

Carnegie Mellon University
18-756 Packet Switching

Simple Routing Information Protocol
Project Report

Annie Cheng (ahcheng)
Tung Fai Chan (tungfai)

Fall 1999

Information Networking Institute
Carnegie Mellon University
A020 Hamburg Hall
5000 Forbes Avenue
Pittsburgh, PA 15213

TABLE OF CONTENTS

OVERVIEWS

DATA STRUCTURES/OBJECTS

PACKETS

DATA LINK LAYER DESIGN

VIRTUAL ROUTER DESIGN

ROUTER DESIGN

DOMAIN NAME SERVER

USER INTERFACE AND MANUAL

DIAGRAM

OVERVIEWS

The Simple Routing Information Protocol (SRIP) is used to advertise reach ability information between adjacent routers in a network environment. The protocol has three main features:

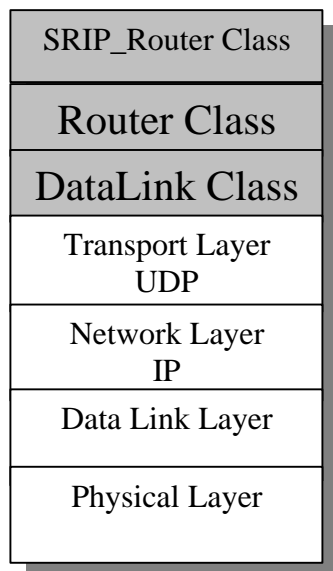
- ❑ Support for neighbor acquisition
- ❑ Support for "alive" messages
- ❑ Support for periodic exchange of routing information

The SRIP implementation of this project:

1. Provides a user interface that can be used to properly configure the router parameters. The user interface also allows viewing of the information contained in the transmitting and received routing table.
2. Handle exchange of routing and control information. Both Router and Virtual Router will be discussed in detail in later sections.
3. Both Sliding Window and Stop & wait protocol were used to handle errors and flow control.
4. Link failures, and data corruption between routers will be simulated. The split horizon *with poison reverse* based approach will be used to resolve the potential looping due to these failures.

In this report document, the detail description of the structure of the user interface is discussed. A modular decomposition of each component of the system and a description of the interface between each pair of components will be provided.

The router is implemented on top of UDP provided by operating systems. Basically, the router is divided into two virtual layers, Router, and DataLink class, which simulate the network, and data link layer respectively. Here is a big picture.



The SRIP_Router class is the graphical user interface layer which provided data representation and router configurations in user-friendly way.

The Router class is the only main component of the design. It contains routing table, timer, and protocol to communicate the router with another host.

The DataLink class is to providing data transfer services to upper layer by using the User Datagram Protocol(UDP) provided by underlying operating platform.

All boxes under these two classes (not shaded) are real network component at kernel level.

We used Java (JDK1.1) as our primary programming language. It is because as you may have noticed, our design are multi-thread and thread is an object in Java which is much more object oriented. In C/C++, the thread is in fact just a function. The other reason is its garbage collection. Multi-thread program will have series memory leak if there is leak at one point and Java can handle this problem well. Also, a java application can easily port on other platform. Furthermore, Java's *abstract window toolkit* (AWT) provided an easy way to implement a graphical interface. We chose to use AWT instead of SWING because it was capable for all JDK1.1 platform while the latter was regarded as add-ins (external packages) in JDK1.1.

This document would omit the detail of thread synchronization because it was not the aim of this project nor Java had provided nice tools on this area, such as Monitor.

DATA STRUCTURES/OBJECTS

Within the Router and DataLink class, there are several important data structures and objects which are defined for easier access and protection. Below are the brief descriptions of some of those.

Objects in Router class

RouterPacket:

- A data structure storing the all data and header information that is required by end-host router.

RoutingTable:

- Store the routing information in a systematic 2-dimensional array and provide fast indexing access and updates.

MappingTable:

- Store necessary information for each connection, such as real IP address, port, router name, etc.

NeighborStateTable:

- Store neighbor acquisition messages information for each neighbor, such as which neighbor acquisition message has been sent and a time out counter for reply of be_neighbor_request.

AliveStateTable:

- Store information regarding update and alive message sending state. Message time counters are created for both reply of alive message request and update routing table message. Hello and update interval timer are also stored in this table.
- Store information regarding whether a neighbor is a valid neighbor and whether a neighbor is alive.

RecvRoutingTable:

- A data structure to hold temporary tables received from a neighbor/virtual router. After all the table entries are received, updates can be done at once to the routing table.

Objects in DataLink class

DLPacket:

- A data structure designs for packet to encapsulate the RouterPacket and pass into the UDP for transmission.

DLConnection:

- An object which reside inside the DataLink class. Its main responsibility is to communicate with corresponding to its own destination.
- Provided scheduling of packets to be sent.

DLSendConnectionTable/DLRecvConnectionRecvTable:

- These two tables maintain the states for sliding window protocol. They kept the acceptable window size for sending and receiving sides respectively.

PushBuffer:

- This is a high priority buffer specially designed for alive, and be_neighbor messages. All packets in these buffers have higher priority than those in ordinary buffers. This buffer employed stop-and-wait scheme.

DLTimerThread:

- A thread wakes up every period to decrement the count corresponding each packet which was waiting for acknowledgement or re-transmission.

