

Resource Control and Reservation

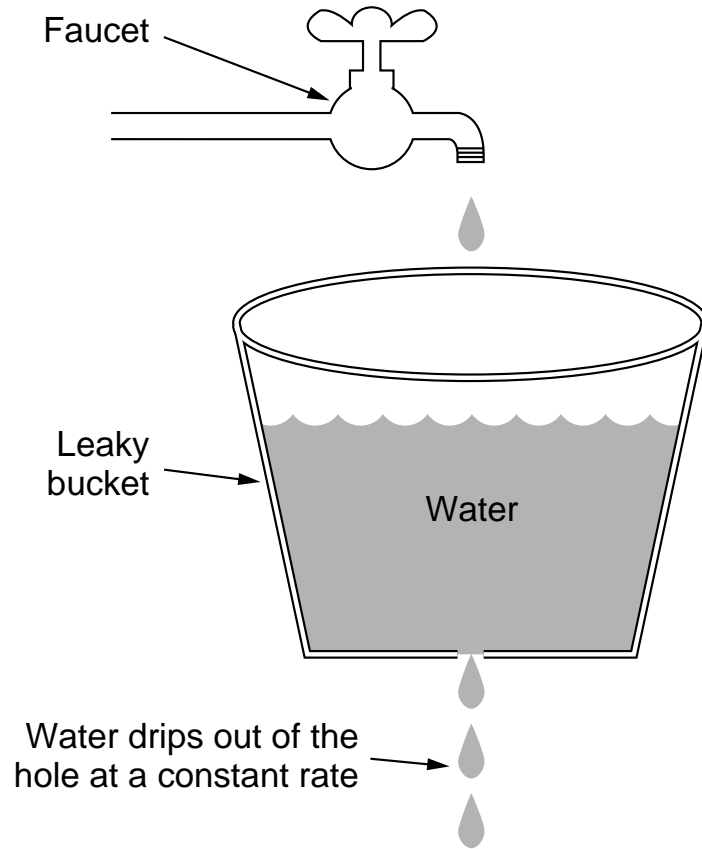
Resource Control and Reservation

- policing: hold sources to committed resources
- scheduling: isolate flows, guarantees
- resource reservation: establish flows

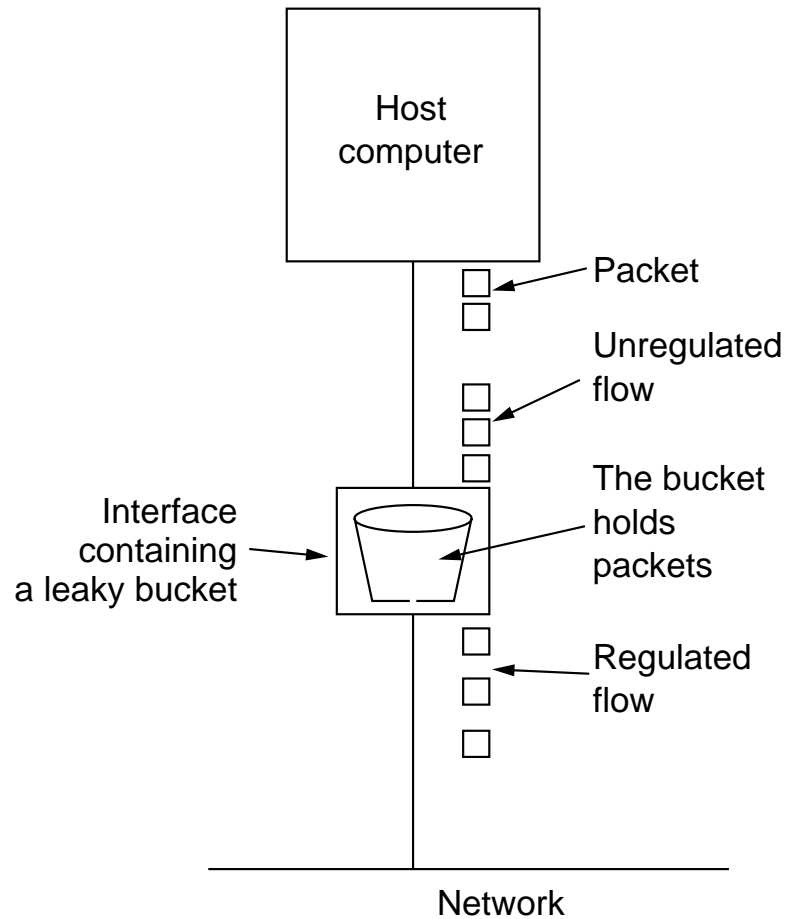
Usage parameter control: leaky bucket algorithm

