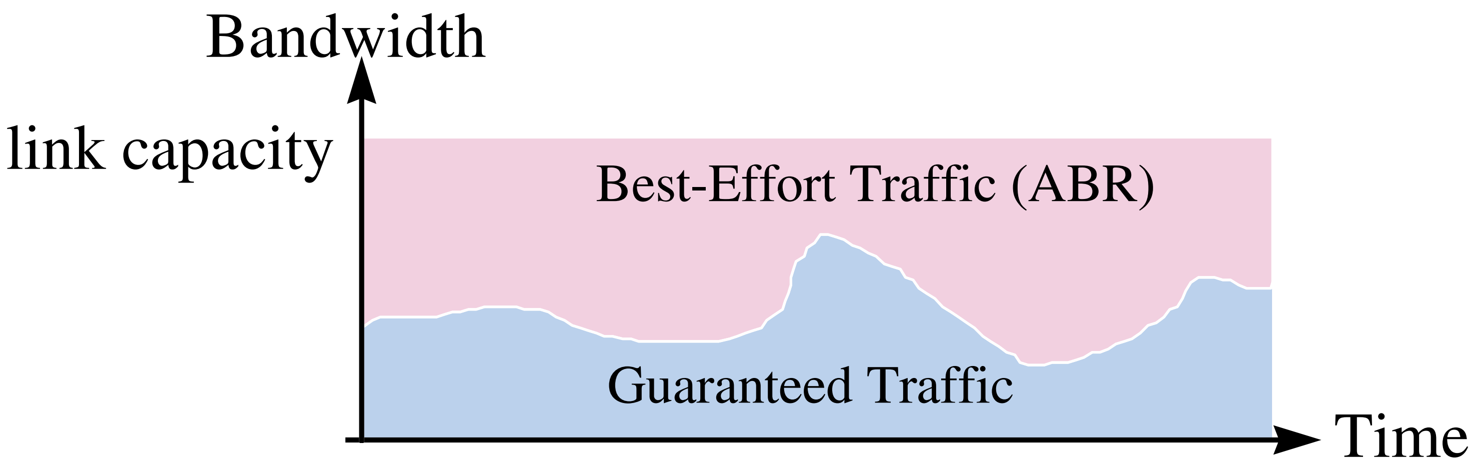
Queue Management for Explicit Rate-Based Congestion Control

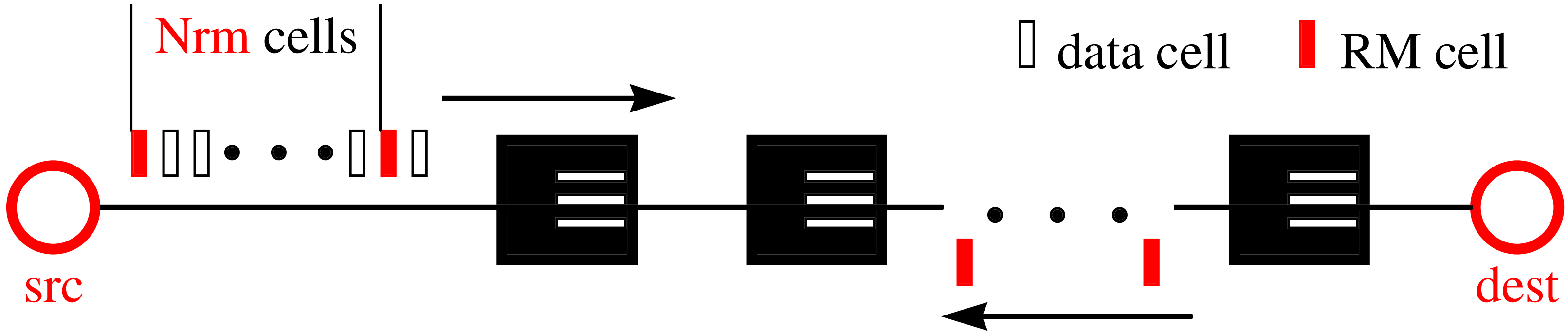
Qingming Ma	K.K. Ramakrishnan
Carnegie Mellon University	AT&T Labs-Research
qma@cs.cmu.edu	kkrama@research.att.com

