Version 5 (4/9)

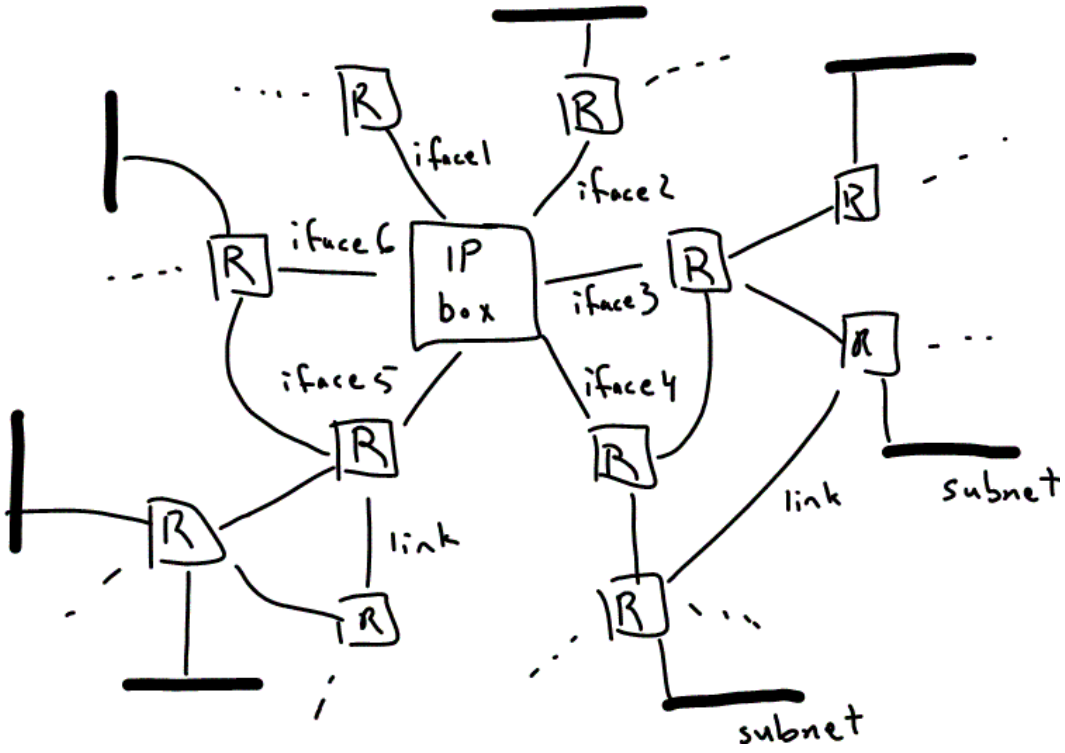
Project 3: Routing and Forwarding

Goal: To build a spanning tree routing algorithm and a best match first forwarding table

Task: This task will build on the functionality you implemented for project 2. Specifically, you will use the **send-config** streams socket, the **app** socket, and six UDP sockets, **iface1** through **iface6**. Once again the **send-config** socket will be used to control the test. The **IP box** will do the following:

1. Receive configuration information from the **test box** over the send-config socket. This configuration information will include an IP address for each ifaceX interface;
2. Receive a series of link-state update "packets" via the send-config socket. From these, the IP box will build a best match first forwarding table.
3. Receive UDP packets from the ifaceX and app interfaces. Unlike project 2, these packets will have valid IP addresses associated with them. The IP box must use the forwarding table to determine over which interface to forward the packet.

