An integrated congestion management architecture for Internet hosts

Hari Balakrishnan¹, Hariharan S. ahul¹, and Srinivasan Seshan²

¹M.I.T. Laboratory for Computer Science
²IBM T. J. Watson Research Center

ACM SIGCOMM, 1999
Outline

• Introduction

• CM algorithms
  – Rate-based control is TCP-friendly
  – Receiver feedback
  – Better than best-effort networks
  – CM Scheduler

• The CM API

• Application performance

• Conclusions
Introduction (1/2)

• Several trends in traffic patterns that threaten the long-term stability.
  – multiple independent concurrent flows by Web application
  – transport protocols and applications that do not adapt to congestion

• Ensure proper congestion behavior and allows applications adapt to network congestion and varying bandwidth.
Fig. 1  New sender architecture with centered around the Congestion Manager.
CM algorithms

- Rate-based AIMD control
- Loss-resilient feedback protocol
- Exponential aging when feedback is infrequent
- Flow segregation to handle non-best-effort networks
- Scheduler to apportion bandwidth between flows
Rate-based control is TCP-friendly (1/2)

- Ensure proper congestion behavior
  - *rate-based* AIMD control scheme
- Rate changes as
  - learns from active flows about the state of the network
  - probes for spare capacity
- AIMD

- Why choose rate-based instead of window-based scheme?
Rate-based control is TCP-friendly (2/2)

- TCP-friendliness relationship:
  \[ \lambda = \frac{K}{\sqrt{p}} \]
  \( \lambda \): throughput, \( p \): loss rate, \( K \): constant depends on the packet size and RTT
  \[ \lambda \propto n_r \]
  \[ p \propto n_d/n \]

  \[ n_r^2 = K^2(n_r/n_d) + K^2 \]

Fig.2 CM’s rate control is TCP-friendly.
Receiver feedback (1/3)

• Why do we need feedback?
  – to be communicated to the sender
• Implicit: hints from application
• Explicit: CM probes
  – Probe every half RTT
  – tracks of the number of packets sent per flow, loss rate, and updates RTT estimate.
Receiver feedback (2/3)

Sending a probe to the receiver

```plaintext
message = <probe, probeseqnum>;
send(message);
probe(probeseqnum) = {probeseqnum, now, nsent};
nsent = 0;
probeseqnum = probeseqnum+1;
```

Responding to probe number thisprobe

```plaintext
message <response, thisprobe, lastprobe, nrecd>;
send(message);
lastprobe = thisprobe;
nrecd = 0;
```
Receiver feedback (3/3)

Sender action on receiving a response

<response, thisprobe, lastprobe, nrecd>

nsent = 0;
for(i=lastprobe+1; i<=thisprobe; i++) do
    nsent += probe(i).nsent;
end;
lossprob = nrecd/nsent;
Delete all entries in probe less than thisprobe;
Handling infrequent feedback

• During times of congestion, probe messages or responses are lost

• **Exponential aging**: reduce rate by half, every *silent* RTT
  – Continues transmissions at safe rate without clamping apps
Better than best-effort networks

• Future networks will not treat all flows equally
  – differentiated services, prioritization based on flow identifiers, etc

• Solution: flow segregation
  – If an application knows beforehand, it can inform the CM
  – the CM incorporates a segregation algorithm
  – based on per-flow loss rates and bandwidths
CM Scheduler

• Using Hierarchical Round Robin (HRR) scheduler for rate allocation

• Uses **receiver hints** to apportion bandwidth between flows

• Exploring other scheduling algorithms for **delay management** as well
  – currently implemented only bandwidth allocation
The CM API

• Goal: To enable easy application adaptation

• Guiding principles:
  – Put the application in control
  – Accommodate application heterogeneity
  – Learn from the application
Put the application in control

- Application decides *what* to transmit
- CM does not buffer any data
  - allows applications the opportunity to adapt to unexpected network changes
- Request/callback/notify API
  - `cm_request(nsend);`
  - `app_notify(can_send);`
  - `cm_notify(nsent);`
- learn about available bandwidth and the RTT
  - `cm_query(&rate, &srtt);`
Accommodate application heterogeneity

• API should not force particular application style

• Asynchronous transmitters
  – triggered by events (ex. file reads) rather than periodic clocks
  – request/callback/notify works well

• Synchronous transmitters
  – Maintain internal timer for transmissions
  – Need rate change triggers from CM
    \[ \text{change\_rate(newrate)}; \]
Learn from the application

• \texttt{cm\_notify(\textit{nsent})}: upon each transmission

• \texttt{cm\_update(\textit{nrecd, duration, loss\_occurred, rtt})}
  – hint to internally update CM sustainable sending rate and RTT estimates.

• \texttt{cm\_close()}: a flow is terminated and allows the CM to destroy the internal state associated with it.
Application Performance (1/4)

- Application 1: Web/TCP
- Web server uses `change_rate()` to pick convenient source encoding

```
1. cm_open
2. cm_request
3. app_notify
4. cm_notify
5. ctm_update
6. cm_close
```

Steps 2, 3, 4 and 5 occur multiple times
Application Performance (2/4)

• **Application 1: Web/TCP**
  - 1Mbps bottleneck link, 120ms propagation delay

Fig. 5 sequence traces for a Web-like workload using 4 concurrent TCP Newreno connections.

Fig. 6 the same workload over TCP/CM.
Application Performance (3/4)

- **Application 2: Layered Streaming Audio**
- The CM enables the audio server to adapt its choice of audio encoding to the congestion state.
  - `cm_open()`
  - `cm_query()`
  - `cm_notify()`
Application Performance (4/4)

- 0.5Mbps bottleneck link, 120ms propagation delay
- choose encodings of 10, 20, 40, 80, 160 and 320 Kbps.
- transmissions of 1KB packets.

Fig. 7  Performance of an adaptive audio application
Conclusions

• CM ensures proper and stable congestion behavior
  – CM tells flows their rates

• Simple, yet powerful API to enable application adaptation
  – Application is in control of what to send

• Improves performance consistency and predictability for individual applications