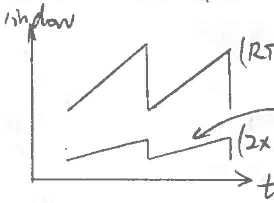
AQM

Drop-tail Queues

Good: easy to implement

Bad: Filled buffers (buffer delay), synchronized



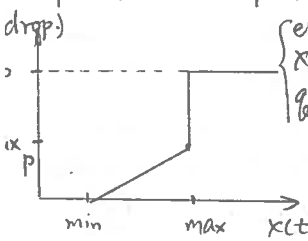
half the slope → half W_{max}

2xRTT → 1/4 throughput

* RED

Idea: Drop probabilistically to prevent cong. early and desynchronize flows

Implementation: Drop based on avg. queue len. $X(t)$



every T: $X(t) \leftarrow w_q \cdot q(t) + (1-w_q) X(t-T)$

$q(t)$ instant queue len.

Problems:
① Params hard to tune
② Queue depend on RTT and number of flows

* PI (Proportional Integral)

Ideas: ① Remove EWMA → responds faster than RED
② Integral Control → Decouple queue len. & num. flow
③ Use derivative of queue → more stable

$$P(t) \leftarrow P(t-T) + \alpha(q(t) - q_{ref}) + \beta(q(t) - q(t-T))$$

Integral control, to drive q to q_{ref}

derivative, we should respond faster when $\frac{dq}{dt}$ is big

* PI Enhanced

Ideas: ① Control delay instead of length
② Auto tune params (α, β) based on P

XCP

Sender reports RTT and CWND, router specify ΔCWND

No per-flow state at router. CWND += ΔCWND

* Efficiency Controller

Match input to output capacity to keep queue short

look at aggregated traffic info

S: Spare Bandwidth

d: average RTT

$\Delta = \boxed{\alpha \cdot d \cdot S} - \boxed{\beta \cdot Q}$

→ Bandwidth → match B_{in} and B_{out}

→ drain the queue

Q: queue len.

* Fairness Controller

Divide Δ between flows (look at info reported by Tx)

$\Delta > 0$: distribute evenly → same throughput increase

Per-flow $\Delta_{CWND_i} \propto RTT_i$

Per-packet $p_i \propto RTT_i \Big/ \frac{CWND_i}{RTT_i}$ ← Rate of incoming packets

$\sum \frac{p_i}{RTT_i} = \Delta/d$ ← need to change Δ by time d

↑ this change takes effect after time RTT_i

for all packets during control interval d

$\Rightarrow k \sum \frac{RTT_i}{CWND_i} = \Delta/d$, $k = \frac{\Delta}{d} \Big/ \sum \frac{RTT_i}{CWND_i}$

$\Delta < 0$: distribute prop. to throughput

↓ faster flow decrease faster (∝ throughput)

Per-flow $\Delta_{CWND_i} \propto CWND_i$

Per-packet $p_i \propto CWND_i \Big/ \frac{CWND_i}{RTT_i} = RTT_i$

Fair Queueing & CSFQ

Work conserving: Never idle if has packet in Q

Max-min Fairness: converge to α that
request $r_i < \alpha$: give r_i
$r_i > \alpha$: give α

Scheduling: Which pkt to send next?

How to fair? Bit-by-bit Round-robin

Real world: Emulate bit-by-bit RR

* Fair Queueing

Round: each queue sends one bit

μ: output rate ; N: num of flows

$\frac{dR}{dt} = \frac{\mu}{N}$ (Num of rounds to finish a pkt is independent of N)

P_i^α: i-th pkt of queue α (the size of)

S_i^α: when it reaches head of queue

F_i^α: when it is finished transmitting

$S_i^\alpha = \max(R(t), F_{i-1}^\alpha)$; $F_i^\alpha = S_i^\alpha + P_i^\alpha$

Send packet with smallest F_i^α

Deficit Round Robin: for each queue, credit increases at rate of fair rate, decrease by pkt sz.

