ELEC-A7150 C++ Programming

Network Simulator
(nsLite)

Project documentation
December 16, 2016

Members: Karthika Krishnamurthy, Naveed Mohammed Vaheed, Tahir Irshad, Tan Nguyen

1. Overview

Network simulator models a group of network nodes, and links that connect the nodes. The nodes communicate with each other by sending and receiving packets. Each node can have multiple interfaces and each interface will have its own IP address and MAC address. There are roughly two kinds of nodes: routers, that forward packets intended to other nodes; and end-hosts that run an application that either sends or receives packets (or does both). Sometimes a node can act in both roles. Each link can be described by two basic characteristics: propagation delay, amount of time it takes for the head of the signal to travel from the sender to the receiver. It can be computed as the ratio between the link length and the propagation speed over the specific medium and link cost. Network simulator measures the network performance with the given network say delay measurement, throughput measurement etc.

nsLite is a simple network simulator that allows defining different network setups (i.e., collections of routers and links between them), and features that generate and receive packets from/to all other routers present in the network. The different network setups will be retrieved from the respective xml configuration file. Each network setup can be differentiated by the number of routers, number of links, link costs, link propagation delay and queue size. nsLite will have a graphical UI, showing the network topology. The graphical UI will display statistics of how long does it take for packets to reach their destination i.e. one-way delay and traceroute of the packets, shortest path to the neighbors in the network in terms of link cost and propagation delay (any one) for each simulation. nsLite will also produce logs for ARP, routing table of each router. The input taken from the xml file cannot be modified during the run time of the program. The software does only the delay measurement between various routers present in the network and it does not do any throughput measurements or link utilization calculations.

nsLite simulates the most of the real time network characteristics like:

- Each interface present in the network has unique MAC address and IP address.
- Construction of packet for delay measurements. Each packet has a IP header, L2 header and packet data.
- Shortest path calculation.
- Routing table creation
- ARP table creation.
- Packet routing.

- Changing MAC address during each hop.
- Changing TTL during each hop.
- Checking TTL expiry.
- Parsing of packet.
- Calculating delay using the timestamp.
- Packet sending and receiving on different threads without affecting each other.
- Interface buffers to send/receive packets.

Limitations:

- All delay measurements are done only with 68Bytes (with FCS) packets.
- Only propagation delay of the link and packet processing delay are considered.
- Software measures only one way delay.
- Shortest path algorithms are very primitive considering only one network parameter at a time like link cost or link propagation delay.

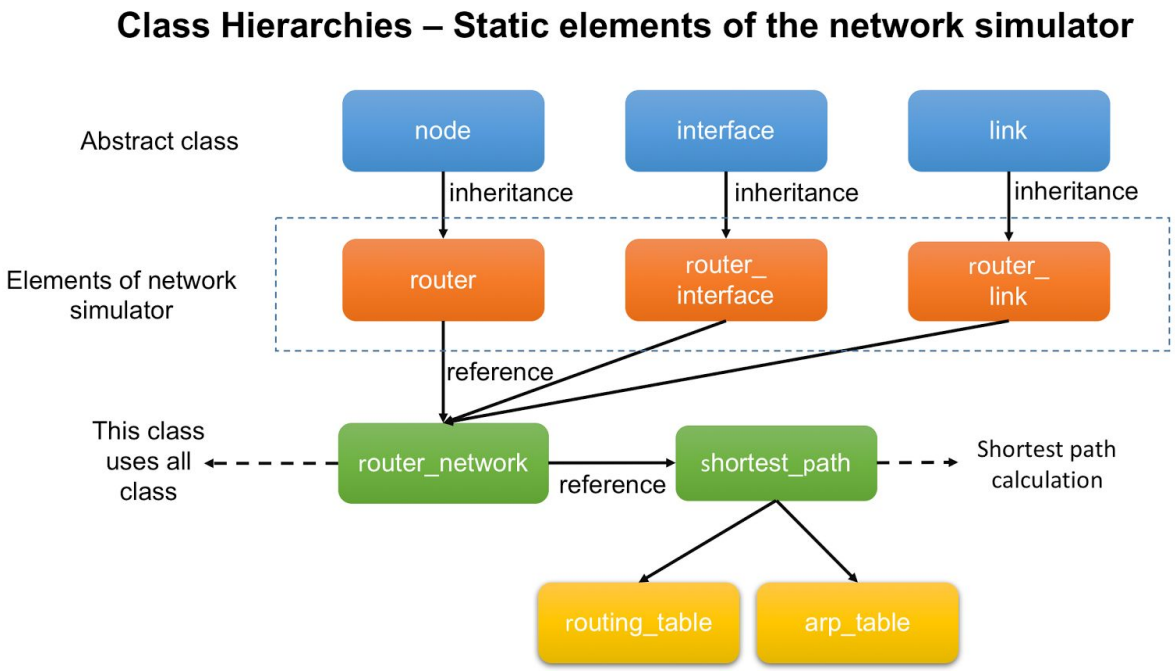
2. Software structure

Network simulator software consists of two elements. They are static element and dynamic element. Static elements involves routers, links, interfaces present in the network. These elements remain constant in the network. The different types of tests run on the static elements (i.e) network are called the dynamic elements of the network. In this nsLite software, delay measurement test is the dynamic element.

Static elements:

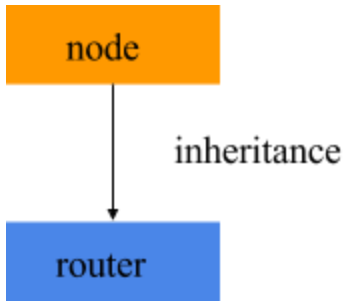
Static element implementation is as follows.
It is broadly categorized into:

1) Routers
2) Interfaces
3) Links
4) Networks



2.1 Node/Routers:

node class is an abstract class as it may denote broader category of elements like router or switch or hosts. Thus router class inherits node class.



