CSEE 4119: Computer Networks, Fall 2014

Programming Assignment 3: Distributed Bellman-Ford

Due Thursday, December 4th 11:55 pm

Academic Honesty Policy

You are permitted and encouraged to help each other through Piazza's web board. This only means that you can discuss and understand concepts learnt in class. However, you may NOT share source code or hardcopies of source code. Refrain from sharing any material that could cause your source code to APPEAR TO BE similar to another student's source code enrolled in this or previous years. Refrain from getting any code off the Internet. Cheating will be dealt with severely. Cheaters will be penalized. Source code should be yours and yours only. Do not cheat.

---

1. Introduction

In this assignment you will implement a simple version of the distributed Bellman-Ford algorithm. The algorithm will operate using a set of distributed client processes. The clients perform the distributed distance computation and support a user interface, e.g., it allows the user to edit links to the neighbors and view the routing table. Clients may be distributed across different machines and more than one client can be on the same machine. Clients are identified by an <IP address, Port> tuple. Each client process gets as input the set of neighbors, the link costs and a timeout value. Each client has a read-only UDP socket to which it listens for incoming messages and its neighbors know the number of that port. Each client maintains a distance vector, that is, a list of <destination, cost> tuples, one tuple per client, where cost is the current estimate of the total link cost on the shortest path to the other client. Clients exchange distance vector information using a ROUTE UPDATE message, i.e., each client uses this message to send a copy of its current distance vector to its neighbors. Each client uses a set of write only sockets to send these messages to its neighbors. The clients wait on their sockets until their distance vector changes or until TIMEOUT seconds pass, whichever arrives sooner, and then

transmit their distance vectors to all neighbors. Each client also provides a command line interface to the user. Four types of commands are required to be supported:

- LINKDOWN {ip_address port} – This allows the user to destroy an existing link, i.e., change the link cost to infinity to the mentioned neighbor.
- LINKUP {ip_address port}– This allows the user to restore the link to the mentioned neighbor to the original value after it was destroyed by a LINKDOWN.
- SHOWRT – This allows the user to view the current routing table of the client. It should indicate for each other client in the network, the cost and neighbor used to reach that client.
- CLOSE – With this command the client process should close/shutdown. Link failures is also assumed when a client doesn't receive a ROUTE UPDATE message from a neighbor (i.e., hasn't 'heard' from a neighbor) for 3*TIMEOUT seconds. This happens when the neighbor client crashes or if the user calls the CLOSE command for it. When this happens, the link cost should be set to infinity and the client should stop sending ROUTE UPDATE messages to that neighbor. The link is assumed to be dead until the process comes up and a ROUTE UPDATE message is received again from that neighbor.

---

2. User Interface

A client process receives command-line arguments as follows:

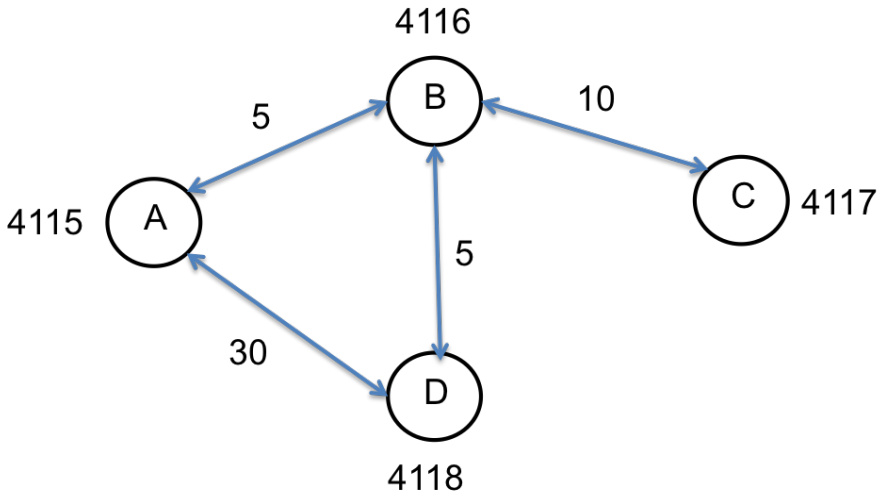
%> ./bfclient localport timeout [ipaddress1 port1 weight1 ...]

where:

- bfclient is the name of the local process (your executable).
- The localport argument is the number of the local port that the process should listen to (This is used to distinguish multiple clients on the same machine).
- The timeout argument specifies the inter-transmission time of ROUTE UPDATE messages in steady state. It is specified in seconds.
- The remaining command line consists of triples (at least one) indicating incident links to neighbors. Each triple consists of:
    - Ipaddress: the IP address of the neighbor, given in dotted decimal notation.

- port: The port number on which the neighbor is listening.
- weight: A real number indicating the cost of the link.

For example, consider the following scenario.