ConnectionSendThread:

- A routine thread which keep asking for packet for sending. This thread will be put to 'sleep' if no packet in the buffers.

ReceiveThread:

- A routine thread waits at the port for incoming packets. Once packet arrived and error-free, it will spawn a new thread to take up the responsibility.

Objects in Virtual Router class

VirtualRouterTable:

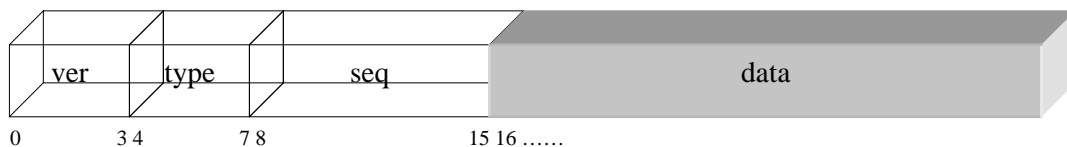
- A table to store all the routing table entries generated by the virtual router.

PACKETS

This section will discuss the different types of packets. As mentioned, there will be two distinct types of packet using in our design, RouterPacket, and DLPacket. They are used to communicate by different layer protocols.

RouterPacket

There are altogether 4 fields in each RouterPacket. They are: Version number, Sequence Number, Packet Type, and Data. The maximum length (including header) is determined to be 255 bytes.



The version number, sequence number fields, and packet type are the header of this type of packet.

Version Number (ver)

- It is a 4 bits unsigned number that carry the version using by the router.
- It can carry up to 16 different version numbers.

Packet Type (type)

- It is a 4 bits unsigned number that indicate the type of packet.
- It is capable to differential up to 16 different packet types.

Sequence Number (seq)

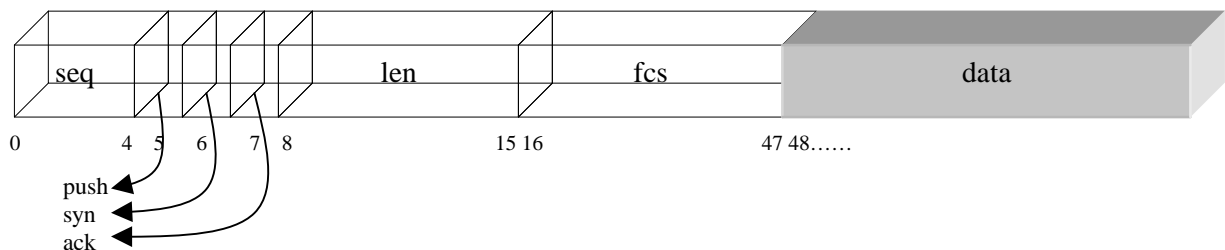
- It is a 8 bits unsigned number that carry the sequence number of the routing update information.
- It permits 256 distinct values on sequence number.

Data (data)

- The maximum length is 12 bytes.

DLPacket

There are altogether 6 fields in each DLPacket. They are: Sequence number, Length, Frame Check Sequence, and Data. The maximum length (including header) is determined to be 261 bytes.



The sequence number, length, frame check sequence are the header of this type of packet.

Sequence Number (seq)

- It is a 4 bits unsigned number that carry the sequence number (if using sliding windows algorithm).

- It can carry up to 16 different sequence number.

Push flag (push)

- A flag to indicate whether it is a 'push' packet. If it is, it will be put in front of the queue for Router.

Synchronization flag (syn)

- A flag to indicate it is a synchronization packet. The `DLConnection` should reset the buffer and sequence number accordingly.

Acknowledgment flag (ack)

- A flag to indicate if it is an acknowledgment packet.

Length (len)

- It indicates the number of bytes of data field (`RouterPacket`).

Frame Check Sequence (fcs)

- It carries the checksum performed by Data Link layer.

Interface

- The Router will pass a `RouterPacket` into the `DataLink` layer. Then the underlying layer will encapsulate it in a `DLPacket`. In another, the data fields in `DLPacket` is used to store `RouterPacket`.
- When a packet is received, the `DLPacket` header will be taken off before return it to Router together with corresponding size.

DATA LINK LAYER DESIGN

The Data Link Layer functionalities are essential provided by `DataLink` class. Although in ISO/OSI and TCP/IP reference model, Data Link Layer is a separate layer under Network protocol, in our design, we regarded `DataLink` class as an *object* inside Router. In another word, the Router *has-a* `DataLink` class.

Here are the interfaces provided for upper layer.

- `Data_in()` – provide function for router to pass in packet which is to be sent with given network ID.
- `Data_out()` – provide function for router to get packet with given network ID.

