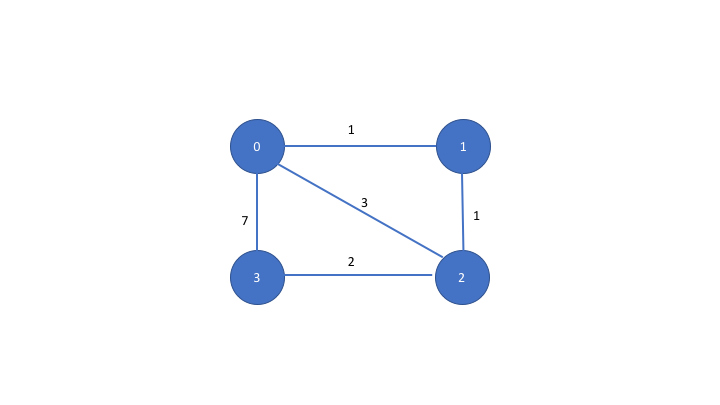
**CS356 Programming Assignment**

This project will be done in phases. The final phase will require four instances of a program running across a network that emulate the operation of a distance-vector routing protocol. Given a network with initial link costs, the output will print out the initial routing table of Router0, and all updates and resulting changes to that routing table based on information from neighboring routers.

**Network Topology**

Network Topology showing four nodes or routers (Router0, Router1, Router2, Router3) with link costs to other routers.



**Implementation Options**

You may use C, C++, Java or Python. CS majors are strongly encouraged to use C or C++ to gain “native” experience with the socket API.

**Computer Systems**

The following computers can be used for running the programs:

* afsaccess1.njit.edu through afsaccess4.njit.edu
* afsconnect1.njit.edu or afsconnect2.njit.edu

**Phase 1 Description**

In this phase, you will implement code for only 2 routers: Router0 and Router1. The code will exchange the initial routing tables between client and server only (processing of the router updates will be done in later phases). Router0 will act as a “client” and Router1 as a “server”. Router0 and Router1 must run on different machines and communicate over the network.

Router0 and Router1 start with an initial routing table based on the topology above. Each router starts off knowing only the link cost of each of its directly connected interfaces. For example, the initial link costs of Router0 to directly connected routers will look as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Destination Routers** | | | |
| **Source Router** | **Router0** | **Router1** | **Router2** | **Router3** |
| Router0 | 0 | 1 | 3 | 7 |

Router1’s initial table can be determined similarly by looking at its directly connected interfaces in the network topology above. Note: It is possible that the link cost is unknown since there may not be a directly connected interface to all routers. Hence, there must be a way in your program to represent an “unknown” link cost.

In Phase 1, you will design a protocol to enable Router0 and Router1 to request and advertise routing updates as follows over the UDP transport protocol.

The protocol message should have a **header** and a **body**. The header will include the following fields: version number, message type (e.g. request or response) and an identifier (to match requests and response messages). The body of the message will include a routing update (which in Phase 1 is just the initial known costs from the source router to all destination routers as shown in the table above). The protocol should be designed to allow a router to **request** a routing update from another router (“pull” model). and the ability for a router to **advertise** its routing update to another router **asynchronously** (“push” model).

Phase 1 includes the following functionality on the Client:

1. Client (Router0) prints its own initial routing table
2. Client (Router0) requests a routing update from the Server (Router1). Router1 responds with its initial table. Client prints Router1’s response.
3. Client (Router 0) advertises its routing update to the Server (Router1) asynchronously

Phase 1 includes the following functionality on the Server:

1. Server (Router1) prints its own initial routing table
2. Server (Router1) waits for at least two messages from Client ( Router 0). This message may be either a request for a routing update from the server, or it may be an asynchronous update (the information in the message header should allow the Server to distinguish between these two cases).
3. Server (Router1) prints the update it received from Router0

Phase 1 Deliverables:

The deliverables must clearly indicate that the client and server are running on two different machines and communicate over the network.

Deliverables must include:

1. Short description of your protocol including message format, message types and message flow
2. Code listing of Client and Server
3. Screenshot showing that the code compiled on the target machine successfully without errors
4. Screenshot of output described above (client and server routing tables)
5. Evidence that messages are being exchanged across the network. This may include using a tool such as wireshark and/or screenshots of trace messages in your program to indicate messages sent and received, including information such as message type, data being sent/received, hostname, IP address and ports used, and timestamps.
6. List References consulted when designing and building your program (if any). Clearly indicate what code is yours versus based on others.

**Academic Integrity:** The [University Code of Academic Integrity](https://www5.njit.edu/doss/sites/doss/files/University%20Policy%20on%20Academic%20Integrity.pdf) will be upheld, and any violations will be brought to the immediate attention of the Dean of Students.

Remember, the program must be your own work. You must give credit where your answers are based on help from others. For example, It is OK to reference socket programming tutorials in textbooks and elsewhere to research socket API details and usage, as long as you give credit, and it is clear what is your own work versus others.