Computer Networks Project Overview

Benjamin Ferrell
benjamin.ferrell@utdallas.edu

Purpose

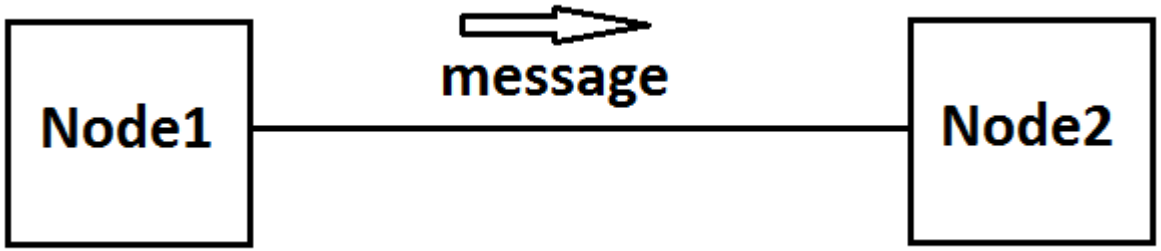
- Simulate a network by implementing the following layers in a node:
  - Transport
  - Network
  - Datalink
  - Channel

Node setup

./node <nodeId> <duration> <destination> [msg] [neighbor]*

- <nodeId> = id of the node
- <duration> = how long the node lives
- <destination> = destination of the message
  - If nodeId == destination, then no message being sent
- [msg] = message being sent
- [neighbor]* = zero or more neighbors

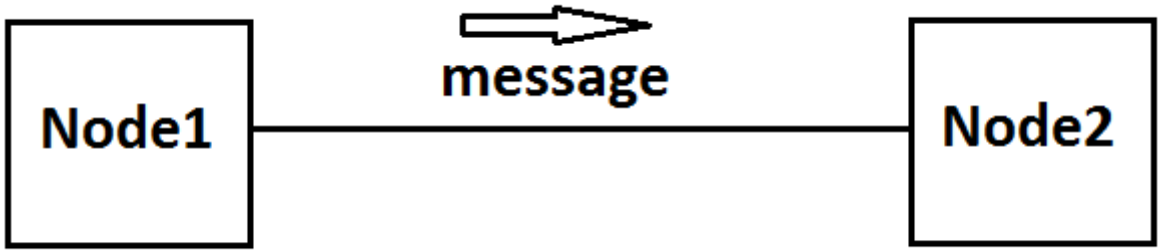
Example



- ./node 1 30 2 “This is a message from 1 to 2” 2
  - Node1 lives for 30 seconds, wants to send a message to 2, and has 2 as a neighbor
- ./node 2 30 2 1
  - Node2 lives for 30 seconds and only has the neighbor node1