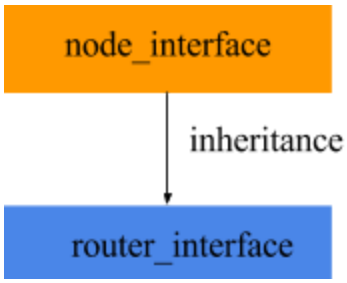
Router class has following private elements:

- ***Router ID***: This member is actually present in the node class and it is available in the router class due to inheritance. This field denotes the unique id of the router.
- ***List of interfaces***: This member is a list which has references to the interface class objects. Initially this list is empty and it grows when a new interface class object is created for a router.
- ***ARP table***: This member has reference to the arp_table class object. This reference is created only after executing the shortest path algorithm.
- ***Routing table***: This member has the reference to the routing_table class object. This reference is created only after executing the shortest path algorithm.

- ***Location***
  This x-axis, y-axis value denotes position of the router in the GUI.

2.2 Interfaces

node_interface is an abstract class which may denote atm interface, ethernet interface, TDM interface. This router_interface class inherits node_interface class. router_interface class denotes ethernet type interfaces.

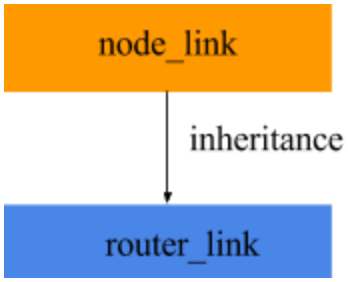


Interface class has following private elements:

- ***Interface ID***: This member is actually present in the node_interface class and it is available in the router_interface class due to inheritance. This field denotes the unique id of the router_interface.
- ***Router ID***: This field denotes the unique id of the router.
- ***MAC address***: This denotes MAC address of the interface and it is a unique value.
- ***IP Address with mask***: This denotes IP address of the interface and it is a unique value and the mask denotes network mask of the IP address.
- ***Queue size***: This denotes number of RX and TX buffers that needs to be allocated for a interface.
- ***Send buffer***: This denotes TX buffers and number of buffers are allocated according to the queue size value.
- ***Receive buffers***: This denotes RX buffers and number of buffers are allocated according to the queue size value.
- ***Interface speed***: This interface speed denotes the supported speed of the interface.

2.3 Links

Similar to other elements, node_link denotes broader category of links like Ethernet link, TDM link and ATM link. router_link inherits node_link class. router_link denotes Ethernet links.



Link class has following private elements:

- ***Link ID***: Unique value that denotes the link present in the network.
- ***Link cost***: It denotes the cost of the link which is used the shortest path calculation.
- ***Link delay***: This value denotes propagation delay of the link which is used in the delay measurement.
- ***Source node***: This field holds reference to the source router of the link. (from which router the link starts)
- ***Source interface***: This field holds reference to the source interface of the link. (from which router interface the link starts)
- ***Destination node***: This field holds reference to the destination router of the link. (till which router the link lasts)
- ***Destination interface***: This field holds reference to the destination interface of the link. (till which router interface the link lasts)

2.4 Network

router_network class has only two private members.

- ***List of routers***: This member has a list of reference to the *router* class objects present in the network.
- ***List of links***: This member has a list of reference to the *router_link* class objects present in the network.

2.5 Shortest path

shortest_path class calculates the shortest path between vertices (routers) present in the network. This *shortest_path* needs information about the links present in the network. These references are present in the *router_network* class. Thus *shortest_path* class instance must be created and its relevant functions must be called by the *router_network* class.

2.6 Routing table

routing_table class consists of the routing table which is linked to the router class through reference of class object. This routing table is created after the shortest path

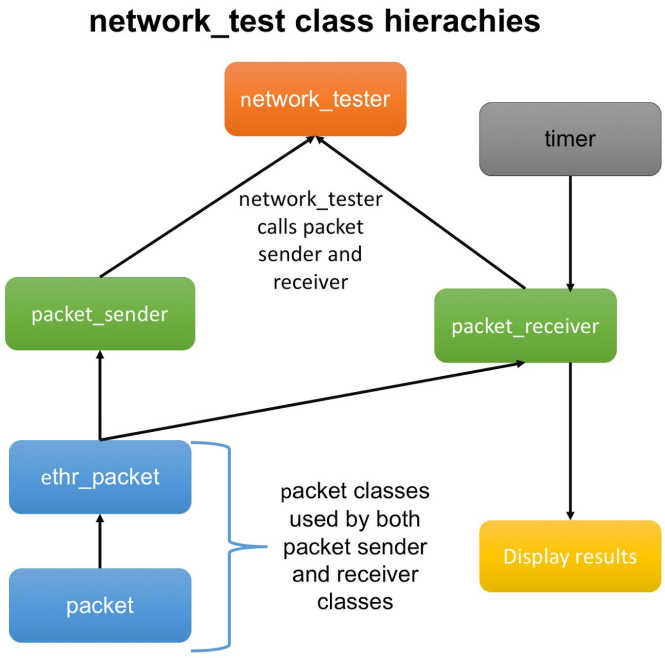
calculation. Thus *routing_table* class object is created and called by *shortest_path* class.

2.7 ARP table

arp_table class consists of the arp table which is linked to the router class through reference. This arp table is created after the shortest path calculation.Thus *arp_table* class object is created and called by *shortest_path* class.

Dynamic elements:

Dynamic elements are categorized as follows:



2.8 Timer

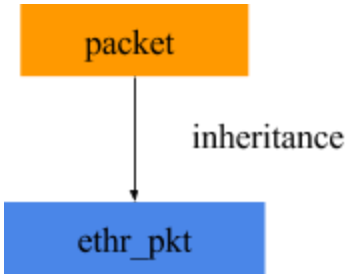
timer class is used to process packets received in the RX buffers. Poller is needed to scan all ports of the routers to check whether there are any packets present in the RX buffer. This poller uses timer to poll interfaces at regular intervals. Ideally poller speed must be greater than combined speed of the interfaces. Poller and its functions are implemented as timer class.