- constrain what host can inject into the network
- single server queue with fixed service time
- finite-size bucket \Rightarrow either throttle source or loose packets
- no burstiness allowed

Leaky bucket algorithm



(a)



(b)

Token bucket

- *tokens* allow bursts into the network
- tokens generated at constant rate up to maximum burst size
- if no token, either quench source or drop packet
- implementation: token counter, incremented periodically

Generic Cell Rate Algorithm (GCRA)

Mechanism used by UNI 3.1 to police either peak or mean cell rate.

PCR: peak cell rate

SCR: sustainable cell rate = mean cell rate

CDVT: cell delay variation tolerance

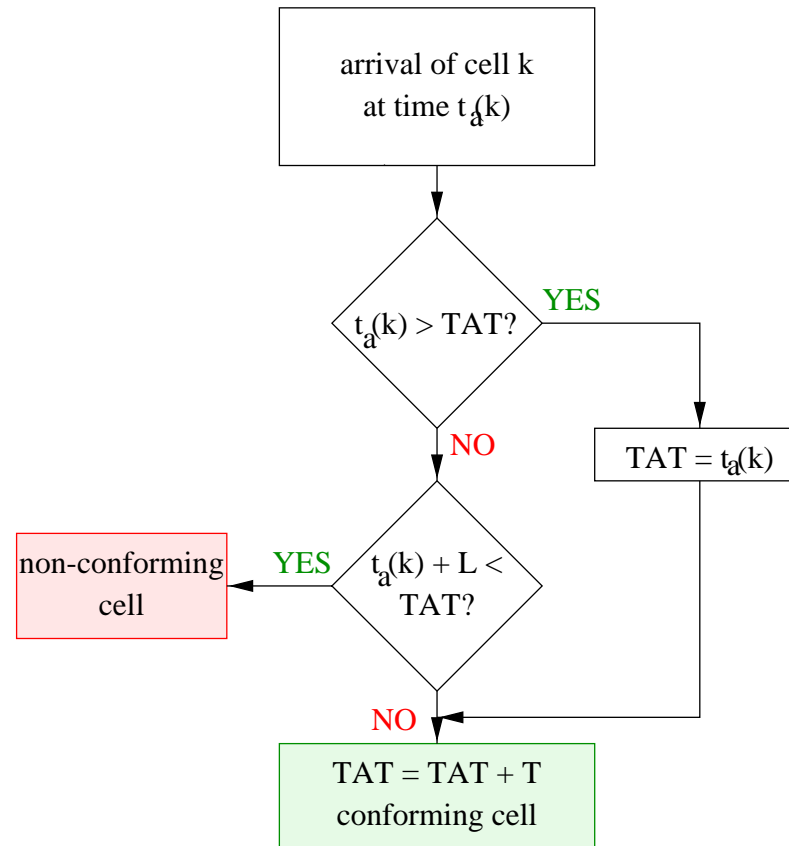
τ_s : burst tolerance

| | peak rate | mean rate |
|-----|-----------|-----------|
| T | 1/PCR | 1/SCR |
| L | CDVT | τ_s |