Channel Layer

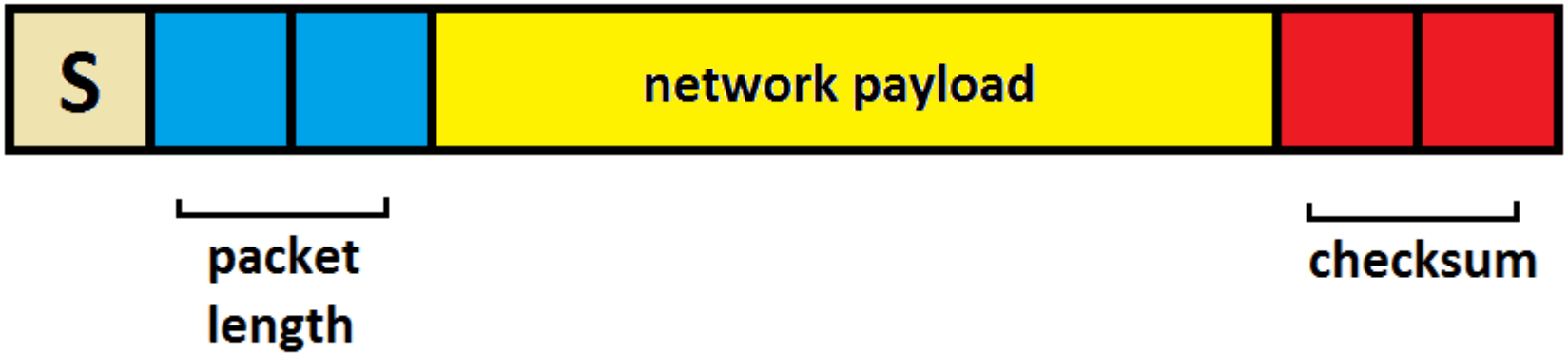
- Simulated by using text files



Writes: from1to2
Reads: from2to1

Writes: from2to1
Reads: from1to2

Datalink Layer



- Packet format (all in ASCII):
  - "S" (start-of-header byte – 1 byte)
  - [0-9][0-9] (length of entire packet – 2 bytes)
  - [payload] (will be sent to network layer)
  - [0-9][0-9] (checksum – 2 bytes)
- Checksum = sum(packet bytes)%100

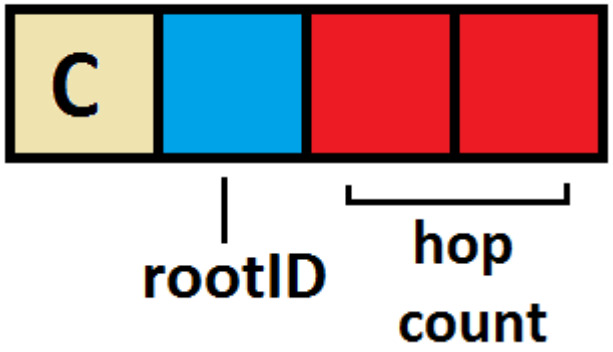
Functions

- datalink_receive_from_channel()
  - Read from all input files (all neighbors)
  - Parse and verify input based on packet specification
    - Send verified payload to network layer
    - If error is encountered, try to recover by searching for next start-of-header byte
  - Keep track of how much has been read from file
    - Only want to read new data in a file (appended)

Functions

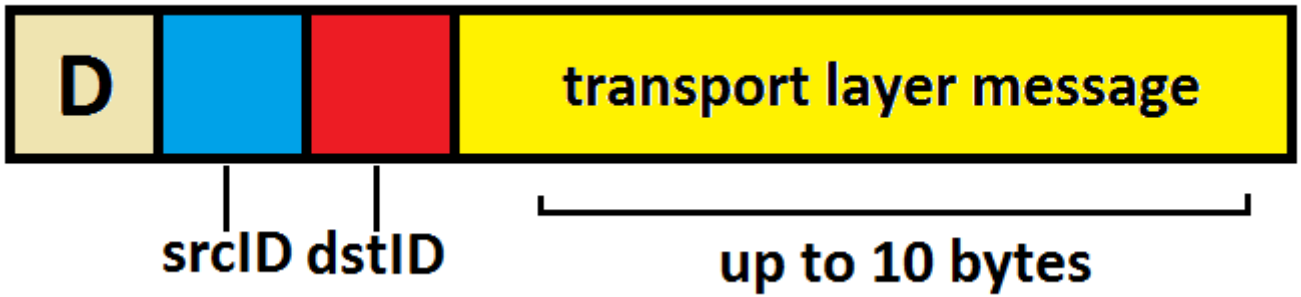
- datalink_receive_from_network()
  - Called by the network layer
  - Builds outgoing packet and writes to file
  - (fairly simple method)

Network Layer



- Configuration Message
  - "C" (Header – 1 byte)
  - [0-9] (root ID – 1 byte)
  - [0-9][0-9] (hop count to root – 2 bytes)

Network Layer



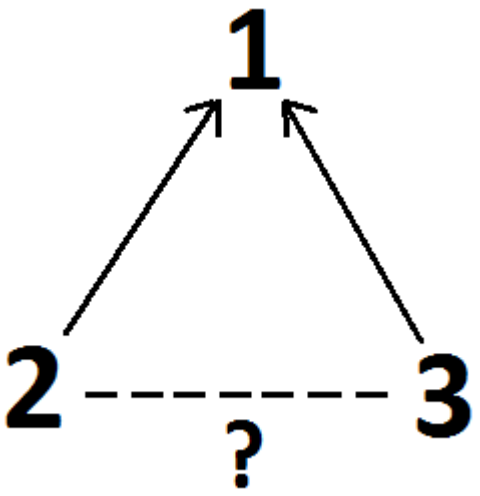
- Data Message
  - "D" (Header – 1 byte)
  - [0-9] (source id – 1 byte)
  - [0-9] (destination id – 1 byte)
  - [message] (sent to transport – up to 10 bytes)

