Computer Networks

Network Layer

Topics

- Introduction ←
- Routing
- Congestion Control
- Internetworking
- The Internet

Introduction to Network Layer

- Getting packets from source to destination
  - may require many hops
  - data link layer from one end of wire to another
- Must know topology of subnet
- Avoid overloading routes
- Deal with different networks

Network Layer Services

- Services to Transport Layer
- Often carrier to customer
  - very well defined
- Goals
  - services independent of subnet technology
  - shield transport layer from topology
  - uniform number of network addesses, across LANs or WANS
- Lots of freedom, but two factions:
  - connection-oriented and connectionless

Connectionless

- Internet camp
  - 30 years of experience with real networks
  - subnet is unreliable, no matter how well designed
  - hosts should accept this and do error control and flow control
  - SEND_PACKET and RECV_PACKET
  - each packet full information
  - no ordering, flow control since will be redundant

Connection Oriented

- Telephone company camp
  - 100 years of international experience
  - set up connection between end hosts
  - negotiate about parameters, quality and cost
  - communicate in both directions
  - all packets delivered in sequence
    - some might still be lost
  - flow control to help slow senders

Connected vs Connectionless

- Really, where to put the complexity
  - transport layer (connectionless)
    - computers cheap
    - don't clutter network layer since relied upon for years
    - some applications don't want all those services
  - subnet (connected)
    - most users don't want complex protocols on their machines
      - embedded systems don't
    - real-time services much better on connected
- (Un) Connected, (Un) Reliable
  - 4 classes, but two most popular

Internal Organization

- *Virtual Circuit*
  - do not choose new route per packet
  - establish route and re-use
  - terminate route when terminate connection
- *Datagrams*
  - no advance routes
  - each packet routed independently
  - more work but more robust

Summary Comparison

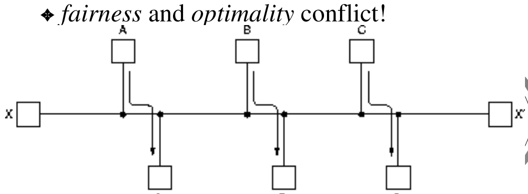
Issue	Datagram subnet	VC subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Subnet does not hold state information	Each VC requires subnet table space
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow this route
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Congestion control	Difficult	Easy if enough buffers can be allocated in advance for each VC

Examples of Services

Upper layer	Type of subnet: Datagram	Type of subnet: Virtual circuit
Connectionless	UDP over IP	UDP over IP over ATM
Connection-oriented	TCP over IP	ATM AAL1 over ATM

RoutingAlgorithms

- *correctness* and *simplicity* (obviously)
- *robustness*
  - parts can fail, but system should not
  - topology can change
- *stability*
- *fairness* and *optimality* conflict!



Review

- Describe each of the following in terms of network layers
  - Repeater
  - Hub
  - Bridge
  - Router

Optimality vs. Fairness

- What to optimize?
  - Minimize delay
  - Maximize network throughput
  - But basic queuing theory says if system near capacity then long delays!
- Compromise: minimize hops
  - Improves delay
  - Reduces bandwidth, so usually increases throughput

Two Classes of Routing Algorithms

- *Nonadaptive algorithms*
  - decisions not based on measurements
  - routes computed offline in advance
  - also called *Static Routing*
- *Adaptive algorithms*
  - change routes based on topology and traffic
  - info: locally, adjacent routers, all routers
  - freq: every ΔT seconds, load change, topology change
- Metric?
  - distance, number of hops, transit time

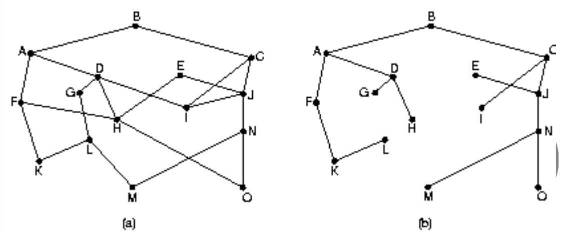
Optimality Principal

"If *J* is on optimal path from *I* to *K*, then optimal path from *J* to *K* is also on that path"

- Explanation by contradiction:
  - Call *I* to *J* r_1 and the rest r_2
  - Assume *J* to *K* has a route better than r_2, say r_3
  - Then r_1r_3 is shorter than r_1r_2
    - contradiction!
- Useful when analyzing specific algorithms