The Available Bit Rate service



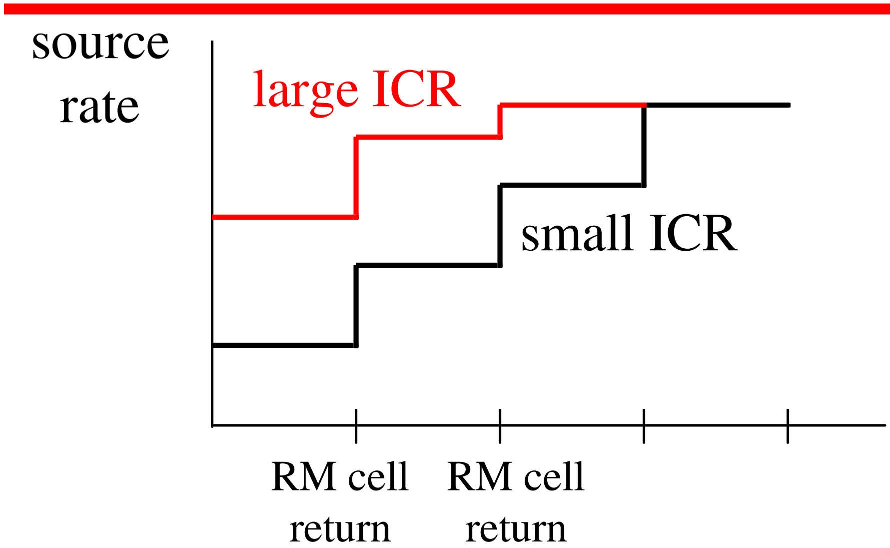
- Goals:
  - opportunistically use available bandwidth
  - achieve low loss rate and maintain small queue
- Our focus: Explicit rate-based congestion control
  - ensures feasibility
  - achieves max-min share allocation for ABR sessions

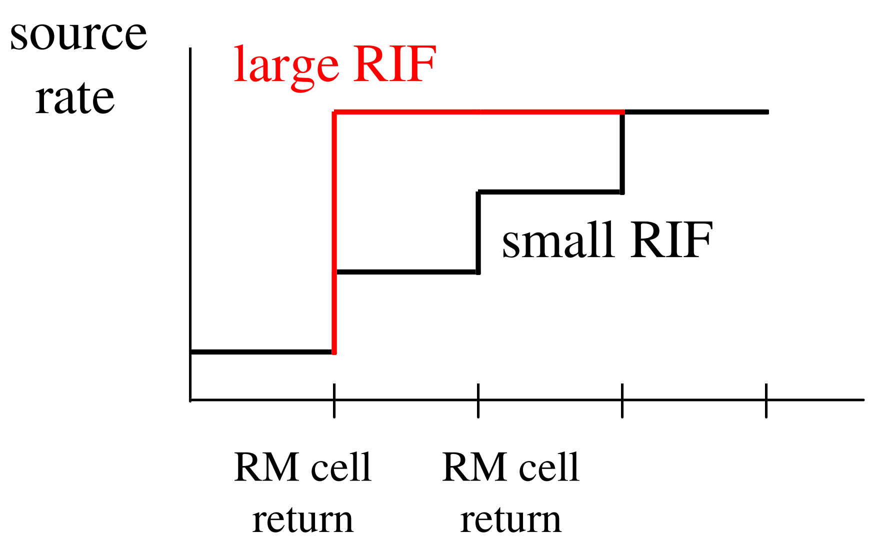
Explicit rate control scheme



- Source sends Resource Management (RM) cell every Nrm cells --- Nrm may be 16, 32, …
- RM cell carries demand of VC: Explicit Rate (ER)
  - switch may mark down the ER value
- Destination turns around RM cell back to source
- Source adjusts rate based on
  - rate allocated (ER) returned in RM cell

Explicit rate scheme operation: source





- Initial cell rate (ICR): source startup rate
  - large ICR ==> aggressive startup
- Rate Increase Factor (RIF): maximal one-step increase
  - large RIF ==> ramp up to max-min fair rate quickly

