CSE4/589: PA3 Description

CSE 489/589 Spring 2016
Programming Assignment 3
Software Defined Routing

Due Time: 05/06/2016 @ 23:59:59 EDT

1. Objective

*Implement a simplified version of the **Distance Vector** routing protocol over simulated routers.*

2. Getting Started

2.1 Distance Vector routing algorithm
Text Book: Page 371-377

2.2 Install the PA3 template
Read the document at https://goo.gl/OvNTSs in full and install the template.
You should complete this step before you start working on your implementation.
It is mandatory to use this template.

3. Implementation

3.1 Programming environment
You will write C (or C++) code that compiles under the GCC (GNU Compiler Collection) environment. Furthermore, you should ensure that your code compiles and operates correctly on the following CSE servers:

1. stones.cse.buffalo.edu
2. euston.cse.buffalo.edu
3. embankment.cse.buffalo.edu
4. underground.cse.buffalo.edu
5. highgate.cse.buffalo.edu

Your code should successfully compile using the version of gcc (for C code) or g++ (for C++ code) found on the CSE servers and should function correctly when executed.

NOTE: You are NOT allowed to use any external (not present by default on the above servers) libraries for any part of the programming assignment. Bundling of code (or part of it) from external libraries with your source will not be accepted either. Further, your implementation should NOT invoke any external binaries (e.g., ifconfig, nslookup, etc.).

3.2 Sockets
- Use the **select() system call** only for handling multiple socket connections. Do not use multi-threading or fork-exec.
- Use select timeout to implement multiple timers. **Any other implementation of multiple timers will not be accepted.**

3.3 Running your program
Your program will take 1 command line parameter:

1. The first parameter is the port number on which your process will listen for control messages from the controller (see section 6.3). In the rest of the document, this port is referred to as the **<control port>**.

E.g., if your executable is named router:
- To run with control port number 4322

./router 4322

3.4 CSE Servers
For the purpose of this assignment, you should only use (for development and/or testing) the directory located at **/local/Spring_2016/<Your-UBIT-Name>/** on each of the 5 servers listed in section 3.1. Change

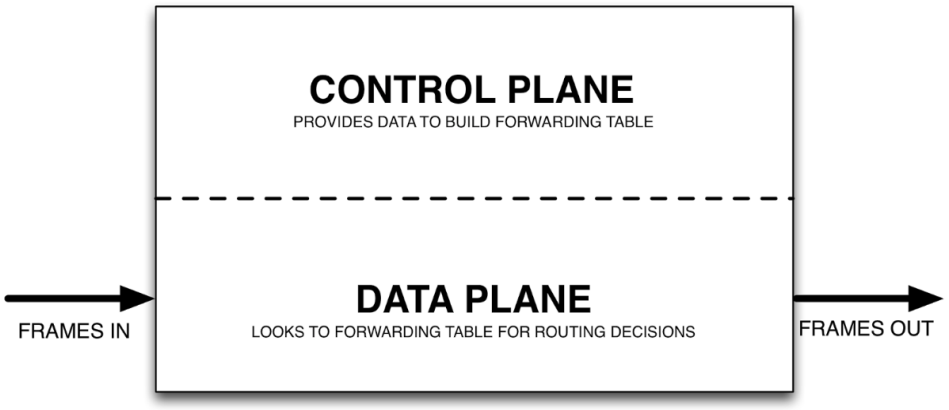
the access permission to this directory so that only you are allowed to access the contents of the directory (*chmod –R 700 /local/*Spring_2016/<*Your-UBIT-Name>/*). This is to prevent others from getting access to your code.

4. Packet Format

We will use automated tests to grade this assignment. The grader, among other things, will also look at the *data*, *routing* and the *control-response* packets generated by your program on the receipt of *control* packets. Towards this end, **ALL the *required* packets**, (as described in sections 6 & 7) generated by your program need to **follow a specific format**. Later sections provide the exact structure of the routing, control, control-response packets, and data packets, **which needs to be strictly followed.**

5. Detailed Description

Routers are one of the most important elements in any network, responsible for forwarding packets to the next-hop in the network and eventually to their correct destination. To be able to perform this critical task, routers rely on the forwarding table. The forwarding table itself is constructed using the distance vector routing protocol. A router's operations are divided into two dimensions: *Data Plane* and *Control Plane* (Figure 2).



