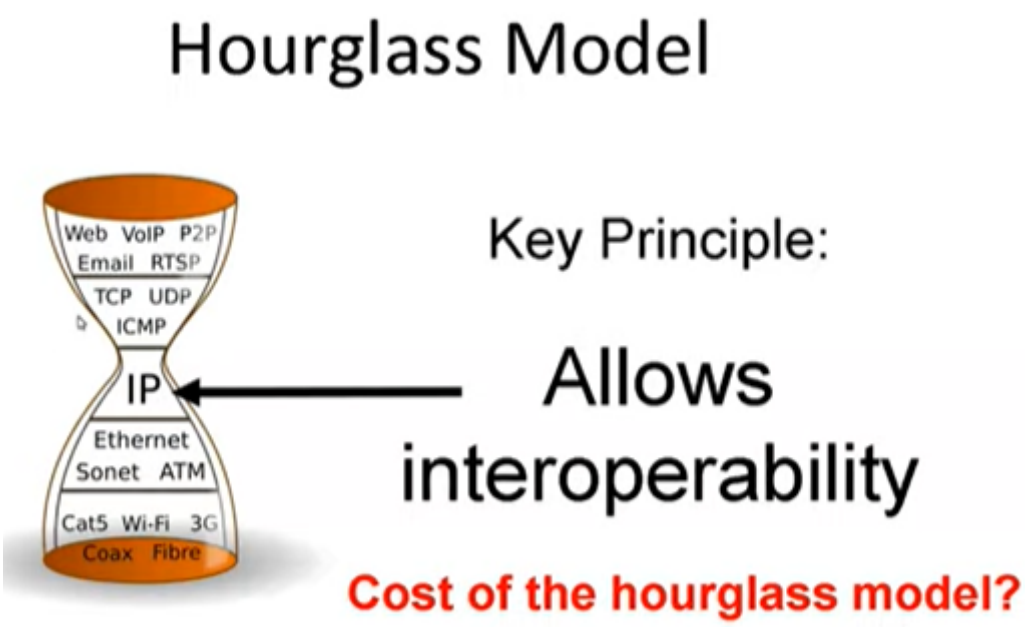
CS5229 Notes

Review of the Term

1. Protocol layer abstraction
    1. application - transport - network physical
2. core principles & design decisions
    1. provide useful interconnection between heterogeneous machines
    2. support independent administrative boundaries of control
3. hourglass model
    1. changing IP layer becomes involved
    2. layering has drawbacks (duplicate functionality, big headers, losing observation of low level details, etc.)
    3. ref: https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F21/index_lectures.html
4. 

4. principle 1: Best effort:
    1. can try but no guarantee
    2. try hard -> cost goes up
    3. best effort -> make design simple
    4. TCP & IP: initially one thing, but sometimes reliability is not required (packets go outdated when retransmitted)
5. principle 2: End-to-end argument
6. Fate sharing (TODO) 21:20

Congestion Control

1. slow start: cwnd += MSS (TODO)
2. congestion avoidance (additive increase) cwnd += MSS/cwnd(TODO)
3. packet loss (multiplicative decrease) cwnd /= 2
4. AIMD -> converge to fairness & efficiency

5. synchronization of TCP flows
    1. flows experience losses together -> half cwnd at the same time
    2. MAY lead to poor utils (not necessarily)
6. explicit congestion notification ECN
    1. notified by switches
    2. latter bit echoed back
        1. Nonce is used in ECT state repr (01 and 10) to detect replay attacks
7. Buffer sizing view:
    1. 1994 Buffer sizing rule of thumb(1994): $B = RTT \times C$ (actually $B >= RTT \times C$)
        1. C is line rate
    2. 2004 small buffers $C \times RTT/\sqrt{N}$
    3. 2006 tiny buffers ($\log W$, 20~50p) with paced traffic
8. CoDel
    1. track local minimum queue delay (time)
        1. time between packet entering queue and leaving queue
    2. start dropping when delay > target(threshold)
    3. next drop time ~ 1/sqrt(num of packets dropped)
        1. num of packets dropped goes quadratically with time since (sqrt(x))' ~ 1/sqrt(x)
    4. until delay < target
    5. looks like a $target \times C$ sized buffer
9. BBR - rate based CC
    1. core idea: trying to operate at the best turning point (pkts in flight, rate) = (BDP, BtlBw)
    2. startup: rate += 2.89 times every RTT (TODO)
        1. until: throughput increase < 25% for 3RTTs
        2. REF https://github.com/google/bbr/blob/master/Documentation/bbr-faq.md#where-does-the-value-of-the-bbr-startup-pacing_gain-come-from
        3. Drain: rate = 4 pkts
            1. until: pkts in flight == BDP (BDP is measured)
    3. PROBE_BW
        1. period: 1.25 probing -> 0.75 buf draining -> 1.0 * 6
    4. PROBE_RTT
        1. empty the buffer to measure RTT(TODO)
    5. cwnd = 2*BDP
10. BBR vs CUBIC (TODO)
    1. BBR wins in shallow buffer
    2. CUBIC wins in deep buffer
    3. but the more BBR flows are, the less BBR wins
11. XCP
    1. efficiency controller
        1. $\phi = \alpha dS - \beta Q$
    2. fairness controller
        1. AIMD principle
12. Max-min fairness

1. idea of max-min fair: maximize minimum bandwidth
2. progressive filling
    1. simultaneously increase flow size for all flows at the same rate until some links are filled up
    2. stop increasing the choked flows and continue increasing other flows; repeat

Data Center Networks

1. Project: jellyfish & fat-tree (part 2)
    1. why RRG works better?
2. characteristics
    1. high throughput, high burst tolerance
        1. deep buffers
        2. reduced RTO
    2. low latency
        1. shallow buffers
        2. AQM-RED
3. buffer sizing
    1. measurements: typically 1-2(up to 4) big flows -> low variance in sending rate -> small buffers (buffer sizing 2004)
4. DCTCP
    1. congestion signal: ECN marking
        1. react in proportion to extent of congestion (low variance in rates lowers queueing requirements)
        2. mark based on instantaneous queue length (fast feedback)
5. TIMELY
    1. use delay as sign of congestion?
        1. compete with loss-based flows (TODO why not a problem?)
        2. get an accurate RTT estimate (hard, for RTTs are short)
    2. solve measuring with modern NICs
    3. TIMELY is suitable for datacenters but not for Internet (TODO)
    4. gradient-based inc/dec
        1. measure normalized gradient and inc/dec rate to compensate
6. Ad Hoc
    1. Proactive
        1. DSDV
    2. Reactive
        1. DSR
        2. AODV
        3. TORA
    3. geographic routing
        1. greedy forwarding (works in most situation)
        2. recover from dead end
        3. back to greedy forwarding
        4. GG, RNG
        5. CLDP(TODO)

        6. hull tree
7. peer to peer system
    1. Chord
    2. fingers stepping by power of 2
    3. reactive cache management
8. BitTorrent
    1. choking algorithm
        1. unchoke top 4
        2. optimistic unchoke 1 random
    2. randomly select finite peers
    3. rarest chunks first
    4. parallel connections in end-phase to get last chunks faster
9. MapReduce
    1. split, map, reduce
    2. ideally M+R>>\# of workers
10. Dynamo
    1. key-value store
    2. consistent hashing
        1. node in charge of indices between its predecessor and itself
        2. virtual nodes(TODO)
    3. vector clocks
    4. quorum + sloppiness
        1. R+W>N
    5. Merkle trees
        1. parent node = hash of sum of hash of children nodes
    6. gossip(TODO)
11. distributed system co-design
    1. NOPaxos - indirection with sequencer, sorted but unreliable delivery
    2. pegasus: selective replication with coherence directory(TODO)

congestion control

why hard?