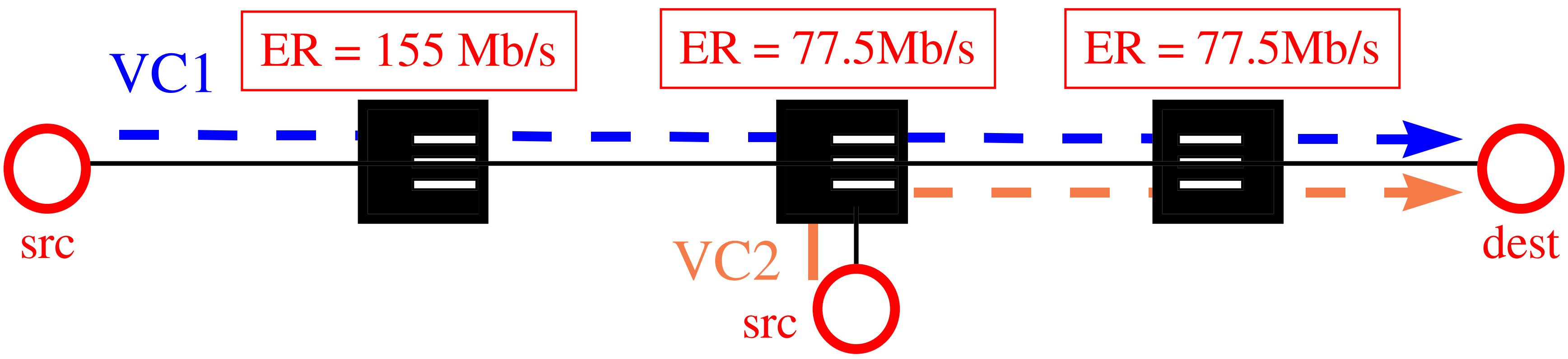
Explicit rate scheme operation: switch

- Explicit rate allocation for max-min fairness

$$A_b = \frac{C - \sum_{i \in S} A_i}{N - \|S\|}$$

C: link ABR capacity
N: total number of ABR VCs
S: the set of satisfied ABR VCs
A_b: rate allocated to bottleneck VC

- Efficient rate allocation mechanism
  - incremental calculation on every RM cell arrival
  - avoid scanning all VCs: O(1) computation



Why do we need queue management

- Large queue builtup
  - feedback delay
  - diverse round trip time of different VCs
  - capacity taken away by high priority traffic
- ABR scheme suggests small ICR & RIF to control queue
  - but queue builds up any way

