CS 425: Assignment 2

April 12, 2021

General instructions:

1. **Some template codes are provided. Do NOT use any libraries beyond what is given in the template. Marks may be deducted based on the functions of the additional libraries used.**
2. **You must submit code that compiles (on GNU C/C++ compiler, or Python3), otherwise your assignment will not be graded.**
3. **You are not allowed to exchange codes with the other teams or to copy code from any source.**

1 Distributed Bellman-Ford Shortest Path Routing Algorithm

Given a network of n routers, you are asked to find the shortest path from a given router to all the other routers using Bellman-Ford shortest path routing algorithm. Network topology is provided in the `Topology.txt` file.

`Topology.txt` **file format** :

```
Router IP Address
Number of Neighbours
Neighbour1-IP    Cost
Neighbour2-IP    Cost
...
...
End
```

Information about the `Topology.txt` file: The first line of the file indicates a router IP address (say '10.1.2.10'). The second line indicates the number of neighbors for above-mentioned router (say 2). Based on the number of neighbors mentioned above (i.e., 2), the consecutive lines (lines 3,4) are reserved for the information related to the neighbor router's IP address and the corresponding cost value separated by a 'tab' space (e.g., 10.1.4.10<tab space>9). A line with 'End' indicates the end of the information for a single router. The `Topology.txt` file contains n such routers information specified in the same format.

Note:

- The format of `Topology.txt` file need not to be validated.
- The given network is **un-directed**; so, the same edge appear twice in the `Topology.txt` file.

Tasks:

1. Write a function `distanceFinder()` that takes router IP address as input and outputs the shortest path cost from source router to every other router using Bellman-Ford shortest path algorithm.
   Output Format: Source IP Address, Destination IP Address, Cost, Next Hop IP Address
   Example Output:
   10.1.2.10, 10.1.2.10, 0, 10.1.2.10
   10.1.2.10, 10.1.3.10, 2.0, 10.1.3.10
   10.1.2.10, 10.1.5.10, 1.0, 10.1.5.10

2. Consider a router in the given topology and you are asked to increase the cost to reach one of its neighbor router by a specific unit (say d).
   Input example: 10.1.3.10, 10.1.4.10, 10, 10.1.2.10
   An explanation for Input format: Cost between router 10.1.3.10 to one of its neighbour 10.1.4.10 is increased by 10 units and your task is to find the Bellman-Ford shortest path from router 10.1.2.10 to all the other routers with the updated cost values.
   As a keen observer, you should differentiate the cost values after the update. i.e., output format should be indicated at two cases.

   (a) Case-1: Considers the case where the cost value is same even after the update.
   (b) Case-2: Considers the case where the cost value is different after the update.

   Write a function `updateRouterCost()` to implement the above-mentioned logic and print the following for cases 1 and 2 respectively.
   Output Format:
   Case-1
   Source IP Address, Destination IP Address, Cost, Next Hop IP Address
   – – – –
   Case-2
   Source IP Address, Destination IP Address, Updated Cost, Next Hop IP Address
   – – – –

3. To add dynamicity for the existing network topology, you are asked to write a function `addRouter()` for adding a new router to the existing network. This function should take input parameters values (New Router's IP Address,{list of Neighbours},{Cost Corresponding to Neighbours}, Source IP) and outputs the shortest path cost from Source IP to every other router after adding the new router to the network.
   Input example: 10.1.6.10,{10.1.5.10, 10.1.4.10}, {3,4}, 10.1.2.10
   An explanation for Input format: A New router 10.1.6.10 is added to the topology and 10.1.5.10,10.1.4.10 are its neighbours. The cost between router 10.1.6.10 and 10.1.5.10 is 3 units and similarly the cost between router 10.1.6.10 and 10.1.4.10 is 4 units. Your task is to find the Bellman-Ford shortest path from router 10.1.2.10 to all other routers after updating the topology.
   Output Format:
   Source IP Address, Destination IP Address, Cost, Next Hop IP Address
   – – – –
   – – – –

Note: We will consider a different network topology while testing. Please make sure that your code can parse any topology given in the prescribed format.

2 Go-Back-N implementation

In this problem, you have to implement Go-Back-N automatic repeat request (ARQ) protocol to transfer a given file in packets using socket programming and carry out some experiments to evaluate its performance.

In the `Q2` folder in the accompanying .zip file you have two folders (`rec` and `sen`). The folder `sen` contains a python file `sender.py` and a `FileToTransfer.txt` file. The folder `rec` contains a python file `receiver.py`. You have to update the code in the `.py` files to connect the sender and the receiver (as explained in the following part of the question) and implement Go-Back-N ARQ to transfer the data from sender to receiver. Note that all the data transfer are only from the sender to the receiver – only ACK packets travel from receiver to sender. In the end, we will have a copy of `FileToTransfer.txt` with the same name in folder `rec`.
The things to do in this question (which will be evaluated) are explained in the following subsections. The summary of things to do and the rules for evaluation are given in the last subsection.