- fairness
- low visibility of network
- churn
- intrinsic delay

window based cc

what is a congestion collapse?

how to set the rate?

- cwnd
- directly

why cwnd bigger????(because some was stored in buffer? or because there are acks?)

- congestion window: unacknowledged packets sent

- packets in flight: packets in the pipe

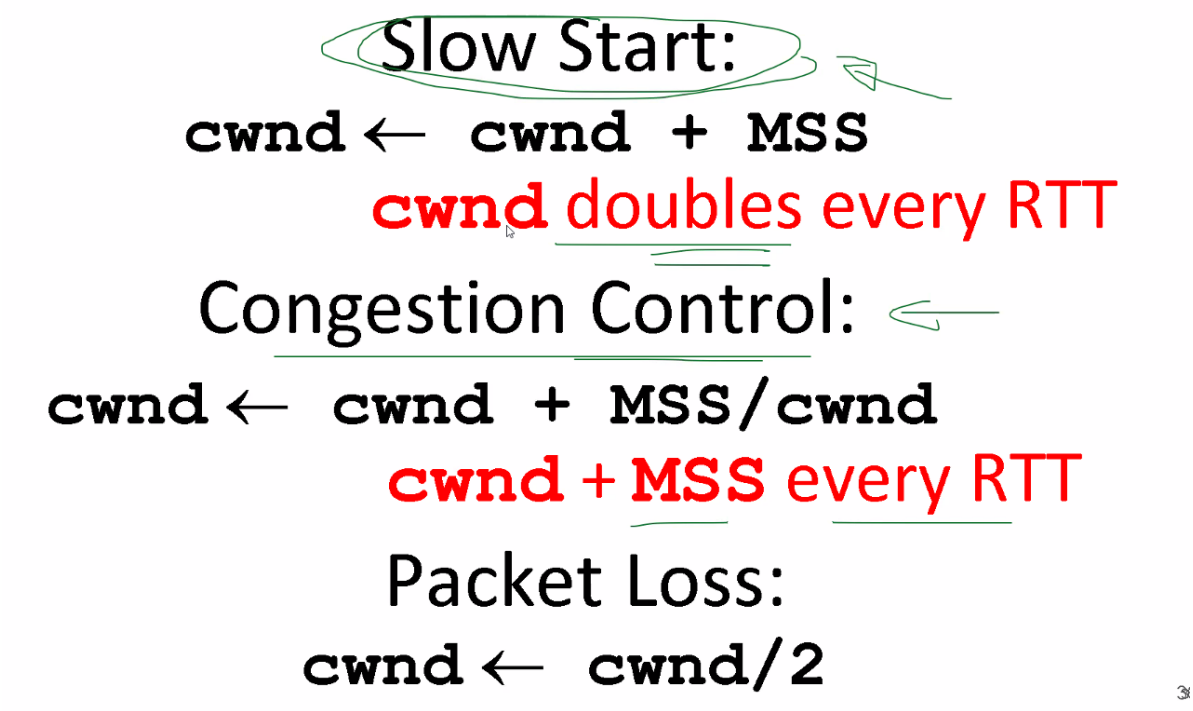
equilibrium (not applicable at start)

how to measure capability of network?

try out until there's package loss

how to know there's package loss?

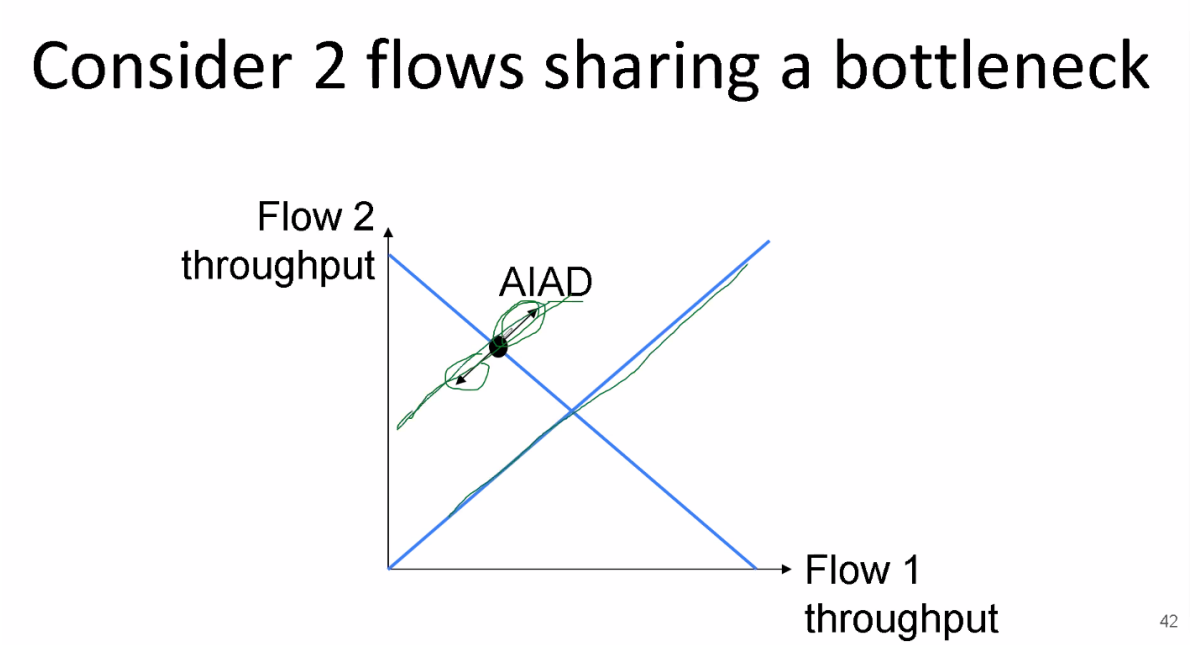
binary backoff



当前网络拥塞控制减半后其他网络加入是否会导致其带宽被其他网络占用?