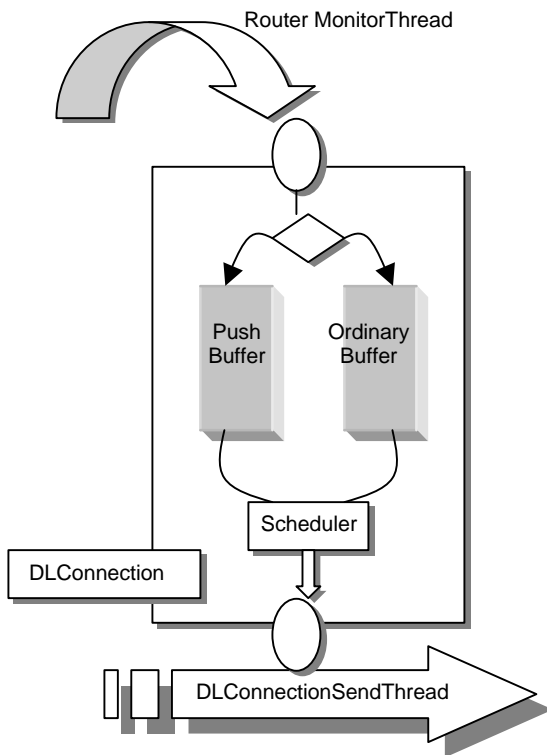
Start Up/Initialization

When the object is initialized, it first obtains the number of connection has to corresponding to. Usually, the number is equal to maximum number of neighbor nodes (connected with direct link) according the `MappingTable` in `Router` class. As we will mention in Router section, this table was set to have a larger size so it gives room to grow in number of neighbor/connection during runtime. Therefore, the number of `DLConnection` is equal to number of *total entry* in the table while only those *currently exist* will be initialized.

It will initial different `DLConnection` object to serve each destination neighbor. There is a `DLConnection` object for each link that connecting to the router logically. In another word, there are different buffers for each connection. The main advantage is to increase concurrent and efficiency. For example, if a router has four neighbor nodes, there will be four instances of `DLConnection` object with separate buffers. Also, it enforces fairness among different connections. If there is only one common buffer for all connections, router can only obtain the packet in a FCFS (First-Come-First-Serve) basis. Therefore, there is possibility that if one connection constantly sends large amount of data, it will cause other packet from other connections waiting in the buffer to be processed. Starvation will occur. Also, if different buffers are used, router can obtain packets from different connection concurrently and there is no need to wait for packets from others to be done first. Because of this reason, we implemented different buffers for each connection.

Priority Transmission

We have implemented two different kinds of data transmission, sliding window and stop-and-wait with two different buffers. The stop-and-wait buffer has a higher priority over sliding window. This buffer is specially designed for packet, which needed reply from other side with time constraint such as alive, and be_neighbor messages. The reasons are that, firstly, Router will start the timer as long as the packet is passed into the DataLink class, if it is placed in sliding window buffer, it will be at the end of the queue and it has to wait all packets before to be sent *successfully* in order to be transmitted. Also, on the receiving side, the same situation happened that the newly received packet has to be in the buffer until all preceding packets are safely received and obtained by Router. This created a long queueing delay that sender might already consider neighbor is dead and clean up the routing table (detail will be described in Router section).



In order to tackle this problem, we employed this priority transmission. Whenever Router needs to send “Time Attack” packets, it will put it into a separate buffer, PushBuffer. All packets in this buffer can preempt the transmission of the ordinary buffer. This action was achieved by packet scheduling taking place in DLConnection. As mentioned, the DLConnectionSendThread will keep waiting for new packet from DLConnection, DLConnection will take check whether there are packets in PushBuffer first. If so, gives it to the sending thread for transmission. Otherwise, check the ordinary buffer for data. The thread will be put to ‘sleep’ state if and only if both buffers are empty.

PushBuffer is using stop-and-wait algorithm to proceed. Therefore, after a packet is selected from PushBuffer for sending, there will not be another packet from this buffer again till the acknowledgement is received. During this period (between packet sent and acknowledgement received), the ordinary has a absolute priority to send packets. When PushBuffer timeout occurs, it will be placed with higher priority again for next sending scheduling. Please note that the stop-and-wait only apply to this buffer but not the whole system, during the ‘wait’ period, the ordinary buffer are free to send packets. This ensures fairness over both buffers.

Flow and Error Controls

Stop-and-wait:

- When packet is sent, it will stop until an acknowledgment is obtained from the receiver side. That acknowledgment is NOT the one for router, but for data link layer only.
- When no acknowledgment is received for a period of time, timeout will trigger the layer to retransmit the packet. After retransmission for 3 times, the packet will either be dropped or even the connection is determined to be down and stop services.

- When packet is received, it will perform checksum immediately. If it passes, the header will be removed and the `RouterPacket` will be stored in the buffers. Then, an acknowledgment will be sent immediately. If test fails, the packet will be dropped and do nothing else.
- If the packet is retransmitted for certain time, it will be dropped *without* notifying the Router.
- Though sequence number is used, it is not for re-ordering incoming packets as stop-and-wait is in order by nature. This sequence number is used for receiver to determine if the packet is duplicated and discards it if so.

Sliding Window:

- Packets will be sent without waiting for acknowledgement up to window size. Our maximum sequence number chosen was $2^5 = 32$ and thus the maximum window size is half of 32 which is 16.
- When receiver received the packet, it will proceed CRC32 check. If fails, the packet will be dropped right away. Otherwise, a new thread will be spawn to take care of this packets, to put it in buffer and sending acknowledgments. In another word, non-cumulative acknowledgment is used.
- After sender sends out packets, a counter will be reset and being counted down whenever the `DLTimerThread` wakes up. If the acknowledgment is received before timeout, the corresponding packet will be cleared (set to null) and the timer will ignore this. When timeout occurs, that packet will be scheduled to be next to be sent (within this ordinary buffer, so packet in `PushBuffer` still have higher priority than this retransmission packet).
- If the packet is retransmitted for certain time, it will be dropped *and* notify the Router that the neighbor is down. (The consequence of notifying Router down will be discussed in details in Router section.)