GCRA

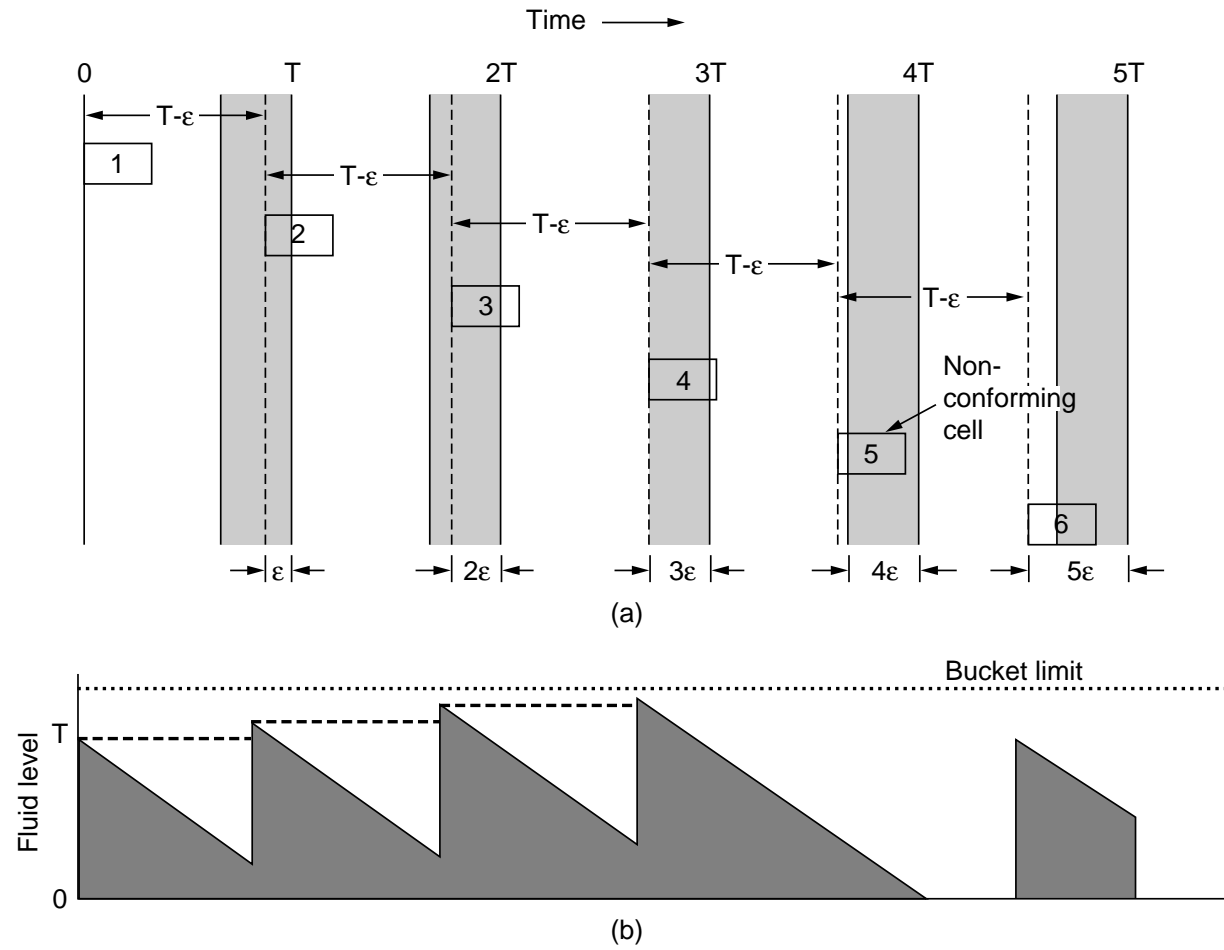
- cell i can arrive at $t_i > t_{i-1} + T - L$; but: arrival time set to $t_i = t_{i-1} + T$
- can't save up late arrivals
- can't accumulate L

GCRA flow chart



TAT = theoretical arrival time

GCRA



Packet scheduling

work conserving: never delay a packet if line is idle \implies no lower bound on jitter

non-work-conserving: minimum residency time \implies jitter bound

Isolation: one misbehaving source can't monopolize resources

FIFO+ and HL

For packets with real-time constraints (deadlines) \Rightarrow give priority to those about to miss their deadline

hop-laxity: $\text{priority} = \frac{\text{hops to go}}{\text{time left}}$

drop packets that have exceeded their deadline or are too close

FIFO+: give priority to packets if travel time $>$ average for class

- both require accumulating delays
- performance better than FIFO
- but: no guarantees, scheduling overhead