同算法会收敛，不同算法之间挤占

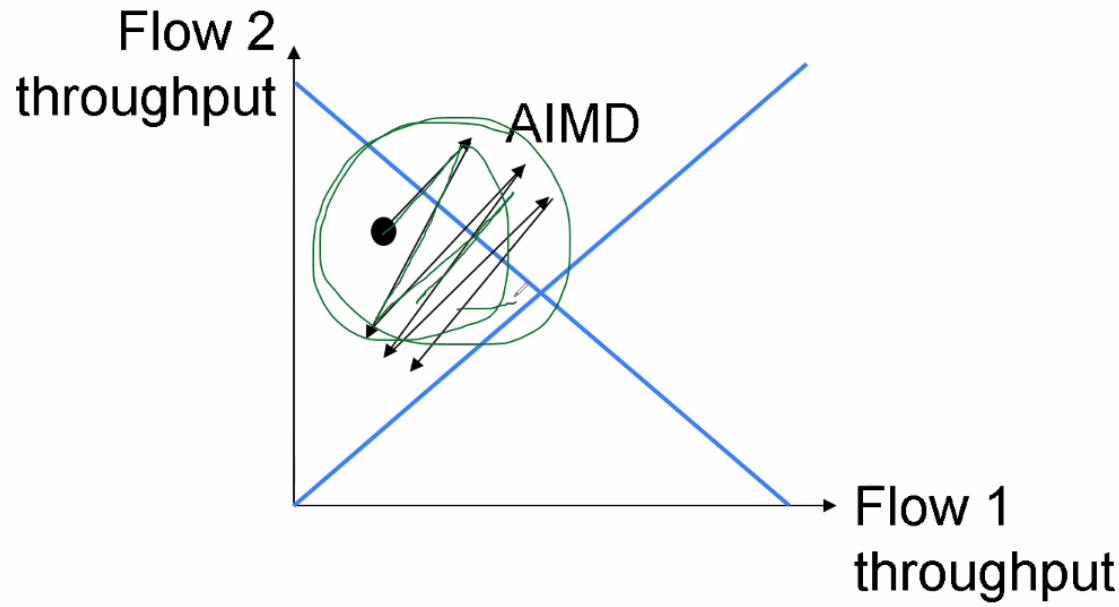
fairness:



AIAD (additive increase additive decrease?)

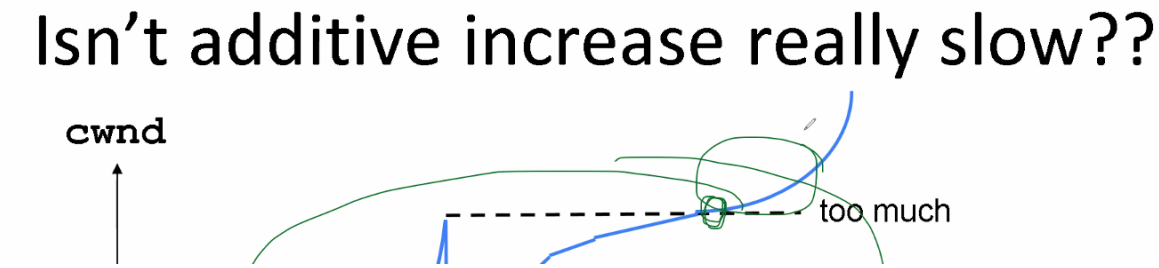
MIMD (multiplicative)

AIMD（也就是前述congestion control+packet loss的做法，每次+1，发生loss时减半，这段时间概括为congestion avoidance）