Synchronization

The `PushBuffer` is using stop-and-wait algorithm and as mentioned, the sequence number is only used for determining duplicate packets. Also, the `PushBuffer` should be empty most of the time as it is only reserved for alive messages. There is no need to synchronize the sequence whatsoever. In another word, Router can *always* send packets through this buffer most of the time.

However, in ordinary buffer, as sliding window protocol is used, synchronization is necessary because of different of sequence number on both sides would freeze the transmission. To synchronize both ends, we used 2-ways handshake in which one side will send a synchronization packet (with SYN field set to true), when receiver gets this packet, it will reset sequence number and buffers and then send back a synchronization acknowledgment. When the sender gets this acknowledgment, it will proceed reset sequence numbers. This send and receive is carried out in stop-and-wait mode and during this synchronization period, the `DLConnection` will be locked until this process is finished.

Interaction with Router Object

After a packet is passed in, it just simply passes it into corresponding `DLConnection` according to `MappingTable` index. When router request a packet, the `data_out()` will be invoked. The basic job for this function is to return the address the packet exists in the `DLConnection` object. We choose not to copy the object but return the pointer instead. It will be much more efficient as memory access is slow.

A timer thread will associate with each `DLConnection` objects to trigger retransmission. It simply wakes up every 1msec (can be changed) and decrement the counter of transmitted packet, if any. If counter is zero, retransmits the packet. More details have been cover above.

The `start_services()` and `end_services()` functions are simply let Router to decide when to start and stop. It will call the `DLConnection`'s start and end thread functions.

Packet Drop/Corruption Simulation (Garber)

- The basic function of Garber is to cause error on outgoing packet. It is on the sending function.
- Two kinds of error can be simulated by user-defined probabilities, packet error, and link error.
- The probabilities are per-connection (logical link). Therefore, it could achieve the situation that 1 link is totally loss while others are perfectly fine.
- Packet error can be done by flipping one or more bit inside a packet in a random position.
- Link error can be simulated by dropping the whole packet without sending it.
- The degree of causing errors are depending on the probabilities which is default to be 0.

A More Detail on DataLink/DLConnection objects

- This object is basically responsible for data flow on each link connection with individual buffers according to the flag, PUSH or not.
- After it received the RouterPacket from DataLink class, it will compute the checksum and form a header and attach to the RouterPacket. A DLPacket is formed. Then, it is put into the sending buffer.
- A thread will continue checking the sending buffer and perform UDP send routines provided by underlying platform. It will sleep if the buffer is empty. The sending mechanism is described by following part *Flow and Error Controls*.
- A 'global thread' in DataLink class will monitor the receiving function. When a new packet comes in, it will check the *from* address and put it into appropriate DLConnection receiving buffer through the `recv_packet()`. Then, this function performs the error controls, take off the header and put the RouterPacket into the buffer. The address (not the packet) of the packet will be returned when the `get_packet()` is invoked.
- The *from* address can be found on the DatagramPacket from standard Java API and the function can simply parse in the header for information.

To simulate router error, we can simply 'kill' the program or stop the services. The effect of router down will be discussed in details in following section.

VIRTUAL ROUTER DESIGN

The Virtual Router is used to simulate a part of network topology behind the actual router. It is a global object that can be easily configured. The user can configure this object with a virtual router domain ("128.2"), maximum host address (255.255) and minimum host address (0.0). With the configuration given by this example, the virtual router will generate random addresses between the range 128.2.000.000 to 128.2.255.255. The user can also configure the maximum number of hops and minimum number of hops this random virtual router can generate. The frequency of adding a new entries and editing the hops distances can also be configured. Lastly, the interval this routing table is transfer to the real router also can be configured.

When `VirtualRouter.start()` is invoked, a thread will be started to generate bounded random values corresponding to the add rate and changing the next hop value according to the edit rate. The virtual routing table will be transfer to the routing table of router during each virtual router update interval.

Simulating RoutingTable From Real Router

- ❑ To simulate a virtual routing table as it is from a real neighbor router, the thread in the object, as mentioned, will generate routing table entries in exact format as a `RoutingTableEntry` from a real router and put it in the `RecvRoutingTable` before transfer to the router for updating.
- ❑ This simulates a real routing table updating experience from the point of view of router. The router could not distinguish a table generated from virtual router or tables received from the neighbors. They are the same from the router's table update algorithm. The only difference

between the two kinds of entries is the NextHop value. An entry generated from virtual router will have NextHop value as “VR”.

- ❑ We didn't put the packet to router directly nor through socket are due to following reasons. First-of-all, the Router is already a complex object, and adding new objects and threads will just increase complexity and difficulty to debug. On the other extreme, if we send it out through the socket, the performance seems to be unacceptable and unnecessary too. Open a socket and send out packet induce lots of I/O which will significant degrade the performance and waste resources.

ROUTER DESIGN

Router Object Configuration

- ❑ `hello_interval` - amount of time a router waits before it tests if the neighbor's alive
- ❑ `update_interval` - max. frequency of routing updates
- ❑ `router_version` - routing protocol version used to run this router.