➨ *A queue management scheme*

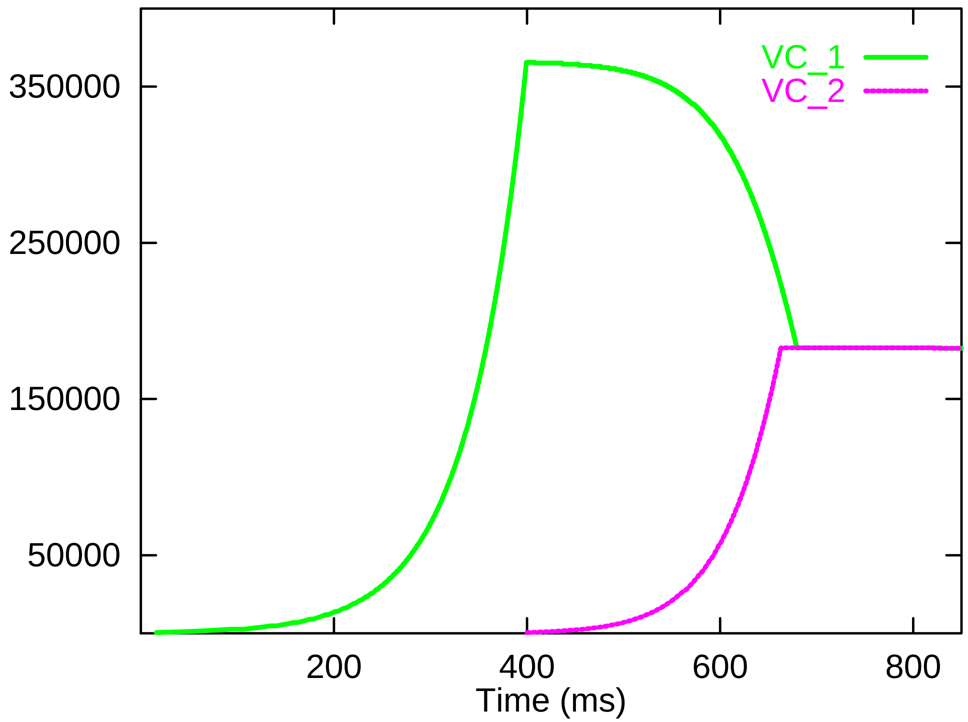
=> reduces available capacity

=> maintains queue around target threshold

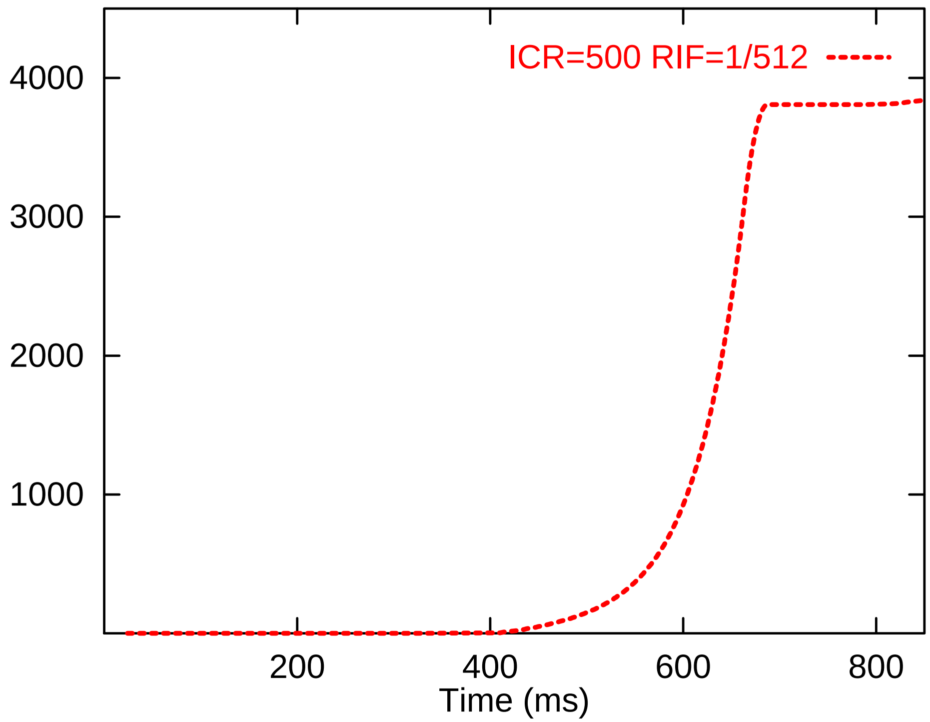
=> eliminates ICR and RIF: aggressive startup and ramp-up

Reasons for queue buildup

source rate (cells/sec)

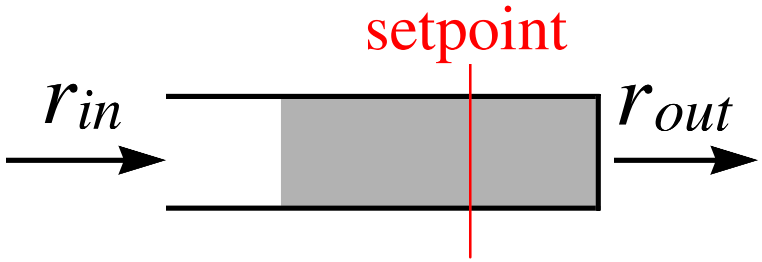


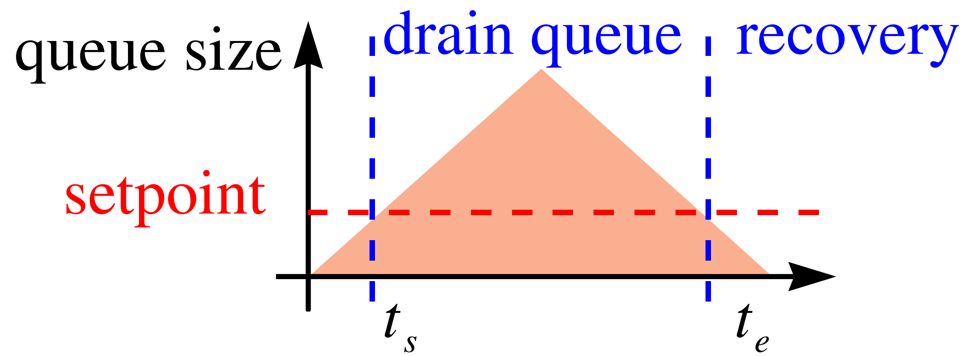
queue size (cells)