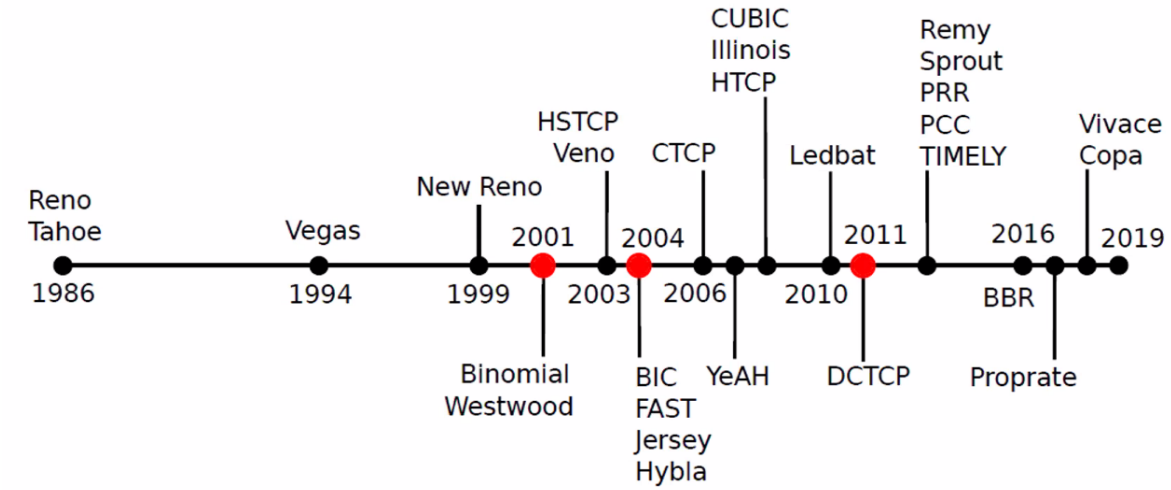
back to signs of loss

- duplicate acks
  - cwnd /= 2
- timeout
  - back to slow start

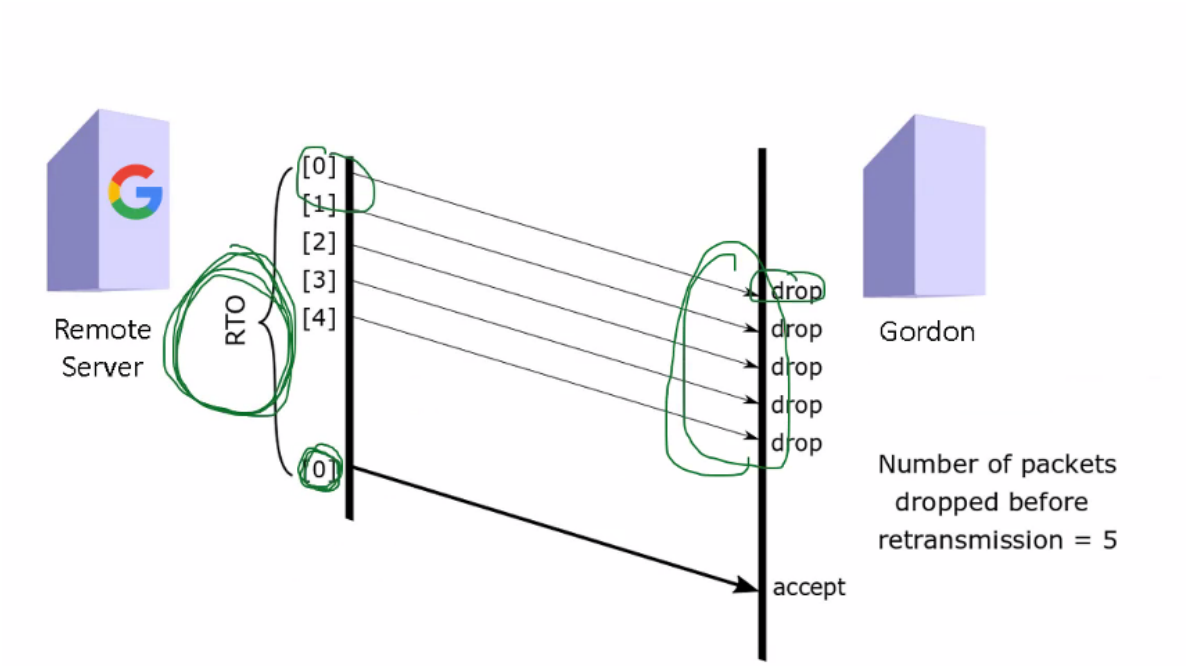


why you can do a fast increase above the upper line? - someone may have released the network resources that are occupied previously

some issues with CC



Gordon: 鉴定一个服务器使用的TCP类型



cwnd = 5

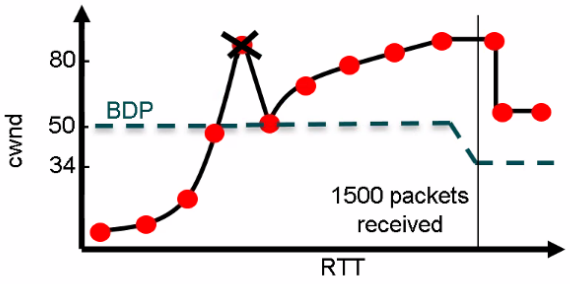
最终用于测试和获取网络特征的策略：

Final sequence of network stimuli

"Network profile":

1. Packet drop at the first `cwnd` that exceeds 80 packets
2. Bandwidth change after receiving 1500 packets
3. Emulating an RTT of 100 ms

Optimal page size for this network profile: **165 kb**



网络在上述情形下的响应为算法特征

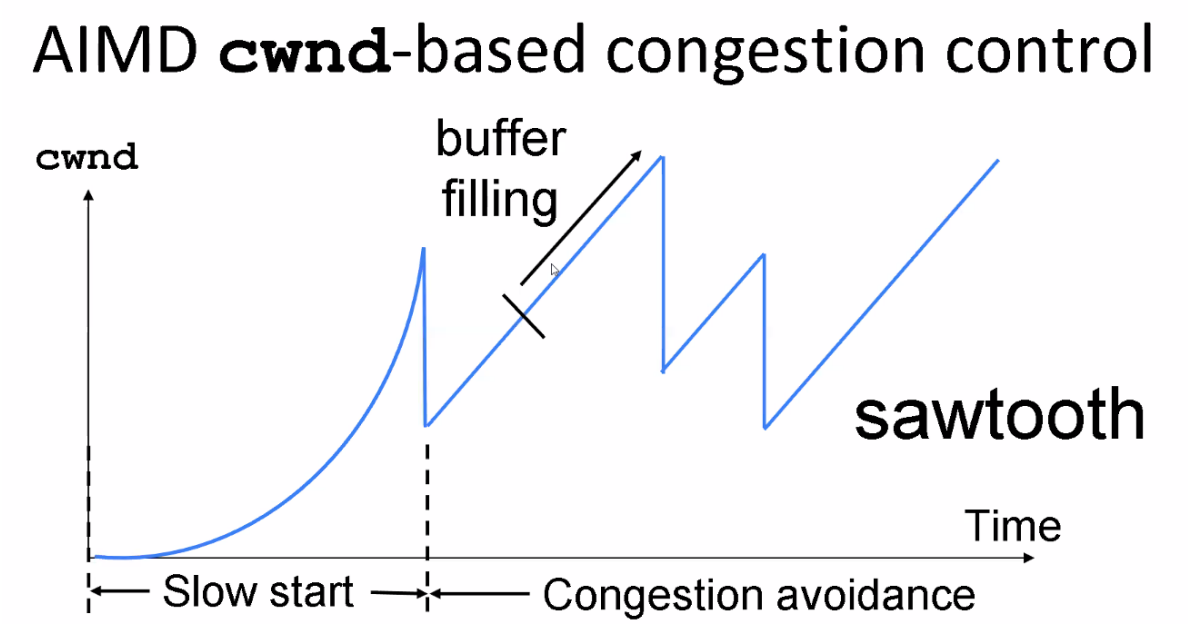
Summary

1. ACK-clocking
2. Congestion window
3. Bandwidth-delay product
4. Slow start
5. AIMD congestion avoidance
6. Binary Backoff
7. TCP Friendliness