Threads

- ❑ `AliveServiceThread`
- ❑ `FuneralThread`
- ❑ `MonitorThread`
- ❑ `NeighborServiceThread`
- ❑ `RouterTimerThread`
- ❑ `UpdateRoutingTableThread`
- ❑ `UpdateSendThread`

There will be seven types of thread in the router system. The most important threads that control the messages flow are Global Timer Thread and Monitor Thread. The responsibility of Global Timer Thread is to wake up and decrement the time in the state tables. Whenever the value of an entry in the time table reaches 0, a corresponding service thread will be waken up and corresponding actions will be taken or the corresponding packets will be send. The monitor thread will be waken up by the data link layer whenever it receive a packet. It will execute appropriate event handler for each packet type. For example, a timer will be set for sending the alive message, whenever that timer time-out, the alive message will be send.

Alive Message Service Thread - Sending out Alive Message Packet during each alive interval.

This thread is called when time counter in the alive message table reaches the value of the hello interval. The index of the entries ready to be served will be passed in to this thread. The thread will service the time out entry. It then reset the timer accordingly after its service. This thread dies after its service. This thread serves two purposes. First, it sends the alive message if the host is a neighbor. Secondly, it sends the `BeNeighborRequest` message if the host is an UNKNOWN neighbor type. This happens when we cease being neighbor with the neighbor before, but wanting to be neighbor with the neighbor again. This second feature will be useful when the router that we want to be neighbor with is not up yet. By resending the `BeNeighborRequest` periodically, we will be able to detect the neighbor router when it's up and be its neighbor as soon as possible.

Funeral Thread - Detect/Clean up the router down condition.

After the router sent out the alive message `Router_Alive_Query`, a timer is set in the alive state table. If the router doesn't receive the `Router_Alive_Ack` within the timeout period, the Neighbor will be considered as down. The Funeral Thread will be invoked. The index of the time out entries in the Alive State Table will be passed into this thread. The thread will service the timeout entry then reset the timer accordingly after its service. The timeout entries represent the neighbor is dead. This thread dies after its service. The only service this thread provided is to notify the routing table that the neighbor is down. The

routing table object will mark the `num_hop` of all the entries that contain this neighbor as `next_hop` to infinity.

Monitor Threads – Thread that handles all the messages received from its neighbor

There will be `n` number of monitor thread where `n` equals the number of neighbor a router has. Each monitor thread is related to a connection in the data link layer.

The monitor thread will be wakening up by the data link layer whenever it receives a packet that belongs to its connection. It will get the packet, look at its header for packet type and execute appropriate event handler. Each message type will have its corresponding event handler to handle the packet. For example, if the monitor receives the `be_neighbors_request` packet, it will execute the `handle_neighbors_request()` event handler.

We choose to implement monitor using multiple threads instead of one thread because the connections will receive packets concurrently. The monitor will become a bottleneck if we use only one thread to handle the distribution of the packets. Also, we decide to have the data link layer trigger corresponding monitor thread whenever data is available instead of having the thread polling the data link layer. This will save some CPU cycle of busy waiting.

Neighbor Service Thread – Retransmission of Neighbor Acquisition Messages

This thread handles the retransmission of `CEASE_NEIGHBORS_REQUEST` and `BE_NEIGHBORS_REQUEST` messages. For the cease neighbor request messages, if we don't get the confirm message, this thread will retransmit the message as long as the alive state indicates the neighbor is alive. If the neighbor is not alive, it will retransmit the cease neighbor request message when the neighbor is up again. This ensures that we are able to cease the neighbor relationships.

The `BeNeighborRequest` message will be retransmitted if the host is an `UNKNOWN` neighbor type. This happens when the router just starts up, the neighbor router that we want to be neighbor with is not up yet. By resending the `BeNeighborRequest` periodically, we will be able to detect the neighbor router when it's up and be its neighbor as soon as possible.

This Thread is called when timer in some entries of the neighbor state table times out. The index of the time out entries will be passed into this thread. The thread will service the timeout entries then reset the timer accordingly after its service. This thread dies after its service.

Global Timer Thread - RouterTimerThread

Timer wakes up, call functions to exam several timing tables and if there are time out for an entry in the table, act appropriately. We imagine there would be several timing table that keeps track of the states of transmission. The messages types that will be triggered by the timer are the alive messages, and the routing update messages.

When the alive messages timer is time out, the alive message will be send by one router to test whether the neighbor is still alive. This period is determined by the hello interval. The monitor thread will be waiting for the alive message acks from the neighbor routers, if the alive messages come back after a set time, the neighbor is consider alive, else, the neighbor is considered dead, the routing table entry for that neighbor will be marked as -1 and the routing array will be marked as down.

When the routing update message timer is triggered, the `Tbl_Hdr` packet will be send to initiate the routing table exchange routine. After the `Tbl_ACK` has been received, the `Tbl_Entry` packets will be send. After all the `Tbl_Entry` packet are sent, the `Tbl_Ovr` packet signals the end of transmission of a routing table. Only `Tbl_Hdr` packet will actually be send out because of the triggering of the global timer. The `Tbl_ACK`, `Tbl_Entry`, `Tbl_Ovr` packet will be send out base on the `Tbl_Ack` received by the monitor thread.

Update Routing Table Thread – Updating the routing table after receiving the table from neighbor

This thread can be invoked in two ways. First, the monitor thread, upon receiving the table over message can invoke this thread. Receiving the table over message indicates the transmission of the routing table from its neighbor has been completed. Secondly, the virtual router can invoke this thread when the virtual router update interval is up. The table update algorithm will be described in detail in the following session. This thread exits after its service.

