Limitations of IP

- IP provides only best effort service
- IP does not participate in resource management
  - Cannot provide service guarantees on a per flow basis
  - Cannot provide service differentiation among traffic aggregates
- Early efforts
  - Tenet group at Berkeley
  - ATM
- IETF efforts
  - Integrated services initiative
  - Differentiated services initiative

So, what is required?

- Flow differentiation
  - Simple FIFO scheduling will not work!
- Admission control
- Resource reservation
- Flow specification

Integrated Services Internet

- Enhance IP's service model
  - Old model: single best-effort service class
  - New model: multiple service classes, including best-effort and QoS classes
- Create protocols and algorithms to support new service models
  - Old model: no resource management at IP level
  - New model: explicit resource management at IP level
- Key architecture difference
  - Old model: stateless
  - New model: per flow state maintained at routers
    - used for admission control and scheduling
    - set up by signaling protocol
Integrated Services Example

- Achieve per-flow bandwidth and delay guarantees
  Example: guarantee 1Mbps and < 100 ms delay to a flow

- Allocate resources - perform per-flow admission control

Integrated Services Example

- Install per-flow state
Integrated Services Example: Data Path
- Per-flow classification

Integrated Services Example: Data Path
- Per-flow buffer management

Integrated Services Example
- Per-flow scheduling

How Things Fit Together
- Admission Control
  - Routing
  - RSVP
  - Per-Flow QoS Table
  - Forwarding Table
  - Route Lookup
  - Classifier
  - Scheduler
**Service Classes**

- Service can be viewed as a contract between network and communication client
  - end-to-end service
  - other service scopes possible
- Three common services
  - best-effort ("elastic" applications)
  - hard real-time ("real-time" applications)
  - soft real-time ("tolerant" applications)

**Hard Real Time: Guaranteed Services**

- Service contract
  - network to client: guarantee a deterministic upper bound on delay for each packet in a session
  - client to network: the session does not send more than it specifies
- Algorithm support
  - admission control based on worst-case analysis
  - per-flow classification/scheduling at routers

**Soft Real Time: Controlled Load Service**

- Service contract:
  - network to client: similar performance as an unloaded best-effort network
  - client to network: the session does not send more than it specifies
- Algorithm Support
  - admission control based on measurement of aggregates
  - scheduling for aggregate possible

**Role of RSVP in the Architecture**

- Signaling protocol for establishing per flow state
- Carry resource requests from hosts to routers
- Collect needed information from routers to hosts
- At each hop
  - consults admission control and policy module
  - sets up admission state or informs the requester of the failure
What is still Missing?
- Classification algorithm
- Scheduling algorithm
- Admission control algorithm
- QoS Routing algorithm

Why did IntServ fail?
- Economic factors
  - Deployment cost vs Benefit
- Is reservation, the right approach?
  - Multicast centric view
- Is per-flow state maintenance an issue?
- What about QoS in general?

What is the Problem?
- Goal: provide support for wide variety of applications:
  - Interactive TV, IP telephony, on-line gaming (distributed simulations), VPNs, etc
- Problem:
  - Best-effort cannot do it (see previous lecture)
  - Intserv can support all these applications, but
    - Too complex
    - Not scalable

Differentiated Services (Diffserv)
- Build around the concept of domain
- Domain - a contiguous region of network under the same administrative ownership
- Differentiate between edge and core routers
- Edge routers
  - Perform per aggregate shaping or policing
  - Mark packets with a small number of bits; each bit encoding represents a class (subclass)
- Core routers
  - Process packets based on packet marking
- Far more scalable than Intserv, but provides weaker services
**Diffserv Architecture**

- **Ingress routers**
  - Police/shape traffic
  - Set Differentiated Service Code Point (DSCP) in Diffserv (DS) field

- **Core routers**
  - Implement Per Hop Behavior (PHB) for each DSCP
  - Process packets based on DSCP

**Differentiated Service (DS) Field**

- DS Filed

- Version
- TOS
- Length
- Identification
- Flags
- Fragment offset
- TTL
- Protocol
- Header checksum
- Source address
- Destination address
- Data

- DS filed reuse the first 6 bits from the former Type of Service (TOS) byte
- The other two bits are proposed to be used by ECN

**Differentiated Services**

- Two types of service
  - Assured service
  - Premium service

- Plus, best-effort service

**Assured Service**

[Clark & Wroclawski '97]

- Defined in terms of user profile, how much assured traffic is a user allowed to inject into the network

- Network: provides a lower loss rate than best-effort
  - In case of congestion best-effort packets are dropped first

- User: sends no more assured traffic than its profile
  - If it sends more, the excess traffic is converted to best-effort
**Assured Service**

- Large spatial granularity service
- Theoretically, user profile is defined irrespective of destination
  - All other services we learnt are end-to-end, i.e., we know destination(s) a priori
- This makes service very useful, but hard to provision (why?)