8. Stability of Future Internet Not Guaranteed

7和8没有细讲

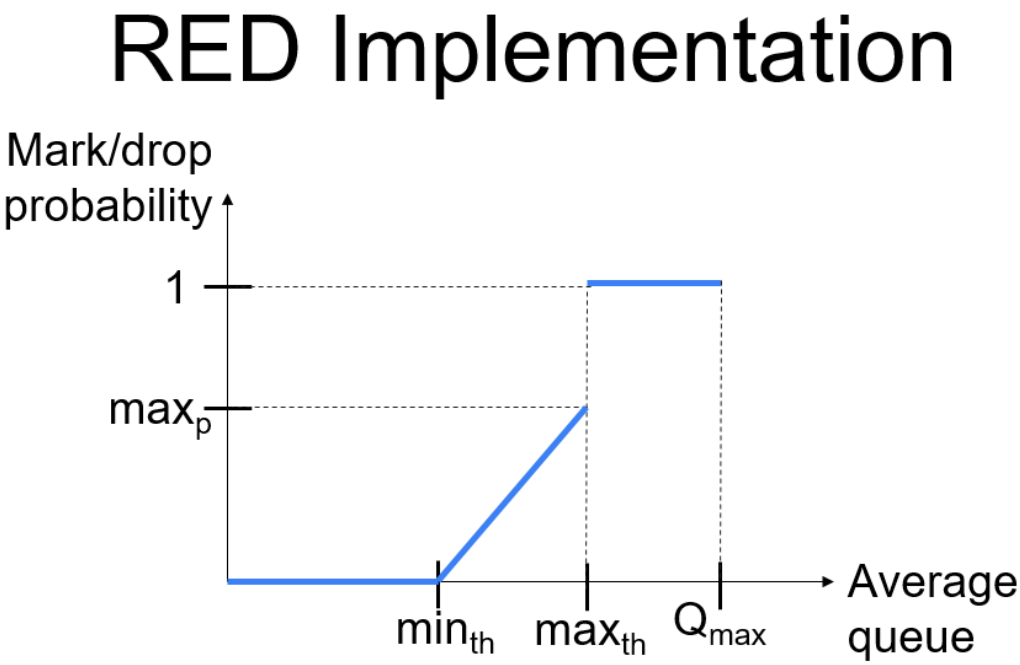
congestion avoidance



ECN

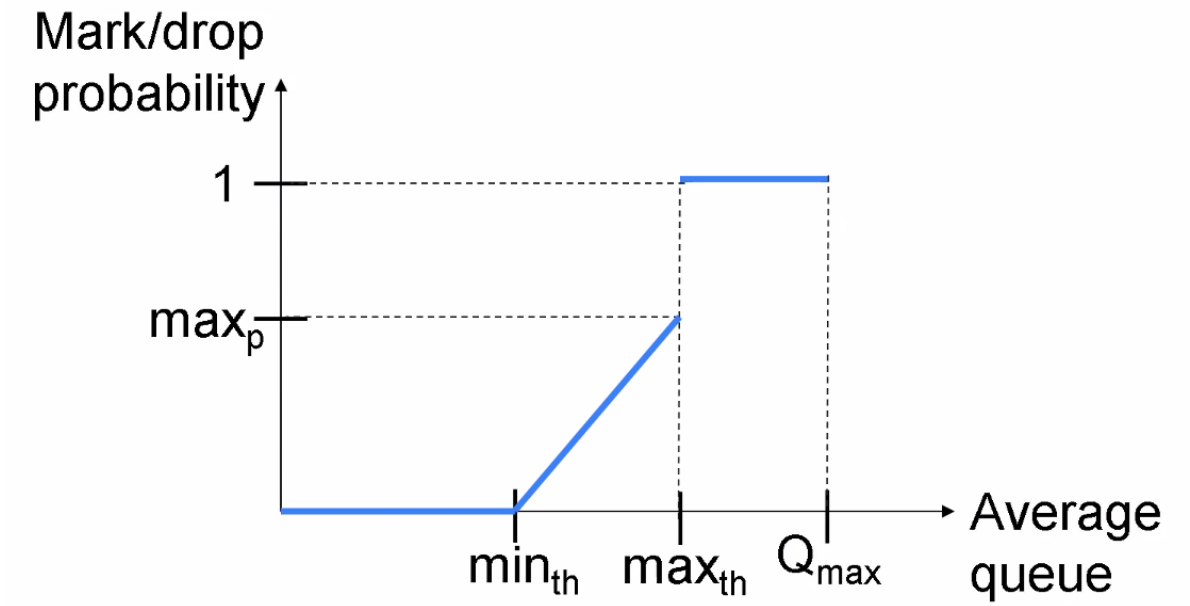
RED

1. random early detection
    1. monitor average queue length
    2. early report congestion

3. 

4.