Update Send Thread – Transmitting routing table entries to its neighbors

This thread is responsible for sending the routing table entries to its neighbors. This thread will only be invoked when the update interval is up and when the neighbor is valid and up and running. The sending algorithm will be discussed in detail in the following session. This thread exits after its service.

Messages and Related Algorithms

Neighbor Acquisition Message

- ❑ Used by 2 adjacent routers to establish "neighborhood" for routing info exchange
- ❑ Messages are
 - BE_NEIGHBORS_REQUEST
 - BE_NEIGHBORS_CONFIRM
 - BE_NEIGHBORS_REFUSE
 - CEASE_NEIGHBORS_REQUEST
 - CEASE_NEIGHBORS_CONFIRM

Be Neighbor Algorithm

When a router started, a BE_NEIGHBORS_REQUEST message will be send. Upon receiving the message, the neighbor router will decide whether to be neighbor upon the routing protocol version number and its own neighbor list. If any of those two is not a match, a BE_NEIGHBORS_REFUSE message will be sent as a reply, else a BE_NEIGHBORS_CONFIRM message will be sent.

Upon receiving a reply message, the router will check whether the delay is too long, if the reply is not received within the time out period, the reply message will be dropped, a BE_NEIGHBORS_REQUEST message will be retransmitted by the router. Note that the time out period in the upper level is long enough for the data link to retransmit the packet 4-5 times. The design decision is for the application level to do retransmission so that the data link layer can give up resending after 4-5 times.

Upon receiving a BE_NEIGHBORS_CONFIRM message, the neighbor will be marked as a VALID_NEIGHBOR. The neighbor will be marked as alive at the same time. Upon receiving a BE_NEIGHBORS_REFUSE message, the neighbor will be marked as an INVALID_NEIGHBOR.

Cease Neighbor Algorithm

Cease Neighbor Request message will be sent when a router administrator decide to invoke this option through the user interface. Upon receiving a CEASE_NEIGHBORS_REQUEST, a router will mark the neighbor who sends out this request as an INVALID_NEIGHBOR. At the same time, we will set this neighbor as down and clean up the routing table entries, setting the related entries to infinity, by notify the routing table that this neighbor is down. Also, a CEASE_NEIGHBORS_CONFIRM message will be sent to the original requestor.

Upon receiving a CEASE_NEIGHBORS_CONFIRM message, the router will mark the neighbor as an INVALID_NEIGHBOR. At the same time, we will set this neighbor as down and clean up the routing table entries, setting the related entries to infinity, by notify the routing table that this neighbor is down. If within the time out period, this message hasn't been received, the router will resent the CEASE_NEIGHBORS_REQUEST message. This retransmission is useful if a neighbor is temporary down or the link is temporary in congestion state, we want to make sure the neighbor receive this request and both agree to stop the neighborhood relationship.

Alive Messages

- ❑ Used by router to check if its neighbor is alive.
- ❑ Messages are
 - ROUTER_ALIVE_QUERY
 - ROUTER_ALIVE_ACK

Neighbor Alive Algorithm

Alive messages are sent every `hello_interval` to a router's neighbors. Upon sending, a timer will be set in the alive state table, when the timer times out, the router will consider its neighbor is down and notify the routing table to mark the corresponding entries to infinity. This time out value is set large enough for the data link layer to retransmit this message 4-5 times in case of message dropped during the network congestion. Upon receiving a ROUTER_ALIVE_QUERY message, a ROUTER_ALIVE_ACK message would be sent as a reply. Upon receiving a ROUTER_ALIVE_ACK message within the time out period, the timer will be reset and the routing table will be notified. The routing table will update its entries accordingly if this neighbor was down and now it's up again. Exactly one entry will be updated; the destination of neighbor will be mark with distance 1. The alive state table will mark this neighbor as up.

For Routing Update Message and its corresponding event handler

- ❑ Used by router to exchange its routing table to the neighbors.
- ❑ Messages are
 - TBL_HDR
 - TBL_ENTRY
 - TBL_OVR
 - TBL_ACK

Routing Table Entries

Routing tables has entries with columns destination, number of hops and next hop.

Sending Routing Table

For each update interval, the routing table will be sent to its neighbor. A clone copy of routing table has been made first. Then, a TBL_HDR message will be sent to the neighbor. Upon receiving a TBL_ACK message, the rest of table entries with type TBL_ENTRY, will be transmitted to its neighbor. After all the entries in the table are transmitted, a TBL_OVR message will be sent to indicate the end of transmission of a routing table.

If TBL_ACK message arrives too late, the whole table will be throw away without continue to retransmit. The router will wait for the next update interval to send the routing table. The late arrival of TBL_ACK message indicates an unstable link; therefore, it is safer not to transmit the routing table until the network is stable enough to do so.

Routing table entries are sent 10 entries at a time if table entries are enough to pack for 10 entries. Else, whatever left will be transmitted in a pack.

Receiving Routing Table

Upon receiving a TBL_HDR message, the TBL_ACK message will be sent to the neighbors. At the same time, a receiving routing table will be created ready to receive table entries. The TBL_HDR message carries the size of the routing table that will be transmitted. All the entries in the receiving routing table have a timer set on it. As a result, if anything entries have been dropped (by router's point of view, DL level may retransmitted it many times already), the whole table will be disregarded. The router will get a

new routing table for the next update interval. Upon receiving the TBL_OVR message, the real routing table update will begin.