2.9 Network tester

Network tester (*network_tester*) is a separate class which initiates the delay measurement tests. It creates two threads per router. One thread is to send packets and one thread to receive packets. The send threads instantiate a *packet_sender* class object whereas receive thread instantiate a *packet_receive* class object. This class does not store any data.

2.10 Packet structure

packet class is an abstract class which may denote any packet like ethernet, ATM, TDM. This class is inherited by *ethr_pkt* class which denotes only ethernet packets. This class has lot of private members which denotes all fields of the ethernet II frame header and IP header. This class can construct a packet, parse a packet and even modify the fields of a packet.

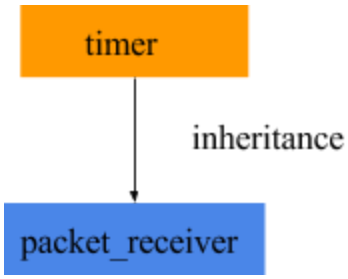


2.11 Packet sender

Packet sender class object created from the of *network_tester* class for each router fetches its own routing table and send packets to all other routers in the network. This class uses *ethr_pkt* class.

2.12 Packet receiver

Packet receiver thread of each router has a poller which checks whether there are any packets present RX buffers of all interfaces. This class inherits the *timer* class.



Besides, there are three more modules are used:

2.13 XML Layout

- XML is used to store network topology. Basic XML layout used in the project is shown below:

```
<network>
	<node id="0" interfaces="2" x="1.0" y="1.0">
```

```
      <interface id="0" mac="00:00:00:00:00:11" ip="192.168.0.1" mask="24" queue="10"
speed="100"/>
      <interface id="1" mac="00:00:00:00:00:12" ip="192.168.2.1" mask="24" queue="10"
speed="100"/>
      </node>
      more nodes ...
      <link id="1" source="0.0" destination="1.0" delay="1.0" cost="1.0"/>
      more links ...
</network>
```

Network can have a number of nodes <nodes> and links <link>. Each node has following attributes:

- id = Identity of the node in whole network (Unique)
- interfaces = Number of interfaces of node
- x = x-axis location of node in GUI window
- y = y-axis location of node in GUI window

Each link is between two nodes and has following attributes:

- id = Link identity in whole network (Unique)
- source = A end of the link in a.b notation. a= node id, b=node interface
- destination = B end of the link in a.b notation. a= node id, b=node interface
- delay = Delay offered by link.
- cost = Cost of packet traversal through that link.

Each node can have multiple interfaces with following attributes:

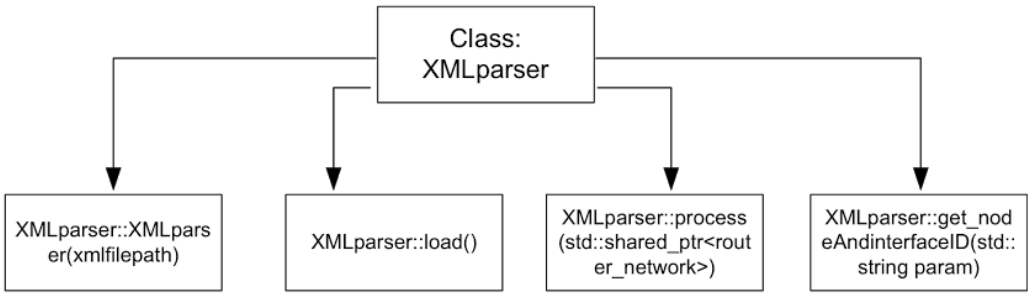
- interfaceid = Identity of the interface on a particular node.
- mac = 48 bit HW address of the interface
- ip = IPv4 address of the interface
- mask = network mask
- queue = Buffer size
- speed = Link speed in Mbps

XML files are stored under **nwsimulator1/resources**

2.14 XML Parser

- XML parser is based on *tinyxml2* library. It reads-in XML file and generates a network from it.
- Class structure:

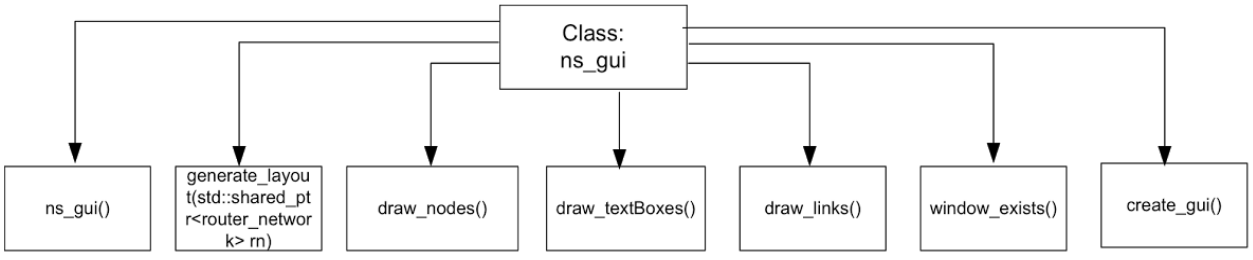
XMLparser class is implemented in ns_xml_parser.h. XMLparser class has following architecture:



XMLparser(filepath): Class constructor. Take XML file path as parameter obtained from command line while running the programme.
load(): Loads the XML file to memory.
process(shared_ptr<router_network>): Parses the XML and pass the nodes, links and node interfaces to router_network through shared pointer.
get_nodeAndinterfaceID(std::string param): Returns vector of strings containing node and interface id from link's source and destination attribute.