2.1 `sender.py`

- The given code contains:
  - the created Sender class and an object with name 'Sender'. Asks the user to enter the window size, and time out and stores them.
  - `connection establishment:` the created sockets in main function to initiate file exchange using accept(). `sender.py` shares the window size, the timeout, and the name of the file to transfer, with the receiver. The code for this part from the receiver's side is already there in `receiver.py`.
  - a function `SendPacketsFromFile()` to send packets from the file, which
    1. reads the file and divide the data in chunks to send.
    2. sends all the data chunks using the function `SendMessage()`

    Please do not update the function `SendPacketsFromFile()`.
- The template of the function `SendMessage()` is written in the provided code. **To do: update the function `SendMessage()` as explained in the comments in the code.**
- The function `SendMessage()` calls three partially written functions: `makePack()`, `acc_Acks()` and `resend()`. **To do: write and update these functions as explained in the comments in the code.**
- Some useful functions are already written in the code. Do not edit these function. You can use them wherever required. Their description is as follows (the exact name of the function and of the parameters is written here, to avoid confusion):
  - `canAdd(self)`: checks if a packet can be added to the send window, returns 0 if the the window is full.
  - `check_sum(data)`: returns the encoded data in hexadecimal format.
  - `divide(self, data, num)`: divides the `data` in chunks of size `num`.
  - `addAndSend(self, pack)`: adds a packet to the window, sends it to the receiver and updates the variables.
  - `separateChunks(self, message)`: There is a possibility that the TCP packages are merged, before the sender reads them from the pipe. If the message received by the sender (using `socket.recv(1024)`) has concatenated packets sent by the receiver, then this function can be used to separate them and this function returns a list of separated packets.
- To test that the Go-back-N protocol works correctly, we use probabilistic error. For every packet sent from sender to receiver, a random number using `random.random(0,100)` is generated, added with the packet, and the packet is sent to the receiver. If the randomly generated number is (< 50), then we assume that the packet is corrupted or lost and hence the receiver must discard it. (also mentioned in

comments in sender.py in function `makePack()`, and is taken care of in `receiver.py` as well). You do not have write or edit code in `receiver.py` for this bullet point. This point is written only to increase clarity, ease in understanding the codes and the comments, and testing.

2.2 `receiver.py`

This file contains the complete code for receiver's side. You do not have to edit or update this file. The receiver sends the packets in format

```
Checksum of the Sequence Number/////Sequence number/////Type
```

where, Type can be `ACK` (acknowledgement), or `NAK` (negative acknowledgement).

2.3 Evaluation

You have to add lines of codes in four partially written functions in `sender.py` to implement Go-back-N ARQ protocol.

1. `SendMessage()`
2. `makePack()`
3. `acc_Acks()`
4. `resend()`

You can break the tasks of these functions and make new functions (add appropriate comments in that case). Remember to encode and decode the message (to/from the byte format from/to string) to be transferred, using `message.encode('utf-8')` and `message.decode('utf-8')`. The tasks to be done by the combination of these functions will be evaluated individually using hidden test cases, and are as follows:

1. We should get the last print line of the function `addAndSend()` on console for each packet. This line prints the packet (in the original form) sent to the receiver, after it is sent. The packet has the format:

   ```
   Checksum of the data/////Sequence number/////Length of data/////Data in byte
   format/////A number generated using random.randint(0,100)
   ```

   Test case: The input file `FileToTransfer.txt`

2. If the last packet sent by the receiver contains `NAK`, print the last packet sent by the receiver on console as `The received packet from receiver is: xxxxx` (xxxxx is the received packet), and the sender should call `resend()` to resend all the packets in the current window.

   Test case: The received packet is `c81e728d9d4c2f636f067f89cc14862c/////2/////NAK`

3. If the last packet sent by the receiver contains `ACK`, the sender should slide the window for packet sequence(s). Print the last packet on console as `The received packet from receiver is: xxxxx`. Print the variables `last_ack_seqnum, active_spaces, self.window` before the sender receives the `ACK` and after sliding the window.

   Test case: The received packet is `c81e728d9d4c2f636f067f89cc14862c/////2/////ACK`

   And, at the beginning of `acc_Acks(self)`, the variables are:

   ```
   self.window=["a16fe651f7f861c77c48e93c3221f61d/////0/////10/////b'Go-Back-N
   '/////57", "b9fd12d34fdbeff054b0c917df19343c/////1/////10/////b'ARQ is a s'/////30",
   "8381a9c41b6e2213c855aa95dfd97732/////2/////10/////b'pecific in'/////62",
   "4f87807702dc7fd4698c43bea0469e76/////3/////7/////b'stance.'/////52"]
   self.w = 4, self.active_spaces=0, self.last_ack_seqnum = 0
   ```

4. Whenever the sender has to resend the packets in the current window, use function `resend()`. This function should send and print (after the packet is sent) each packet in the form it is sent to the receiver (similar to point(1)).

   Test case: Same as that for point (3).