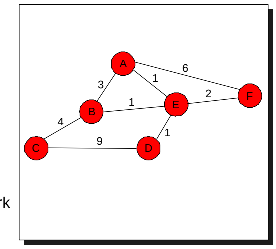
Network Routing and Network Protocols

Arvind Krishnamurthy
Spring 2004

Routing

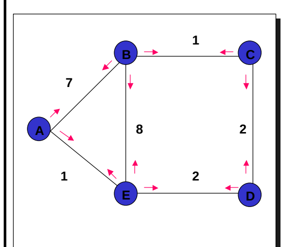
- Routing algorithms view the network as a graph
- Problem: find lowest cost path between two nodes
- Factors:
  - Static topology
  - Dynamic load
  - Policy
- Two main approaches:
  - Link state protocol
    - Each node builds a local copy of the entire network
  - Distance-vector protocol



Distributed Bellman-Ford

- Start Conditions:
  - Each router starts with a vector of distances to all directly attached networks
- Send step:
  - Each router advertises its current vector to all neighboring routers
- Receive step:
  - For every network X, router finds shortest distance to X
    - Considers current distance to X
    - Then takes into account distance to X from its neighbors
  - Router updates its cost to X
  - After doing this for all X, router goes to send step

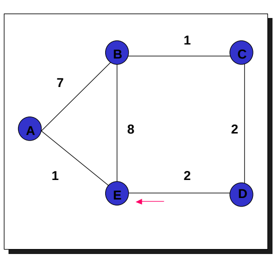
Example - Initial Distances



Distance to Node

Info at Node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	~	2	0

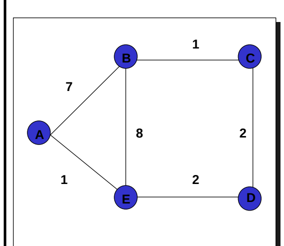
E receives D's routes; Updates Costs



Distance to Node

Info at Node	A	B	C	D	E
A	0	7	~	~	1
B	7	0	1	~	8
C	~	1	0	2	~
D	~	~	2	0	2
E	1	8	4	2	0

Final Distances



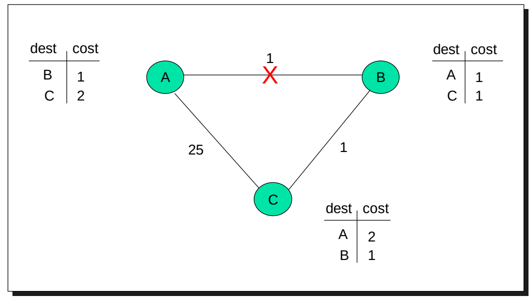
Distance to Node

Info at Node	A	B	C	D	E
A	0	6	5	3	1
B	6	0	1	3	5
C	5	1	0	2	4
D	3	3	2	0	2
E	1	5	4	2	0

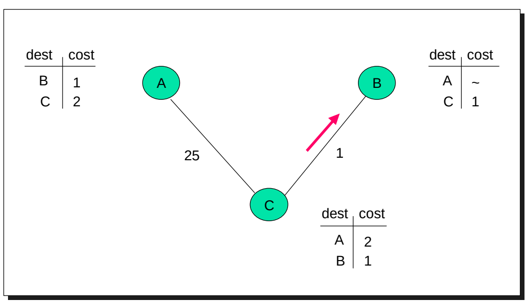
Complexity

- How many steps does it take to converge?
- What is the message complexity of the algorithm?
- How does this compare to link state routing protocol?