- Allocating on RM cell arrival
  - short VC ramp up before long VC reduces
  - small over-allocation accumulates queue unless being drained

Queue management: challenges





$$\text{Qbuiltup}(t_1) = \int_{t_0}^{t_1} (r_{in}(t) - r_{out}(t))dt \qquad \text{Qdrain}(t_e) = \int_{t_s}^{t_e} (C - c(t))dt$$

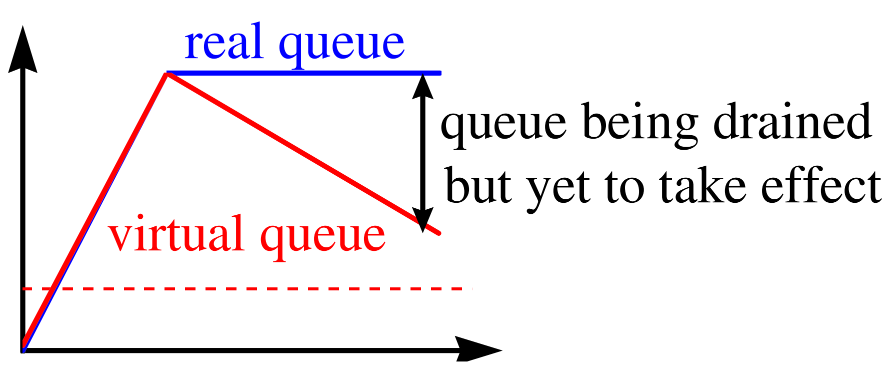
Reduction target: $\text{Qdrain}(t_e) = \text{Qbuiltup}(t_e)$

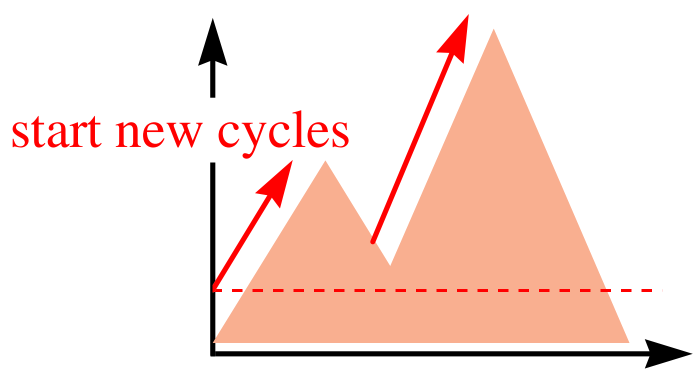
- Reduction based on current queue causes *oscillation*
  - amount to drain is not the queue we see
  - how much to reduce, and when should reduction stop?
- Feedback delay may cause over-drain: *underutilization*