Routing Algorithm

- Build spanning tree of the network
- At each node keep track of (in order of precedence)
  - bestRootID
  - bestHopCount
  - bestParentID
- Edge in spanning tree -> send data msg
- Edge not in spanning tree -> don't send data msg

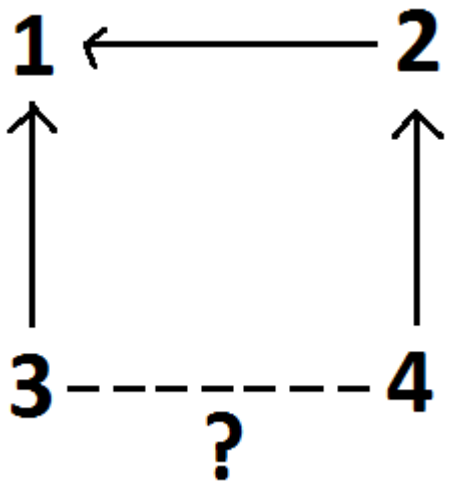
Breaking ties

- 3->2: C101
- 2 learns that 3 is one hop away from root. Since 2 is one hop away from root as well, upon receiving configuration packet, make link inactive. (and vice-versa)



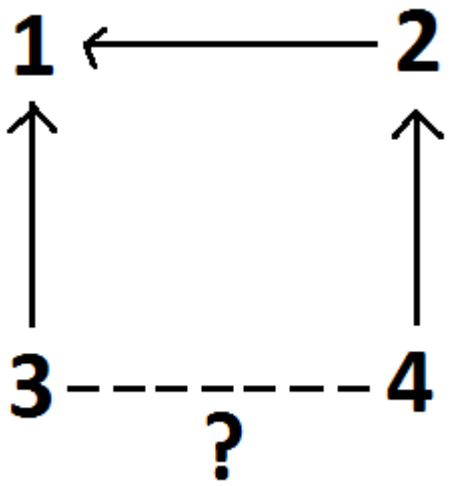
Breaking ties

- 4->3: C102
- How does 3 know it's not 4's parent?
  - Can't restructure config packet
  - Solution: don't send config packets to parents.
- EDIT: Please see slide 15 for alternative solution to slides 13&14.



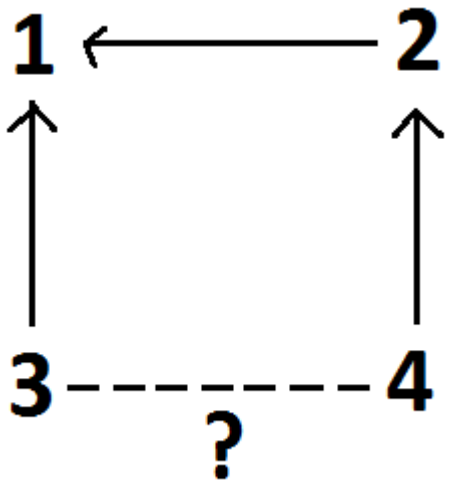
Breaking ties

- 4->3: C102
- Upon receiving config packet with matching root
- If either of the following are true, set link inactive
  - Config # of hops differs by +1
  - Config # of hops differs by -1 AND nodeParent < neighbor



Breaking ties

- 4->3: C102
- How does 3 know it's not 4's parent?
- Don't worry about it, go ahead and send a data packet to 4 when the time arises. Node 4 will know that 3 is not a parent or child so it can drop any data packets it receives from 3. In this setup, you CAN send config packets to ALL neighbors (even your parent).