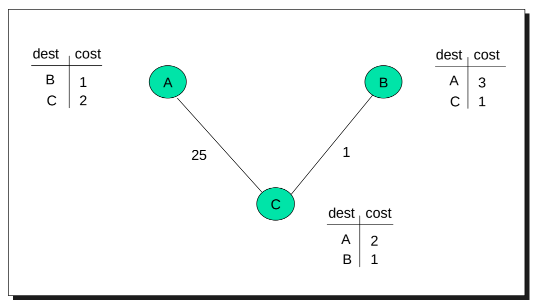
The Bouncing Effect



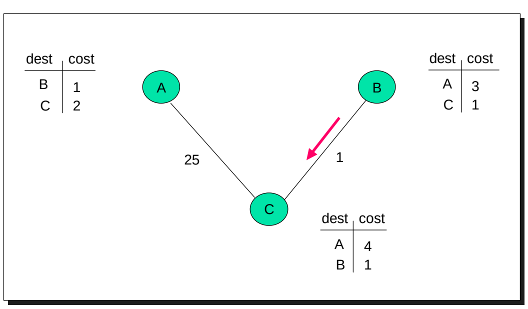
C Sends Routes to B



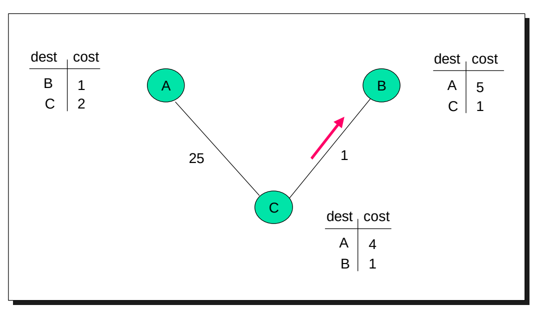
B Updates Distance to A



B Sends Routes to C



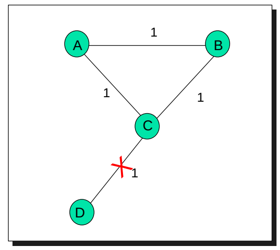
C Sends Routes to B



Solutions

- Problems arise:
  - When metric increases
  - Implicit path has loops
- "Solutions":
  - If metric increases, delay propagating information
    - Adversely affects convergence
  - Split horizon: C does not advertise route to B
  - Poisoned reverse: C advertises route to B with infinite distance
- Works for two node loops
  - Does not work for loops with more nodes

Example Where Split Horizon Fails



- When link breaks, C marks D as unreachable and reports that to A and B
- Suppose A learns it first
  - A now thinks best path to D is through B
  - A reports D unreachable to B and a route of cost=3 to C
- C thinks D is reachable through A at cost 4 and reports that to B
- B reports a cost 5 to A who reports new cost to C
- etc...

Solution: Enhanced Distance Vector

- Each routing update carries the entire path
- Loops are detected as follows:
  - When node gets route check if node is already in path
    - If yes, reject route
    - If no, add self and (possibly) advertise route further
- Advantage:
  - Metrics are local - node chooses path, protocol ensures no loops

Border Gateway Protocol (BGP)

- Designed for scalability
- Granularity is at the level of "autonomous systems" (ASs)
- Usual BGP table has a few thousand entries
- Each entries contains the entire AS-path for getting to a destination
- Uses simple hop-count metric – does not propagate information about bandwidth or congestion in the system
- Some problems:
  - ASes do not necessarily convey packets through shortest paths
    - Some adopt "early exit" strategy – get rid of packet as soon as possible
    - Some send packets only through other ASes with which they have contractual agreements

Networking Software Goals

- Simple
- Scalability
  - Predict what will happen in the future: everything will have a network address
- Heterogeneity (not a goal – but have to support it)
- Robustness: failure, structural changes
  - Something is changing; not a clean reboot
- Performance:
  - Latency: minimum cost (or amount of work to get nothing done!)
    - Measured in time
  - Bandwidth: incremental cost; measured in bytes/second
  - Latency more important than bandwidth
    - Most common mistake in systems is to ignore latency

Issues (Problems to solve)

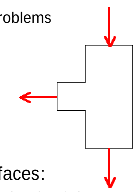
- Link transmission: how do you get a packet of data from one machine to another machine "connected" to it
- Routing
- Naming: mapping names to network addresses
- Multiplexing (how do you share resources, protocols)
- Reliable delivery (cannot guarantee that every packet will be delivered) [ack, timeout, retransmit]
  - Duplicate packets
- Sequencing (process packets in the same order as it was sent; one approach is to have only packet outstanding)

Issues (contd.)

- Fragmentation & reassembly
- Flow control
  - Sender generating data faster than the receiver can handle
  - Feedback required from receiver to sender
- Congestion control
  - Related to flow control; similar in many ways
  - There is more than the sender & receiver
  - Problem gets rediscovered every once in a while!
- Presentation
  - Endian-ness, floating point format
- Security (authentication)

Solution: Layered Protocols

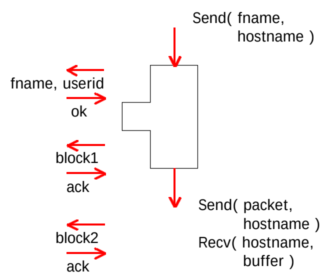
- Collection of protocols
  - Stacked together
  - Each solves one of the problems