Graphics Source: http://blogs.salleurl.edu/data-center-solutions/2016/03/sdn-the-latest-trend-in-data-centers-e2016/

This assignment requires the implementation of a very basic, simplified router, able to perform the functions described below:

● CONTROL PLANE
The distance vector protocol will be implemented to run on top of the servers (behaving as routers) on a pre-defined port number. In the rest of the document, this port is referred to as the **<router port>**. Note that this is different from the control port number passed as argument during startup. Your implementation will in fact emulate a router at the application layer and use **UDP** as the transport layer protocol to exchange routing messages. Further, it will not modify the actual kernel routing table on the servers, but rather maintain a separate one. Lastly, you only need to implement the basic algorithm: **count to infinity, NOT poison reverse**. Since we are limited to the five CSE servers (mentioned in section 3.1), we will at most have a network consisting of five routers.
In addition to the router functionality, your implementation will listen for and respond to *control messages* (see section 6.4) from a **controller** (see section 6.3), that will itself run on one of the five CSE servers. The router, when launched, should start listening for **TCP** connections on the control port number passed as argument and respond to control messages. The control messages could either require functions to be performed on the router, or require it to send a response back (over the established TCP session) to the controller, or both.

● DATA PLANE
Once forwarding tables stabilize, data packets can be routed between a given pair of source and destination routers. On the receipt of a specific control message, a router should read and packetize a given file using a fixed packet structure (see section 7.1) and forward it to the next hop router by looking at the forwarding table. The router that receives this packet should then forward it further and so on, until the packet reaches its destination. All data plane packets will use **TCP** on a pre-specified port number. In the rest of the document, this port is referred to as the **<data port>**.

5.1 Control and Data Plane Dual Functionality
When launched, your router application will listen for control messages on the control port. The very first control message received (of type INIT) will actually initialize the network. Apart from the list of neighbors and link-costs to them, it will contain the router and data port numbers. After the receipt of this initialization message, the application should start listening for routing updates/messages on the router port and data packets (that may require forwarding) on the data port. From this point onwards, the application will listen for messages/connections on the router (UDP), control (TCP), and data (TCP) ports **simultaneously** (using the ***select()*** system call) and respond to those messages (if required).

5.2 Packet byte-order
All packets will use the **network** (big-endian) byte order for all multi-byte fields.

5.3 Topology
After launching your application on the two or more CSE servers, the controller can be launched by supplying a topology file that it will use to build the network. The topology file contains an entire snapshot of the network. The controller, after reading the topology file, will send out the required INIT messages to each of the routers in the network.

The topology file is structured follows:

- The 1st line contains a non-zero positive integer ***n***, the number of routers in the network.

- The next *n* lines contain the ID, IP address, control port, router port and data port number of each router in the network, formatted as:
<ID> <IP address> <control port> <router port> <data port>

Notes
➔ Entries in a line are separated by a single space (‘ ‘) character
➔ There is no space character after last entry in a line
➔ Lines themselves are separated by a single newline (‘\n’) character

- The next ***l*** lines contain the links existing between the routers and their cost, formatted as:
<router ID 1> <router ID 2> <cost>

Notes
➔ Entries in a line are separated by a single space (‘ ‘) character
➔ There is no space character after last entry in a line
➔ Lines themselves are separated by a single newline (‘\n’) character
➔ All links are **bidirectional**
➔ All link costs are **symmetrical**

E.g., Consider the topology in Figure 1.

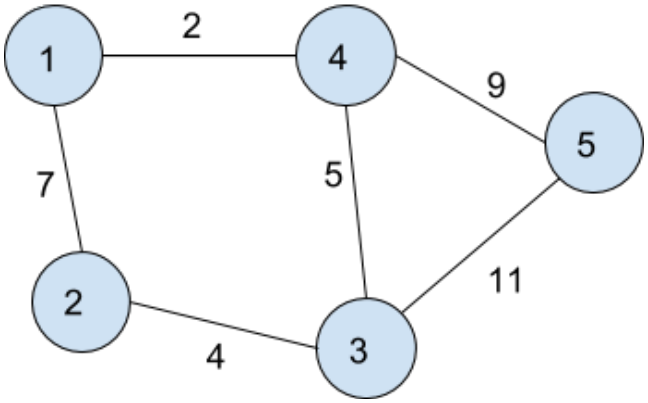


Figure 1: Example topology

Line #	Line entry
1	5
2	1 128.205.36.34 4091 3452 2344
3	2 128.205.35.46 4094 4562 2345

4	3 128.205.36.33 4096 8356 1635
5	4 128.205.36.35 7091 4573 1678
6	5 128.205.36.36 7864 3456 1946
7	1 2 7
8	4 5 9
9	1 4 2
10	3 4 5
11	3 2 4
12	3 5 11