Weighted Fair Queueing (WFQ)

- fair queueing: separate queues for each input stream, round-robin \Rightarrow favors long packets, wait for n other queues if a bit too late
- \Rightarrow WFQ: order transmissions by when last bit would have been sent under bit-by-bit round robin
- need ordered queue of size q : $O(\log q)$ \Rightarrow expensive
- divide bandwidth into m -bit cycles and distribute unequally

Weighted Fair Queueing

Delay D_i of flow i if token bucket at edge:

$$D_i = \frac{\beta_i}{g_i} + \frac{(h_i - 1)l_i}{g_i} + \sum_{m=1}^{h_i} \frac{l_\star}{r_m}$$

where β : bucket size; g_i : fraction; l_i : maximum packet length for i ; l_\star : maximum packet length in network; h_i : number of hops; r_m : outbound bandwidth

Reservations

First approach: everybody is the same \Rightarrow best effort \Rightarrow

- enough bandwidth for everybody (telephone network)
- “human backoff” if unusable
- TCP for data applications (but: also minimum usable bandwidth)
- adjust audio or video coding to best possible \Rightarrow application control (later)
- pick least congested route: telephone system, but Internet too large

Reservations

Some are more equal than others \Rightarrow

- incumbency protection
- priorities (general over PFC)
- bulk service vs. priority delivery \Rightarrow cost

Reservations

\$/kb/s may be dynamic \Rightarrow

- reservation may change during the lifetime of an application
- networks may not be homogeneous \Rightarrow different multicast groups for different *layers* or versions

RSVP

Receiver-oriented, out-of-band reservation protocol standardized by IETF:

- not a routing protocol, but interacts with routing
- may need *QOS routing* to pick appropriate path
- transports *opaque* QOS and policy parameters for sessions
- flow: group of packets being treated the same \Rightarrow same multicast group or destination, IPv6 flow id, ...
- simplex \Rightarrow setup for unidirectional data flows

RSVP, cont'd.

- does not prescribe admission or policy control
- sets up packet classifier, but does not handle packets
- independent sessions (can't tie video and audio session)
- multicast (and unicast)
- either own protocol type or UDP encapsulated

RSVP Objects

Flow descriptor =

Flowspec: • service class

- Rspec \Rightarrow desired QoS
- Tspec \Rightarrow describes traffic characteristics

Filterspec: which packets get this treatment \Rightarrow sender IP address/port, protocol, other fields \Rightarrow complex (regular expressions? IP options!)
 \Rightarrow currently, sender IP address and UDP/TCP port \Rightarrow no fragmentation

Reservation Styles

| sender selection | reservations | |
|------------------|--------------------------|----------------------|
| | distinct for each sender | shared |
| explicit | fixed filter (FF) | shared-explicit (SE) |
| wildcard (all) | – | wildcard filter (WF) |

☞ mutually incompatible

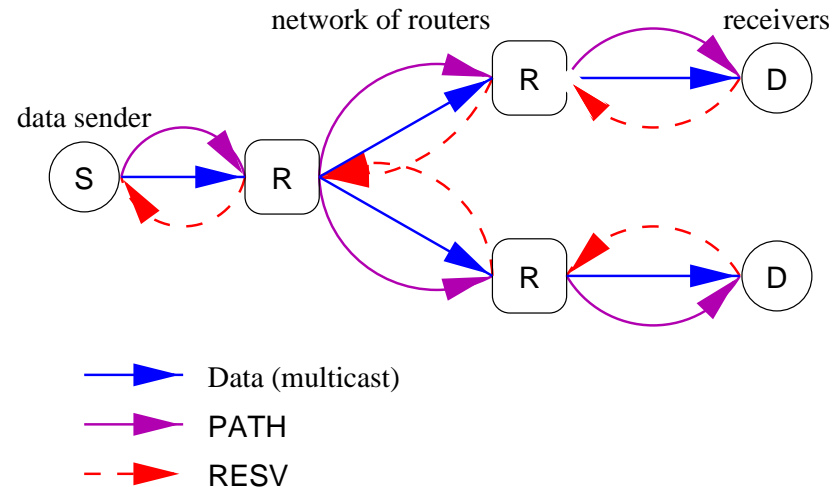
explicit: list senders by address

wildcard: any sender with a specific port (e.g.)