➨ Virtual queue, smooth reduction, and slow recovery

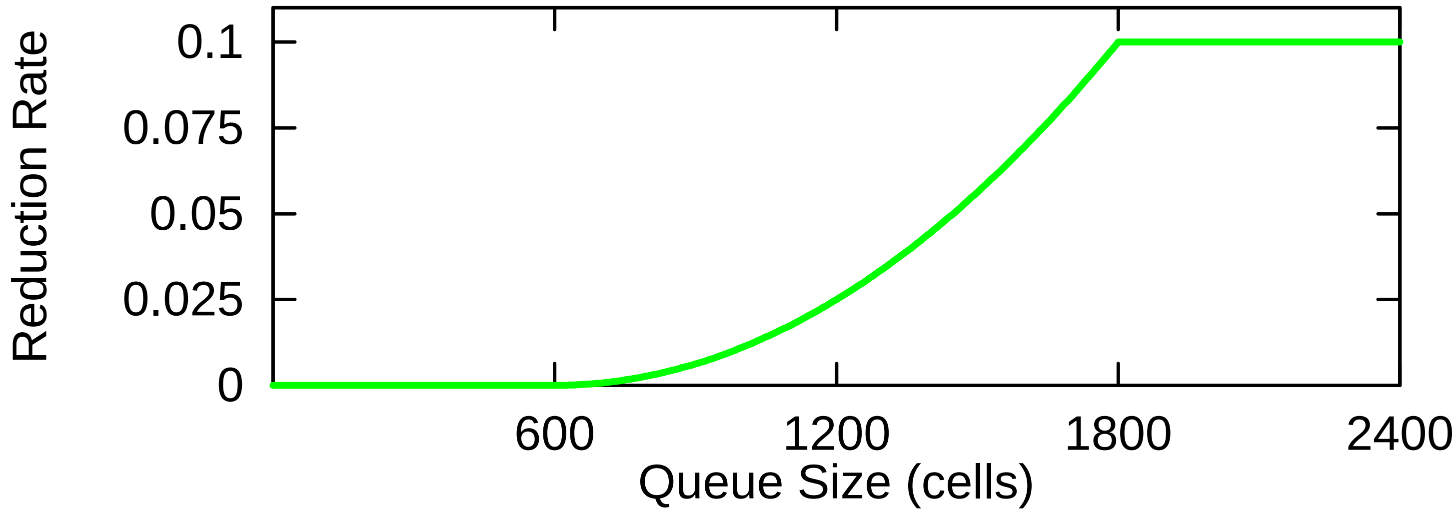
Feedback delay: virtual queue

- Track the size of queue that has to be drained
  - but may have not yet taken effect
- Virtual queue = max_queue - queue_reduced
  - track queue reduced for each VC
  - reduction_rate * interval between two RM cells
- Queue reduction is applied over a regeneration cycle
  - when real queue increase, a new cycle starts over again
  - the cycle lasts until queue is below setpoint





Reduction as a function of virtual queue



$$R = a(q - \text{setpoint})^2 + b(x - \text{setpoint}) + c$$

- Select a setpoint
  - avoid underutilization, keep small delay
  - a good heuristic: bandwidth * delay * max_reduction_rate
- Small reduction when close to setpoint
- Conservative ceiling on amount of capacity reduced

Releasing capacity: slow recovery

- Consequence of releasing capacity
  - virtual queue may still be large, the amount reduced is large
  - every VC takes advantage of unrecovered rate of all others
  - oscillation substantial with large number of VCs
  - queue builds up again
- Recover a VC rate gradually: set recovery ceiling
  - currently reduced capacity / # of bottlenecked VCs
  - maximal reduction / # of bottlenecked VCs
  - correct for ABR capacity changes
  - spread recovery over one RTT

Support aggressive startup: two options

- Having a fixed ICR poses difficulties
  - either not the “fastest” startup
  - or higher than its max-min fair rate
- Ideal startup rate: max-min fair rate
  - appropriately fast startup
  - take precisely the fair share of the resource
- Use connection setup information

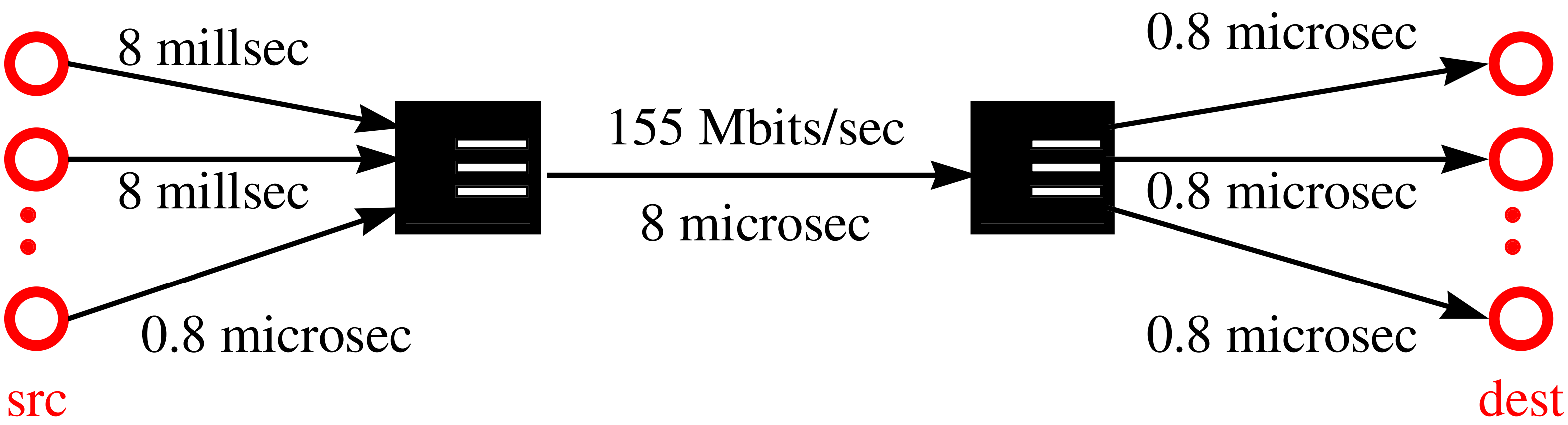
  RATE: switch computes max-min fair rate on connection setup, but allocates only on the first RM cell

