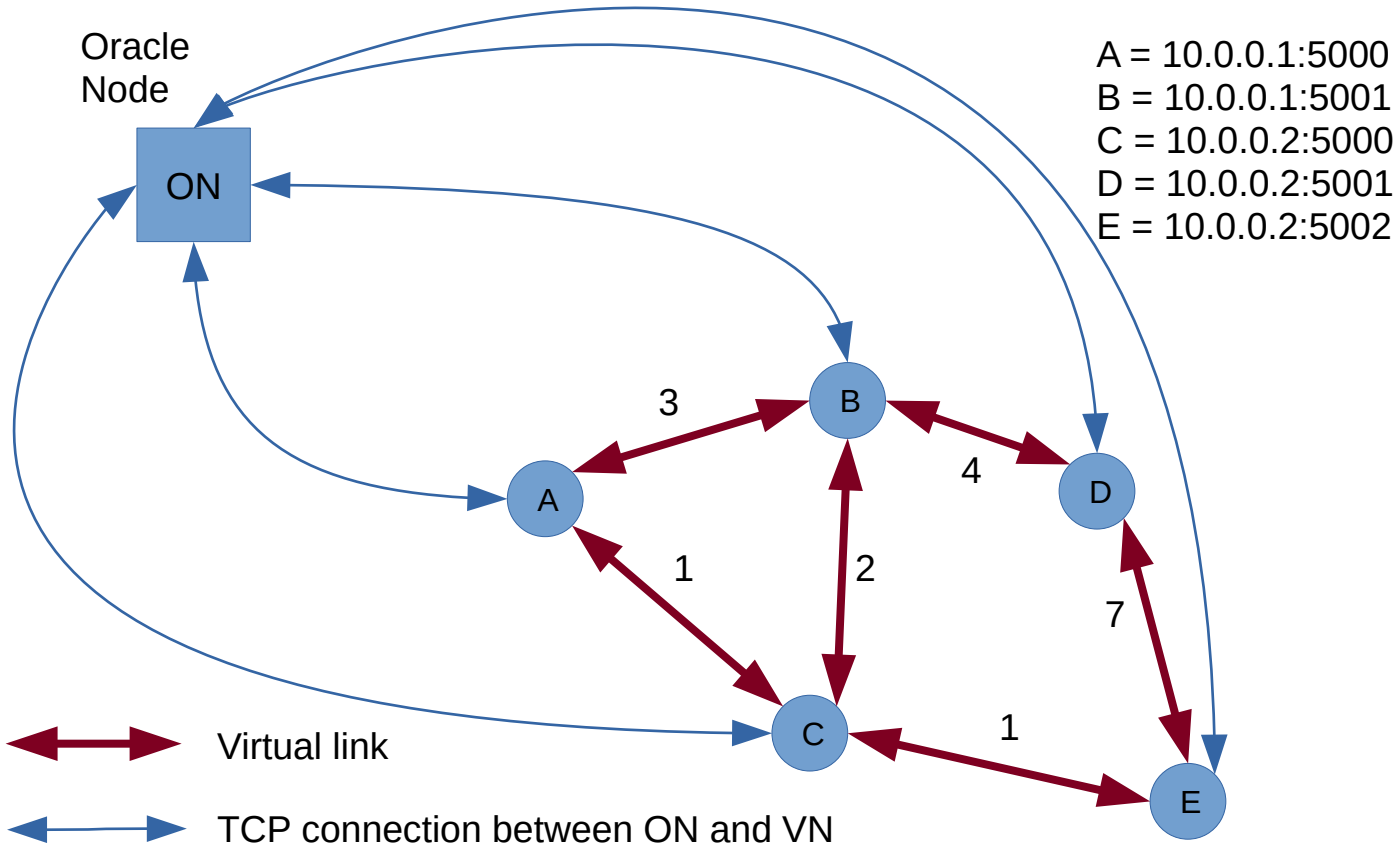
Socket Programming Exercise: Link State Emulator

Overview

- In this socket programming exercise, you will be *emulating* a simple link state protocol.
- It is *emulation*, and not simulation, since you will be sending real messages. It is *emulation*, and not a fully real implementation, since you will be using virtual network nodes.
- In the emulated network, each *virtual node (VN)* is a process, identified by an (IPAddr, Port) tuple. This way you can have multiple virtual nodes per physical machine (your laptop). You can also run a virtual network with virtual nodes spread across multiple physical machines.
- What is the virtual network itself? This will be determined by an *oracle node (ON)*. Each VN has a TCP connection with the ON, and this TCP connection is used to convey *link-state* information from the ON to each VN.
- Note that the use of ON is two-fold: (1) for ease of configuration of each VN, dynamically, instead of having to configure each of them individually via possibly command-line, and (2) you get to experience TCP sockets as well, as part of this exercise.
- After receiving the neighbourhood information from the ON, each VN exchanges link-state messages using UDP sockets, as per a link-state protocol.
- The following figure gives an example virtual network with 5 nodes & link costs as shown.