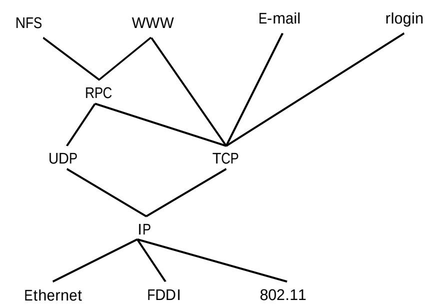
- Protocol has three interfaces:
  - Provides service to higher levels of the protocol stack
  - Depends on some lower transport protocol
  - Has a peer-to-peer interface

Simple File Transfer

- Copy file to remote machine



Protocol Stack



Internet Protocol (IP)

- Datagram protocol (as opposed to stream protocol)
  - No sequencing
  - Stateless
  - Unreliable
  - Host-to-host (not program-to-program)
- IP Functions:
  - Addressing and routing (not naming)
    - Does not know about names
    - Understands addresses
    - Uses route information computed by some other entity
  - Fragmentation (controversial functionality)
    - Other option: let network layer take care of fragmentation

Fragmentation

- If a network has a small packet size, two approaches:
  - Transparent approach at the network level
  - IP fragments:
    - Packet stays fragmented till it reaches destination
    - Reassembled at destination
    - Makes it not stateless!
    - Destination needs to wait for all the fragments to dribble in
      - Keeps track of a partial datagram, and a map of useful parts
      - Packet needs to have a:

    host-id (32 bits), datagram id (16 bits), position (16 bits), length
- IP approach vs. network layer fragmentation/reassembly
  - Question: which is better?

"Time-to-live"

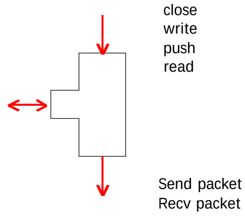
- Field on an IP packet header:
  - 8 bit header (255 secs or ticks)
  - Every router/gateway forwards a packet, it subtracts at least 1 tick
  - When it gets to zero, packet is trashed
  - Prevents packets from roaming around for ever
  - Question: what are the implications of time-to-live?

Features and Limitations

- IP packet headers are variable length:
  - Route that a packet takes can be recorded
  - Source routing: specify the route from the source
- What are the IP limits?
  - 32 bits of address
  - Reliability: requires to get to destination in one shot
  - Speed limitations?

Transmission Control Protocol

- Connection oriented
- End-to-end reliable
- Flow controlled
- Congestion controlled



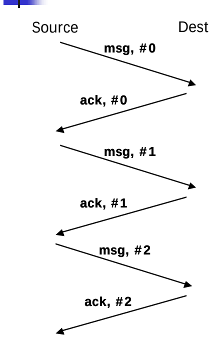
Overall Features

- Reliable
  - Sequence numbers (per byte basis)
  - Acknowledgements
  - Timeout/retransmit
- Flow control
  - "sliding window protocol"
  - Purpose: pipeline communication through overlap
- Multiplexing
  - Several connections to be open (sockets: host, port number)
- Connection-based: state kept at both ends
- Out of band data: "urgent"

Reliable Message Delivery

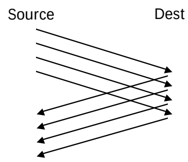
- All of these networks can garble, drop messages
  - Physical media can garble packets or have interference
  - Congestion: too many packets at an intermediate node
  - Destination cannot receive packets as fast as the sender
- What can we do?
  - Detect garbling using checksums
  - Receiver ack's if received properly and timeout at sender
    - If ack gets dropped, sender retransmits
    - Put sequence number in message to identify retransmissions
      - Requires sender to keep copy of all packets sent
      - Receiver must keep track of message ids that could be a duplicate (When can receiver know it's ok to forget?)
  - Destination controls window to indicate its willingness to receive messages
- Solutions:
  - Alternating bit protocol
  - Window based protocol (TCP)

Alternating Bit Protocol



- Send one message at a time
- Don't send next message until ack received
- Receiver keeps track of sequence # of last message received
- Simple
- Small overhead
- Poor performance:
  Bandwidth = packet_size/RTT

Window Based Protocol



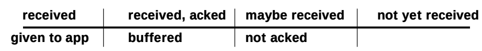
Peak Bandwidth:
Packets_in_window * packet_size/ RTT
or
packet_size/message_init_overhead

- Send up to N messages at a time without waiting for acks
- "Window" also reflects storage at receiver – sender shouldn't overrun receiver's buffer space
- Each message has sequence number. Receiver can discard state of messages outside the window
- If messages are received out of order:
  - Keep copy until sender fills in the missing pieces