2.15 GUI

- GUI is based on SFML library. GUI is implemented in *ns_gui* class which gets the overall network information from *router_network* class and renders it to the screen.
- The class has following structure:

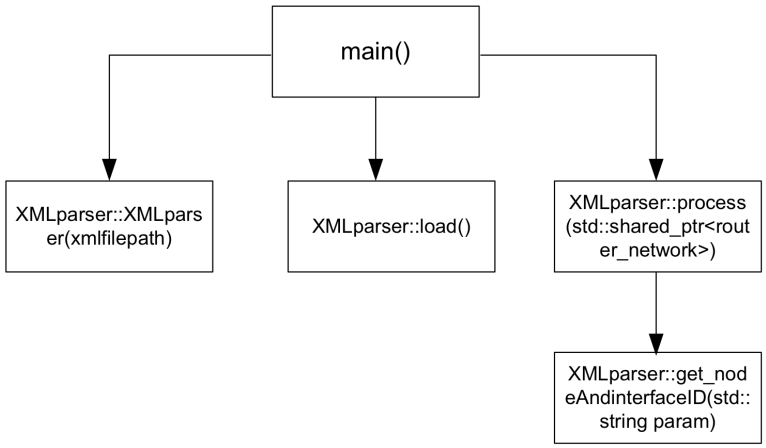


ns_gui(): Class constructor. Creates window of certain width and height. Sets the text style, node colors.
generate_layout(std::shared_ptr<router_network>): Generate basic layout and apply scaling for GUI. Takes shared pointer of type router_network as parameter.
draw_nodes(): Draws nodes on the GUI winodw.
draw_textBoxes(): Draws text boxes along the nodes showing source/destination IP addresses and link parameters.
*draw_links():*Draws links between nodes.
window_exists(): Returns true if window is successfully rendered to the display.
create_gui(): Calls all the rendering functions to create the GUI window and displays on the screen.

3. Description of software logic and most important function interfaces.

3.1 XMLParser

Working logic of XMLparser class is shown as flow diagram in figure below:



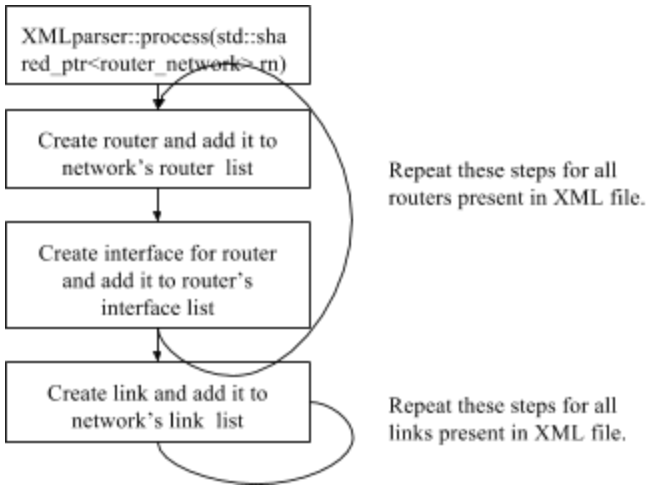
main function calls *XMLparser* constructor and passed path to the XML file as parameter. XMLparser then load the file to memory by calling *load* member function. After loading the file, *main* calls *process* and passes shared pointer of *router_network* type. In this function, XML is parsed and all the nodes, links and node interface information is passed to shared pointer to make it available in router_network class. After this, routing and ARP table is created and delay measurement between nodes is called.

3.2 Network creation , routing table creation and ARP table creation.

Network is created using the XML input files.

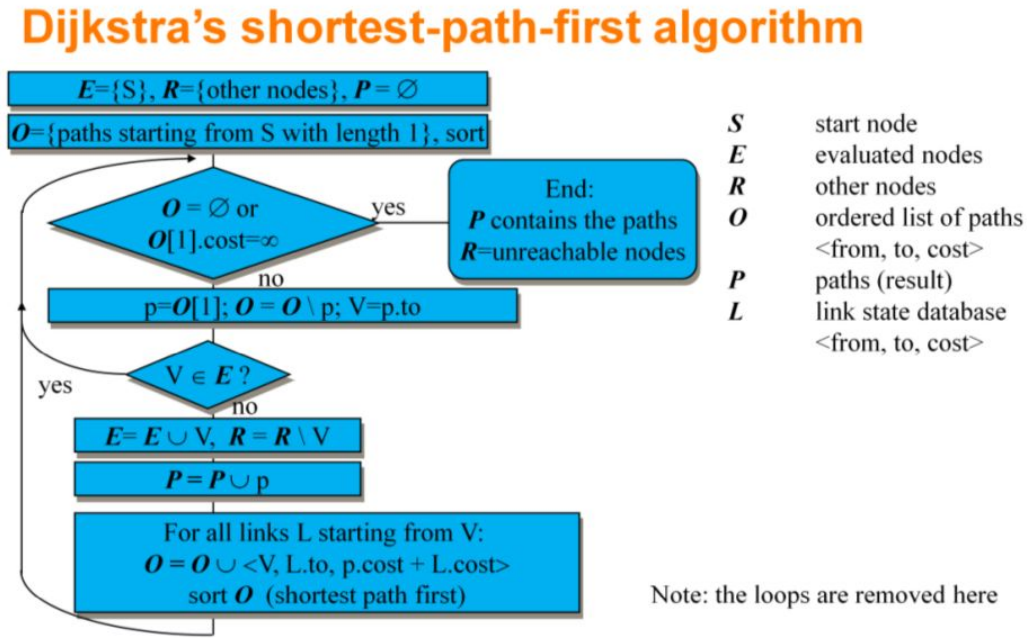
XML process() function takes router_network class object reference as input parameter.

