CSE160: Computer Networks

Project 2 – LS Routing High Level Design

2020-10-02

Professor
Alberto E. Cerpa

Design Choices

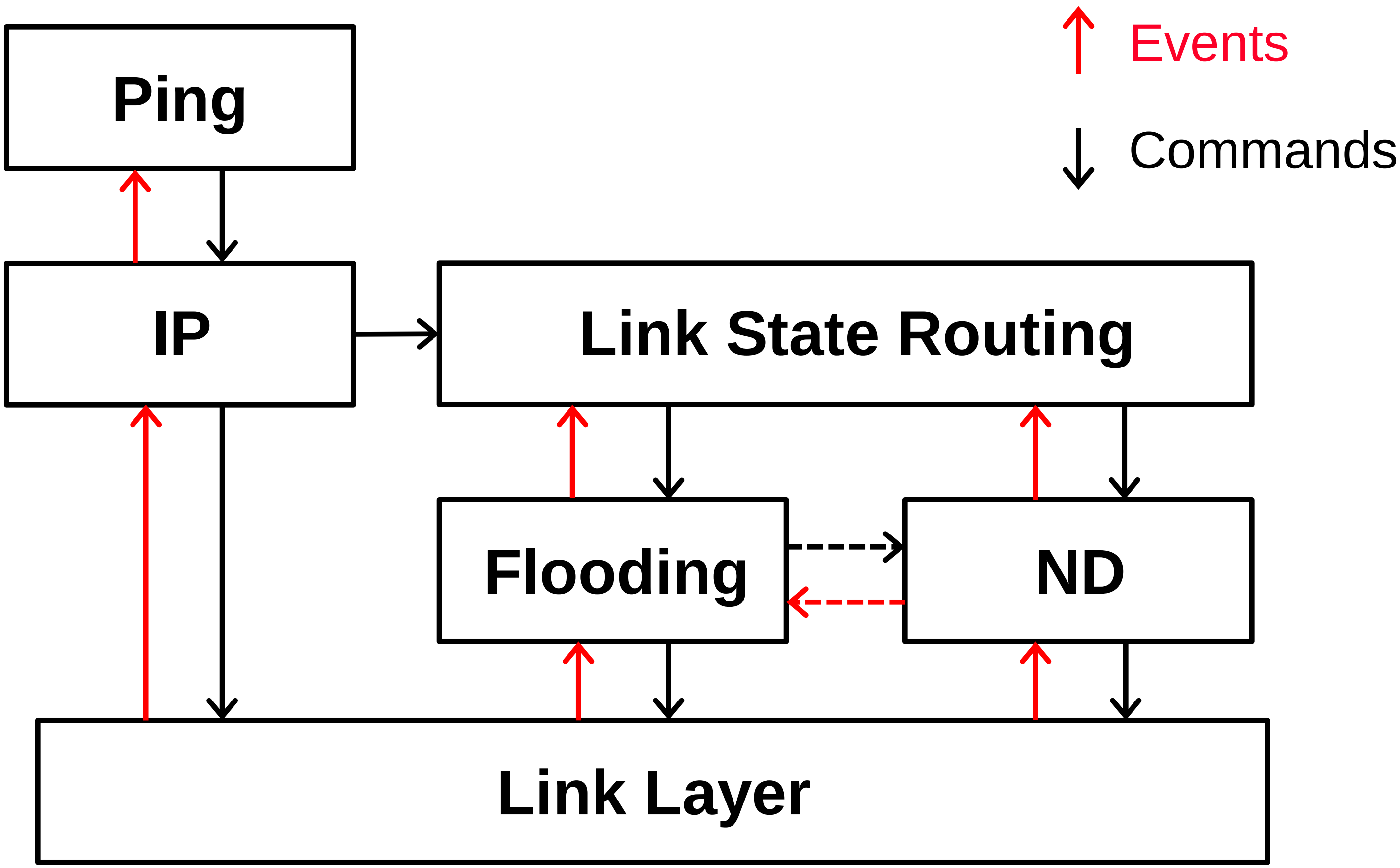
- In the following slides, I propose a high-level design scheme for project 2
- **HOWEVER**, this is **NOT THE ONLY** way you can design project 2
- The bottom line is that whatever it is your design, you should try to work on:
  - Scalability: modular design with clean interfaces → easy to use by other modules
  - Efficiency: reduce the overhead as much as possible (not at the expense of scalability)
- Ultimately, **you** are in control of your code

Routing Design

- The cleanest design involves having a module just for Link-State Routing (LS)
- In addition, you will need to develop an IP module (like the one suggested for the Link Layer in project one)
- The IP module will do all the data forwarding, using the routing table information provided by LS
- This IP module can be used by other applications when ever they need to use the IP service (**Hint: project 3!!**)
- In this project, we can test the IP data forwarding functionality by using Ping on top of the IP service
- The LS module will oversee the update of the Routing Table used by the IP module

Routing Design (2)