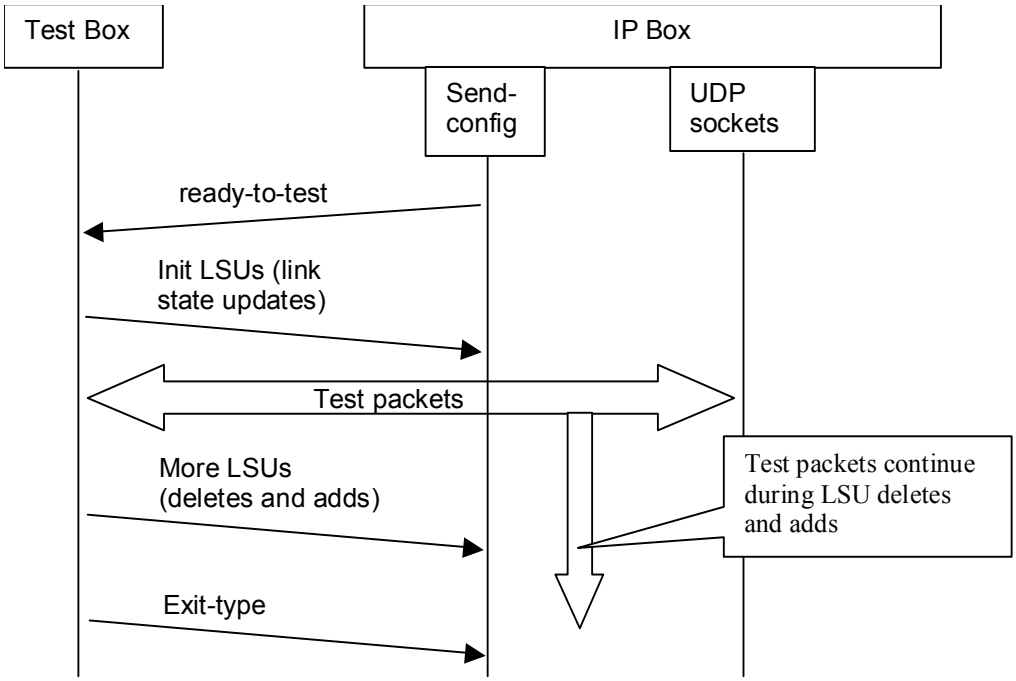
The figure below illustrates the emulated scenario. Specifically, the IP box appears as a single router in a network of routers and subnets. It has a set of up to 6 neighbor routers (correspdonding to iface1 – iface6). These routers are in turn connected to other routers. Any of the routers may also have subnets attached to them, with each subnet having a subnet address (address and mask) associated with it. The test box, through the send-config socket, will configure all of this information into the IP box as a series of link-state update packets.



As with project 2, packets received and sent over the app socket will have the "API header":

| src-ip (32) | dst-ip (32) | src-port (16) | dst-port (16) | TTL (8) | ToS (8) | Length (16) |

Note that neither incorrect nor segmented packets will be sent to the IP box for project 3. (For the speed test, however, you can expect to get fragmented packets, and you must set the checksum correctly.)



The test operates as follows (and is summarized in the figure above):

1. The IP box opens the eight sockets described above.

2. The **send-config** socket connects to the **Test box** (on an IP address and port number hard coded into your program) and sends out a "ready-to-test" command.

This command contains the IP addresses and port numbers of the eight sockets. The format of this command is as follows:

addr1\nport1\n . . . addr8\nport8\nstring\ntest-type\nnet-id\n

where,

addr1	the ip address of the send-config socket in string format
port1	the port of the send-config socket in string format
addr2	the ip address of the app socket in string format
port2	the port of the app socket in string format
addrX	the ip address of the ifaceX socket in string format (where X goes from 1 – 6)
portX	the port of the ifaceX socket in string format (where X goes from 1 – 6)
seed	the seed digit of a random number generator when you are still testing out your program, and is the string 'final' when you have completed your testing and want to submit it run the program for grading. If you set the seed digit to 0, then the test box will randomly generate its own seed. If you set the seed digit to 1, then the test box will generate only a small set of initial LSUs.
net-id	your net-id (this allows us keep record of your program's performance)

\n is the newline character

Note that this format is essentially the same as project 2, except that the app socket has been moved to be the second pair of entries, and that there are 4 additional iface sockets (total of 6).

3. The test box then sends a set of address assignments via the send-config socket. These will take the following form:

"list-of-addrs\nu_addr1\nu_addr2\nu_addr3\nu_addr4\nu_addr5\nu_addr6\n\n"

where u_addrX is in string notation ("a.b.c.d"). These addresses are assigned to interfaces iface1 – iface6 correspondingly. Note that these addresses (u_addrX) are "emulated" IP addresses—that is the IP addresses that may appear in your user-level IP headers. This is not to be confused with the addresses conveyed over the **send-config** socket in step two, which correspond to actual physical addresses and tell the test box how to actually talk to your IP box.

When the router receives a packet addressed to one of these addresses (over the interface corresponding to the address), then the router must deliver that address to the app socket. Also, the IP box will consider its own ID (for the purpose of interpreting LSUs, see step 5) to be u_addr1.

4. The test box then sends a list of neighbors to the test box. This models the neighbor configuration that would typically occur at a router. This list takes the following form:

"list-of-nbrs\nid1\nid2\nid3\nid4\nid5\nid6\n\n"

where idX is the identifier of the neighbor (in IP address string notation "a.b.c.d"). The IP box interprets this list to mean that its neighbor over iface1 is id1, its neighbor over iface2 is id2, and so on.

5. The test box then sends an initial series of link LSUs via the send-config socket. There are two types of these. One has the following form:

"lsu-link: action from-id to-id cost\n"

where:
action is either "add" or "delete",
from-id is the router at the sending end of the (unidirectional) link,
to-id is the router at the receiving end of the link,
and cost is the metric value to use in calculating the spanning tree.

