CS118 Project 2

Communication in the Internet of Things

Simple Distance Vector Routing Protocol in C/C++

Team 11
Kevin Lu 504129624
Ryan Voong 404317866
Hee Hwang 804212513

Demonstration

- `$ make`
- `$ ./start-router`
- `$ ./inject "Hello World"`

Implementation description

- Files
  - **my-router.cpp**: our main file
  - **DV.h**: the interface file for the DV class and other structures
  - **DV.cpp**: the implementation of DV.h
- Structures
  - **dv_entry**: each router maintains its own routing table implemented as an array of dv_entry structures. Each dv_entry contains the cost of the shortest path to a particular destination and the next hop along the shortest path.
  - **node**: contains basic information about a single router. Each router maintains a vector of node structures, representing the information for the router's neighboring routers. Such information includes the ID of the neighbor, the port number to reach the neighbor, the timer (to determine if the router is still valid), and the neighbor's socket address.
  - **DV**: the wrapper class that contains all information that the current router holds. It includes its address, ID, routing table, and neighbor node structures.
- Routing Table
  - Each router maintains an up-to-date routing table, as well as a backup routing table containing the initial routing information. The up-to-date routing table updates its entries based on the information contained in a received advertisement. It also keeps track of which routers in the network are invalid, making sure to route in a manner that avoids such invalid routers.
- Packet Details
  - A packet typically contains a header followed by a payload. Depending on the type of packet, some packets have empty payloads.
  - **header**: contains the type of the packet, the ID of the packet source router, the ID of the packet destination router, and the length of the payload. On certain occasions, we use the header fields to carry other information that we want to pass along to other routers (for example, TYPE_RESET packets have an empty payload, so it carries the "hop count" in its header length field).

    - **payload**: we can specify what we want to carry in the payload for TYPE_DATA packets through the argument to the "inject" script (`./inject <message>`).
- Packet Types
    - **Data**: indicates that the packet contains data that should be forwarded to a particular destination.
    - **Advertisement**: contains a router's distance vector, used to update neighboring routers' routing tables. This packet is sent to the router's neighbors periodically.
    - **Wakeup**: each router has a child process that periodically sends this packet to its parent process, waking up the parent from listening for packets. Through this method, the parent process is able to perform periodic tasks like sending its distance vector advertisement to neighboring routers and checking whether or not neighbors' timers have expired (to determine if they are invalid).
    - **Reset**: alerts the router that one of the routers in the network has become invalid, causing this router to reset its routing table to its initial configuration (loaded from file), and also to tell its neighbors to reset their routing tables as well.
- Other Details:
    - For efficiency purposes, each router broadcasts its distance vector every second. Also, if a router does not get any advertisements from its neighbor for 5 seconds, that neighbor is rendered invalid.

Difficulties

- **Determining how a router receives and sends packets at the same time**: Although most of the packet sending is in response to received messages, each router must also send periodic advertisements to its neighbors, regardless of whether or not it has received a packet. Using a simple receive-and-then-send loop would potentially cause the entire network to become inactive, as one router's inability to send packets could cause a neighbor to never receive any packets, causing that neighbor to also never send any packets. To fix this problem, we forked the program and maintained two separate processes for each router. While the parent process uses a receive-and-then-send loop, the child process periodically sends a packet to the parent process. That way, the parent process can wake up periodically and send advertisements to it neighbors.
- **Determining how to update a router's routing table based on a received advertisement**: We decided that each advertisement would contain the distance vector of the sender, and the receiver router would update its routing table by comparing its current shortest cost to each destination with the cost of the path to the destination going through the routing that sent the advertisement. This cost is simply the sum of the cost to reach the sender and the cost from the sender to the destination. Through this routing table updating scheme, we were able to see each routing table in the network converge to an optimum set of routes.
- **Having a router detect if a neighboring router is invalid**: A router performs detection by maintaining a timer for each of its neighboring routers. Every time the router receives an advertisement packet, the router resets the timer corresponding to the neighbor that delivered the packet. Because each valid neighboring router advertises periodically, the router detects that a neighbor has become invalid if it hasn't sent any advertisements in a while. Once the router receives an advertisement from the invalid neighboring router, that neighbor is treated as valid again.

- **Alerting the rest of the network that a router is invalid**: Our method for reacting to detection of invalid routers is for every other router in the network to reset its routing table to its initial configuration (as loaded from the topology.txt file). This requires that the routers that detect the invalid router must broadcast to the rest of the network. To prevent broadcast storms, we set a limit on the number of hops a "reset" packet can travel. Whenever a "reset" packet is received, a counter in the packet header is decremented before passing the packet off to the neighboring routers. Whenever a "reset" packet's counter reaches 0, the packet is no longer forwarded. We initialize the counter to n-2, where n is the number of routers in the network. This ensures that every router receives the packet at least once: consider the worst case, a network where routers R1, R2, … Rn are connected in a single file line. When R1 becomes invalid, R2 detects that R1 is invalid, and sends a packet to R3 with counter n-2. We can see that when the packet has propagated to Rn, the counter will equal 1. So all routers in the network receive the packet at least once, and the packet will not propagate forever.

Routing Tables (after convergence)

- Note: for a given router, -1 in the "nexthop" and "cost" fields means that the destination is either the router itself, is unreachable, or is invalid.

- Router A

dst	nexthop	cost
A	-1	-1
B	10001	3
C	10001	5
D	10001	7
E	10005	1
F	10001	4

- Router B

dst	nexthop	cost
A	10000	3
B	-1	-1
C	10004	2
D	10002	4
E	10005	2
F	10004	1

- Router C

dst	nexthop	cost
A	10001	5
B	10004	2
C	-1	-1
D	10003	2
E	10001	4
F	10004	1

- Router D

dst	nexthop	cost
A	10002	7
B	10002	4
C	10002	2
D	-1	-1
E	10002	6
F	10002	3

- Router E

dst	nexthop	cost
A	10000	1
B	10001	2
C	10001	4
D	10001	6
E	-1	-1
F	10001	3

- Router F

dst	nexthop	cost
A	10001	4
B	10001	1
C	10002	1
D	10002	3
E	10001	3
F	-1	-1