Sink Tree

- Set of optimal nodes to a given destination
- Not necessarily unique
- Routing algorithms want sink trees



Sink Trees

- No loops
  - each packet delivered in finite time
  - well, routers go up and down and have different notions of sink trees
- How is sink tree information collected?
  - we'll talk about this later
- Next up: *static routing algorithms*
- On deck: *adaptive algorithms*

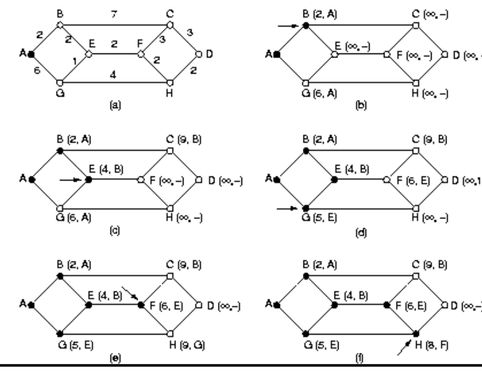
Static Routing - Start Simple

- *Shortest path routing*
- How do we measure shortest?
- Number of hops
- Geographic distance
- Mean queuing and transmission delay
- Combination of above

Computing the Shortest Path

- Dijkstra’s Algorithm (1959)
- Label each node with distance from source
  - if unknown, then ∞
- As algorithm proceeds, labels change
  - tentative at first
  - permanent when “added” to tree

Djikstra’s Algorithm: A to D



Flooding

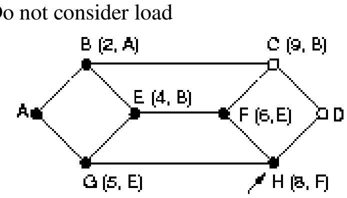
- Send every incoming packet on every outgoing link
  - problems?
- Vast numbers of duplicate packets
  - infinite, actually, unless we stop. How?
- Hop count: decrease each hop
- Sequence number: don’t flood twice
- *Selective flooding*: send only in about the right direction

Uses of Flooding

- Military applications
  - redundancy is nice
  - routers can be blown to bits
- Distributed databases
  - multiple sources
  - update all at once
- Baseline
  - flooding always chooses shortest path
  - compare other algorithm to flooding

Flow Based Routing

- Above algorithms only consider topology
  - Do not consider load



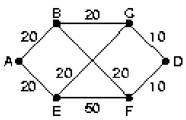
- Ex: if huge traffic from *A* to *B* then better path would be *AGEFC*

Flow-Based Routing

- For a given line, if capacity and avg flow are known, can compute mean packet delay
- Do for all lines
- Find routing algorithm that has minimum average delay for the entire subnet
- Need:
  - topology
  - traffic matrix, F_{ij}
  - capacity matrix, C_{ij}

Flow-Based Routing Example

Capacities



Ex: B to *D*
- 3 packets/sec
- route: *BFD*
3 to BF, 3 to FD

Destination (columns), Source (rows):

Source	A	B	C	D	E	F
A		9 AB	4 ABC	1 ABFD	7 AE	4 AEF
B	9 BA		8 BC	3 BFD	2 BFE	4 BF
C	4 CBA	8 CB		3 CD	3 CE	2 CEF
D	1 DFBA	3 DFB	3 DC		3 DCE	4 DF
E	7 EA	2 EFB	3 EC	3 ECD		5 EF
F	4 FEA	4 FB	2 FEC	4 FD	5 FE	

Flow-Based Routing Example

i	Line	λ_i (pkts/sec)	C_i (kbps)	μC_i (pkts/sec)
1	AB	14	20	25
2	BC	12	20	25
3	CD	6	10	12.5
4	AE	11	20	25
5	EF	13	50	62.5
6	FD	8	10	12.5
7	BF	10	20	25
8	EC	8	20	25

Delay on Each Line

- $T = \dfrac{1}{\mu C - \lambda}$
- $1/\mu$ is mean packet size
- C is capacity in bps
- λ is mean flow in packets/sec

Line	T_i (msec)
AB	91
BC	77
CD	154
AE	71
EF	20
FD	222
BF	67
EC	59

Mean Delay for Entire Subnet

- Weights are fraction of traffic on line
- Weighted average is mean delay
  – (86) in this example
- Can repeat for other flows
- Offline, so can be slow