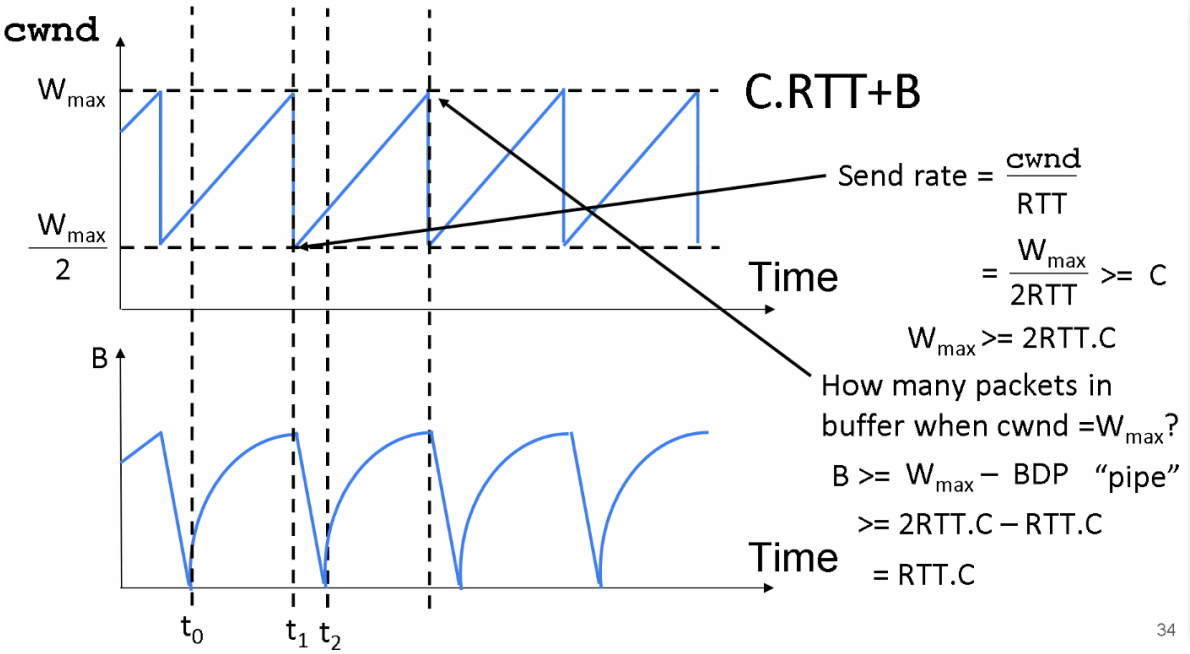
RED Implementation

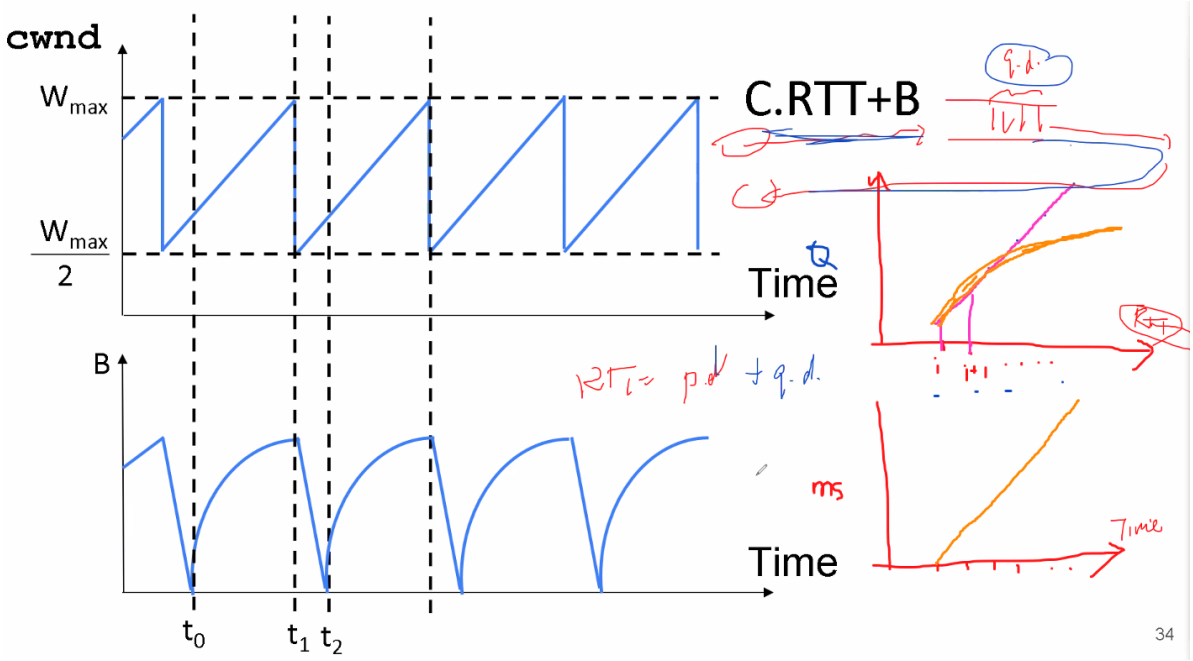


tracking the average queue length with EWMA(exponentially weighted moving average)

buffer sizing

rule of thumb (1994): buffer size = RTT*bottleneck





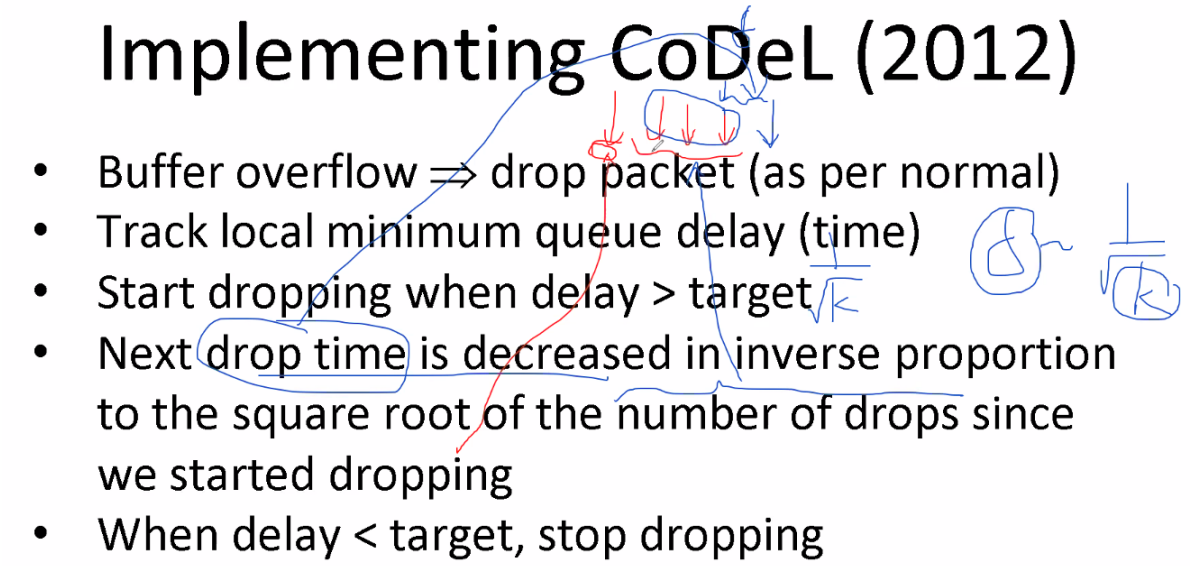
new proposal (2006): B = RTT * bottleneck / sqrt(n)

more flows, smaller buffers?

tiny buffers: log(W) (W is average of cwnd)

CoDel: drop when delay > target

the more you drop, the quicker you drop



fairness

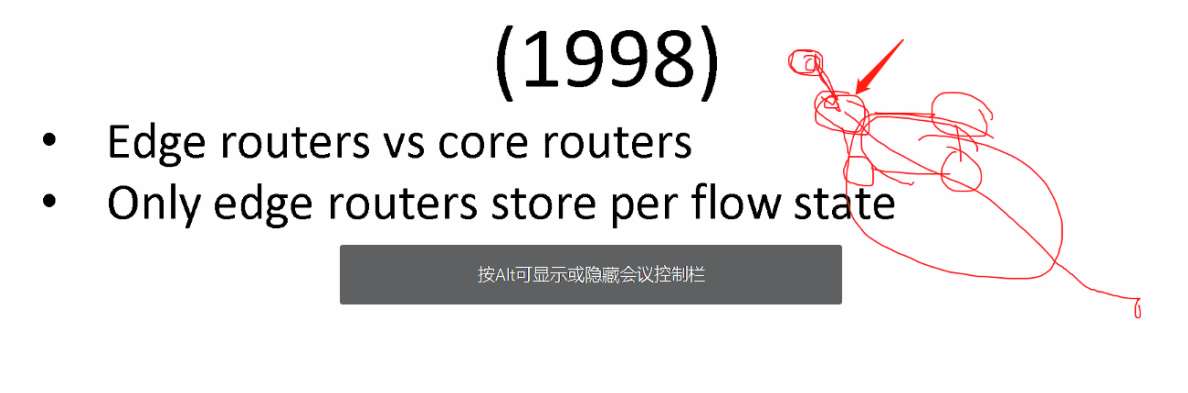
TCP friendliness

fairness? - maximize minimum

hard to achieve:

- have to maintain per-flow state at line rates
- balance between long flows & short
- bursty flow
- hard to predict future of flow
- routing changes