- Routers are created one by one from the XML file and these routers are added to list for routers (references) of the network.
- After creating each router, interfaces are created and linked to the router as interface references.
- Once all routers and interfaces creation are completed, links are created and added to list for links (references) of the network.



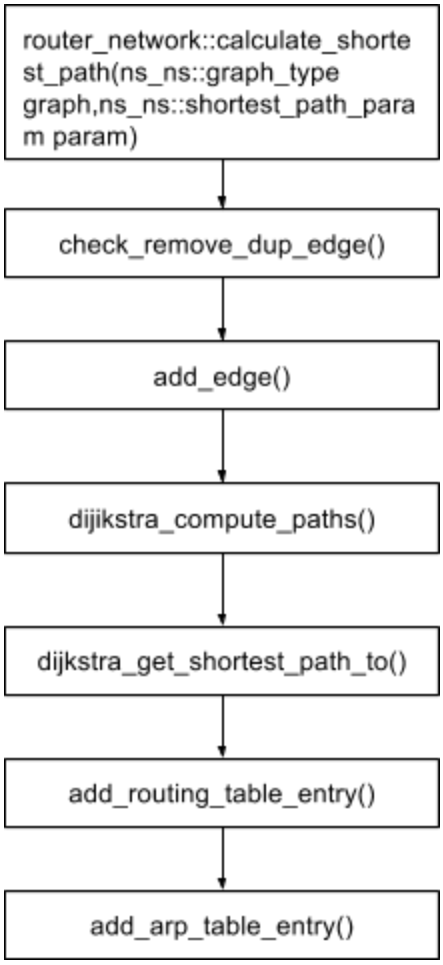
Shortest path algorithm.

Shortest path algorithm is implemented using the following instructions.



Once network is created from the XML file, *calculate_shortest_path()* function is called from the main file.

- *calculate_shortest_path()* function goes through all links present in the network list and forms the adjacency list needed for the shortest path calculation.
- Duplicate links present between routers are removed before forming the adjacency list.
- Routers without any links or interfaces do not participate in the shortest path calculation.
- Once shortest path is calculated, routing table and ARP table are created.



3.3 Delay measurements: (Packet sender and receiver)

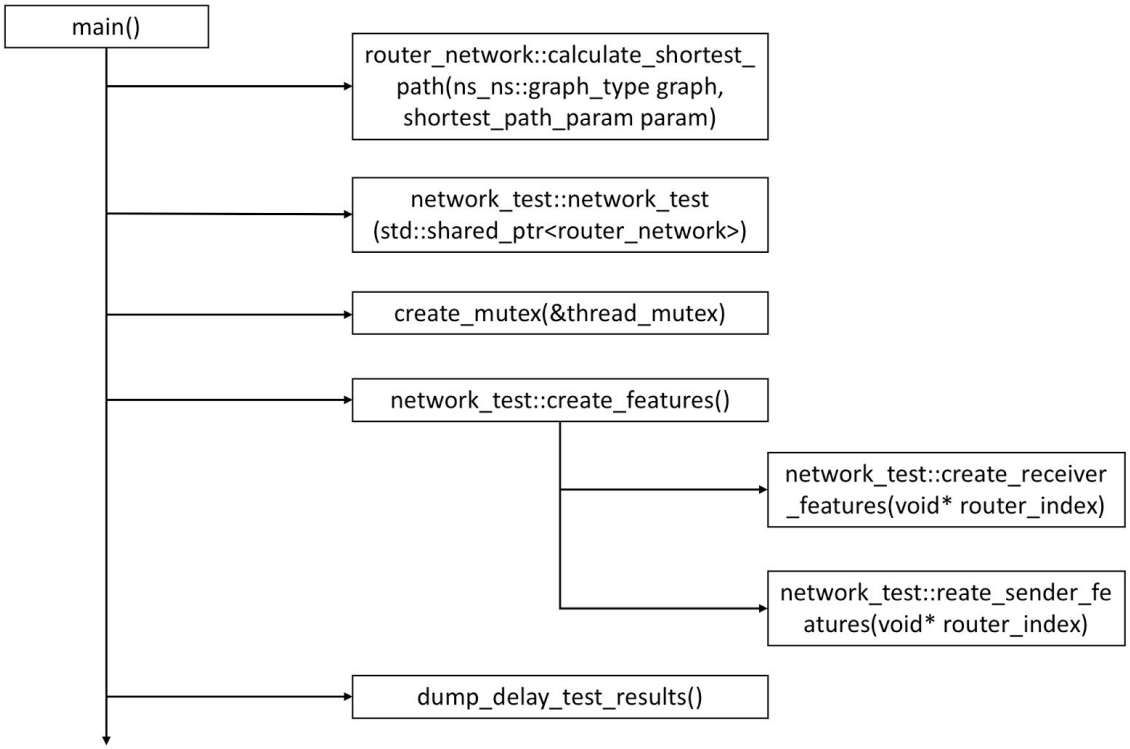
network_test class implements delay measurement tests. *network_test* implements delay measurement in the following way:

- *network_test* class starts two threads per router present in the network. One thread is to send packets for delay measurement and one thread to receive packet.

- **Send thread:** Send thread sends packets to all the destinations (routers) present in its routing table. This way, packets are sent to all routers present in the network if a path is present to that router. Path of the packet is determined by routing algorithm. After constructing the (68B) packet, packets are put in the RX buffer of the gateway router with a timestamp which is used to calculate the delay. If the RX buffer is full, packets are dropped.