**Premium Service**

*Jacobson '97*

- Provides the abstraction of a virtual pipe between an ingress and an egress router
- Network: guarantees that premium packets are not dropped and they experience low delay
- User: does not send more than the size of the pipe
  - If it sends more, excess traffic is delayed, and dropped when buffer overflows

**Control Path**

- Each domain is assigned a Bandwidth Broker (BB)
  - Usually, used to perform ingress-egress bandwidth allocation
- BB is responsible to perform admission control in the entire domain
- BB not easy to implement
  - Require complete knowledge about domain
  - Single point of failure, may be performance bottleneck
  - Designing BB still a research problem

**Example**

- Achieve end-to-end bandwidth guarantee
**Comparison to Best-Effort and Intserv**

<table>
<thead>
<tr>
<th></th>
<th>Best-Effort</th>
<th>Diffserv</th>
<th>Intserv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>Per aggregate isolation</td>
<td>Per aggregate guarantee</td>
<td>Per flow isolation</td>
</tr>
<tr>
<td>No isolation</td>
<td>Per aggregate guarantee</td>
<td>Per flow guarantee</td>
<td></td>
</tr>
<tr>
<td>Service scope</td>
<td>End-to-end</td>
<td>Domain</td>
<td>End-to-end</td>
</tr>
<tr>
<td>Complexity</td>
<td>No setup</td>
<td>Long term setup</td>
<td>Per flow setup</td>
</tr>
<tr>
<td>Scalability</td>
<td>Highly scalable (routers maintain only routing state)</td>
<td>Scalable (routers maintain per aggregate state, core routers per class state)</td>
<td>Not scalable (each router maintains per flow state)</td>
</tr>
</tbody>
</table>

**Summary**

- Diffserv more scalable than Intserv
  - Edge routers maintain per aggregate state
  - Core routers maintain state only for a few traffic classes
- But, provides weaker services than Intserv, e.g.,
  - Per aggregate bandwidth guarantees (premium service) vs. per flow bandwidth and delay guarantees
- BB is not an entirely solved problem
  - Single point of failure
  - Handle only long term reservations (hours, days)

**What is the Problem?**

- Internet has limited resources and management capabilities
  - Prone to congestion, and denial of service
  - Cannot provide guarantees

- Existing solutions
  - Stateless - scalable and robust, but weak network services
  - Stateful - powerful services, but much less scalable and robust

**Stateless vs. Stateful Solutions**

- **Stateless** solutions - routers maintain no fine grained state about traffic
  - Scalable, robust
  - Weak services
- **Stateful** solutions - routers maintain per-flow state
  - Powerful services
  - Guaranteed services + high resource utilization
  - Fine grained differentiation
  - Protection
  - Much less scalable and robust
Existing Solutions

<table>
<thead>
<tr>
<th>QoS</th>
<th>Stateful</th>
<th>Stateless</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tenet [Ferrari &amp; Verma '89]</td>
<td>Diffserv</td>
</tr>
<tr>
<td></td>
<td>IntServ [Clark et al '89]</td>
<td>[Clark &amp; Wroclawski '97]</td>
</tr>
<tr>
<td></td>
<td>ATM [late '80s]</td>
<td>[Nichols et al '97]</td>
</tr>
</tbody>
</table>

Network support for congestion control
- Round Robin [Nagle '85]
- Fair Queueing [Demers et al '89]
- Flow Random Early Drop (FRED) [Lin & Morris '97]
- DECbit [Ramakrishnan & Jain '88]
- Random Early Detection (RED) [Floyd & Jacobson '93]
- BLUE [Feng et al '99]

Question

- Can we achieve the best of two worlds, i.e., provide services implemented by stateful networks while maintaining advantages of stateless architectures?

Scalable Core (SCORE)

- A trusted and contiguous region of network in which
  - edge nodes - perform per flow management
  - core nodes - do not perform per flow management

The Approach

1. Define a reference stateful network that implements the desired service
2. Emulate the functionality of the reference network in a SCORE network
The Idea
- Instead of having core routers maintaining per-flow state, have packets carry per-flow state.