Line	λ_i (pkts/sec)	T_i (msec)	Weight
AB	14	91	0.171
BC	12	77	0.146
CD	6	154	0.073
AE	11	71	0.134
EF	13	20	0.159
FD	8	222	0.098
BF	10	67	0.122
EC	8	59	0.098

Topics

- Introduction ✔
- Routing
  – static ✔
  – adaptive ←
- Congestion Control
- Internetworking
- The Internet

Modern Routing

- Most of today's computer networks use dynamic routing
- Distance vector routing
  – Original Internet routing algorithm
- Link state routing
  – Modern Internet routing algorithm

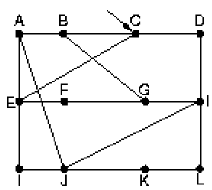
Distance Vector Routing

- Each router has table
  - preferred outgoing line
  - estimate of "distance" to get there
- Assume knows "distance" to each neighbor
  - if hops, just 1 hop
  - if queue length, measure the queues
  - if delay, can send PING packet
- Exchange tables with neighbors periodically

Distance Vector Routing Computation

- Just got *Routing Table* from *X*
  - X_i is estimate of time from *X* to *i*
- Delay to *X* is *m* msec
- Know distance to X (say, from ECHO's)
  - Can reach router *i* via *X* in $X_i + m$ msec
- Do for all neighbors
- Closest to *i* as "preferred outgoing line"
- Can then make new routing table

Distance Vector Example



To	A	I	H	K	New estimated delay from J	Line
A	0	24	20	21	8	A
B	12	36	31	28	20	A
C	25	18	19	36	28	I
D	40	27	8	24	20	H
E	14	7	30	22	17	I
F	23	20	19	40	30	I
G	18	31	6	31	18	H
H	17	20	0	19	12	H
I	21	0	14	22	10	I
J	9	11	7	10	0	–
K	24	22	22	0	6	K
L	29	33	9	9	15	K
	JA delay is 8	JI delay is 10	JH delay is 12	JK delay is 6	New routing table for J	

Vectors received from

Good News Travels Fast

A	B	C	D	E	
	∞	∞	∞	∞	Initially
	1	∞	∞	∞	After 1 exchange
	1	2	∞	∞	After 2 exchanges
	1	2	3	∞	After 3 exchanges
	1	2	3	4	After 4 exchanges

- A is initially down
- Path to A updated every exchange
- Stable in 4 exchanges

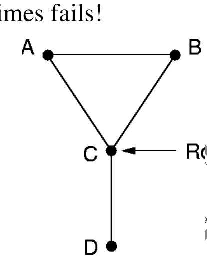
Bad News Travels Slowly

A	B	C	D	E	
	1	2	3	4	Initially
	3	2	3	4	After 1 exchange
	3	4	3	4	After 2 exchanges
	5	4	5	4	After 3 exchanges
	5	6	5	6	After 4 exchanges
	7	6	7	6	After 5 exchanges
	7	8	7	8	After 6 exchanges
		⋮			
	∞	∞	∞	∞	

- Sloooowly converges to ∞ (*count to infinity*)
- Better to set infinity to max + 1

Split Horizon

- Report ∞ to router along path
  - ex: C says ∞ to reach A when talking to B
- Widely used … but sometimes fails!
- If D goes down
  - C can say ∞ to D quickly
- A and B have route through other
  - A and B count to ∞ as slowly as before!
- Other Ad Hoc also fail

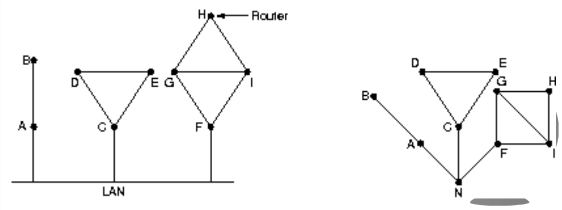


Link State Routing

- Used on Internet since 1979
- Steps
  - Discover neighbors
  - Measure delay to each
  - Construct a packet telling what learned
  - Send to all other routers
  - Compute shortest path
- Basically
  - Experimentally measure distance
  - Use Dijkstra's shortest path

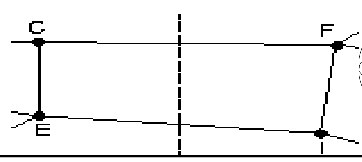
Learning Neighbors

- Upon boot, send *HELLO* packet along point-to-point line
  - names must be unique
- Routers attached to LAN?