Implementation constraints, assumptions, advice

- You MUST work in groups of two [if unable to find assignment partner, talk to instructor]
- Ideally, the same group can also work in pair for the next lab on sysad-giri too, but this is not a constraint
- DO NOT use AI tools like ChatGPT. It is in general a *really bad idea* to use these tools *while learning*, since (a) you will not learn, (b) you may learn wrong things since AI tools make mistakes, (c) you are unprepared for when the unsustainable AI bubble will burst
- DO NOT use any http library – use only TCP and UDP sockets
- You can use any programming language – e.g. Java, C++, C, python. My personal preference and recommendation as instructor is in the above decreasing order.
- You MUST use at least two *different* programming languages – one for ON, and one for VN. This will make you better learn network socket programming and the need for protocols. Lite students are allowed to choose only a single programming language. Lite students need to implement only the TCP connection part between ON and VN.
- You can assume that nodes are *named* with capital English alphabets (as in the example) and that there are at most 26 nodes.
- Assume that link costs are symmetric (same in both direction) and are positive integers > 0
- DO NOT use loopback IP address (127.0.0.1 or 0.0.0.0) anywhere as during demo VNs may be split across multiple physical machines
- My recommendation is to implement *single-threaded process* for each of ON and VN, and use *event-driven* sockets using the *select* system call. But this is not a constraint – you can use multiple-threads. But in my experience, multi-threaded socket programs are not only more difficult to debug but also perform less well (you may not experience performance issues in this simple assignment though).
- Beware of nasty loop bugs leading your laptop to freeze – it can happen if you leak memory or leak sockets. The best is to artificially introduce large sleep delays, like 500ms or 1sec, in your main loop. While debugging, you may even make these delays even higher, like10sec.
- Avoid dynamic memory allocation as much as possible – you will be avoiding a major source of bugs and crashes.

Evaluation

- This lab will be worth three weeks of weightage: Thu 02 Oct, Thu 09 Oct and Thu 16 Oct
- Evaluation will be based on demo to TAs
- Either lab partner may be asked to show any part of the demo, and each question will be directed to specific student – so ensure fair participation between both students in team

Operation of Oracle Node

1. Take one command-line argument, the configuration file – example shown in oracle-node-config.txt (see end of this file)
2. Read configuration file to learn network info; while you can assume that the format of the config file is as expected, it is always a good programming habit to put sanity checks
3. Listen for incoming connections from VNs on TCP port 5000
    a) First incoming connection can be assumed to be virtual node A, next one B, and so on
    b) If there are *more* incoming connections that nodes specified in config file, then those additional nodes are all disconnected islands
4. Also constantly (periodically) monitor changes to config file – this part is optional for lite students. This will be used during demo to introduce topology changes.
5. Listen for (app layer) messages in each ON-VN connection
6. In VN-to-ON direction, there is only one type of message called CONNECT whose body will specify the identity (IPAddress, UDPPort) of the VN. Use binary format, not strings in the message body.
    a) For example, in the example network, the CONNECT message from A virtual node will specify (10.0.0.1, 5000)
7. In the ON-to-VN direction, there is only one type of message called LINK-STATE whose body will specify a list of tuples. Like earlier, use binary format, not strings in the message body.
    a) Each tuple will be (NodeAlphabet, IPAddress, UDPPort, Cost).
    b) Exactly one tuple in this message will have zero cost, which will specify the alphabet mapping of a VN
    c) All other tuples will correspond to the neighbours of that VN
    d) For example, in the example network, the LINK-STATE message from ON to virtual node E will specify the three tuples (C, 10.0.0.2, 5000, 1), (D, 10.0.0.2, 5001, 7), (E, 10.0.0.2, 5002, 0).
8. When will VN send CONNECT message to ON ? Right after TCP connection formation.
9. When will ON send LINK-STATE message to VN ? ~~Right after CONNECT message~~ After receiving CONNECT message from N virtual nodes where N is the number of nodes indicated by the configuration file, and after each change to contents of the configuration file

Operation of Virtual Node

1. Initialization
   a) Take IPAddress (and perhaps port) of ON in command-line argument
   b) Also take own UDP port as argument (can take own IP address also as argument or find it programmatically)
   c) Open and bind socket to own UDP port – exit on failure with appropriate error message
2. Interaction with Oracle Node
   a) Make TCP connection to ON at port 5000, soon after start
   b) Send CONNECT message to ON
   c) Listen (constantly/periodically) for LINK-STATE message
   d) On receiving LINK-STATE message, print to STDOUT in human-readable format, the neighbourhood link state. E.g. node E in given example network can print "E=0,C=1,D=7"
3. Lite students need to implement only until the above
4. The other major part of the VN is the *emulated* link-state protocol operation, detailed below (optional for lite students)

Operation of Link-State Protocol

1. You only need to define one kind of message – Link-State-Packet (LSP)
2. This is to be sent and received using the UDP sockets
3. LSP needs to have the following info: originID, sequence number, link-state of originID. Define a suitable *binary* message format for this (not human readable string). Include alphabet-ID of a node in this.
4. For simplicity, you do not need to consider sequence number restarting from 0 (due to node reboot)
5. Also for simplicity, you can assume that LSPs are not lost. UDP packets will rarely be lost in the test/demo scenario, for additional robustness you can simply send each LSP twice or thrice.
6. Note that LSP needs to be sent on each link-state change (new LINK-STATE message from ON), as well as periodically – you can set period to be a small value like 10-20 sec, just for test/demo purposes.
7. Periodically, simply print the currently known graph to STDOUT; format can be similar/same as the ON config file

ON config file example oracle-node-config.txt

```
# Lines beginning with hash are comments, to be ignored


# Empty lines like above and below are also to be ignored


# Config file specifies the graph in adjacency matrix format

# Only right-upper triangle is specified, as link costs are symmetric in either direction

# Cost of -1 implies infinite cost (not a neighbour)

# Each entry is separated from next by a tab, for readability

# Tabs at the beginning of a line are to be ignored (for readability - see below)


# First row is for virtual node A, next for B, and so on

# Example matrix for the example network shown in instructions is below


# Costs from A to B, C, D, E

3       1       -1      -1


# Costs from B to C, D, E; note the tab at the beginning, to overall give view of upper-right triangle

        2       4       -1


# Costs from C to D, E

                -1      1


# Costs from D to E

                        7
```