Flow Control Details

Sender messages

Receiver messages

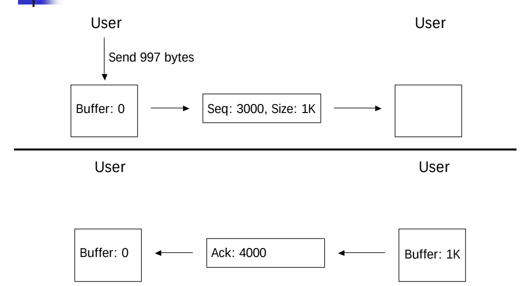


- Receiver acks: "got all messages up to #"
- Duplicate acks implies holes in received message sequence
  - Sender can perform "fast retransmit"

TCP Flow Control

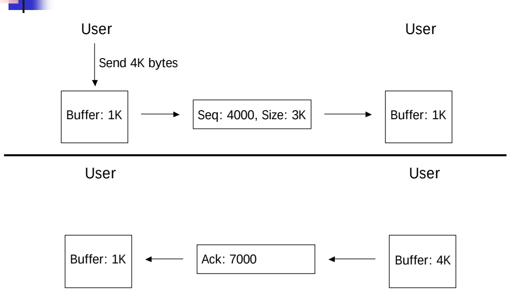
- Assume:
  - Receiver window size = 8K
  - TCP minimum threshold for sending = 1K
  - Initial sequence number = 3000

TCP Flow Control 2



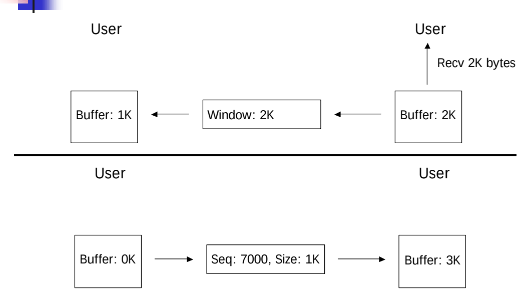
- Acknowledge message: "expecting byte # x"

TCP Flow Control 3



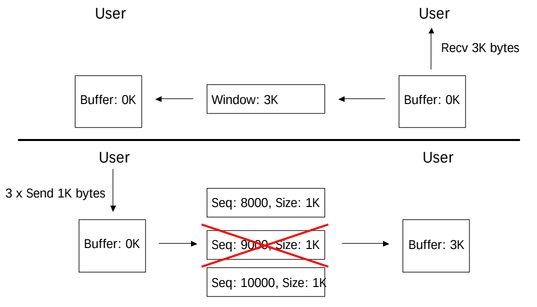
- Send only up to the receiver window

TCP Flow Control 4

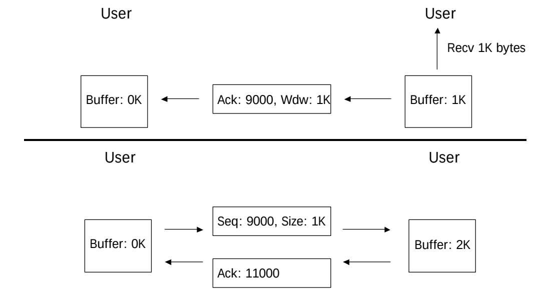


- Receiver sends a window update when user picks up data

TCP Flow Control 5

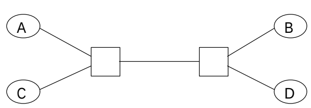


TCP Flow Control 6



Congestion Control

- Window size controls flow and congestion
  - Receiver advertised window is maximum amount of data that can be outstanding
  - Have a smaller window if there is congestion in the system
- Canonical congestion problem:
  - Flow between A-B uses up link capacity
  - Flow between C-D starts, resulting in congestion on the link

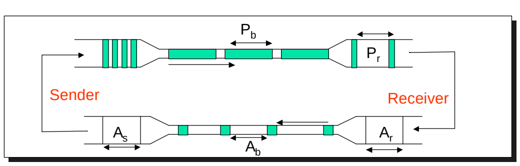


Issues

- Flows need to find a fair use of link resources
  - When a flow starts, it needs to find what is available reasonably fast and under different network capacities
- Flows need to distinguish packet losses from packet delays
  - Spurious detection of packet loss results in more traffic, more congestion, more delays, and so on
- Flows need to adapt to changing network conditions
  - Sometimes increase its utilization, sometimes lower its utilization