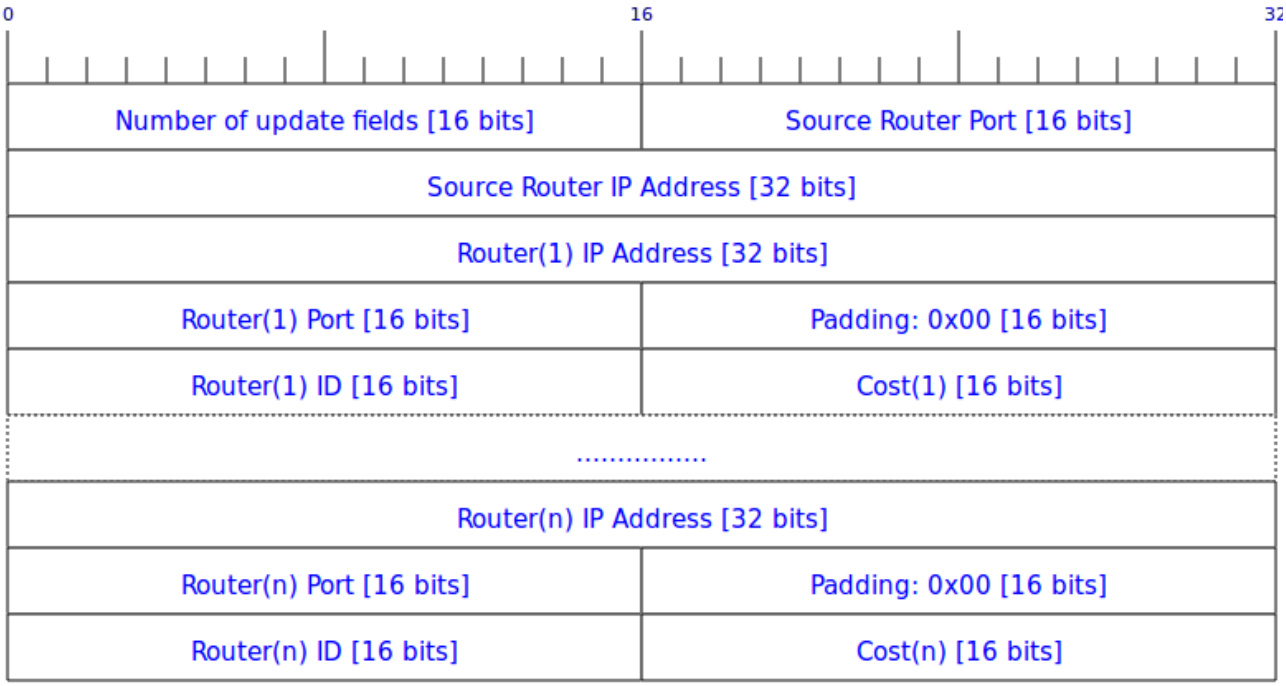
Actual topology files will contain only the Line entry part (2nd column, see topology_example file included in the template). You can use your own topology files to test your code. However, we will use our topology files to test your program.

5.4 Numeric Types
All numeric fields (ID, cost, port numbers, control and response codes etc.), unless mentioned otherwise, should be represented/encoded as their **unsigned** versions.
INF (infinity) value: Largest unsigned integer that can be represented in 2 bytes.

6. Control Plane

6.1 Routing Updates
Routing updates (distance vectors) should be sent *periodically* by a router to all its neighbors (directly connected to the router) based on a time interval specified in the INIT control message, using the packet format described in section 6.2. All routing updates will be sent over UDP. Your application should NOT send routing updates to routers in the network that are not directly attached to it.
Routers can also be removed from the network as a result of receipt of certain control messages. When a router has been removed from the network, it should no longer send routing updates to its neighbors. Further, when a router does not receive distance vector updates from its neighbor for **three consecutive update intervals**, it assumes that the neighbor no longer exists in the network and makes the appropriate changes to its routing table (set link cost to this neighbor to **infinity** but do NOT remove it from the routing table). This information is propagated to other routers in the network with the exchange of routing updates. Please note that, although a router might be specified as a neighbor with a valid link cost in the topology file, the absence of three consecutive routing updates from that server will imply that it is no longer present in the network.
Also note that each router uses a **different timer** for each neighbor (to check for updates from that neighbor) which is *set for first time when the router receives* the first distance vector update message from that neighbor. E.g., if the update period is 3 sec and a router receives the first update from neighbor *n1* at *t=2*s and from neighbor n2 at *t=4s*, then it will expect future updates from *n1* at *t=5, 8, 11, …* and from *n2* at *t=7, 10, 13, …* . Remember that you have to use **select()** to implement multiple timers.