- **Receive thread:** Receive thread has a poller which polls all interfaces sequentially present in the router to check whether there are any packets present in the RX buffers of a router. If present, it copies the packet from the buffer and empties the RX buffer. It parses the packet in the following way. It parses packet using the parse_packet from ethr_packet class.
  - First, *receive_thread* checks the destination MAC address of the packet with the received interface MAC address. If MAC addresses are different, then the packet is dropped.
  - Next, destination IP address of the packet is checked with the IP address of the received interface. If they are same, then the received router is the destination router and delay results are updated by getting current time-stamp and the timestamp from the packet. This gives only the processing delay of the packet. To add the propagation delay, the delay of the each link the packet takes to added to the final delay. Once the total delay is calculated, results are updated to the delay result structure.
  - If IP addresses are different, then packet is routed to the respective gateway depending upon the routing table result.

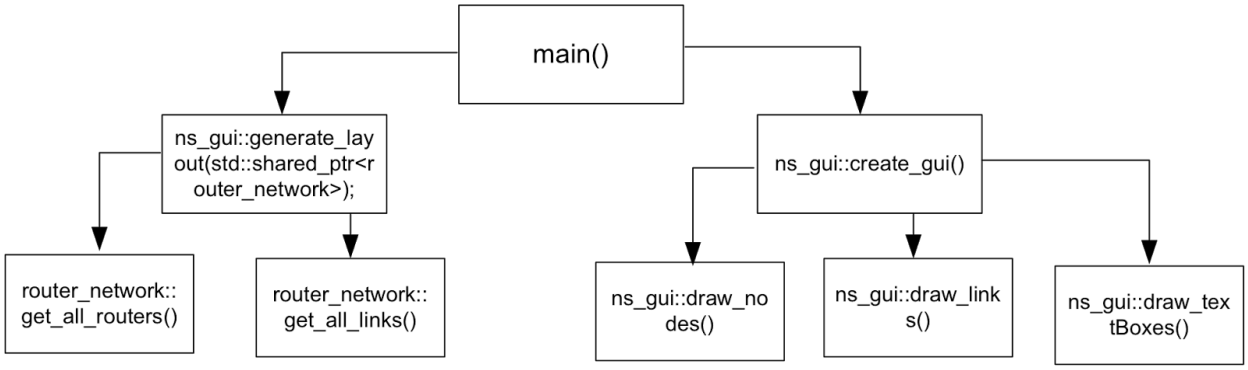
  Delay measurement is called from main as following:



main function calls *calculate_shortest_path* to generate routing and ARP table. There are two types of shortest_path_param: LINK_COST and LINK_DELAY. If LINK_COST is selected, shortest path algorithm will be called based on link cost. If LINK_DELAY is chosen, link delay will be used to calculate path. Then *main* calls *network_test* to create a test network class from *router_network* class mentioned above. Next, *main* calls *create_features* to start two threads for each node in the test network. Before starting the threads, *mutex* is also created to protect shared data from being simultaneously accessed by multiple threads. Finally, *dump_delay_test_results* is called to process the calculation of delay and display the dump. Next step is generating a graphic representation which is explained as below.

3.4 GUI (ns_gui)

Working logic of ns_gui class is shown as flow diagram in figure below:



main function calls *generate_layout* and passes *router_network* object as parameter. The object has complete network information and that is made available to *generate_layout* by calling *get_all_routers* and *get_all_links* member functions of *router_network* class. *ns_gui* object then calls the *create_gui* function that renders complete network along with IP addresses and link details in a text box.

4. Instructions for building and using the software

1) Execute software (all commands) from **nwsimulator1/src** folder.
2) Compile code using make file.
Execute "**make clean**" and then "**make**".
3) Software execution needs xml file (i.e) Network simulation needs input network. XML files are present in **nwsimulator1/resources** folder.
Thus software execution command must be for example,
./network_simulator ../resources/demo_7nodes_cost_delay.xml
network_simulator is the binary file name.
4) Once the binary file is executed as mentioned above, it opens a GUI displaying delay measurements and also text based output.
5) All logs will be present in **nwsimulator1/src/logs** folder.

- **ns_arp_table_logs.txt** – Has arp table for all routers.
- **ns_routing_table_logs.txt** - Has routing table for all routers.
- **ns_output_logs.txt** - Has results for delay measurements.
- **ns_traceroute_logs.txt** – Has trace route of packets from all routers to all routers.
- **ns_packet_logs.txt** – Has packets logs from packet sender and packet receiver. By default, this log is not enabled as this may cause delay difference in packet processing. To collect this log, execute following command instead of above command.

./network_simulator ../resources/demo_7nodes_cost_delay.xml log_enable

- External libraries:
- Software needs SFML library for execution of the software. SFML library is needed for Graphical UI.
- Software also needs **Google unit test (GUT)** framework to execute google unit test testcases but it is needed only for testing purposes and explained in detail in next section.

5. Testing

The testing is divided into three phases:

5.1 Unit Test

GUT is used for unit testing of each and every module of nsLite software. It is required to have GUT installed beforehand. There is a Makefile of each module where we specify the location of GUT:
GTEST_DIR = /home/$(USER)/googletest

Each module is tested thoroughly using separate test cases that are available at **nwsimulator1/src/test/src**
Various module test cases are:

1) node - All router class related functions.
2) interface - All interface class related functionalities.
3) link - All link class related functionalities.
4) network - All network class related functionalities.
5) packet - All packet construction, parsing related functionalities.
6) feature - All packet sender and receiver related functionalities.
7) main - Mix of all modules.