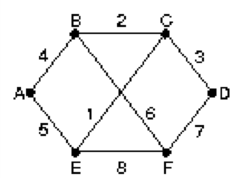
Measuring Line "Cost"

- Send ECHO packet, other router returns
  - delay
- Factor in load (queue length)?
  - Yes, if other distance equal, will improve perf
  - No, oscillating routing tables
  - Ex: Back and forth between C-F and E-I



Building Link State Packets

- Identity of sender, age, list of neighbors



Link State

A		B		C		D	
Seq.		Seq.		Seq.		Seq.	
Age		Age		Age		Age	
B	4	A	4	B	2	C	3
E	5	C	2	D	3	F	7
		F	6	E	1		

- *When* to send them?

Distributing Link State Packets

- Tricky if topology changes as packets travel
  - routes will change "mid-air" based on new topology
- Basically, use *flooding* with checks
  - increment sequence each time new packet sent
- Forward all new packets
- Discard all duplicates
- If sequence number lower than max for sending station
  - then packet is obsolete and discard

Distribution Problems

- Sequence numbers wrap around
  - use 32 bits and will take 137 years
- Router crashes … start sequence number at 0?
  - next packet it sends will be ignored
- Corrupted packet (65540)
  - packets 5 - 65540 will be ignored
- Use *age* field
  - decrement every second
  - if 0, then discard info for that router
- Hold for a bit before processing

Keeping Track of Packets

Station B

Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F
A	21	60	0	1	1	1	0	0
F	21	60	1	1	0	0	0	1
E	21	59	0	1	0	1	0	1
C	20	60	1	0	1	0	1	0
D	21	59	1	0	0	0	1	1

- A arrived
  - ack A
  - forward C and F
- F arrived
  - ack F
  - forward A and C

Keeping Track of Packets

Source	Seq.	Age	Send flags A	Send flags C	Send flags F	ACK flags A	ACK flags C	ACK flags F
A	21	60	0	1	1	1	0	0
F	21	60	1	1	0	0	0	1
E	21	59	0	1	0	1	0	1
C	20	60	1	0	1	0	1	0
D	21	59	1	0	0	0	1	1

- E arrived via EAB and via EFB
  - send only to C
- If C arrives via F before forwarded, updated bits and don't send to F

Computing New Routes

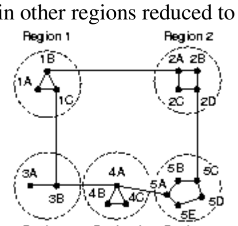
- Router has all link state packets
  - build subnet graph
- *N* routers degree *K*, O(*KN*) space
- Problems
  - router lies: forgets link, claims low distance
  - router fails to forward, or corrupts packets
  - router runs out of memory, calculates wrong
  - with large subnets, becomes probable
- Limit damage from above when happens

Link State Routing Today

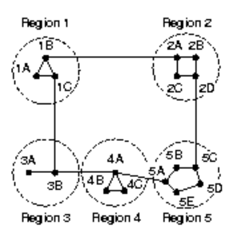
- Open Shortest Path First (OSPF) (5.5.5)
  - used in Internet today
- Intermediate System Intermediate Syst (IS-IS)
  - used in Internet backbones
  - variant used for IPX in Novell networks
  - carry multiple network layer protocols

Hierarchical Routing

- Global picture difficult for large networks
- Divide into regions
  - Router knows detail of its region
  - Routers in other regions reduced to a point



Reduce Routing Table



Full table for 1A

est.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

Hierarchical table for 1A

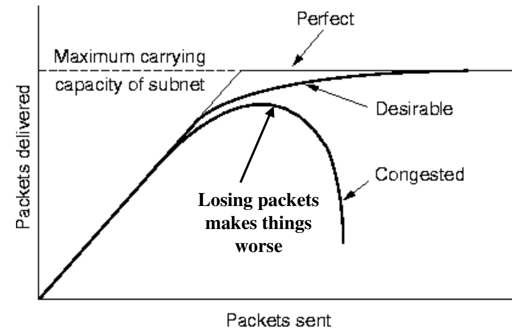
Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