6.2 Routing Update Packet Format



Packet structure

- Number of update fields: Number of router entries that follow
- Source Router Port: **<router port>** of the router sending this packet (src router)
- Source Router IP Address: **<IP address>** of the router sending this packet
- Router(x) IP Address: **<IP address>** of the x^{th} router in the forwarding table of the src router
- Router(x) Port: **<router port>** of the x^{th} router in the forwarding table of the src router
- Router(x) ID: **<ID>** of the x^{th} router in the forwarding table of the src router
- Cost(x): **<cost>** estimate of the **path** from the src router to the x^{th} router in the routing table of the src router

Notes

➔ Routers can be listed in any order inside the packet
➔ Total number of entries (n) should be equal to the number of routers in the network, including an entry to self with cost 0.

6.3 Controller

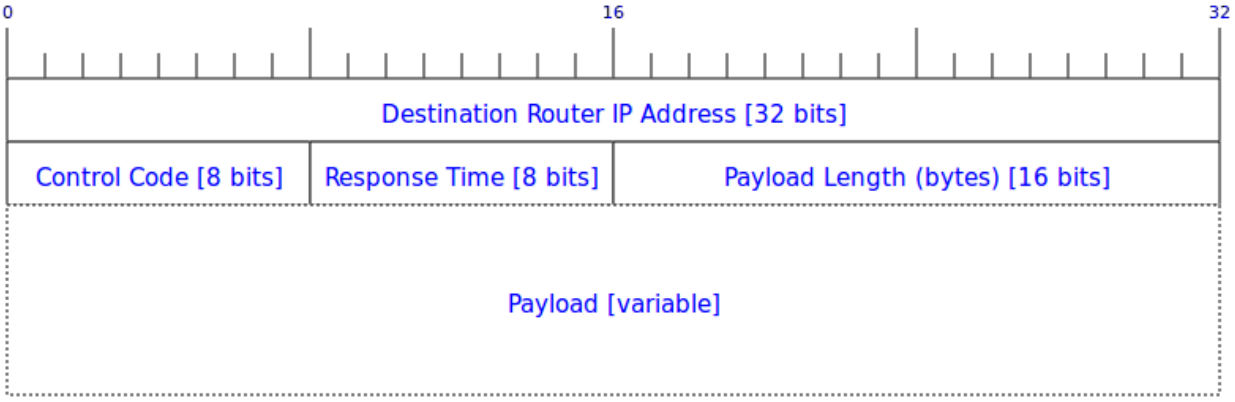
The controller is a separate application that generates control messages for the router application. We will provide the controller application and it need not be implemented. It sends control messages to the routers and expects a response within a time-limit. The controller will connect to a router on the router's control port and may send any number of control messages or even none after establishing the connection. Further, the controller can terminate the connection to a router any time and initiate a new one sometime in the future. The router should always be ready to accept a connection and listen for control messages on the control port. The controller itself is stateless and does not retain any information across runs.

6.4 Control Messages and Response

Control messages generated by the controller and responses generated by the routers will use the packet formats described below. These control messages can be generated by supplying the required command line arguments to the controller application provided with the template. The router application will need to parse the messages and generate response messages which will then be read by the controller. The response message should be sent after completing all the operations required for a command, unless otherwise stated.

Both the control messages and control-response messages will use the **header** format described below. The **payload** for the packets will be different for each control message and its response is listed separately for each control message below. The payload may very well be empty (0 bytes). If there is no description of the payload for a control message or its corresponding response, you can assume it to be empty. In such cases, you should still expect/respond with just the header of the packet.

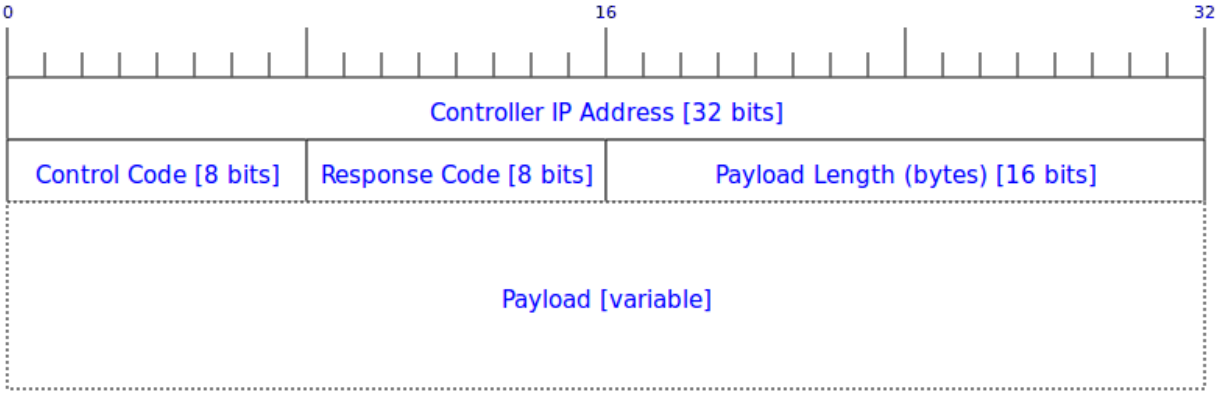
6.4.1 Control Message Header



Packet structure

- Destination Router IP Address: **<IP address>** of the router to which the control message is addressed
- Control Code: A 8-bit unsigned integer indicating the type of control message. The control code for each type of message is mentioned below next to name of the message
- Response Time: Expected response time (in seconds) within which the controller expects a response from the router
- Payload Length: Size of the payload in bytes, excluding the header fields