  RM: switch also allocates on connection setup

Summary of queue management

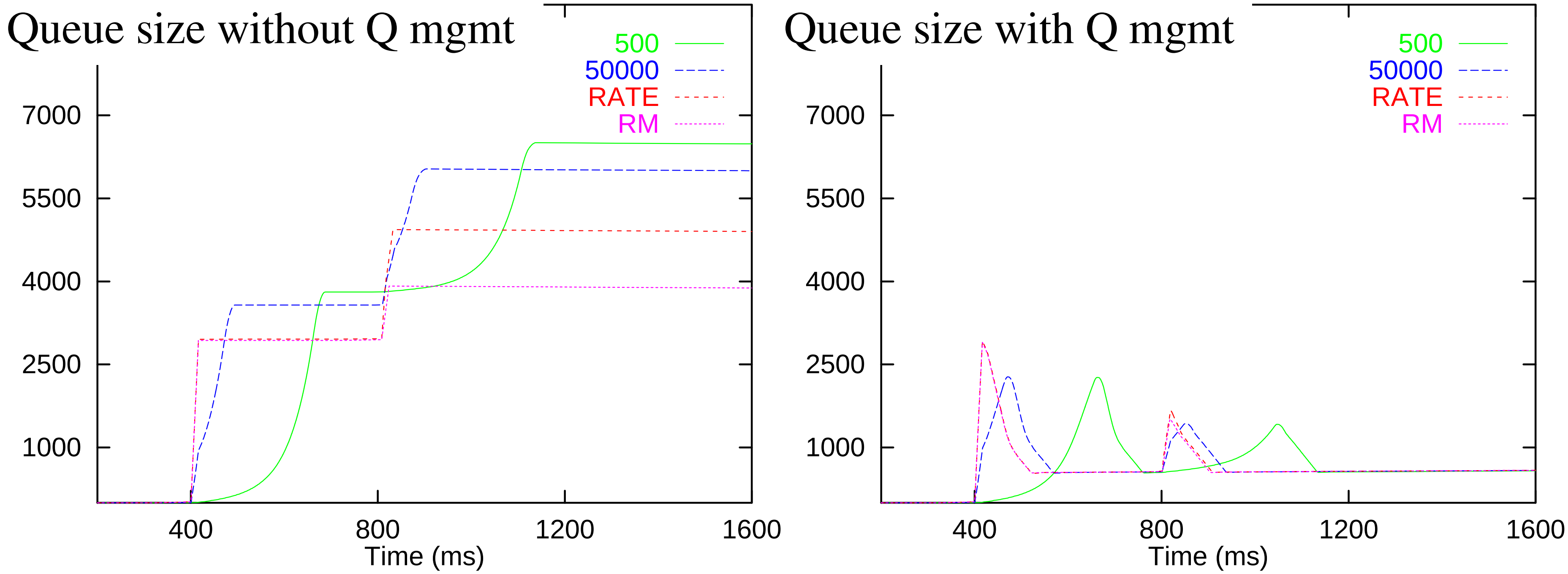
- Reduce output link ABR capacity
  - start when queue size > setpoint
  - pick a setpoint to avoid underutilization
  - distribute reduction over bottleneck VCs: max-min fairness
- Reduction as a function of virtual queue
  - keep track the amount being reduced but yet to take effect
  - smooth reduction to minimize oscillation
  - start a “regeneration” cycle when real queue increases
- Release reduced capacity slowly when queue drained
- Allow aggressive startup and ramp-up
  - set ICR to max-min rate, set RIF to 1