In the router level, TBL_NCK and TBL_FTL will not be used because the router will rely on the data link layer to check the error status

Updating Routing Table

Upon receiving routing table from the router's neighbor, the router will be updated according the distance vector algorithm. Loops are dealt with using the split horizon algorithm described on page 168, 169 in the book "Computer Network" by Peterson & Davie.

Each time the router has been updated; the sequence number will be updated.

Transmission error handling in the router level:

All the retransmission will be handled in the data link layer. If an error detected, the data link layer will re-transmit the packet for maximum of 4-5 times. If the packet still doesn't get through, the data link layer will give up. So, in the router layer, the time out period will be about 4-5 times longer then the timeout period in the data link layer. If an ACK is not received, the router will consider it's a packet lost, and will handle the situation accordingly. For an alive_ack, the router will just assume the neighbor is dead.

DOMAIN NAME SERVER

Due to the fact that Router can start on any machine with any port number, we have implemented a DomainNameServer as a centralized database to store all routers' address information. The basic design is very similar to the real DNS Server. During the Router starts up, it will carry out the registration process and add in an entry of router name, IP address, and port number. Whenever a router adds new neighbors, users only required to input the Router Name as unique and a request will be sent to DomainNameServer and the server will send back the result back to router.

The core part of the server is simply a hashtable which used the Router Name as the key while RouterAddress as the object to pass in. The main advantage is that the search time is fixed given that the table grows and the hashtable is already defined in Java.

USER INTERFACE AND MANUAL

The SRIP_Router have been test successfully on following platforms with certain configuration.

Platforms:

Microsoft Window NT 4.0 (X86)

Sun Solaris 2.6 (SPARC)

Requirements:

Xserver (for Unix)

JDK 1.0/1.1/1.2 installed (can be obtained from www.javasoft.com)

Compilation:

On the xterm window (Solaris) or Command Prompt (NT), run the following statement on current working directory with all necessary source codes.

```
> javac -deprecation *.java
```

Execution:

On the xterm window (Solaris) or Command Prompt (NT), run the following statement where all .class files located

To start the router:

```
> java SRIP_Router
```

To start the DNS Server

```
> java DomainNameServer <port number>
```

Main Program

After execute the command above, the following window should prompt up on screen (Figure UI.1)

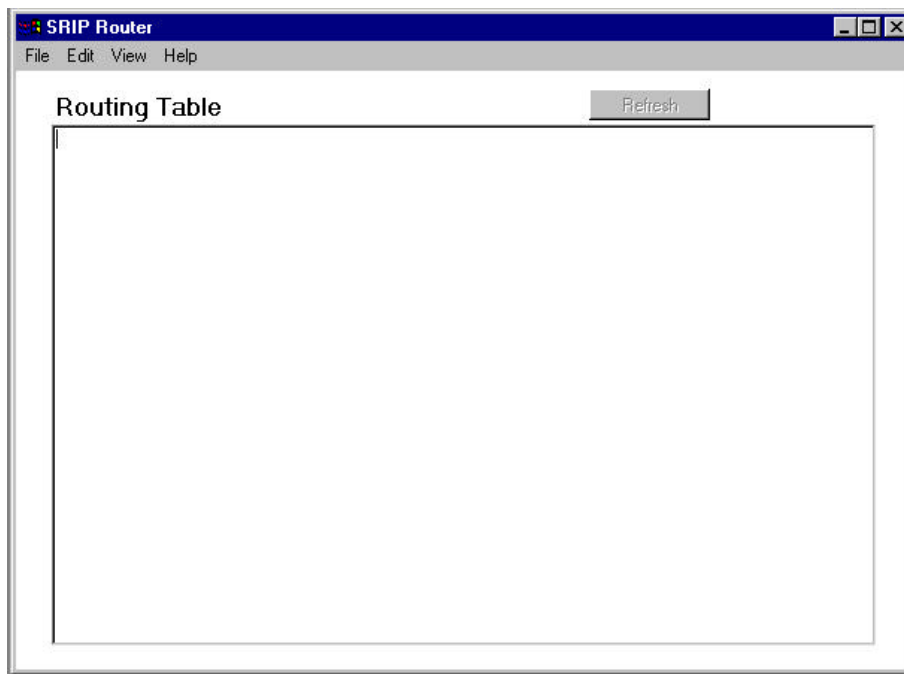


Figure UI.1 (Actual appearance may different from each operating platform.)

To bring up the router, please follow the procedure below:

Step 1: Registration with Domain Name Server

- 1) Go to **File**, click **Register**
- 2) Dialog box (Figure UI.2)
- 3) Insert the DNS Server address and port number
- 4) Enter the Router Name (must be unique)
- 5) Input the Routing Table Update Interval
- 6) Input the Hello Interval
- 7) Click **OK** to proceed and the title of the application will show up the Router Name