6.4.2 Control-Response Message Header



Packet structure

- Controller IP Address: **<IP address>** of the controller
- Control Code: A 8-bit unsigned integer indicating the type of control message to which the response was generated.
- Response Code: 0x00 is treated as success. Any other value is error. Controller will discard a packet with an error code
- Payload Length: Size of the payload in bytes, excluding the header fields

6.4.3 Control/Control-Response Payloads

- *AUTHOR* [Control Code: 0x00]

Control-Response Payload

I, <ubitname>, have read and understood the course academic integrity policy.

Notes

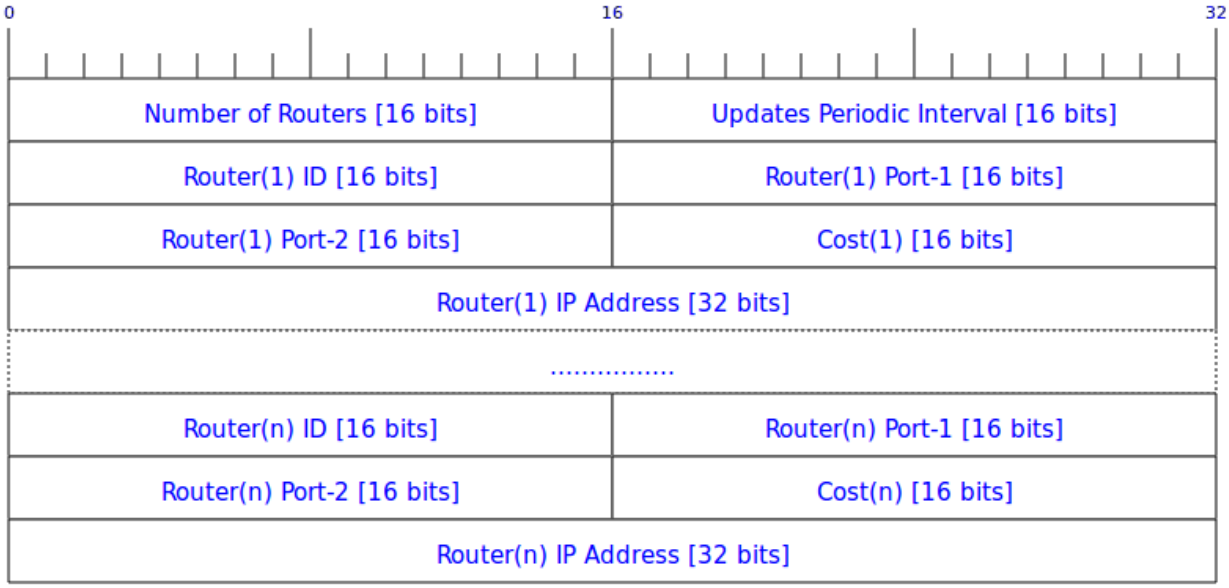
➔ **<ubitname>** is your UBIT-Name in lower case
➔ Both the **<ubitname>** and the academic integrity statement will be ASCII coded

Your submission will not be graded if the AUTHOR response is incorrect.

- *INIT* [Control Code: 0x01]

Contains the data required to initialize a router. It lists the number of routers in the network, their IP address, router and data port numbers, and, the initial costs of the links to all routers. It also contains the periodic interval (in seconds) for the routing updates to be broadcast to all neighbors (routers directly connected).
Note that the link cost value to all the routers, except the neighbors, will be INF (infinity). Further, there will be an entry to self as well with cost set to 0.
The router, after receiving this message, should start broadcasting the routing updates and performing other operations required by the distance vector protocol, immediately.