shared: only one active data source ☞ e.g., reserve for twice needed for audio

distinct: video

RSVP: basic operation



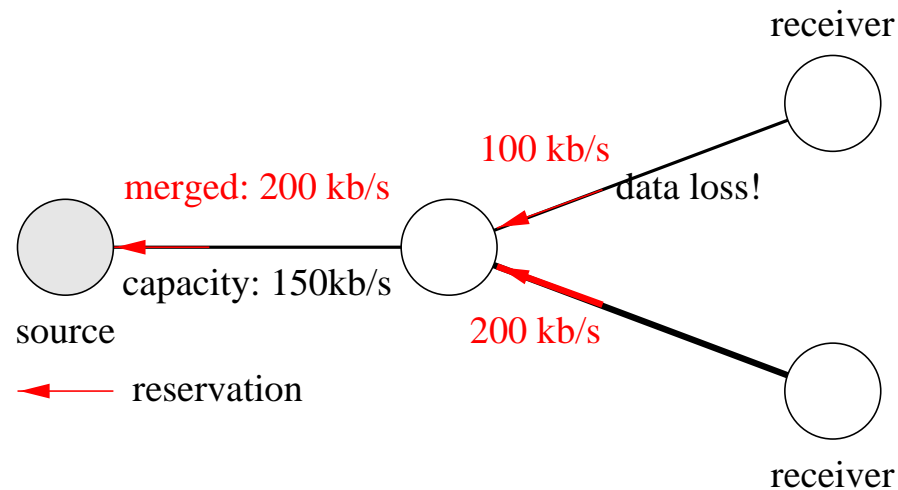
- receiver joins group via IGMP
- source sends PATH messages to receivers \Rightarrow same path as data: previous hop to source, Tspec \leftrightarrow RESV one path, data another
- receivers send RESV messages back to senders

RSVP: basic operation

- reservations may be lowered
- reservations are merged at each node for same sender: max. flowspec
- merge point or data sender may send confirmation (if requested)
- reservations *may* get merged between senders (audio!)
- one-pass \Rightarrow receiver doesn't know final QoS \Rightarrow One Pass With Advertising
- application *should* explicitly tear down reservations

Killer Reservations

1. small reservation in place; another receiver larger reservation \Rightarrow failure? \Rightarrow keep old
2. large reservation fails again and again \Rightarrow blocks new, smaller one



RSVP service classes

guaranteed: no loss, upper bound on delay

controlled load: “few” losses, “like unloaded network” \Rightarrow
delay-adaptive applications

best effort: no guarantees; current IP service model \Rightarrow delay +
bandwidth adaptive services

others: research

RSVP vs. ATM resource reservation

| | IP, RSVP | ATM |
|-----------------------------|------------|------------------------------------|
| multicast tree, reservation | sequential | same time |
| origin | receiver | sender (root) \Rightarrow UNI4.0 |
| change reservations | yes | no |
| routing changes | time-out | re-establish VC |
| routing | IP routing | PNNI (QOS) |
| flow merging (audio) | yes | no (separate VCs) |
| receiver diversity | not yet | no |
| state | soft | hard |

The recurring costs of reservations

Signaling: processing and state maintenance, APIs

Routing: QoS path selection, state distribution

Policy: who gets what (and who doesn't)

Charging, billing, accounting, service contracts: right party pays for usage, ensure QoS is delivered as promised

