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 either throttle source or loose packets
- no burstiness allowed

Leaky bucket algorithm
**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

\[ \tau_s \]: burst tolerance

\[
\begin{align*}
\text{peak rate} & = \frac{1}{\text{PCR}} \\
\text{mean rate} & = \frac{1}{\text{SCR}} \\
\tau_s & = \frac{1}{\text{CDVT}}
\end{align*}
\]

**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

\[ TAT = t_{(k)} + \text{L} \]

\[ TAT = TAT + T \]

\[ TAT = TAT + L \]

\[ TAT = TAT + \text{I} \]

GCRA

Packet scheduling

**work conserving:** never delay a packet if line is idle \(\Rightarrow\) no lower bound on jitter

**non-work-conserving:** minimum residency time \(\Rightarrow\) 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:** 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:

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

where $\beta$: bucket size; $g_i$: fraction; $l_i$: maximum packet length for $i$; $l^*$: 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 ➟ same multicast group or destination, IPv6 flow id, ...
- simplex ➟ 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 ➟ desired QoS
  - Tspec ➟ describes traffic characteristics

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

RSVP, cont’d.

**Reservation Styles**

<table>
<thead>
<tr>
<th>sender selection</th>
<th>reservations</th>
</tr>
</thead>
<tbody>
<tr>
<td>explicit fixed filter (FF)</td>
<td>shared-explicit (SE)</td>
</tr>
<tr>
<td>wildcard (all)</td>
<td>—</td>
</tr>
</tbody>
</table>

⇒ 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

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**

<table>
<thead>
<tr>
<th></th>
<th>IP, RSVP</th>
<th>ATM</th>
</tr>
</thead>
<tbody>
<tr>
<td>multicast tree, reservation</td>
<td>IP, sequential</td>
<td>ATM same time</td>
</tr>
<tr>
<td>origin</td>
<td>receiver</td>
<td>sender (root) UNI4.0</td>
</tr>
<tr>
<td>change reservations</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>routing changes</td>
<td>time-out</td>
<td>re-establish VC</td>
</tr>
<tr>
<td>routing</td>
<td>IP routing</td>
<td>PNNI (QOS)</td>
</tr>
<tr>
<td>flow merging (audio)</td>
<td>yes</td>
<td>no (separate VCs)</td>
</tr>
<tr>
<td>receiver diversity</td>
<td>no yet</td>
<td>no</td>
</tr>
<tr>
<td>state</td>
<td>soft</td>
<td>hard</td>
</tr>
</tbody>
</table>

October 30, 2001

**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

October 30, 2001

**RSVP implementation**

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

October 30, 2001

**Resource reservation: general comments**

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

October 30, 2001
RSVP problems

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

RSVP scaling

Scaling issues:
- number of flow states → refresh, memory, time-outs
- large number of packet queues

Alternatives:
- “tunnels” = encapsulation IP-in-IP → overhead
- aggregation for sender reservation → flow classes
- drop and delay preferences

YESSIR: Yet another Sender Session Internet Reservation

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

YESSIR

plain RTCP SRs or additional information:

YESSIR message:
- reservation command: active/passive
- reservation style, refresh interval
- reservation flow specification
- link resource collection
- reservation failure report

IP Header with Router-Alert Option

UDP Header

RTCP message:

- Sender Report:
  - detailed report for each source

YESSIR message:

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

**YESSIR**

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

**SRP: Scalable Reservation Protocol**

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

**SRP packet processing**

<table>
<thead>
<tr>
<th>SRP estimator</th>
<th>Packet scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reserved</strong></td>
<td><strong>Reserved</strong></td>
</tr>
<tr>
<td>Is an update of the estimated bandwidth acceptable?</td>
<td>Can the packet be schedule in the reserved service class?</td>
</tr>
<tr>
<td><strong>Request</strong></td>
<td><strong>Request</strong></td>
</tr>
<tr>
<td><strong>Best effort</strong></td>
<td><strong>Best effort</strong></td>
</tr>
<tr>
<td><strong>Best effort</strong></td>
<td><strong>Best effort</strong></td>
</tr>
<tr>
<td><strong>Best effort</strong></td>
<td><strong>Best effort</strong></td>
</tr>
</tbody>
</table>

Discard

---

October 30, 2001