- Cost is efficiency
- Consider 1A to 5C
  - via 3 better for most of 5

Topics

- The Network Layer
  - Introduction ✔
  - Routing ✔
  - Congestion Control ←
  - Internetworking ✗
  - The Internet ✗
- The Transport Layer

Congestion



Causes of Congestion

- Queue build up until full
  - Many input lines to one output line
  - Slow processors
  - Low-bandwidth lines
    - system components mismatch (*bottleneck*)
  - Insufficient memory to buffer
- If condition continues, infinite memory makes worse!
  - timeouts cause even more transmission
  - congestion feeds upon itself until collapse

Flow Control vs. Congestion Control

- Congestion control (network layer)
  - make sure subnet can carry offered traffic
  - global issues, including hosts and routers
- Flow control (data link layer)
  - point-to-point between sender and receiver
  - fast sender does not overpower receiver
  - involves direct feedback to sender by receiver
- Ex: Super-computer to PC w/1Gbps line
- Ex: 1000 computers w/1 Mbps lines transferring files at 1kbps to other half

Principles of Congestion Control

- Control theory: *open loop* and *closed loop*
- Open loop: ahead of time
  - solve problem by making sure doesn't happen
  - when to accept new traffic
  - deciding to discard packets and which ones
  - scheduling decisions within the network
- Closed loop: feedback
  - detect congestion … *how*?
  - pass information to system that can adjust

Feedback Loop (cont)

- Metrics to detect congestion:
  - percentage of dropped packets
  - average queue length
  - number of timed out packets
  - average packet delay (and std dev of delay)
- Transfer info:
  - router to send packet to traffic source(s)
    - but this *increases* the load!
  - set bit in acks going back (ECN)
- Send probe packets out to ask other routers
  - ala traffic copters to help route cars

Congestion Control Algorithms

- Taxonomy (Yang and Reddy 1995)
- Open or Closed (as above)
- Source or Destination
- Explicit or Implicit feedback (for closed)
  - explicit: send congestion info back to source
  - implicit: source deduced congestion (by looking at round-trip time for acks, say)

Congestion Fix

- Load is greater than resources
  - Increase resources or decrease load
- Increase resources
  - adding extra leased bandwidth
  - boost satellite power
  - split traffic over multiple routes
  - use backup, fault-tolerant routers
  - but difficult under many systems
- Decrease load
  - at *data link*, *network* or *transport* layer

Preventing Congestion

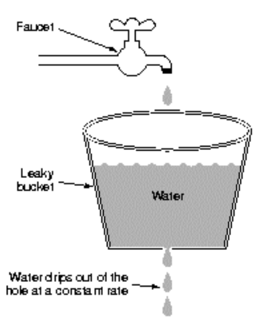
- Traffic is often bursty
- If steady rate, easier to avoid congestion
- Open loop method to help manage congestion by forcing packets at more predicable rate
  - *Traffic Shaping*

Traffic Shaping

- Limit rate data is sent
- User and subnet agree upon certain pattern (shape) of traffic
  - especially important for real-time traffic
  - easier on virtual circuit, but possible on datagram
- Monitoring agreement is *traffic policing*

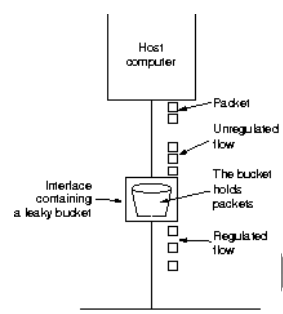
The Leaky Bucket

- No matter how fast water enters bucket, drips out at same rate
  - ρ
- If bucket is empty,
  - then ρ is 0
- If bucket is full, then spills over sides
  - i.e. - lost



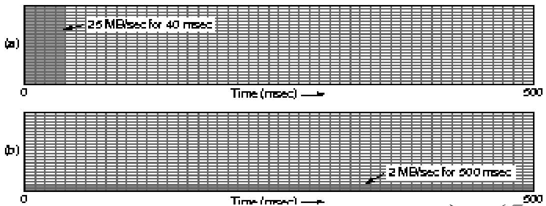
The Leaky Bucket Algorithm

- Each router has finite internal queue
  - excess packets discarded
- One packet per tick sent
  - or fixed bytes, if different sized packets