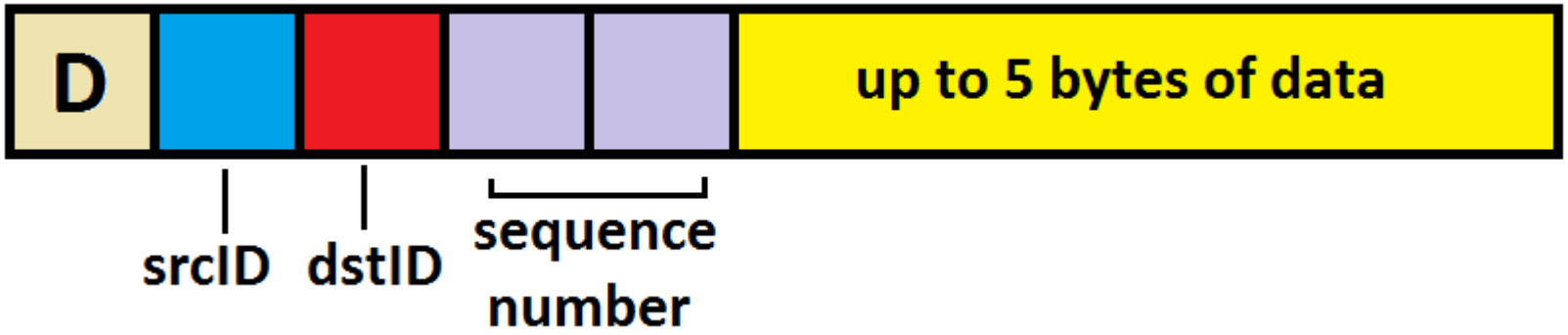
Functions

- network_check_if_root()
  - Sends configuration messages every 5 seconds
  - If >= 20 seconds have passed since last received config message, then become root
- network_receive_from_transport()
  - Builds payload from the message and sends to datalink (fairly simple method)

Functions

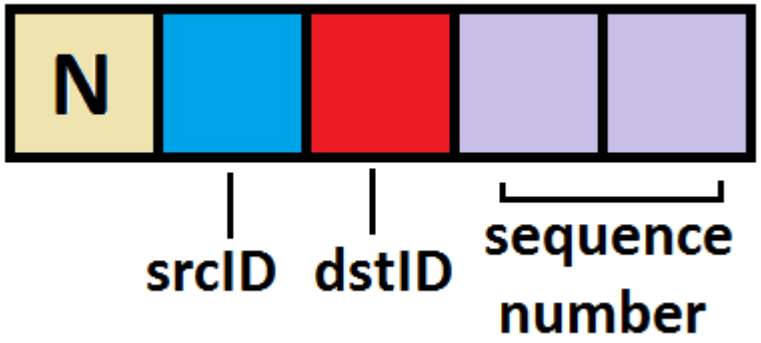
- network_receive_from_datalink()
  - If configuration message, use to maintain spanning tree
  - If data message
    - If at destination, send to transport layer
    - Otherwise, broadcast on all active channels except the neighbor from whence it came.

Transport Layer



- Data message
  - "D" (Header – 1 byte)
  - [0-9] (source ID – 1 byte)
  - [0-9] (destination ID – 1 byte)
  - [0-9][0-9] (sequence number – 2 bytes)
  - [data] (up to 5 bytes of data)

Transport Layer



- Negative Acknowledgement (NACK)
  - "N" (Header – 1 byte)
  - [0-9] (source ID – 1 byte)
  - [0-9] (destination ID – 1 byte)
  - [0-9][0-9] (sequence number – 2 bytes)

Source behavior

- Breaks the message down into 5 byte chunks
- Sends a chunk every iteration (1 second)
- Each chunk has a sequence number
- Keep track of messages sent in case of NACKs

Destination behavior

- Receives packets and reconstructs original message
- Packet with sequence num *i* arrives at time *t*
  - if any packets missing between 0 and *i* at time *t+5*
    - Send NACK for each missing packet
  - If packet *i+1* has not appeared by time *t+5*
    - Send NACK for *i+1*

Functions

- transport_send()
  - Called once per second by main()
  - Sends a packet of data
  - Sends NACKs for data not received
- transport_receive_from_network()
  - Receives message from network
- transport_output_all_received()
  - Output received messages to ‘nodeXreceived’ file

Demo