

**School of Engineering & School of Computer Science and Statistics**

**3D3– Computer Networks**

|  |  |
| --- | --- |
| **Group Number** | 16 |
| **Title** | Assignment 2 Report |
| **Date of Submission** | 12/04/19 |

|  |  |
| --- | --- |
| **Member Name** | **Member Student Number** |
| Adam McQuade | 16326767 |
| Ailish Conneely | 16320336 |
| Cian Levy | 16318564 |
| Edmond Cheng | 15315911 |

By submitting this Laboratory Report we understand that we are agreeing with the following:

* We agree that this is all our own work and has not been taken from the work of others save where appropriately referenced in the body of the assignment.
* We have read and we understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.
* We have also completed the Online Tutorial on avoiding plagiarism ‘Ready Steady Write’, located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write."

**Implementation of Program**

The following classes were implemented in our solution to the task

* **Datagram**
* Encapsulates the datagrams transmitted between routers.
* Used by the router class to both construct and deconstruct datagrams.
* From the sender point of view the class will be used to multiplex the payload data and datagram header.
* From the receiver point of view the class will be used to demultiplex the payload data and header fields.
* **Distance-vector**
* This class encapsulates the distance vectors and their data.
* Used by the router to add all the data required to advertise a new link to a vector (buildDV) which can be sent to the other routers.
* When a router receives a new link advertisement the data will be extracted and parsed by this class.
* **Log**
* Responsible for recording: changes to the routing table, datagrams received and routed.
* The log will write to a file called "routing-outputX.txt" (where X is the router ID) which will be stored in the "resources" folder.
* Storing the state of the table prior to an update requires that the previous state is stored somewhere.
* When recordTableUpdate is called, "recordTableUpdate" will be one update behind. After the required records are written to the file previousTable can be overwritten with the current table state.
* **Router**
* Effectively implements a dynamically updated lookup table.
* When the router receives a datagram whose destination is X, the link in position X in the "table" vector specifies which router the datagram should be forwarded to, based on the current least cost path.
* When a new link is advertised the path cost is calculated based on the values in the vector. If the new cost is less than the current cost the link will be updated.
* Heartbeat protocol is implemented here to check liveness of routers.
* **Bash Script**
* Launches the routers at random order. This is done by shuffling an array that contains the router IDs – namely A, B, C, D, E and F. The array members are passed as arguments when launching the my-router.cpp file.
* Bash script to generate traffic available as well. It will generate a predefined packet from a source to a destination defined by the user via inputs on launch time of script.
* **My-Router**
* Top level file of the program.
* Starts up a router at launch time with the default topology.
* Can start up a router using a topology specified by a csv file.
* Also used to generate traffic with a source ID and destination ID taken from user input.

**How Do The Routers Know When A Router Is Dead?**

Every five seconds, the routers will broadcast their status (Heartbeat Protocol). Doing so they announce to all the other routers that they are still alive. When a router is removed from the network, a timer starts. After six seconds of not receiving any updates from the removed router, the still alive routers recognise that a router has been dropped from their network. They then update their routing tables and broadcast their updates too. A router is killed by implementing the poison reverse method. All alive routers in the network are informed that the link cost to the dead router is infinite (specifically, FF or 255).

**Difficulties Faced And How They Were Resolved**

* Inconsistent receiving of datagrams. The datagrams were sometimes not received at all by some routers, which was caused by logic inside the function used by a router when a packet was received [receive()]. The program proceeded without calling receive() again so datagrams were lost along the way. This was corrected by removing recursion in the broadcasting function [periodicRetransmit()] and receive() would not be skipped by other routers.
* The log file of the first router to receive a datagram was always corrupted. Non-ASCII characters were introduced into the log file of the first router to receive the datagram. All other receiving routers’ log files were not corrupted. This occurred because the log file of the router was opened whilst the packet was being generated. This altered the log file and was fixed by implementing a check to prevent the log file from being opened while the traffic was being generated.
* Routers appeared as dead despite their neighbours were still sending updates. This happened for the routers that had no direct paths to the router that was dropped. The fix was to create a list of the neighbours of each router. This list remains unchanged and only routers that were on the list were sent updates by the respective router. E.g. A is directly connected to B and E and therefore A only sends updates to B and E.

**Suggested Improvements**

Currently, when a router sees two adjacent nodes with the same link cost, the router will alphabetically prioritise the nodes. The program does not take into account of the lower hop count. The program can be amended to calculate the hop count when it encounters this scenario, and choose the path with the lower hop count.

**Experiences Gained and Lessons Learned from Completion of this Project**

* Using version control software such as Git.
  + Allowed us to easily manage the large number of files required in our implementation.
  + Kept each group member accountable as each contribution had a version history.
* Writing batch scripts.
  + Initialising and launching the routers randomly and in parallel at start-up.
* Working in parallel when developing code.
  + Commenting code effectively to ensure universal comprehension.
  + Working under time constraints to deliver code for other group members to use.
* Importance of high level planning.
* Effective and frequent communication is essential.
* Modular testing of code to ensure that each class and function behaves as required and expected.
* The difficulty of fair division of workload in a project where the parts are interdependent.
* Importance of having a clear timeline of deadlines for each part of the project.
* Querying technical problems in search engines as well as with laboratory demonstrators can be helpful.

**Log files showing the stabilised route tables. Full log files included as separate files showing the following conditions: router dropout, router re-joins and packet transported.**

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| **6=** | **A Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A A 10000 0 |
|  | B E 10005 3 |
|  | C E 10005 5 |
|  | D E 10005 7 |
|  | E E 10005 1 |
|  | F E 10005 4 |
|  |  |
|  | **B** |
|  | **Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A E 10005 3 |
|  | B B 10001 0 |
|  | C F 10004 2 |
|  | D F 10004 4 |
|  | E E 10005 2 |
|  | F F 10004 1 |
|  |  |
|  | **C** |
|  | **Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A F 10004 5 |
|  | B F 10004 2 |
|  | C C 10002 0 |
|  | D D 10003 4 |
|  | E F 10004 4 |
|  | F F 10004 1 |
|  |  |
|  |  |
|  | **D Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A F 10004 7 |
|  | B F 10004 4 |
|  | C C 10002 4 |
|  | D D 10003 0 |
|  | E F 10004 6 |
|  | F F 10004 3 |
|  |  |
|  | **E** |
|  | **Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A A 10000 1 |
|  | B B 10001 2 |
|  | C B 10001 4 |
|  | D B 10001 6 |
|  | E E 10005 0 |
|  | F B 10001 3 |
|  |  |
|  | **F** |
|  | **Current DV state:** |
|  | **Destination Next router Port Cost** |
|  | A B 10001 4 |
|  | B B 10001 1 |
|  | C C 10002 1 |
|  | D D 10003 3 |
|  | E B 10001 3 |
|  | F F 10004 0 |