The Congestion Manager

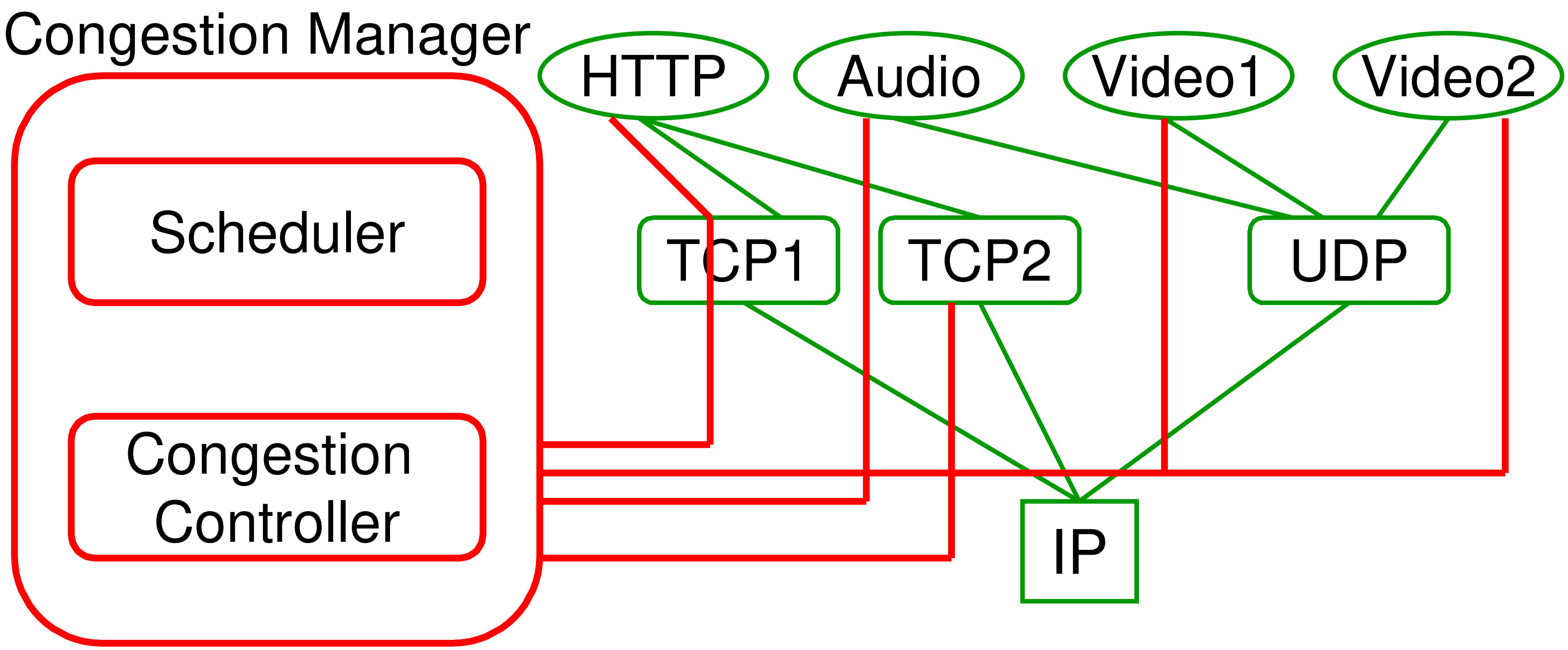
draft-balakrishnan-cm-01.txt

Hari Balakrishnan — MIT LCS

Srinivasan Seshan — IBM Watson

http://wind.lcs.mit.edu/projects/cm/

Architecture



- An integrated approach to end-system congestion management at sender
- Sender-only changes: “well-behaved apps”
  - App-level receiver feedback

Architectural Components

- Shared repository for congestion information across concurrent flows
  - cwnd (or rate), srtt, rttvar, loss rate, etc.
- Congestion controller implements TCP-friendly algorithm (e.g., TCP-style AIMD)
- Scheduler apportions bandwidth amongst flows that share congestion information
- API to enable applications to learn about and adapt to congestion
  - Accommodate ALF-based applications

Sharing

- A *macroflow* is a group of flows between two hosts that share congestion information and behavior (congestion controller and scheduler)
- What should granularity of macroflow be?
  - API allows apps to group their streams into macroflows, and ungroup a macroflow
  - Default macroflow granularity = ?