RSVP implementation

- scheduling: about 10% cost overhead
- low-end 68040: 0.73 ms for PATH, 0.37 ms for RESV
- \Rightarrow approximately 1,000 flow setups/s
- processing of PATH (RESV) refresh: 0.33 ms (0.29 ms)
- \Rightarrow approximate capacity is 1,600 flows
- about 500 bytes/flow
- refresh bandwidth \approx 100 kb/s for 1000 flows (30 s refresh)
- PATH: 208 bytes, RESV: 148 bytes

Resource reservation: general comments

- doesn't help if network capacity \ll demand
- modes:
 - receiver-oriented: RSVP**
 - sender-oriented: YESSIR**
- scaling issues: a reservation for every phone call \leftrightarrow datagram idea, routing aggregation

RSVP problems

- if reservation/tear down request lost, no immediate feedback
 - can increase reservation latency or “phone off hook”
 - large number of refreshes \Rightarrow scaling problems
- \Rightarrow hop-by-hop confirmation (\rightarrow extend refresh interval)

RSVP scaling

Scaling issues:

- number of flow states \implies refresh, memory, time-outs
- large number of packet queues

Alternatives:

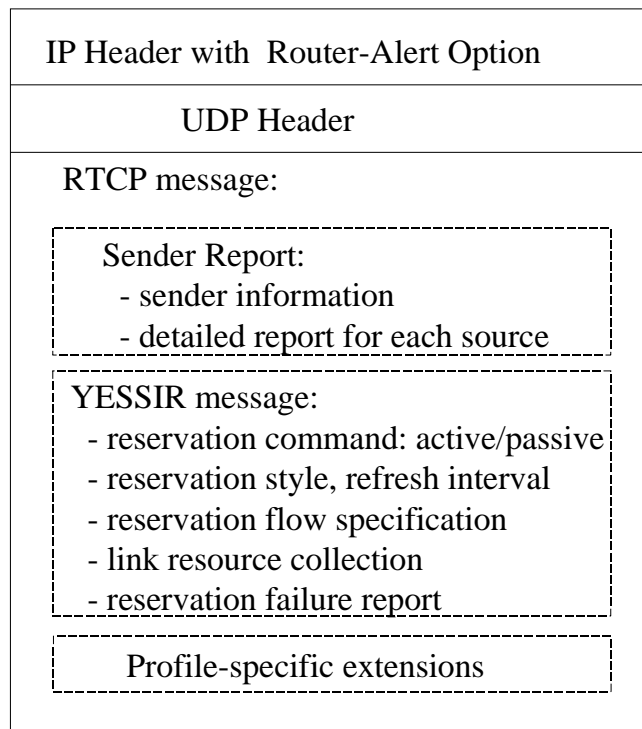
- “tunnels” = encapsulation IP-in-IP \implies overhead
- aggregation for sender reservation \implies flow classes
- drop and delay preferences

YESSIR: Yet another Sender Session Internet Reservation

- RSVP: separate daemon, API
- \Rightarrow integrate into application that needs it (embedded systems!)
- in-band \Rightarrow easier firewall
- router alert option
- soft-state + RTCP BYE
- partial reservations: add links as session ages \leftrightarrow fragmentation

YESSIR

plain RTCP SRs or additional information:



end-to-end refresh (vs. hop-by-hop)

YESSIR

- measurement mode
- IntServ flow specs
- PT-based for well-known PTs
- TOS-based: value
- killer reservations \Rightarrow SR reservation failure
- OPWA: hop count, propagation delay, aggregated bandwidth, delay bounds \Rightarrow updated at router
- cost: $360 \mu s$

SRP: Scalable Reservation Protocol

- sender-oriented, out-of-band
- data packets marked as REQUEST \Rightarrow learn reservation level
- router aggregates requests, downgrades to best effort
- receiver reports rate of successful REQUESTS
- \Rightarrow sender adjusts rate RESERVED data packets
- aggregation by estimation:
- max(observed traffic over several intervals)
- effective bandwidth $e = \sup \frac{\sum n_i}{t_j - t_i + D}$

SRP packet processing