- In addition, you will need both a Neighbor Discovery (ND) and Flooding (F) modules with appropriate interfaces to hook everything up
- The whole stack could look like this:



LS Advertisement (LSA) Packet Format

- The LS advertisement (LSA) packet should include:
  - Source Address of the node sending the info
    - This information may be obtained from the Flooding Header if passed to the LS Event Handler (omitted in LSA then)
  - Monotonically increasing sequence number to uniquely identify the LSA update
  - A list of tuples with the following format:
    - Tuple$_n$: <Neighbor Address$_n$, Cost$_n$>
    - List: [<Neigh Addr$_1$, Cost$_1$>, [<Neigh Addr$_2$, Cost$_2$>, …]
  - You could also add other fields like number of tuples in the list, or any other thing you may deem necessary
- To access the payload at any layer, your header structures should end with a zero-length array
  - Please use the trick/method described in slide 27 (Why a zero-length array?) of the C Tutorial (Files $\rightarrow$ Projects $\rightarrow$ C-Tutorial.pdf)

LSA Update Cache

- It may be useful to maintain an LSA update cache in your design, keeping track of the last sequence number for any advertisement sent by any node
- This is useful to discard old LSA updates with old (stale) information
  – If your node receives an LSA update from a node, with an older sequence number than the one in your LSA update cache, it gets discarded
  – This means that **no** new shortest path computation will be run [see later]
- The structure could be like the flooding cache you already use in the Flooding module
  – You need one entry per node in the topology in the cache (like Flooding)

Bootstrapping

- Initially, you don't want to start the Dijkstra's SP calculation until you receive info from all the nodes in the system
  - This allows the node to calculate SP to all destinations once you have complete topological knowledge
- The timing sequence should be roughly:
  - ND starts immediately, learning the list of local neighbors
  - If Flooding depends on ND, the node needs to wait until ND discovers all neighbors
    - This can be done, for example, by posting a task that periodically checks if the neighbors have changed
    - The Flooding event handler can keep the state whenever ND signals for a neighbor update (e.g. increase the size of a variable)
    - If after the timeout the state has changed, it reposts itself; if not, it starts the process
  - Since LS depends on ND, we can use a similar process as above
  - LS must also wait to receive all the initial LSAs from all the nodes
    - Use a timer before starting the process

Initial SP Computation

- Once the node has received the initial LSA updates from all nodes in the system, the node has all the info to start computing SP to all nodes
- The computation is run once at the beginning, and then after any update [see later]
- For ideas on how to implement Dijkstra, look at Peterson Chapter 3.3 and the class Lecture 09
  - You could also use your CSE 100 code that asked you to implement Dijkstra (I think I assigned it when I taught it)
  - Remember to save not only the **Cost**, but also the **Next Hop** information (it would be a good idea to also save a few alternative paths too, e.g. top shortest, 2nd and 3rd shortest paths)
- The output of the algorithm, should be used to update the Routing Table…

Routing Table

- This is the main data structure that is used for data forwarding
- It looks something like this:

Destination	Next Hop	Cost	Backup Next Hop	Backup Cost
3	5	7	4	9
9	11	5	10	14
…	…	…	…	…

- You can decide what data structure to use (your choice)
- Your LS module should have commands to access the content of the table by other modules
  - This is used by the IP module to perform data forwarding

Adaptation to Dynamics

- When the ND module discovers a change in the neighbors list, it should signal to all the other modules wired to it
  - Changes can be:
    - Adding a new node
    - Removing an old unresponsive node
    - Changing the cost of an existing node
- Note that the ND module should be implemented to signal an event to the LS's event handler every time there is a change in the Neighbor List
  - This should have been already implemented if your Flooding module was wired to the Neighbor Discovery module for proj 1
- This signal triggers two actions (called triggered update):
  1. A new LSA should be flooded to all nodes in the system
  2. A new SP computation should be done [see next]

SP Computation in Steady State

- After the node has run the initial SP computation, the node should only run the SP computation every time there is a change in the topology
- So, in steady state, the SP computation is run every time that:
  - The node receives a new LSA update from another node
  - The local ND module changes the neighbors list (by id or cost of the node) [see previous slide]
  - The points above are the ones that trigger a change in the topology!

IP Module

- The IP module oversees doing data forwarding
- The IP module consults the routing table with the list of shortest paths to every destination maintained by the LS routing module
- The IP header has the following fields:
  - Source address, Destination address, and a TTL field
  - The header could have an optional Protocol field (like the IPv4's Protocol field and the LL Type field, see next slide)
  - This is **NOT** absolutely required, since you are going to use only Ping in this project, and next project you will wire only a TCP module
  - If you ever want to wire multiple modules concurrently, then this field is required for (de)multiplexing
  - This could be very useful for live debugging, so I would add it
    - E.g.: sending pings directly over IP (no reliability) and then over TCP (with reliability) and check the difference