* CSFQ

Ideas: ① Edge routers mark estimate of arrival time
② Core routers use ① and internal measure of fair share to compute prob. of drop.
③ Estimation of fair share converges quickly

(Arrival rate)

$$r_i^{new} = (1 - e^{-T_i^k/K}) \frac{l_i^k}{T_i^k} + e^{-T_i^k/K} r_i^{old}$$

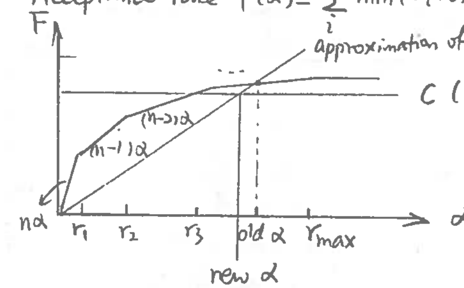
t_i^k: arrival time of kth packet in flow i

l_i^k: length of k th packet in flow i

$T_i^k = t_i^k - t_i^{k-1}$ K: constant

$P(drop) = \max(1 - \frac{\alpha}{r_i(t)}, 0)$, α is fair rate

Acceptance rate $F(\alpha) = \sum_i \min(r_i(t), \alpha)$



BGP

Route: link/next-hop to a dest (local info)

Path: Sequence of edges (the whole path)

Why scalable: ① Nearby in topology → similar IP
② Route announced as prefixes

Longest Prefix Match: use the longest match of prefix

AS: unit of who announces a route

Transit: provider provides access for customer

Peering: mutual access to subset of each other

B advertise P to A: B will forward pkts to P from A

Pricing: 95% of 5-min moving avg. throughput

Pick route: Customer > Peer > Provider (LOCAL PREF)

Announce Route (see reverse side)

To \ from	Customer	Peer	Provider
Customer	✓	✓	✓
Peer	✓	X	X
Provider	✓	X	X

BGP Goals: ① Scalability ② Policy ③ Cooperative Competition

iBGP Route Reflector

From non-client: only to client
From client: to all peer

BGP Attributes:

NEXT HOP: IP of next-hop router

ASPATH: all AS that announcement goes thru

MED: should use entry point with smallest MED

BGP Path Selection:

LOCAL PREF > len(ASPATH) > MED > eBGP or iBGP (learned from?) > IGP Path cost > Smaller Router IP

Measurement

What? Transport: performance, congestion control
Network: routing fail, topology, performance

AS-level Topo: find ASes and BGP links

* Router Alias Resolution: map IP to Router
Methods: ① Probe and see reply IP
② Increasing IPID field

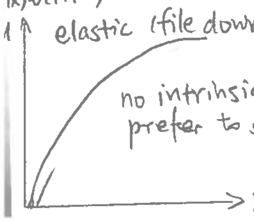
* ISP Topology Inference: combine tracert info
Challenge: how to choose target IP to trace?
Rocketfuel: ① sufficient num of vantage points
② select target IP
③ Deal with tracert issues

* ZMap
Idea: ① bypass TCP/IP and craft Eth frames
② Encode dest info in packet → stateless

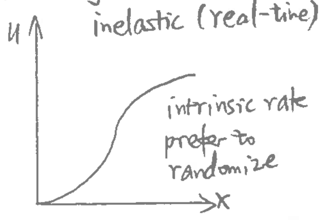
NUM (Network Utility Max.)

U(x) Utility Function: benefit of sending at rate X

elastic (file download)



inelastic (real-time)



Max. $\sum_{i=1}^{N} U_i(x_i)$ N flows, L links

When $R_{L\times N} \times [x_i] \leq [c_i]$ (capacity) (routing matrix)

If U(x) concave, have unique solution.