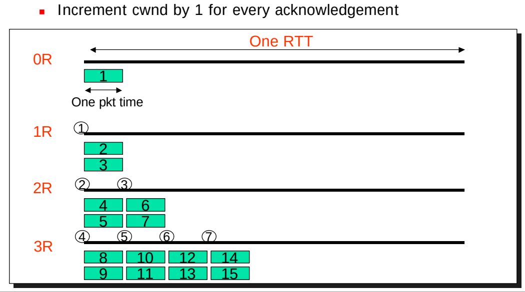
Finding Equilibrium from Startup

- Two features:
  - Self-clocking mechanism
  - "Slow start" mechanism – actually ramps up rather fast!
- Self-clocking:
  - Send a new packet only when a previous packet is acknowledged
  - Soon packets are sent at the rate they are received



Slow Start Mechanism

- Initially set cwnd to be 1
  - Maintain the invariant that cwnd < window given by receiver
  - Increment cwnd by 1 for every acknowledgement



Accurate Round Trip Time Estimates

- How long should timeout be?
  - Too long? Wastes time
  - Too short? Retransmits even though message is not lost
  - Maintain running estimate of "R"

    R = (1-α)*R + α * M

    where M is new measurement, α is decay constant
  - High α makes it unstable
  - Low α makes the system have too much history
- Also measure the error or variance in measurements
- Set timeout to be R + 4*variance

Congestion Avoidance Algorithm

- React to changing network conditions by modifying cwnd
- At loss: (multiplicative decrease)

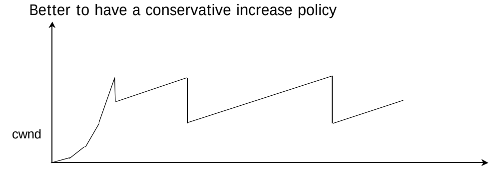
  cwnd = cwnd / 2

  Better to have a drastic decrease when losses occur
- After loss: (additive increase)

  cwnd += 1/cwnd

  Results in slow increase; probes for available bandwidth

  Better to have a conservative increase policy



Announcements

- Assignment 4 has been posted
- Wednesday reading: "Authentication in distributed systems"
- Friday reading: "Andrew File System"
  - Background reading: "Implementing Remote Procedure Calls" and "Sun's Network File Systems"

Congestion Control at Routers

- Router queues can fill up
- When they fill up?
  - What to drop?
  - When to drop?
- Random Early Detection (RED) algorithm: use randomization
- Router can be in one of three states:
  - Few packets in the queues: do not drop any packets (normal operating phase)
  - Lots of packets in the queues: drop for sure (congestion phase)
  - Intermediate number of packets: calculate probability for dropping based on queue length and number of packets since last drop (congestion avoidance phase)

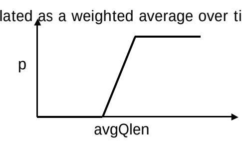
RED Drop Probability

- Voodoo constants: minqThresh, maxqThresh, maxp
- Step 1:

  p = maxp * (avgQlen – minqThresh)/(maxqThresh – minqThresh)

  p <= maxp

  avgQlen is calculated as a weighted average over time



- Step 2:

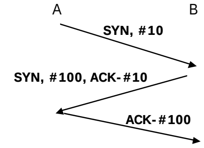
  Drop probability = p / (1 – count*p)

  count is number of packets since last drop

  Try to avoid cascading drops

Connections

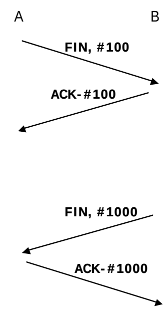
- Requires three-way handshakes
- Setup:
  - Open request packet (SYN, initial sequence number)
  - Acknowledgement (SYN, own sequence number, ack number)
  - Acknowledgement of the acknowledgement
- SYN occupies 1 byte of sequence space



Failure Scenarios

- Cannot reuse sequence number if there are some old live data
  - Keep track of previous recent connections
- What if machines go up and down?
  - Wait for a while when machine reboots
  - Let old packets die
- What if connection packets get lost?
  - Timeout and retransmit
  - But initially very conservative estimate of RTT

Connection Tear Down



- Keep connection state around for some more time
  - FIN occupies 1 byte in sequence space
  - Connection state is last byte received in sequence
- Typically kept for 2*RTT duration
- No clean solution as to when state can be forgotten
- A distributed consensus problem

General's Problem

- Two generals on separate mountains
- Can communicate only via messengers
  - Messengers can be captured
- Need to coordinate an attack
  - If they attack at the same time, they win
  - Else they will all die
- Devise a protocol to coordinate the two generals