Data Forwarding

- The forwarding rules are as follows:
  - If a packet has an IP destination address that is **NOT** the current node, then the packet must be forwarded
  - The next hop is determined by consulting the routing table maintained by the LS routing module
  - If the packet has an IP destination address that **matches** the current node, then the packet is delivered to the local module expecting the packet
  - In this latter case, the packet is **NOT** forwarded to any other node
- Flooding and ND Packets:
  - In this design, Flooding and ND packets are sent directly over the Link layer, so they should **NOT** be carried by the IP layer
  - In the explanation of project 2 (in the description), flooding is somehow integrated with data forwarding, but I find this may be confusing for many students

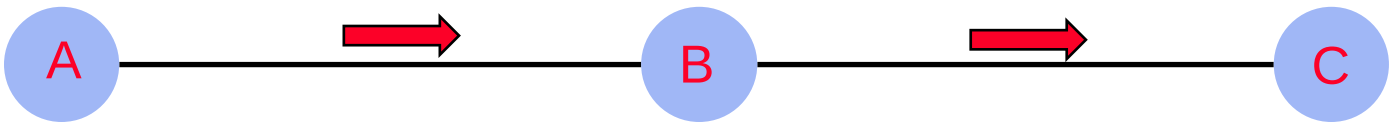
Example: LSA Distribution

LSA Packet	Source	Seq Num	Num Entries	Node	Cost

Flooding Header	Flood Source	Sequence Number	Time to Live

Link Layer Header	Source Address	Destination Address	LL Type

Node A's LSA



LSA	A	27	1	B	1
FH	A	52	15		
LLH	A	B	Flood		

LSA	A	27	1	B	1
FH	A	52	14		
LLH	B	C	Flood		

- Flooding works as before like in project 1 (no explanation)
- Notice that we added a Link Layer Type in the LL Header
  - This is because the Link Layer is multiplexing connections from the IP, Flooding and Neighbor Discovery modules in our design → need identifier in the header to know which module to signal an event when packet arrives at the LL module
  - This is like the Ethertype field in the Ethernet Header

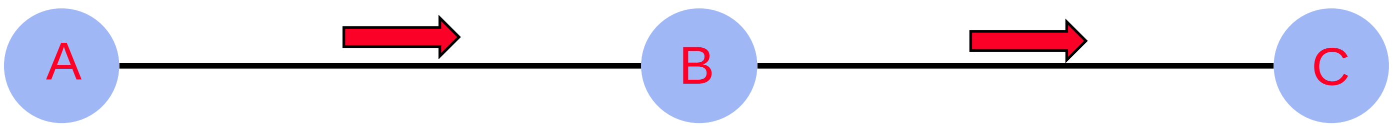
Example: LSA Distribution (2)

- The Link Layer source and destination fields change hop by hop
- The only field that changes hop by hop in the Flooding Header is the TTL
- The LSA Packet never changes hop by hop
  - It should be forwarded by Flooding AS IS
- Please note that in this example, node A has only one neighbor (B), so the list in the LSA is very short (just one tuple)
  - In your design, you should NOT send just one tuple at a time, you want to send as many as you can fit in a single packet
  - In TinyOS, by default, the maximum packet size is 29 bytes, but there is some overhead at the lowest levels of the stack
  - This means that for nodes with lots of neighbors, you may need to send (flood) multiple LSA packets
  - Consult the TOS Packet Format in the code

Example: Data Forwarding

Ping Packet	Ping Payload		
IP Header	Source Address	Destination Address	Time to Live
Link Layer Header	Source Address	Destination Address	LL Type

Node A pings C



Ping	Ping Payload		
IP	A	C	15
LL	A	B	IP

Ping Payload		
A	C	14
B	C	IP

- A sends a ping to C
- C is **NOT** directly connected to A, i.e. multi-hop routing is required
- B forwards the IP packet to its destination

Example: Data Forwarding (2)

- Data forwarding as the packet traverses the network:
  - Node A's IP module receives a Ping payload from the wired Ping module, with destination IP address C
  - Node A's IP module consults the routing table for destination C destination, and finds that next hop is B
  - Node A's IP module assembles a new IP packet with source, destination addresses and TTL (A, C, 15) and the ping payload
  - Node A's IP module sends the packet to the LL module, with LL destination address B, i.e. the next hop from the routing table
  - Node B's IP module receives the IP packet from the LL module
  - Since the IP destination address is **NOT** B, then B consults the routing table and forwards the packet to C (next hop)
  - Node C's IP module receives the IP packet from the LL module
  - Since the IP destination address **matches** C, it delivers the packet to Ping

Food for Thought

- The current design assumes perfect knowledge and no errors
  - What happens if you get errors?
- A more resilient design might be to have LSAs expire after a certain amount of time
- This design involves periodically sending LSAs before they expire, even if nothing changes in the topology, to renew the LSAs
  - E.g., OSPF’s LSAs expire after 1 hour, and they are renewed every 30 minutes (50 minutes with flooding reduction)
  - A design like this involves getting a timer to send the LSAs periodically, in addition to triggered updates