Macroflow API

- cm_getmacroflow(i32 cm_flowid) returns mflow identifier
- cm_setmacroflow(i32 cm_macroflowid, i32 cm_flowid) sets macroflow for flow
  - If macroflowid is -1, new mflow created
  - Iteration over flows allows grouping
  - Each call overrides previous mflow association

Macroflow Granularity

- In the last IETF, there was a feeling that *no* sharing should occur by default
- This choice is problematic because:
  - There is no ability to share across applications
  - And only a little incentive to do so within the same application
  - So…
    - Optimize the common case!
    - A mistake in sharing isn't fatal, it only slows a flow down, IF the congestion control is designed correctly

CM API

- State maintenance
- Data transmission
  - Buffered style
  - Callback-based
  - Synchronous-style
- Application notification
- Sharing granularity

State Maintenance API

- i32 cm_open(u32 dst)
- cm_close(i32 flowid)
- cm_mtu(i32 cm_flowid)
  - Statically configured
  - Path MTU discovery

Data Transmission API

- Buffered transmission
  - cm_send(i32 flowid, (u8 *) data, u32 len)
- Callback-based transmission
  - Decide *what* to transmit as late as possible
  - cm_request(i32 flowid)
  - cmapp_send() callback allows up to PMTU bytes to be sent
  - ALF-based applications use this
  - TCP too

Data Transmission (cont.)

- Synchronous-style
  - Use internal timer for transmissions
  - cmapp_send() callbacks wreck this structure
  - So, use cmapp_update(u64 newrate, u32 srtt, u32 rttdev) callback
  - Granularity of this callback can be registered with CM: cm_thresh(down, up)
- cm_query(u32 flowid, u64 *rate, u32 *srtt, u32* rttdev) for app to learn stats

Application Notification

- cm_notify(u32 dst, u32 nsent)
  - MUST be called when data is sent
  - E.g., by IP output routine
- cm_update(u32 flowid, u32 nsent, u32 nrecd, u8 lossmode, i32 rtt)
  - Pass feedback from receivers to CM
  - Applications MUST do this
  - Lossmode = persistent, transient, ECN, no loss

CM Congestion Controller

- Determines when and how much a macroflow can transmit
  - Each macroflow MAY use different algorithm
  - MUST be “TCP-friendly”
- Interface
  - query(u64 *rate, u32 *srtt, u32 *rttdev)
  - notify(u32 nsent)
  - update(u32 nsent, u32 nrecd, u32 rtt, u8 lossmode)

CM Scheduler

- Determines which flow gets to transmit
  - Each macroflow MAY use different algorithm
- Interface
  - float query_share(u32 flowid)
  - notify(u32 nsent)
  - schedule(u32 nbytes)
    - MUST be called before transmission
    - schedule() MAY call cmapp_send()
- [Can export interface to prioritize flows]