The Technique: Dynamic Packet State (DPS)
- Ingress node: compute and insert flow state in packet's header.
- Core node: process packet based on state it carries and node's state.
  - update both packet and node's state.
The Technique: Dynamic Packet State (DPS)
- Egress node: remove state from packet's header

Why Guaranteed Service Example?
- Illustrate power and flexibility of our solution
  - Guaranteed service - strongest semantic service proposed in context of stateful networks

Example: Guaranteed Services
- Goal: provide per-flow delay and bandwidth guarantees
- How: emulate ideal model in which each flow traverses dedicated links of capacity \( r \)
- Per-hop packet service time = \( \frac{\text{packet length}}{r} \)
Guaranteed Services

- Use DPS to eliminate per-flow state in core
  - control path: emulate per-flow admission control
  - data path: emulate Jitter-VC by Core-Jitter Virtual Clock (CJVC)

Data Path

- Ideal Model
- Stateful solution: Jitter Virtual Clock
- Stateless solution: Core-Jitter Virtual Clock

Ideal Model: Example

- With each packet associate:
  - eligible time – start time of serving packet in ideal model
  - deadline – finish time of serving packet in ideal model

Stateful Solution: Jitter Virtual Clock (Jitter-VC)
Jitter-VC
- Algorithm: schedule eligible packets in increasing order of their deadlines
- Property: guarantees that all packets meet their deadlines

Jitter-VC: Eligible Time Computation
- Minimum between
  - arrival time
  - deadline at previous node + propagation delay
  - deadline of previous packet

eligible time = arrival time

eligible time = packet deadline at previous node

Jitter-VC: Eligible Time Computation
- Minimum between
  - arrival time
  - deadline at previous node + propagation delay
  - deadline of previous packet

eligible time = arrival time

eligible time = packet deadline at previous node

Stateless Solution: Core-Jitter Virtual Clock (CJVC)
- Goal: eliminate per-flow state
  - eliminate dependency on previous packet deadline

using previous packet's deadline ➔ per flow state
Core-Jitter Virtual Clock (CJVC)
- Solution: make eligible time greater or equal to previous packet deadline
  - How: associate to each packet a slack variable $s$
- Delay eligible time at each node by $s$

Theorem: CJVC and Jitter-VC provide the same end-to-end delay bounds
- $s$ can be computed at ingress: depends on
  - current and previous packet eligible times ($e$ and $e_p$)
  - current and previous packet lengths ($l$ and $l_p$)
  - slack variable associated to previous packet ($s_p$)
  - flow reservation ($r$)
  - number of hops ($h$) - computed at admission time

$$s = \max \left( 0, s_p + \frac{l_p - l}{r} + \frac{e_p - e + l_p / r}{h - 1} \right)$$

CJVC Algorithm
- Each packet carries in its header three variable
  - slack variable $s$ (computed and inserted by ingress)
  - flow's reserved rate $r$ (inserted by ingress)
  - ahead of schedule $a$ (inserted by previous node)
- Eligible time = arrival time + $a + s$
- Deadline = eligible time + (packet length) / $r$

NOTE:
- using $a$ instead of the deadline at previous node \(\Rightarrow\) no need for synchronized clocks
**Jitter-VC: Core Router**

- Data path
  - Per-flow classification
  - Per-flow buffer management
  - Per-flow scheduling

- Control path
  - Install and maintain per-flow state for data and control paths

**CJVC: Core Router**

- Data path
  - Per-flow classification
  - Per-flow buffer management
  - Per-flow scheduling

- Control path
  - Install and maintain per-flow state for data and control paths

**Control Path: Admission Control**

- Goal: reserve resources (bandwidth) for each flow along its path
- Approach: light-weight protocol that does not require core nodes to maintain per-flow state

**Per-hop Admission Control**

- A node admits a reservation $r$, if
  - $C$ - output link capacity
  - $R$ - aggregate reservation
    - $R = \sum_{k} r_k 
    - r_k \leq C - R$

- Need: maintain aggregate reservation $R$
- Problem: it requires per flow state to handle partial reservation failures and message loss
Solution
1. Estimate aggregate reservation $R_{est}$
2. Account for approximations and compute an upper bound $R_{bound}$, i.e., $R_{bound} \geq R$
3. Use $R_{bound}$, instead of $R$, to perform admission control, i.e., admit a reservation $r$ if $r \leq C - R_{bound}$

Virtual Length
- Problem: What if flows do not send at their reserved rates?

