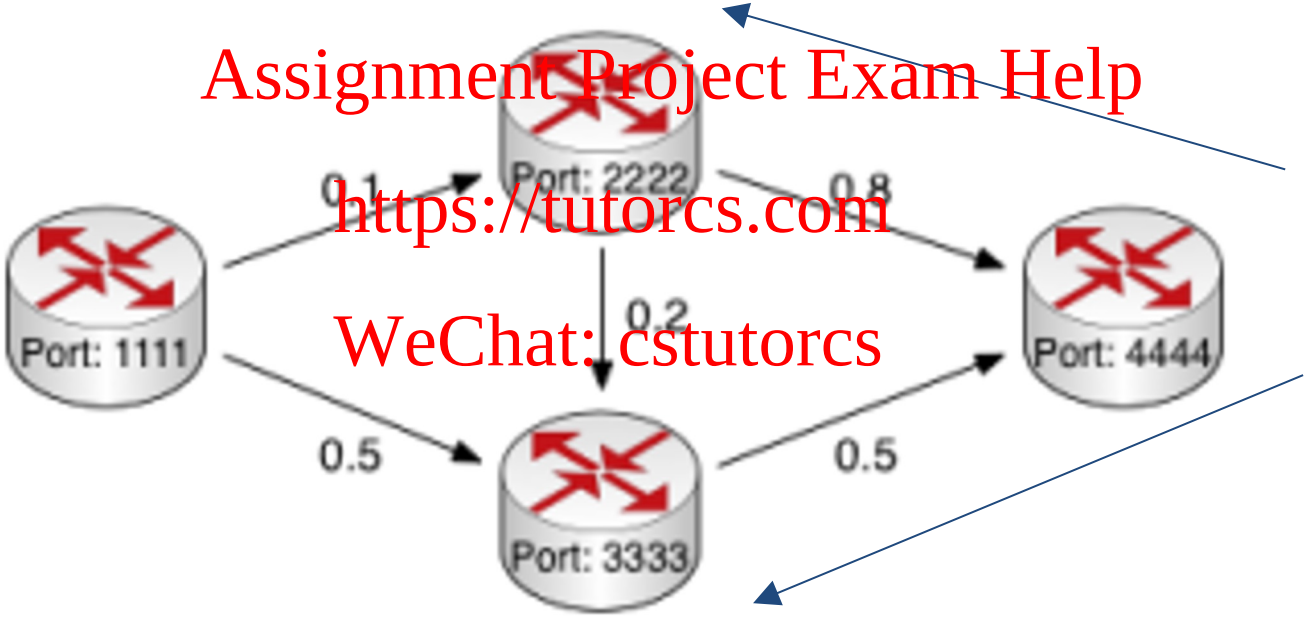
$ ./cnnode 1111 receive send 2222 3333 (receiving list is empty)

$ ./cnnode 2222 receive 1111 .1 send 3333 4444

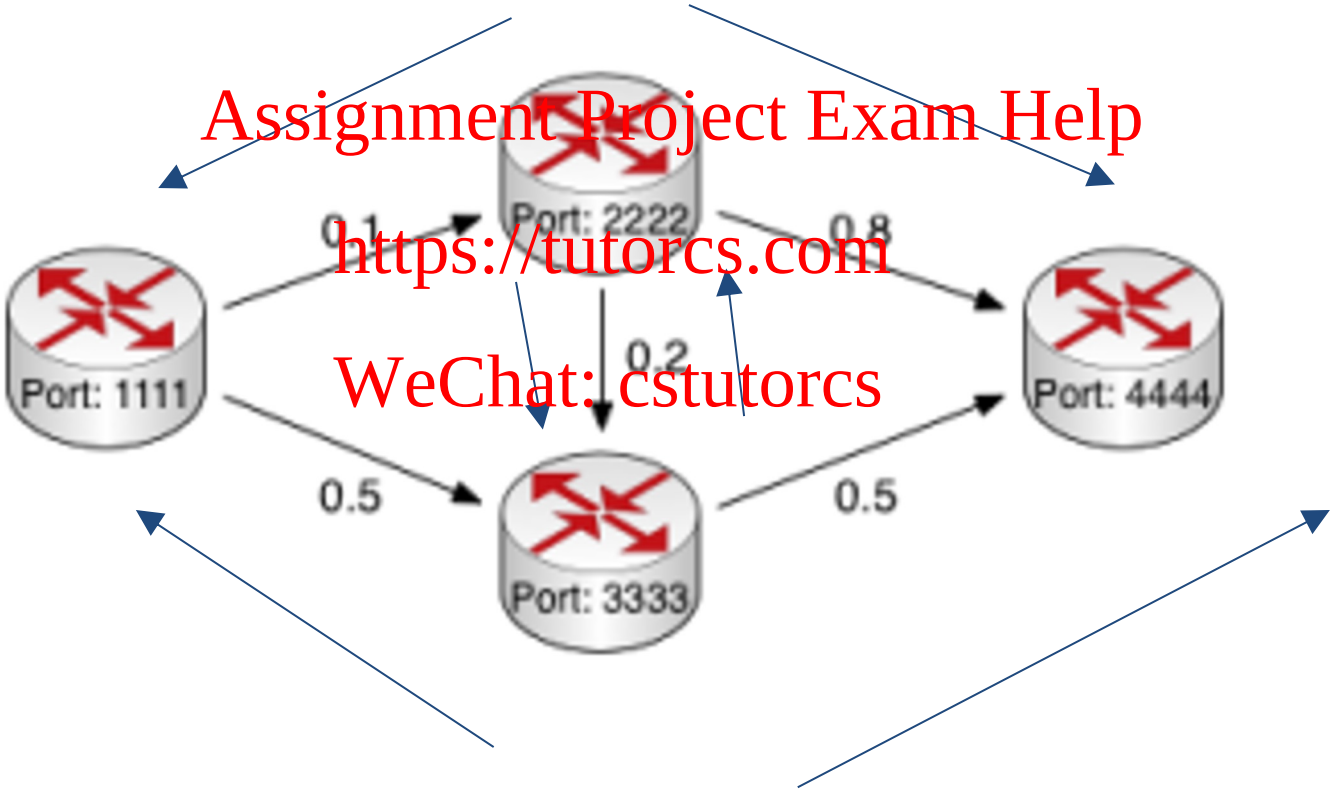
$ ./cnnode 3333 receive 1111 .5 2222 .2 send 4444

$ ./cnnode 4444 receive 2222 .8 3333 .5 send **last** (sending list is empty)

Simple Example



Simple Example



Testing

- We will be running your code on Google Cloud Machines using Ubuntu 14.04 LTS
- ***Please run and test your code in this environment***
- Java, Python, and C allowed (same rules as PA1)
  - We will only be using Java 7 and JDK 1.7 (**Not Java 8)**

Some Tips

- Start early
- Make sure a step works before going on to the next one
- When we ask for command line arguments we actually mean command line arguments
- Don't submit class or executable files
- Don't submit your eclipse package structure
- Java does not need a Makefile as long as the readme specified the main class

Notes from PA1

- Please submit .zip archives
- Remove logging / debugging print statements in final submission
- Submit README as ***text file***