Control Payload



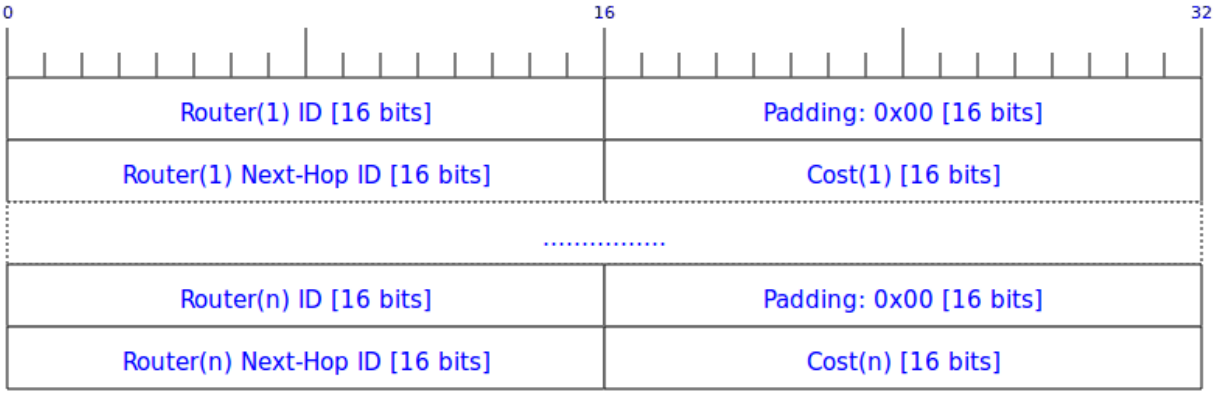
Packet structure

- Number of Routers: Number of routers in the network
- Updates Periodic Interval: Time (in seconds) to be used between two successive routing update packet broadcast
- Router(x) ID: **<ID>** of the x^{th} router
- Router(x) Port-1: **<router port>** of the x^{th} router
- Router(x) Port-2: **<data port>** of the x^{th} router
- Cost(x): **<cost>** of the **link** to the x^{th} router
- Router(x) IP Address: **<IP address>** of the x^{th} router

- *ROUTING-TABLE* [Control Code: 0x02]

The controller uses this to request the current routing/forwarding table from a given router. The table sent as a response should contain an entry for each router in the network (including self) consisting of the next hop router ID (on the least cost path to that router) and the cost of the path to it.

Control-Response Payload



Packet structure

- Router(x) ID: **<ID>** of the xth router
- Router(x) Next-Hop ID: **<ID>** of the next-hop router in the shortest path to the xth router
- Cost(x): **<cost>** of the **path** to the xth router

Notes

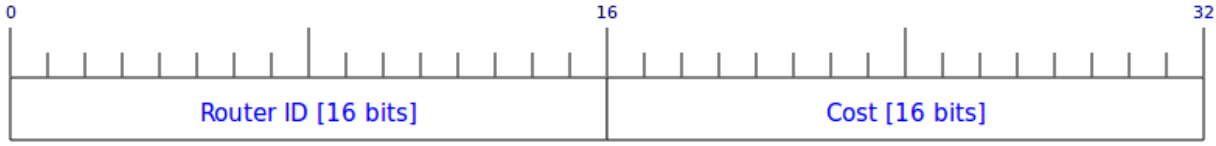
➔ The number of entries should be equal to the number (n) of routers mentioned in the INIT control message
➔ One of the entries should be to self with a cost of 0
➔ If the cost of path to a router is INF (infinity), the next hop router ID would also be INF in such a case

If you do not implement the ROUTING-TABLE response correctly, most automated tests will fail.

- *UPDATE* [Control Code: 0x03]

The controller uses this to change/update the link cost between between the router receiving this message and a neighboring router. Since in our network, links are symmetric, this control message will always be sent in pairs to both the routers involved in a link.

Control Payload



Packet structure

- Router ID: **<ID>** of the router to which the link cost is to be updated
- Cost: New **link <cost>** to the router

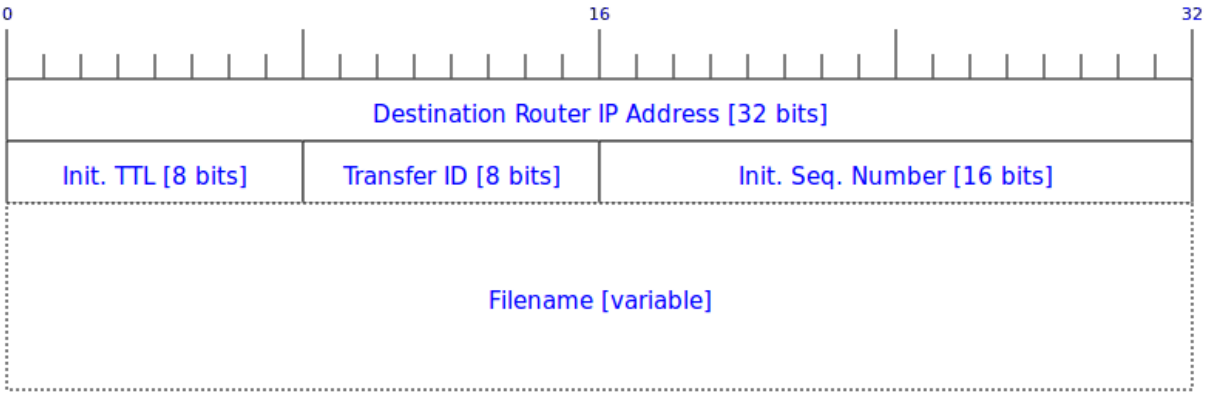
- *CRASH* [Control Code: 0x04]