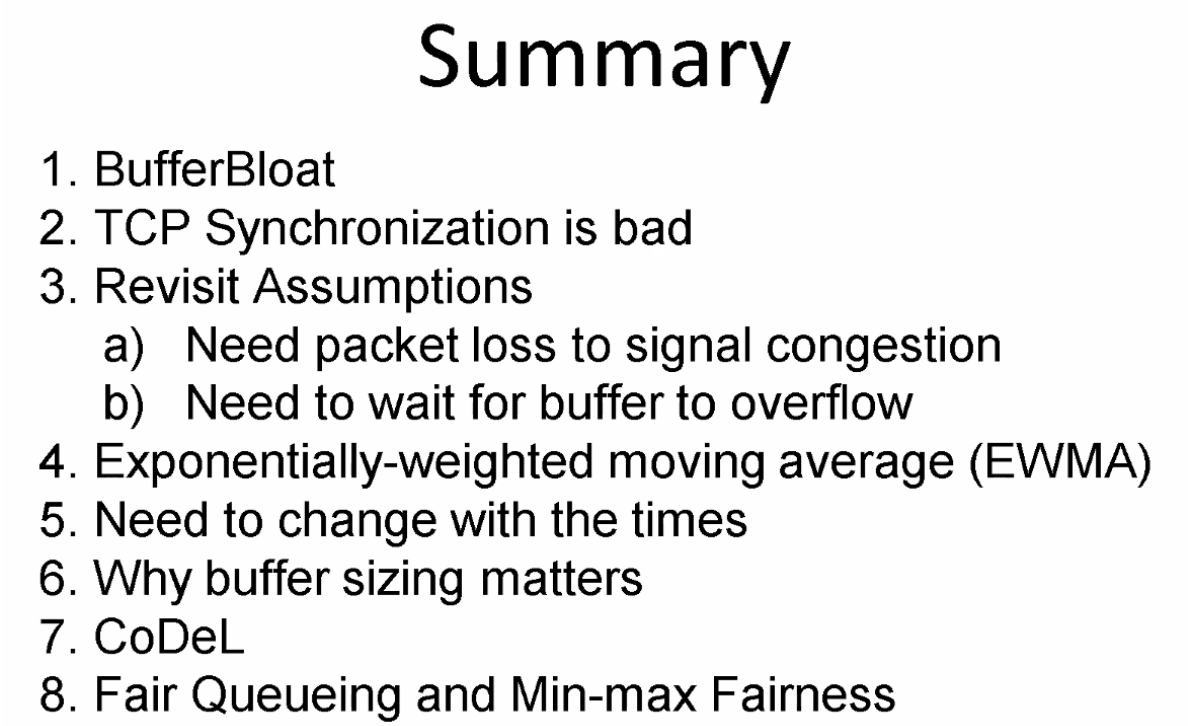
edge router: routers nearest to users



core-stateless fair queueing (1998)

modern AQM

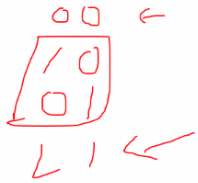
- PIFO queue (可插队的队列)(2016)
- Calendar Queues in P4(2020)



(max-min actually)

ECN & XCP & RCP

- Only knows of RED, but anticipates unknown AQMs
- CE codepoint should not be set based on the instantaneous queue size. ⇒ Why?
- Fragmentation possible
- Middleboxes can cause problems
- Dealing with non-compliant flows
- 1-bit implementation vs 2-bit implementation



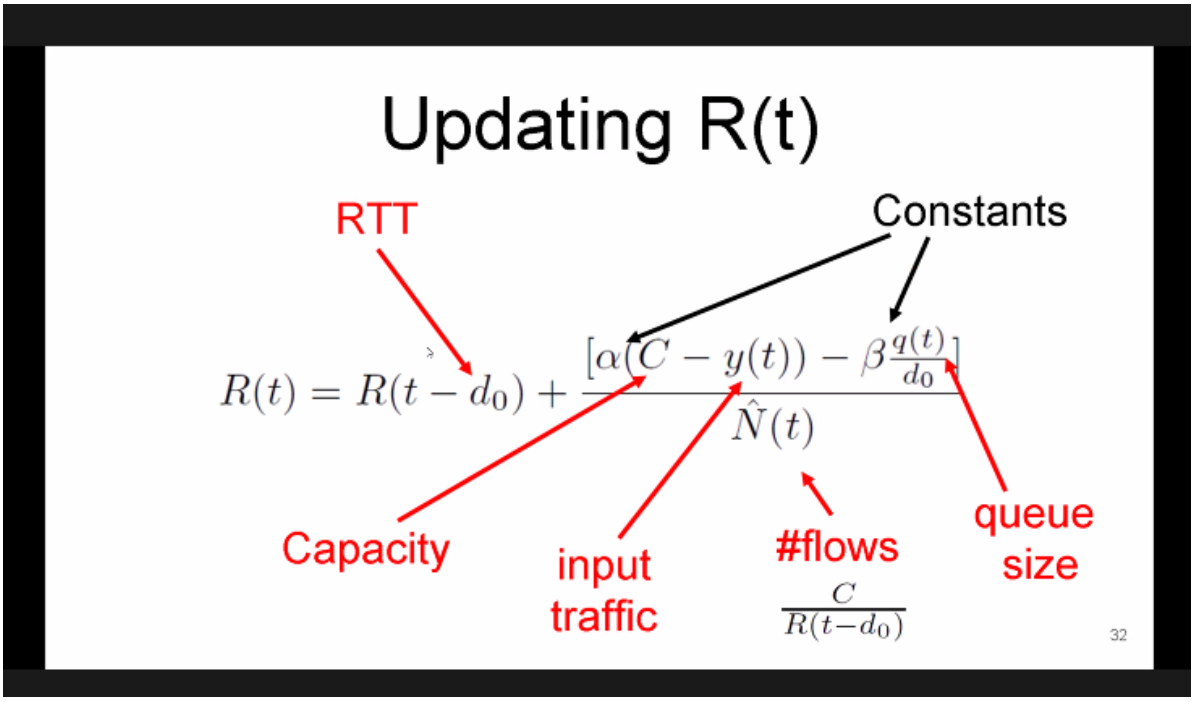
XCP

XCP: Implementation

- Aggregate state, not per-flow state
- Can replace AIMD/MIMD with other controllers
- Robust to estimation errors
- Incremental Deployment
- TCP Friendliness
- Everything needs to be modified ☹

RCP

delay: XCP 反馈到达sender时至少经过了半个 RTT



what if a flow doesn't have enough data to send?