- Execution of the GUT test cases:

Go inside each module folder and execute **"make clean"** followed by "**make run**" command. The result will be something like below (sample: node module).

```
karthika@karthika-Inspiron-3542:~/Desktop/Karthika/ELEC-A7150/Project/nwsimulator1/src/test/src/node$ make clean
rm -f ns_node_router_test gtest.a gtest_main.a *.o
karthika@karthika-Inspiron-3542:~/Desktop/Karthika/ELEC-A7150/Project/nwsimulator1/src/test/src/node$ make run
g++ -isystem /home/karthika/googletest/include --std=c++0x -g -Wall -Wextra -pthread  -c ../../../node/ns_node_router.cpp
g++ -isystem /home/karthika/googletest/include --std=c++0x -g -Wall -Wextra -pthread  -c ../../../interface/ns_node_router_if.cpp
g++ -isystem /home/karthika/googletest/include --std=c++0x -g -Wall -Wextra -pthread  -c ns_node_router_test.cpp
g++ -isystem /home/karthika/googletest/include -I/home/karthika/googletest --std=c++0x -g -Wall -Wextra -pthread  -c \
           /home/karthika/googletest/src/gtest-all.cc
g++ -isystem /home/karthika/googletest/include -I/home/karthika/googletest --std=c++0x -g -Wall -Wextra -pthread  -c \
           /home/karthika/googletest/src/gtest_main.cc
ar rv gtest_main.a gtest-all.o gtest_main.o
ar: creating gtest_main.a
a - gtest-all.o
a - gtest_main.o
g++ -isystem /home/karthika/googletest/include --std=c++0x -g -Wall -Wextra -pthread  -lpthread ns_node_router.o ns_node_router_if.o ns_node_rou
ter_test.o gtest_main.a -o ns_node_router_test
./ns_node_router_test
Running main() from gtest_main.cc
[==========] Running 2 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 2 tests from test_router
[ RUN      ] test_router.router_creation
[       OK ] test_router.router_creation (0 ms)
[ RUN      ] test_router.router_if_list_creation
[       OK ] test_router.router_if_list_creation (0 ms)
[----------] 2 tests from test_router (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test case ran. (0 ms total)
[  PASSED  ] 2 tests.
karthika@karthika-Inspiron-3542:~/Desktop/Karthika/ELEC-A7150/Project/nwsimulator1/src/test/src/node$
```

5.2 System Integration Testing (SIT)

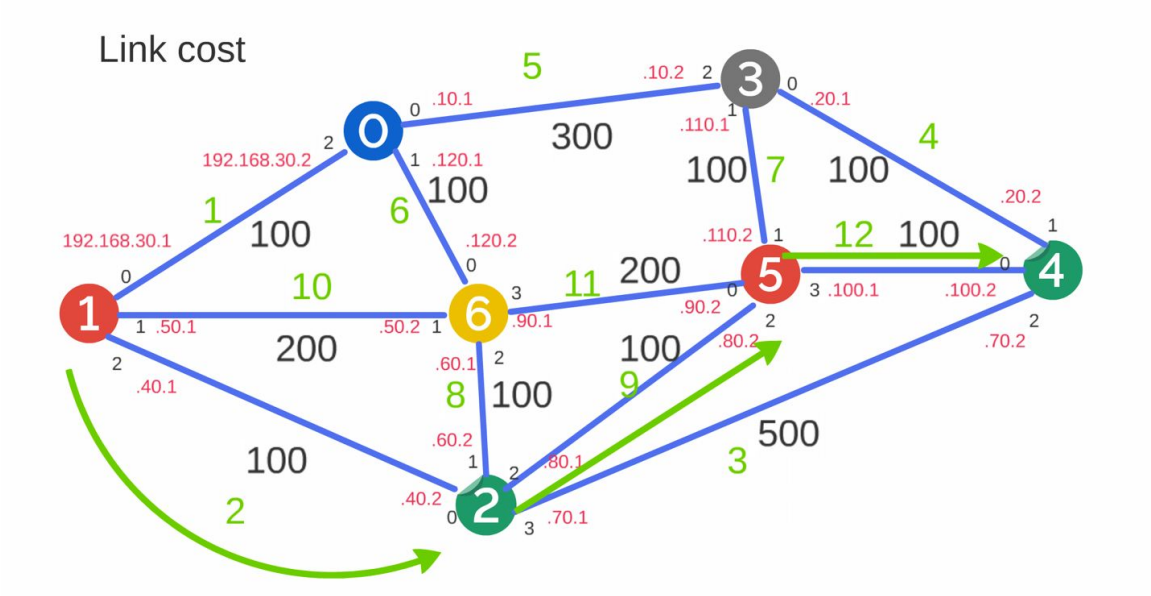
In this phase, we tested the integration between modules:

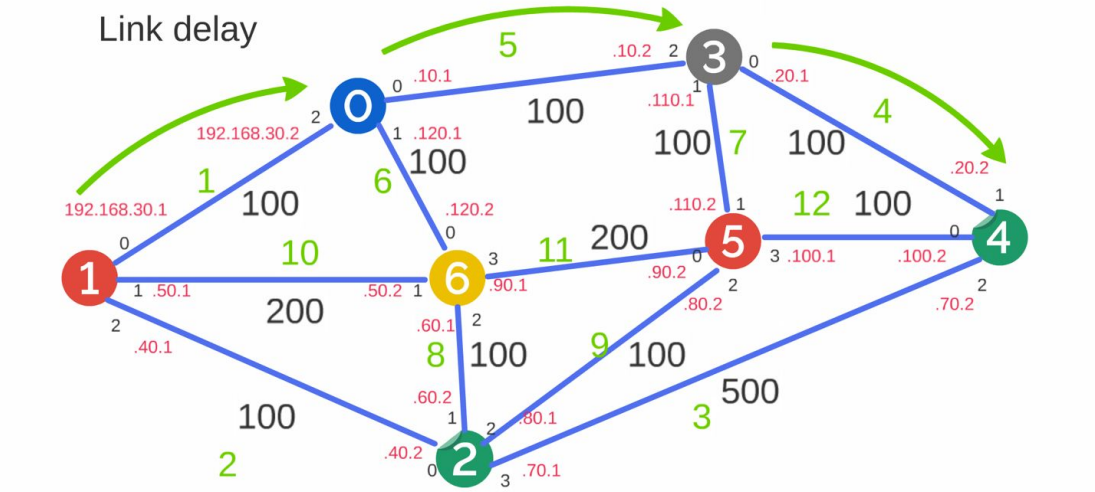
- XML file and XMLParser: specify the location of XML file to see if XMLParser can load XML file
- XMLParser and static elements and shortest_path: call these modules in main file to see if routing and ARP table can be created. All information is available in router_network class.
- Create a network_test class from above router_network class to start the threads on each router and dump the test results (display the delay calculation between nodes)
- Call ns_gui to generate a graphic representation of the network topology and results

5.3 System Testing (ST)

In this phase, we tested the whole system after the integration (including the GUI).