The controller uses this to simulate a crash (unexpected failure) on a router. On receiving this message, the router should exit immediately. Note that you should not send messages to any routers in the network regarding the crash. They will discover this and update their routing tables, in due time, on their own. **Note that in this case you have to send the response message before exiting.** Once a router exits, it will not be respawned.

- *SENDFILE* [Control Code: 0x05]

The controller uses this to initiate a file transfer from the router receiving this message to any other router in the network. See section 7 for details on how to packetize and route the packets.
The file to be sent will reside in the same folder as the binary executable of the router application. The receiving router should store the file in the same folder as the executable, with the name **file-<Transfer ID>**. Your implementation should be able to transfer both text and binary files. You can assume the maximum file size to be **10 MiB**. In this case, you have to send the response message after the packet with the FIN bit set (last packet) has been sent to the next hop.

Control Payload



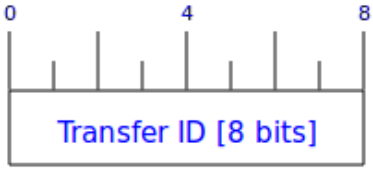
Packet structure

- Destination Router IP Address: **<IP address>** of the router to which the file is to be sent
- Init. TTL: Initial TTL value to be used in data packets (see section 7)
- Transfer ID: Unique number identifying the transfer/flow
- Init. Seq. Number: Initial Sequence number to used in data packets (see section 7)
- Filename: Name of the file (ASCII encoded) to be sent

- *SENDFILE-STATS* [Control Code: 0x06]

The controller uses this to get statistics from the routers involved in a file transfer about the routed data packets. To be able to respond correctly to this control message, each router should store the **TTL** and **Sequence number** of each data packet that it sends/routes. These statistics need to be maintained **per Transfer ID**, on all routers including the source and destination router involved in a transfer.
Unless you are attempting BONUS, there will be only one active data transfer/flow at any given time, in the network.

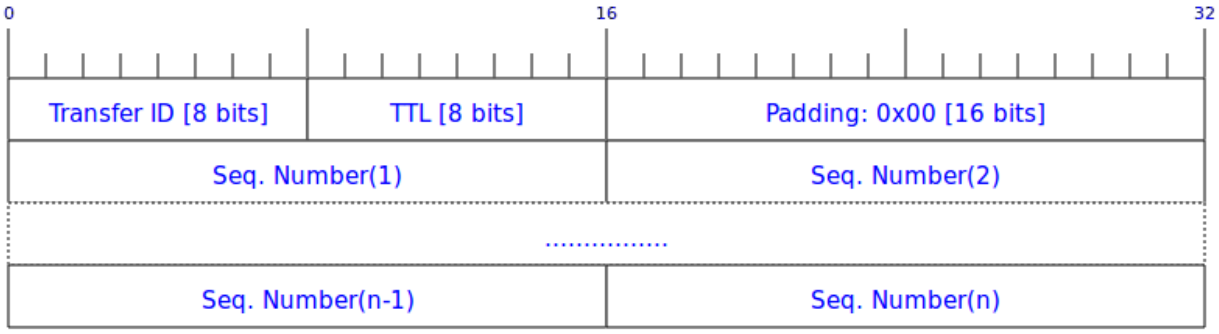
Control Payload



Packet structure

- Transfer ID: **<ID>** of the transfer/flow for which the statistics are requested

Control-Response Payload



Packet structure

- Transfer ID: **<ID>** of the flow for which the statistics are provided
- TTL: TTL value of routed packets after the decrement operation (see section 7)
- Seq. Number(x): Sequence Number of the x^{th} packet routed/sent

- *LAST-DATA-PACKET* [Control Code: 0x07]

The controller uses this to get a copy of the last data packet that was sent/routed through the router. As

a response to this control message, the router will put the last sent/routed packet as payload inside the control-response packet and send it to the controller over the control channel (connection established involving the control port). Note that in order to respond to this control message correctly, each router will always keep a copy of the last packet that it sends/routes which can then be sent to the controller when requested.