Leaky Example

- 200 Mbps network
- 2 Mbps for long intervals
- 25 MB/sec for 40 sec

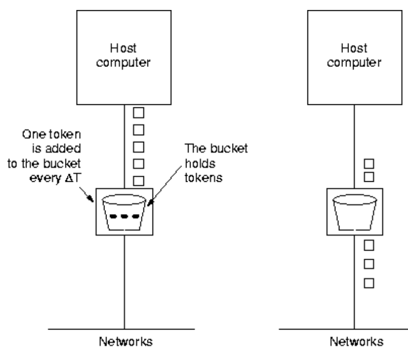


(a) is w/out bucket, (b) is with bucket

Leaky Enhancements

- Leaky enforces rigid output rate
  - instead, allow some speedup of output
  - *token bucket algorithm*
- Token generated every ΔT second
  - to send packet, station must capture and destroy
- Example: station wants to send 5 packets and there are 3 tokens

Token Bucket Example

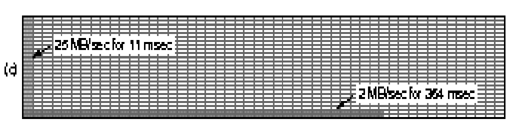


Traffic Shaping with Token Bucket

- Leaky bucket does not allow hosts to "save up" for sending later
- Token bucket host can capture up to some max *n* tokens
- Since hosts must stop transmitting when no tokens, then can avoid lost data
  - leaky bucket will just drop data, resulting in timeouts and retransmissions (or, just lost data)

Token Bucket Example

- 250 KB token bucket
- Token rate allows 2MB/sec
- 25 MB/sec arrives for 40 sec
  - can drain at this rate for about 10 seconds
  - then must cut back to 2 MB/sec



Closed-Loop Congestion Control

- Router monitors utilization (queue, cpu …)
  - ex: each line num 0.0 to 10.0
  - how to sample?
    - f is instantaneous sample (0 or 1)
    - $u_{new} = a u_{old} (1+a) f$
    - a determines how fast "forget"
- *u* above threshold then enters a "*warning*" state
  - router sends chock packet to source
  - original packet is tagged so will not generate more choke packets

Choke Packets (cont)

- When source receives choke packet, reduces traffic by *X* percent
  - reduce window size or bucket parameters
  - decrease 0.5, 0.25, … increase slowly, too
- Ignore new choke packets from destination for some time interval
  - *why*?
- Increase flow at some time
- Variations: degrees of warning

Foul Play

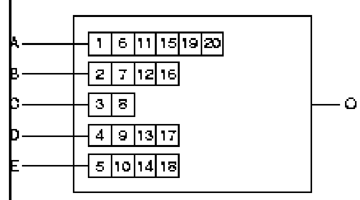
- Consider A, B and C send through Router
- Router detects congestion, sends choke packet to each
- A cuts back packet rate but B and C continue blasting away
  - requires *voluntary* cutback
- Transport protocols:
  - TCP: built in flow-control helps congestion control
  - UDP: mis-behaved flows
- Solution: *fair queuing*

Fair Queuing

- Multiple queues for each output line
  - one per source
- Do round-robin among queues
  - with *n* hosts competing, get 1/*n* of bandwidth
- Sending more packets will not help
- Trouble?
  - More bandwidth to hosts with large packets
- Solution: byte-by-byte round robin

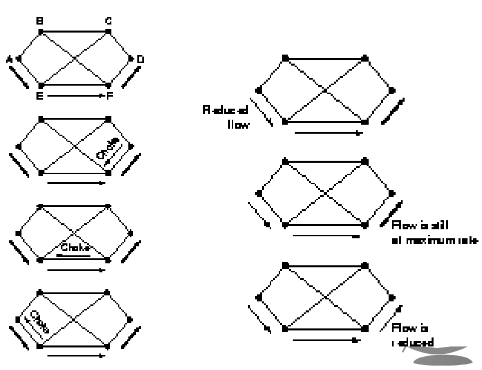
Fair Queuing, Take 2

- Find tick at which each packet is done
- Sort by those ticks



Packet	Finishing time
C	8
B	16
D	17
E	18
A	20

Choke Packets over Distance



Hop-by-Hop Choke Packets