From-id and to-id are IP address strings, and cost is an ASCII string number ranging from 1 – 255. The value 0 is unused. This LSU obviously is used to determine the router topology.

The other LSU has the form:

"lsu-subnet: action id subnet\n"
where:
action is either "add" or "delete",
id is the router that is attached to the subnet, and
subnet is the subnet reachable at that router

The subnet is expressed in "slash" notation (i.e. 20.1.128/17). It is possible to have one subnet that is a subset of another (i.e. 20.1.128/17 and 20.1.129/24, where each is reachable via a

different id). Finally, a default route will be denoted as 0/0. Packets destined to a subnet that is reachable at your IP box should not be transmitted over any socket interface.

Note that the initial series of LSUs will have action of “add” only. No LSUs will have “delete”. This set of LSUs is designed to emulate when a router attaches to the network and gets a routing table dump from one of its neighbor routers.

The lsu-link and lsu-subnet LSUs allow the IP box to calculate the shortest path to every destination subnet.

Note that there is no LSU that adds or deletes a node per se. You will learn the existence of a node by its inclusion in one of the LSU types. Likewise, a node being “deleted” will appear to you as the loss of a number of links (the links from the deleted node’s neighbors to the deleted node).

6. The test box then sends a string that signals the end of the initial LSUs:

“end-of-init-lsu\n”

Upon reception of this string, the IP box must generate its forwarding table. It does this by running the shortest-path-spanning-tree algorithm over the set of LSUs, and then building a forwarding table from the resulting next hops.

7. The test box then sends a series of packets to the IP box. Unlike project 2, the IP addresses in these packets will be valid IP addresses that must be forwarded over the correct interface according to the forwarding table you calculate. Packets may be received via the app socket or any of the ifaceX sockets. Packets destined for one of the IP box’s 6 iface addresses should be forwarded over the app socket. Packets destined for one of the subnets reachable at the IP box should not be forwarded anywhere. All other packets must be forwarded over the appropriate iface socket to the test box.

Note that the test box may still send bad packets, as defined in project 2. The test box must drop these packets accordingly.

During this period of the test, the test box may send additional LSUs!!!! These LSUs may remove links, or subnets previously established, or may add new ones. Your IP box must calculate the new forwarding table while simultaneously still forwarding received UDP packets. The way to do this is to maintain a pointer to the “current” forwarding table. When you receive new LSUs, you calculate a new forwarding table (separate from the current one). Once the new forwarding table is done, you modify the pointer so that subsequent packets are forwarded on the basis of the new forwarding table. In this way, there is no point in time where there isn’t some operating forwarding table.

Note, however, that there is a brief period of time when some packets may be misrouted (i.e. the new forwarding table hasn’t been installed yet). This is normal operation in routers and cannot be avoided. The test box will allow 10 seconds after an LSU during which packets may be forwarded either based on the old table or on the new table. After 10 seconds, however, we expect the IP box to be using the new table.

Note that some LSUs will come in “groups”, in the sense that you may receive several LSUs at about the same time. This represents the case where a node and its links are added, or a node is deleted and all of its links must be deleted. I suggest that you wait a short time (no more than a second) after the reception of an LSU to see if you are going to get any more before calculating the forwarding table. Alternatively, you may start calculating the forwarding table as soon as you get an LSU, but be prepared to interrupt that calculation and start over should you immediately receive another LSU.

8. The **Test-box** then sends an exit message to the **IP-box** at it's send-config socket. The message will be formatted as follows:

"type\nsome set of debug statements\n\n"

type	Debug statements
exit-fail	The debug statements will be whatever appropriate, and will typically include a full copy of the expected forwarding table.
exit-good	The string "ok". This is also only used for a non-final test run.
exit-final	The hashed timestamp. (The hash is sent in string format implying 40 chars for the 160 bit hash) – you must **record** this as proof of your execution!

Test Criteria: In addition to the test criteria of project 2, this test passes if packets are sent over the proper interface (or not sent if they should not be). You can get partial credit if most but not all of your packets are correctly routed.

IP box specifications

The IP box must be able to handle up to 500 "active" LSUs.
The IP box must be able to support a forwarding table of up to 500 entries.

(Note that for the speed test at the end of the semester, these values will be set at 10000 LSUs and 10000 forwarding table entries!!!)

Other details

We will post on the newsgroup when the test box is running, and over what port(s). We will have a web page where any changes may be posted.

Note: you must turn in your code to CMS as well as pass the test box test by the due date!