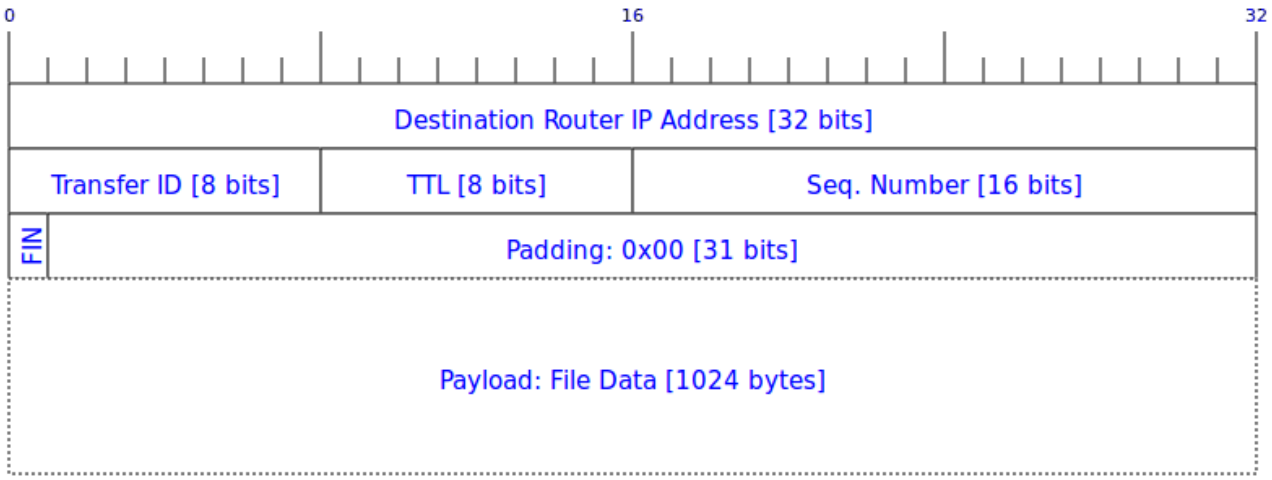
- *PENULTIMATE-DATA-PACKET* [Control Code: 0x08]

The controller uses this to get a copy of the second last data packet that was sent/routed through the router. As a response to this control message, the router will put the second last sent/routed packet as payload inside the control-response packet and send it to the controller over the control channel (connection established involving the control port). Note that in order to respond to this control message correctly, each router will always keep a copy of the second last packet that it sends/routes which can then be sent to the controller when requested.

7. Data Plane

In a typical deployed network, routers will never generate data packets of their own but only route packets generated by end-hosts. However, for this assignment, routers can indeed generate data packets (on receipt of SENDFILE control message), in addition to routing them.
On receiving the SENDFILE control message, the sending (source) router will need to read the file from the disk and packetize it using the packet format described in section 7.1. The destination (sink) router will need to reconstruct the file back from the received data packets and write it to disk.
The SENDFILE control message along with the **<Filename>** of the file to be transferred, specifies a TTL and an initial sequence number value **<seq>**. The source router will put the TTL value in each packet generated. The first packet for a given transfer will have sequence number **<seq>**, the second one **<seq>+1** and so on.
All data plane packets will be sent over TCP connections on the **<data port>** specified in the INIT control message. The routing procedure consists of decrementing the TTL value (by 1) of the data packet received, looking up the next hop router for the destination mentioned in the packet in the forwarding table and forwarding it. The same procedure is repeated at each router, until the packet reaches its destination (the sink router). If a packet's TTL reaches 0 after the decrement operation, it should be dropped and not routed further. Other than the TTL, a router should not modify any other fields in the packet in any way.

7.1 Data Packet Format



Packet structure

- Destination Router IP Address: **<IP address>** of the router to which the packet is destined
- Transfer ID: Unique number identifying the transfer/flow
- TTL: TTL value
- FIN: This bit is set (1) for the last data packet of a given transfer/flow/file sent from the source, unset (0) otherwise
- Sequence Number: Sequence number

Notes

➔ File sizes used will always be a multiple of 1024 bytes

8. BONUS: Multiple File Transfer Flows

Implement additional functionality to allow for multiple file transfers/flows at the same time (simultaneously). Each transfer/flow will have a unique **<Transfer ID>** and can be between any two routers.
If you attempt the BONUS, add an empty file named BONUS inside the src/ directory in your source tree.

9. Grading and Submission

The grading will be done using automated tests. Any deviation from the packet format/syntax described in previous sections will cause the tests to fail. For a detailed breakup of points associated with each command/functions, see https://goo.gl/K3VDuF
For packaging and submission, see the section **Packaging and Submission** in https://goo.gl/OvNTSs