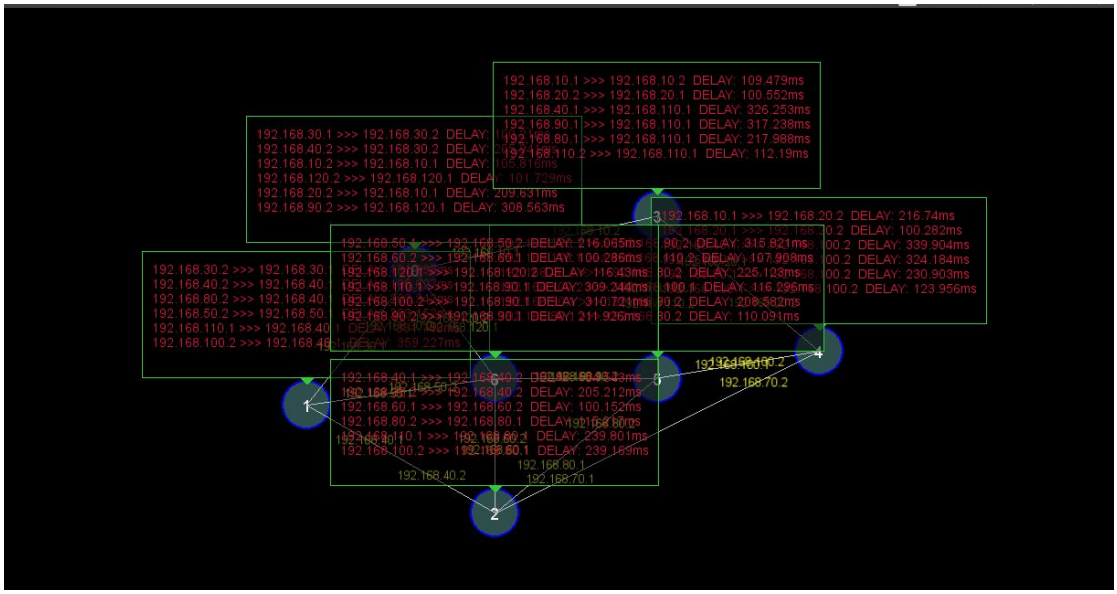
- Input: XML files with different topologies.
  Location of the XML for testing:
  nwsimulator1/resources/demo_7nodes_cost_delay.xml
- Expected result: a similar topology is displayed as below. Delay between each two routers will be shown in a text box at each router.





- Link cost scenario: the route from node 1 to 4 will be **1 – 2 – 5 – 4**
- Link delay scenario: the route from node 1 to 4 will be **1 – 0 – 3 - 4**

- Actual result:



Check the log file in logs folder to see the actual route:
nwsimulator1/src/logs/ns_traceroute_logs.txt

Traceroute log for link cost scenario:

```
Path from Node: 1 to Node: 4
1 2 5 4
```

Traceroute log for link delay scenario:

```
Path from Node: 1 to Node: 4
1 0 3 4
```

6. Future development

- Valgrind memory leaks still exists. Not able to solve all memory leaks.
- Comments are missing in some files.

7. Work log

Distribution of work:

S.No	Task	Task Owner
1	Design phase	All team members
2	Project plan	Karthika
3	Development of static network elements	Karthika

4	Creation of routing table and arp table	Karthika
5	Packet construction, parsing	Karthika
6	Makefile	Karthika
7	Packet sender	Tan
8	Packet receiver	Tan
9	Packet routing	Tan
10	Delay measurement result structure design and implementation	Tan
11	XML	Naveed
12	GUI	Tahir, Naveed
13	Logs	Karthika
14	Packet logging	Tan
15	GUT testing	Karthika, Tan
16	SIT testing	All team members
17	ST testing	All team members
18	Documentation	All team members

Work log:

Week	Task	Duration	Responsible person
02 Nov – 06 Nov	Form the group and select the topic	2 hours	All
07 Nov – 13 Nov	Prepare project plan	8 hours	Karthika
	Development of static network elements.	25 hours	Karthika
	Packet sender	16 hours	Tan
	XML template	2 hours	Naveed
	GUI	2 hours	Tahir, Naveed
14 Nov – 20 Nov	GUT framework and test cases	20 hours	Karthika, Tan
	Packet construction, parsing	8 hours	Karthika
	Packet receiver	15 hours	Tan
	Shortest path algorithm	8 hours	Karthika
	XML	2 hours	Naveed
	GUI	2 hours	Tahir, Naveed
21 Nov – 27 Nov	Makefile GUT minor fixes	4 hours 2 hour	Karthika Naveed
	Packet routing, fixes for packet sender and receiver	5 hours	Tan
	GUI	2 hours	Tahir, Naveed
28 Nov – 04 Dec	Create threads for packet sender and receiver and mutex	8 hours	Tan, Karthika
	GUI	12 hours	Tahir, Naveed
04 Dec – 09 Dec	Create Logs	10 hours	Karthika, Tan
	GUI fine tuning	4 hours	Tahir, Naveed
	Integration and bug fixing	8 hours	All
	XML for demo	2 hours	Tan

	Presentation slide + Topology	3 hours	Tan, Karthika
09 Dec	Demo session	1 hour	All
10 Dec – 16 Dec	Documentation, bug fixing	20 hours	All