Datacenter

Fat Tree

Jellyfish

DCTCP

adaptive window decreases (a factor between 1 and 2, based on fraction of marked pkts due to congestion)

TIMELY

delay based (NIC hardware measuring)

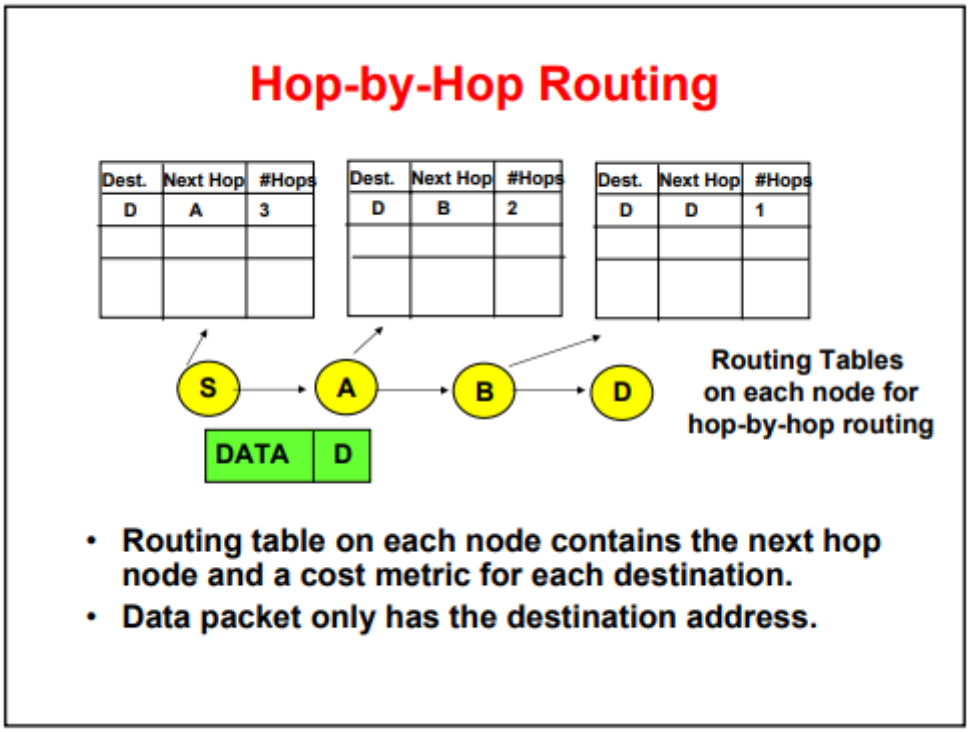
SDN

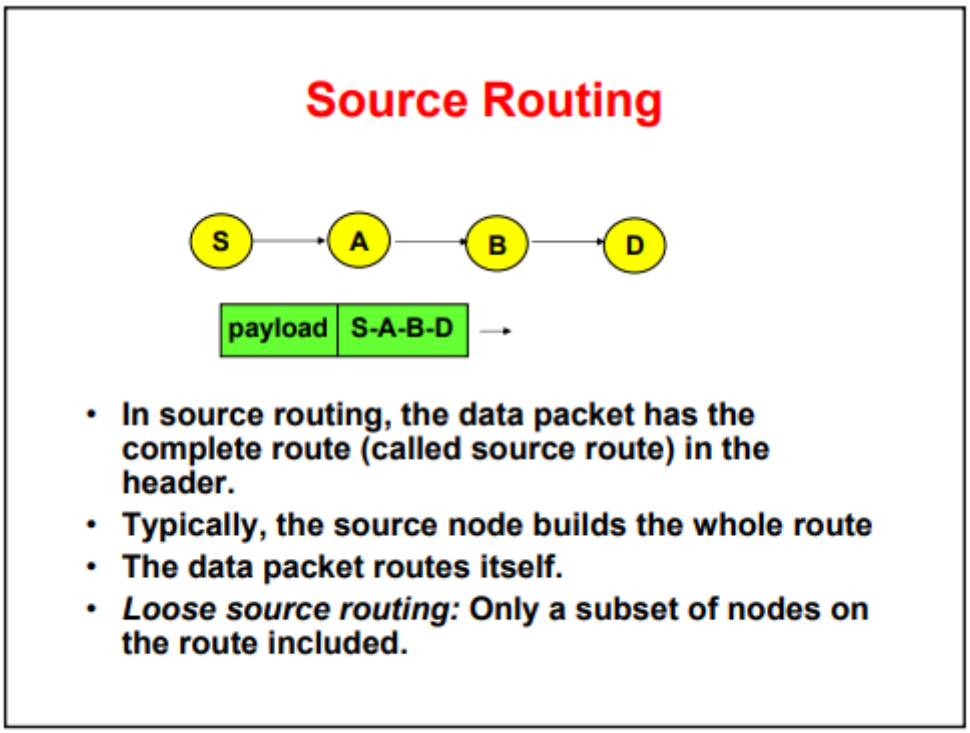
1. control plane & data plane
    1. control: apps, protocols, OS, ASIC drivers
    2. data: match-action tables, packet forwarding on ports
2. SDN 1.0: OpenFlow
    1. decouple control and data plane
    2. a logically centralized open control plane (slow but smart)
    3. data plane containing various of switches (dumb but fast)
    4. the whole picture
        1. control program: python scripts
        2. northbound interface: REST API
        3. control plane (NOS): floodlight
        4. southbound: OpenFlow
        5. CP agent & data plane: mininet network (agent per node)
3. SDN 2.0: P4

1. OpenFlow need to expand to support new protocols -> we need programmable data plane
2.

Ad Hoc Networks

1. ad hoc network
    1. each node serves as host and router at the same time
    2. every node form its own network on the fly
2. what we care about routing algorithm
    1. correctness: find the path to dest with high certainty if the path exists
    2. efficiency: among all paths possible, find the short one
    3. overhead: cost for maintaining states in the algorithm (storing topology, actions needed to update database when topology changes, retransmission, etc.)
3. source routing vs hop-by-hop routing

1. 

2. 

    3. ref: https://www3.cs.stonybrook.edu/~samir/cse534/routing.pdf
    4. DSR is source routing.
4. sequence number
    1. a technique used for versioning routing entries
    2. usually entry with higher sequence number replaces entry with a lower one
5. distance vector routing
    1. every node maintains a table("distance vector") of shortest distance to every destination with a next-hop
    2. distance vector is propagated to neighbors
6. proactive vs reactive
    1. proactive: periodically exchange info to maintain routing table for all dest even with no traffic
    2. reactive: discover path and update routing table only for dest with traffic
    3. DSDV is proactive
    4. AODV, TORA, DSR are reactive
7. TORA
    1. build DAG through broadcasting QUERY and UPDATE; DAG contains multiple paths
    2. multi routing -> topo changes can be fixed at a local level
8. measuring performance
    1. pause time: time step between randomly changes of topology
    2. vary pause time and observe packets sent and received
    3. DSDV performs badly when topo changes quickly
    4. TORA has a large routing overhead when topo changes quickly while DSDV is nearly constant
9. geographic routing
    1. greedy forwarding
    2. handle dead end in greedy forwarding
    3. GPSR
        1. planarization: GG & RNG
        2. perimeter mode forwarding: forward to closer faces towards dest (along $x\overline{D}$)

4. CLDP
    1. avoid costly planarization; send probes to detect overlapping links
5. GDSTR: hull tree
6. geographic is proactive since it maintains the topology all the time

Dynamo

在virtual node中使用k replica：需要保证replica分布在不同的physical node

R+W>N：抽屉原理，保证必至少有一个节点参与完整read-write流程

sloppy for writing

Pegasus