Estimating Aggregate Reservation ($R_{est}$)
- Observation: If all flows were sending at their reserved rates, computing $R_{est}$ is trivial:
  - just measure the traffic throughput, e.g.,
  
  \[
  R_{est} = \frac{\sum\text{length}(i)}{T}
  \]
  where $S(a, a+T)$ contains all packets of all flows received during $[a, a+T)$

- $r_1 = 2 \text{ Mbps}$
- $r_2 = 3 \text{ Mbps}$
  - $R_{sw} = 5 \text{ Mbps}$

Virtual Length
- Problem: What if flows do not send at their reserved rates?
- Solution: associate to each packet a virtual length such that
  - if lengths of all packets of a flow were equal to their virtual lengths, the flow sends at its reserved rate
- Then, use virtual lengths instead of actual packet lengths to compute $R_{est}$
Virtual Length

- **Definition:**
  \[ \text{virtualLength} = r \times (\text{ct}_{\text{r}} - \text{prev}_{\text{r}}) \]
  - \( r \): flow reserved rate
  - \( \text{ct}_{\text{r}} \): transmission time of current packet
  - \( \text{prev}_{\text{r}} \): transmission time of previous packet

- **Example:** assume a flow with reservation \( r = 1 \text{ Mbps} \) sending 1000 bit packets

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>Length (bytes)</th>
<th>virtualLength (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9</td>
<td>1000</td>
<td>1900</td>
</tr>
<tr>
<td>1.3</td>
<td>1200</td>
<td>1800</td>
</tr>
<tr>
<td>1.2</td>
<td>1300</td>
<td>1560</td>
</tr>
<tr>
<td>1.1</td>
<td>1400</td>
<td>1540</td>
</tr>
</tbody>
</table>

Estimating Aggregate Reservation \((R_{\text{est}})\)

- Use Dynamic Packet State (DPS)
- Ingress node: upon each packet departure computes the virtual length and inserts it in the packet header
- Core node: Estimate \( R_{\text{est}} \) on each output link as

\[
R_{\text{est}} = \frac{\sum \text{virtualLength}(i)}{T}
\]

- where \( \text{virtualLength}(i) \) contains of all packets of all flows received during \([a, a+T]\)

Aggregate Reservation Estimation: Discussion

- The estimation algorithm is robust in presence of control message loss and duplication
  - their effect is "forgotten" after one estimation interval
- If no packet of a flow departs during a predefined interval (i.e., maximum inter-departure time), ingress node generates a dummy packet
- Utilization \( \leq 1 - f \)
  - where \( f \) = \( (\text{max. inter-departure time}) / (\text{estimation int.}) \)
  - e.g.: max. inter-departure time = 5s; estimation int. = 30s \( \Rightarrow \) utilization \( \leq 0.83 \)

Core Router

- **Data path**
  - Per-flow state
  - Per-flow buffer management
  - Per-packet scheduling

- **Control path**
  - Install and maintain per flow state for data and control paths

- Control State
  - Buffer management
  - Scheduler
Core Router

- Data path
  - Per-flow buffer management
  - Per-packet scheduling
- Control path
  - Install and maintain per-flow state for data and control paths
  - No need to maintain consistency of per-flow state

Implementation: State Encoding

- Problem: Where to insert the state?
- Possible solutions:
  - Between link layer and network layer headers
  - As an IP option (IP option 23 allocated by IANA)
  - Find room in IP header

Implementation: State Encoding

- Current solution
  - 4 bits in DS field (belong to former TOS)
  - 13 bits by reusing fragment offset
- Encoding techniques
  - Take advantage of implicit dependencies between state values
  - Temporal multiplexing: use one field to encode two states, if these states do not need to be simultaneously presented in each packet

Implementation

- FreeBSD 2.2.6
- Pentium II 400 MHz
- ZNYX network cards 10/100 Mbps Ethernet
- Fully implements control and data path functionalities
- Management and monitoring infrastructure
Monitoring Infrastructure

- Light weight mechanism that allows continuous monitoring at packet level

Implementation

- Record each packet (28 bytes)
  - IP header and port numbers
  - arrival, departure or drop times
- Use raw IP to send this information to a monitoring site

A Simple Experiment

- Three flows sharing a 10 Mbps link
  - Flow 1: 1 Mbps reservation
  - Flow 2: 3 Mbps reservation with ON/OFF traffic
  - Flow 3: best-effort UDP sending at > 8 Mbps

Aruba (ingress) → Cozumel (core)

Aggregate Reservation Computation

- 0.5 Mbps reservation active during entire interval
- 0.5 Mbps reservation starting at 18 sec: ending at 39 sec

Accept reservation (0.5 Mbps) → Terminate reservation (0.5 Mbps)
Conclusions

- SCORE and DPS bridge the gap between stateless and stateful solutions

Key ideas

- Instead of core routers maintain per-flow state have packets carry this state
- Use state to coordinate edge and core router actions

Conclusions (cont’d)

- SCORE architecture can provide:
  - Service guarantees
  - Network support for congestion control
  - Service differentiation
- DPS compatible with Diffserv: can greatly enhance the functionality while requiring minimal changes