%>./bfclient 4115 3 128.59.196.2 4116 5.0 128.59.196.2 4118 30.0

%>SHOWRT

<Current Time>Distance vector list is:

Destination = 128.59.196.2:4116, Cost = 5.0, Link = (128.59.196.2:4116)

Destination = 128.59.196.2:4118, Cost = 30.0, Link = (128.59.196.2:4118)

After some time, new nodes might be added as they are discovered

%>SHOWRT

<Current Time> Distance vector list is:

Destination = 128.59.196.2:4116, Cost = 5.0, Link = (128.59.196.2:4116)

Destination = 128.59.196.2:4118, Cost = 10.0, Link = (128.59.196.2:4116)

Destination = 128.59.196.2:4117, Cost = 15.0, Link = (128.59.196.2:4116)

Break a link

%>LINKDOWN 128.59.196.2 4116

The link cost to this neighbor should be set to infinity (The distance vector needs to be updated) and a LINK DOWN message is sent to the neighbor. On receipt of a LINK

DOWN message, a client should set the link cost to the sender as infinity. The two nodes are not neighbors till the link is restored (by a LINK UP message) and stop exchanging ROUTE UPDATE messages.

%>LINKUP 128.59.196.2:4116

The link cost should now be restored to the original value (The distance vector needs to be updated) and a LINK UP message is sent to the neighbor. On receipt of a LINK UP message, a client should restore the link cost to the sender to the original value. The two nodes are now neighbors again and should resume exchanging ROUTE UPDATE messages.

Note: The above example has all the nodes on the same machine, but we will test using by placing nodes on different machines.

---

3. Requirements and Tips

- The clients should not do busy waiting while listening on its socket.
- The clients should be able to adjust to dynamic networks – i.e. new nodes joining and existing nodes leaving the network.
- When a new node enters the network, its neighbors should also add it to their list of neighbors when they receive the first ROUTE UPDATE from it. They should use their distance from it as the link cost (picked from the first ROUTE UPDATE message they receive from the new neighbor).
- When the CLOSE command is issued, it is like simulating link failure of all the links to that client since the process exits and its neighbors don't receive ROUTE UPDATE messages for more than 3*timeout. (It is the same as hitting CTR-C).
- You don't need to worry about the counting to infinity problem.
- The protocol messages should be compact and avoid containing redundant or unnecessary information.
- You should design your own protocol for the inter-client communications. You may use HTPP-like text protocol encoding or a proprietary designed one. Project

submission should include a description of the designed protocol, including syntax and semantics.

- System time is obtained using gettimeofday and it should be truncated before its display in order to improve readability.
- Design a good timer mechanism. Since you will be using only one timer (implicit in the select system call), you have to manage a queue of all timeout values that should occur in the future (two values for each neighbor; one for the time by which the client should send a ROUTE UPDATE and one for when the client expects a ROUTE UPDATE before concluding that the node is dead).
- Test your program with non-trivial topologies. Test your program behavior in a dynamic environment in which new clients join the system. Use the LINK UP/DOWN commands to test the convergence of your bellman-ford implementation.

---

4. What to submit

Submission will be done by courseworks. Please post a single <UNI>_<Language>.zip file to the Programming Assignment 3 folder. The file should include the following

- README.txt This file contains the project documentation; program features and usage scenarios; protocol specification of your implementation, including a complete list of the data messages used, their encodings (syntax) and semantics; and a description of whatever is unusual about your implementation, such as additional features or a list of any known bugs.
- Makefile The make file is used to build your application. Output file name should be bfclient (bfclient.class for Java).
- The source code files.

Please make sure that your programs compile and run correctly. To grade your program we will extract the received zip file, type make, and run the programs using the same syntax given in sections 2. Please make sure that your submission successfully passes this simple procedure. TA's will compile and test the code you submit on CLIC Linux machines. It is your responsibility to make sure your code is able to be compiled and

runnable on CLIC Linux machines. The CLIC lab has following setup: **Java 1.6, gcc version 4.6.3, python 2.7.3.**

References

1. 'Computer Networking: A Top-Down Approach Featuring the Internet' (3rd Edition) by James F. Kurose and Keith W. Ross. See Chapter 3 for a good description of transport-layer protocol fundamentals (with TCP as an illustrative example).
2. 'TCP/IP Illustarted Vol I' by W. Richard Stevens. An excellent guide to the TCP/IP protocol.
3. 'Beej's Guide' which can be found at http://www.beej.us/guide/bgnet/. An online tutorial for socket programming.
4. Unix man pages for socket, bind, sendto, recvfrom, select.

Grading rubric

Functionality	Max Points awarded / deducted
Creation of stable initial distributed routing table with a network of size 5 with few of the clients on same machine. Use of SHOWRT.	35
Use of LINKDOWN command on one or more clients to show its correct effect	15
Use of LINKUP command on one or more terminals to show its correct effect	15
Use of CLOSE command and show its correct effect 15	15
No proper error messages when required	1 to 5
Code quality	1 to 5
Textual description of design protocol including syntax and semantics	10
Extra feature: The feature should be useful and grading will be done on its innovation, utility and amount of effort.	Max 10 points per extra feature. On TA discretion.
Total max extra feature points	20