* α-fairness

maximize $\sum_{i=1}^{N} \frac{x_i^{1-\alpha}}{1-\alpha}$ $(\alpha \geq 0)$

α=0: throughput max.
α→1: prop. fairness ($\log(x_1)+\log(x_2)+\ldots$)
α→∞: max-min fair

* Solve NUM

P_ℓ: cong. price for link ℓ (price per unit bw)

q_i: total price for source i ($\sum P_\ell$ along path)

profit $= U_i(x_i) - q_i x_i$ (max. done at source indep)

$P_\ell(t+1) = \max(P_\ell(t) + k(\underbrace{y_\ell(t)}_{\text{total traffic}} - \underbrace{C_\ell}_{\text{capacity}}), 0)$ (done at link)

* TCP+PI:

$P(\ell)$ follow the same update rule as prob. drop

solution: $x_i = \frac{1}{RTT_i \sqrt{q_i}}$ (TCP formula)

$U_i(x_i) = \frac{-1}{RTT_i^2 x_i}$ (α-fair, α=2)

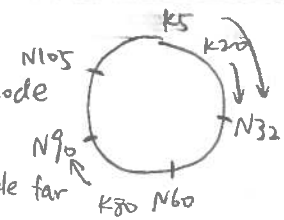
DHT / Chord

Node and key share a ring

Key stored on next higher node

Consistent hashing

Each node point to 1/2, 1/4, cycle far



* Fault tolerance:

Each node keep next r nodes

Each key stored by r nodes after owner

* Joining

1. look up itself to find the successor
2. set self. successor
3. copy keys from successor
4. call stablize
① find succ. pred. and set self.succ
② notify self.succ about itself

* Lookup

Each node stores $succ(n+2^{i-1})$, i.e. node that have n+2

Get k: search j in table that j is closest smaller to k

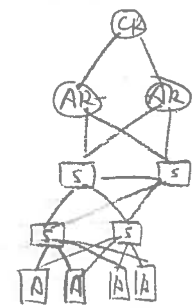
VL2 / Datacenter

Agility: any service at any server

Layer 2 Good: Auto-config, failover
Bad: Broadcast (ARP), no multipath (STP)

Layer 3 Good: Scalable, multipath
Bad: Hard to migrate (change IP), config

* Conventional Problem



* Goal of VL2

1. L2 semantics
2. Uniform high capacity
3. Performance isolation

* ECMP LB

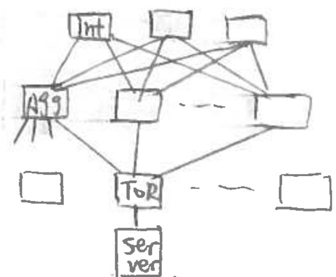
→ Pick equal-cost paths by hash of 5-tuples (flow-level LB)

Problem: hash collisions

Not Problem:
① Flows are many and sma
② Switch-switch link thick than flow (NIC) size

* Name-Loc Separation

* Clos Topo

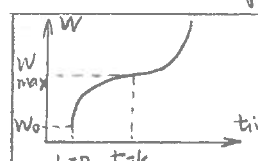


Number of servers: (24-port switch)

Top: all 24 ports connect to next level

other layers: half talk to upper level

Congestion Ctrl

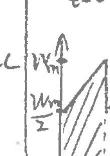


* Cubic:

$W(t) = \alpha(t-k)^3 + W_{max}$

Loss: $t=0$, $W(0) = (1-\beta) W_{max}$

$P \propto \frac{1}{W_m^{4/3}}$, Thruput $\propto \frac{1}{P^{3/4}}$



* Reno:

$BDP + Q = W_{max}$, $\frac{W_{max}}{2} \geq BDP \Rightarrow Q > BDP$ for ma util

$P \propto \frac{1}{W_m^2}$, $W_m \propto \frac{1}{\sqrt{P}}$, Thruput $\propto \frac{1}{\sqrt{P}\, RTT}$