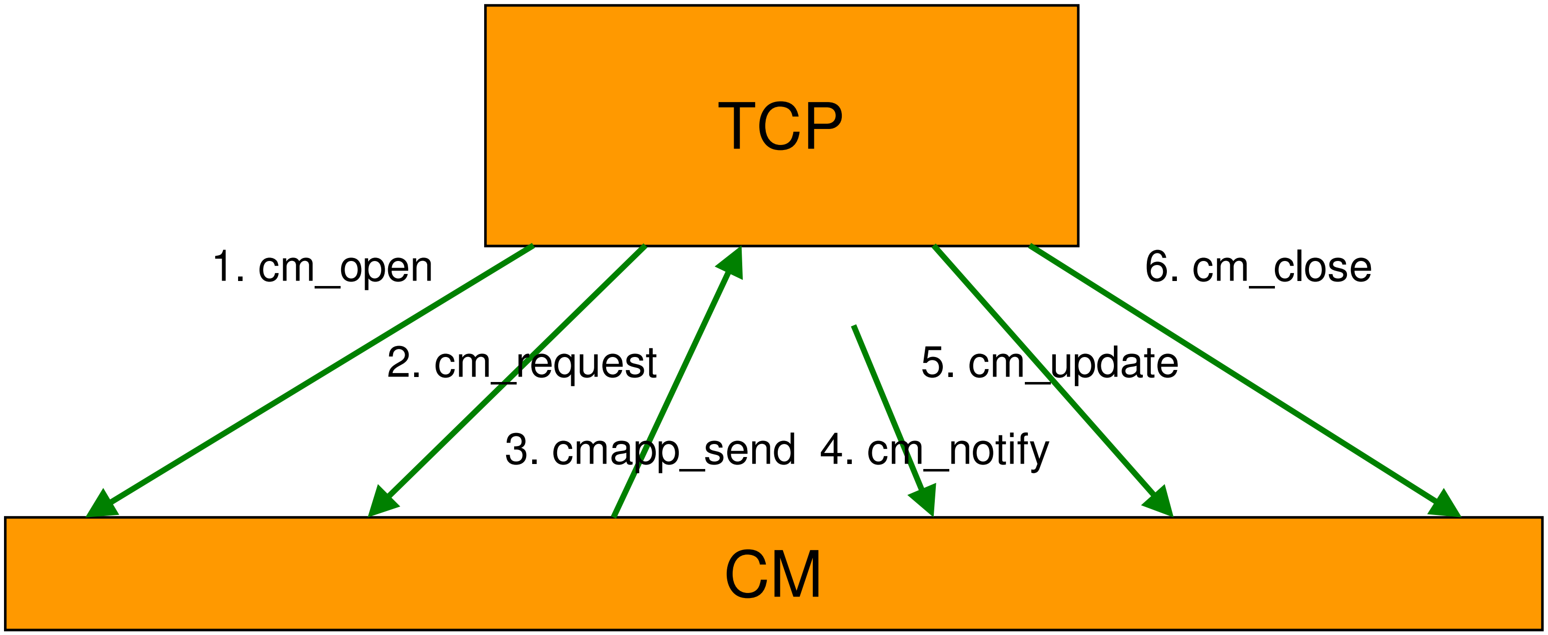
Applications

- TCP/CM
  - Like RFC 2140, TCP Sessions, TCP-Int
- Congestion-controlled UDP
- Conferencing applications
  - Use of simultaneous TCP & UDP flows
- Web server
  - Use of multiple simultaneous TCP flows

How will they use CM?

- TCP/CM
  - Asynchronous API
- Congestion-controlled UDP
  - Buffered API
- Web server + TCP
  - Query + Asynchronous API
- Conferencing applications
  - Synchronous API for real-time streams
  - Combination of others for other streams

TCP/CM



- Implements cmapp_send() callback
- Has per-connection tcp_ownd value
- Keeps track of avg_seg as tcp_ownd/seg_cnt

TCP Output Changes

- Remove all cwnd checks & calculations
- When app data are available, perform all standard checks
  - Flow control, Nagle, etc.
  - If checks pass, enqueue and cm_request()
- On loss, enqueue rxmission and call cm_request()
- cmapp_send() is simple
  - prioritize retransmissions over normal data
  - Only one segment per call is sent

TCP Input Changes

- RTT & RTO algorithms as usual
- Remove all cwnd and ssthresh calculations
- When ACK *a* arrives, calculate inflight = snd_max - *a*
  - cm_update(fid, tcp_ownd - in_flight, 0, CM_NOLOSS, rtt): new ACK without ECN
  - cm_update(fid, tcp_ownd - in_flight, 0, CM_ECN, rtt): with ECN
  - cm_update(fid, 4*avg_seg_size, 3*avg_seg_size, CM_TRANSIENT, rtt): 3rd dup ACK
  - cm_update(fid, avg_seg_size, avg_seg_size, CM_NOLOSS, rtt)
- RTO: cm_update(…, CM_PERSISTENT, ...)

Example Components

- Draft has examples of a congestion controller and a simple scheduler
- AIMD_CC faithfully mimics TCP
- RR_sched is a round-robin flow scheduler

Security Issues

- Incorrect reports of losses or congestion; absence of reports when there's congestion
- Malicious application can wreck other flows in macroflow
- These are all examples of "NOT-well-behaved applications"

Status & Summary

- Implementation being evaluated
  - Single-connection TCP performance similar to without CM (preliminary)
  - Efficient even at hundreds of Mbps
  - Callback-based structure doesn't have to be inefficient
    - Precise timing not crucial
- Separate architecture and API docs?
- On-going work on scheduler, apps, and other control algorithms

Future Evolution

- Support for non-well behaved applications
  - Likely use of separate headers
- Support for DiffServ, IntServ
  - Delay and loss based segregation
- Aging of congestion information for idle periods
- Better sharing of congestion info
  - Within cluster and across macroflows

Not ready for IETF yet!