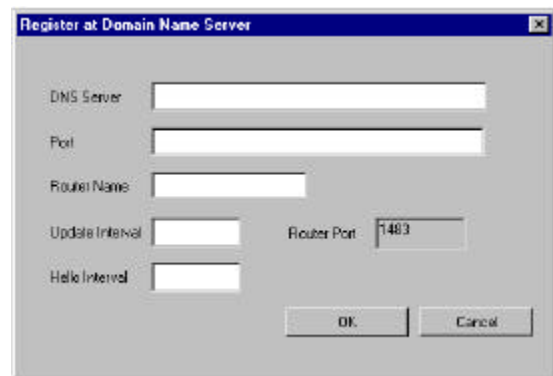


Figure UI.2

Step 2: Add Neighbor for current Router

- 1) Go to **Edit**, click **Add New Neighbor**
- 2) A dialog box will prompt up as Figure UI.3
- 3) Input the Router Name (Not IP Address) on the text box
- 4) Press **Enter**
- 5) The name will show up in the list box.
(Note: Router will not accept the same Router Name as itself)
- 6) After entering all desired Router Name, click **NSLookup** button.
- 7) The corresponding IP address and port number will show up on the next screen. (Note: All invalid Router Name will be omitted.)
- 8) Click **OK** to proceed.

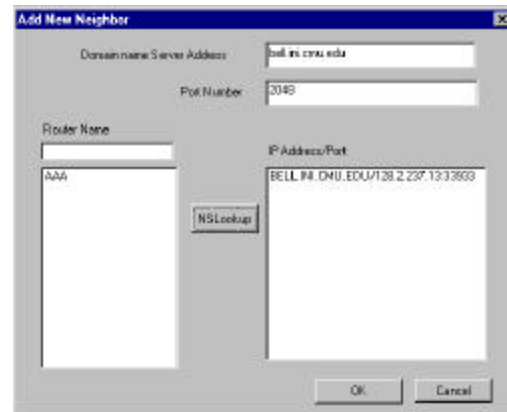


Figure UI.3

Step 3: Starting the Router

- 1) Go to **File**, click **Start Router**

At this stage, the Router is running and try to connect to neighbor. To see the Routing Table information, click **Refresh** button or go to **View** and click **Refreshing Routing Table**. The table will be pulled up in the text area in main window as shown in Figure UI.1.

Users are free to change some of the configuration during runtime.

To **add new neighbor**, please follow the same *Step 2*.

To **configure virtual router**:

- 1) Go to **Edit**, click **Edit Virtual Router**
- 2) A dialog box will prompt up as Figure UI.4
- 3) Please input all necessary

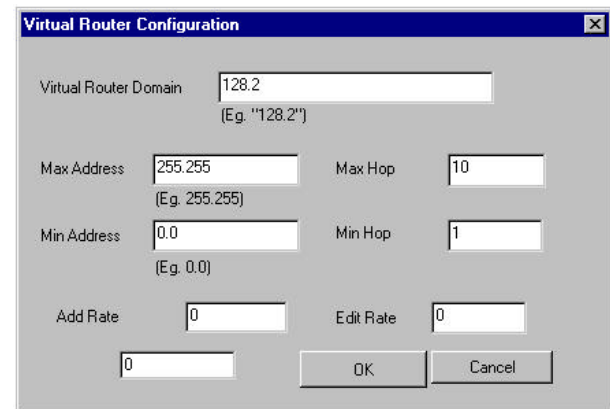


Figure UI.4

To carry out **Be/Cease Neighbor Request**:

- 1) Go to **View**, click **Neighbor List**
- 2) A dialog box will be shown as Figure UI.5
- 3) Double click the desired neighbor name
- 4) Check/Uncheck the **Neighbor** checkbox
- 5) Click **OK** to proceed.

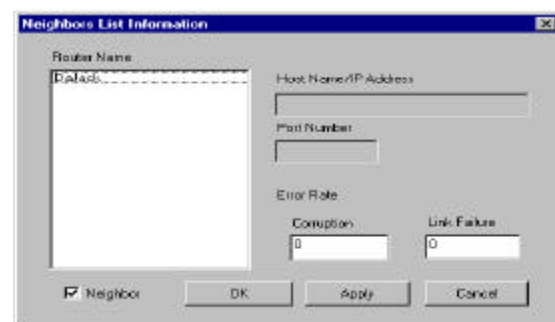


Figure UI.5

To view Neighbor Information, please follow the steps above.

To look at the statistic of packets transmitted:

- 1) Go to **View**, click **Statistics**
- 2) A dialog box will be shown as Figure UI.6
- 3) Select the desired connection from the selection box
- 4) The corresponding data will be shown on screen
- 5) Click **Refresh** to get updated data

The Statistics dialog box displays the following data:

	Actual Value:	Percentages
Attempt to send	258	N/A
Corrupted	0	0.0
Dropped	0	0.0
Actually sent	258	100.0
Received	0	N/A
Error Free	0	NaN

Buttons: Refresh, OK

Figure UI.6

There is an help provided, please go to **Help**, click **Help Topics**, the help menu will show up as Figure UI.7. Please click the desired topics for help statements.

The Help dialog box contains the following content:

Help

- Register Router
- Start Router
- Add New Neighbors
- Edit Virtual Router
- Neighbors List
- Statistics
- Getting Start**

To get started, please following this procedure:

- 1) Register the current Router (File->Register)
- 2) Add new neighbor lists onto the Router (Edit->Add New Neighbor)
- 3a) Configure the Virtual Router if necessary (Edit->Edit Virtual Router)
- 3b) Set the corruption and drop rates if necessary (View->Neighbor List)
- 4) Start the Router (File->Start Router)
- 5) User can configure the Virtual Router, corruption and drop rates during runtime.

Button: OK

Figure UI.7

DIAGRAM