Simulation results



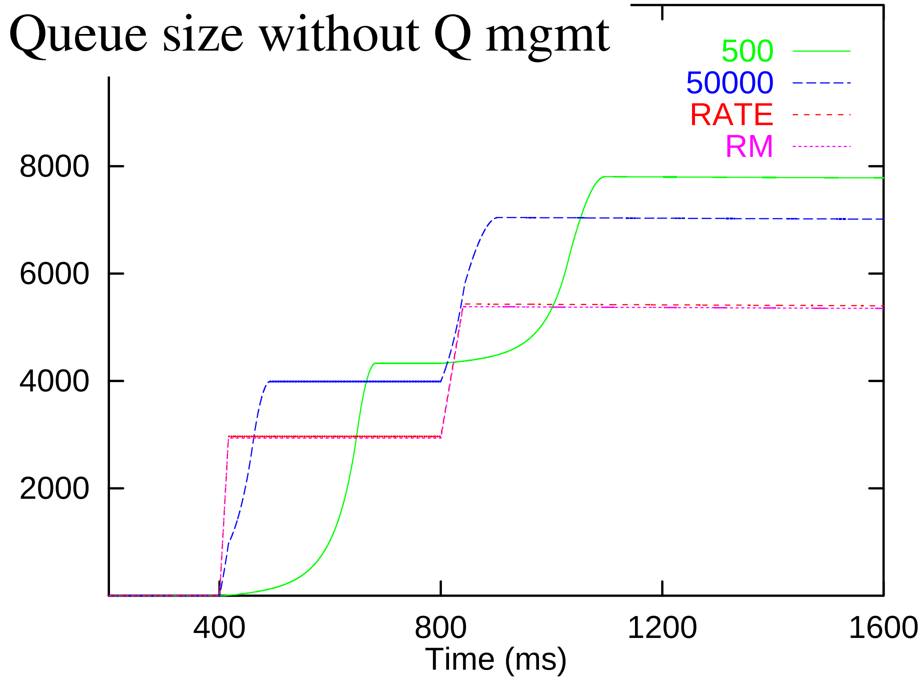
- Parameters:
  - setpoint: 600 cells
  - ICR=500 or 50000 cells/sec
  - RIF = 1 or 1/512
- Workload:
  - persistent cell sources: start up in staggered fashion
  - long VCs vs. short VCs

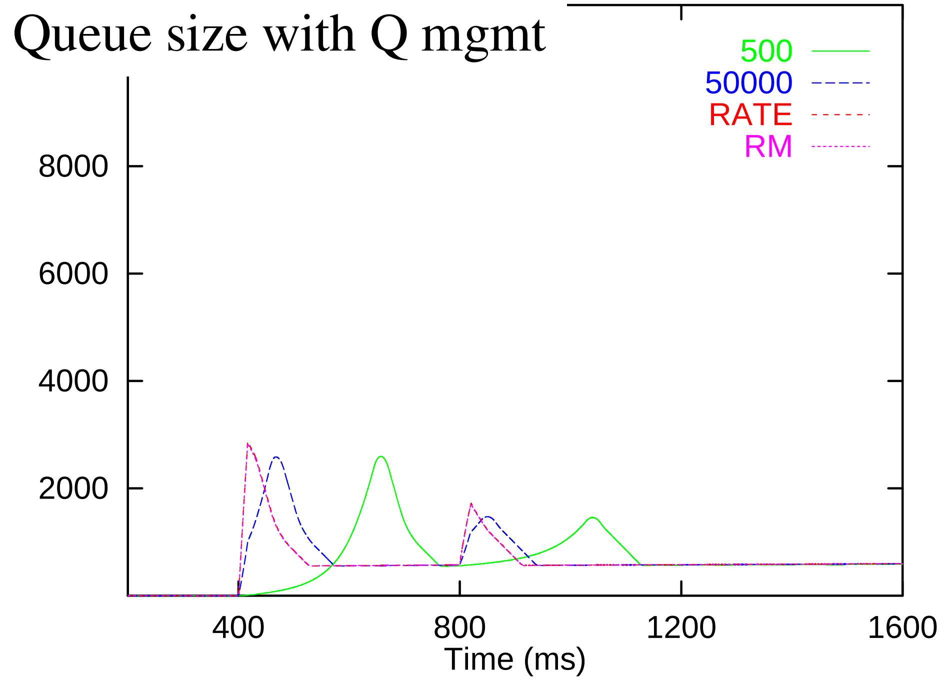
Effects of ICR with FIFO queueing



- Workload: 1 long & 2 short VCs arrive at 0, 400, 800 ms
- Queue builds up regardless the values of ICR used
  - short VCs ramp-up quickly but reallocation to long VC is slow
- Queue management: works well for all ICR options
  - queue builtup lasts for very short time, no steady queue builtup

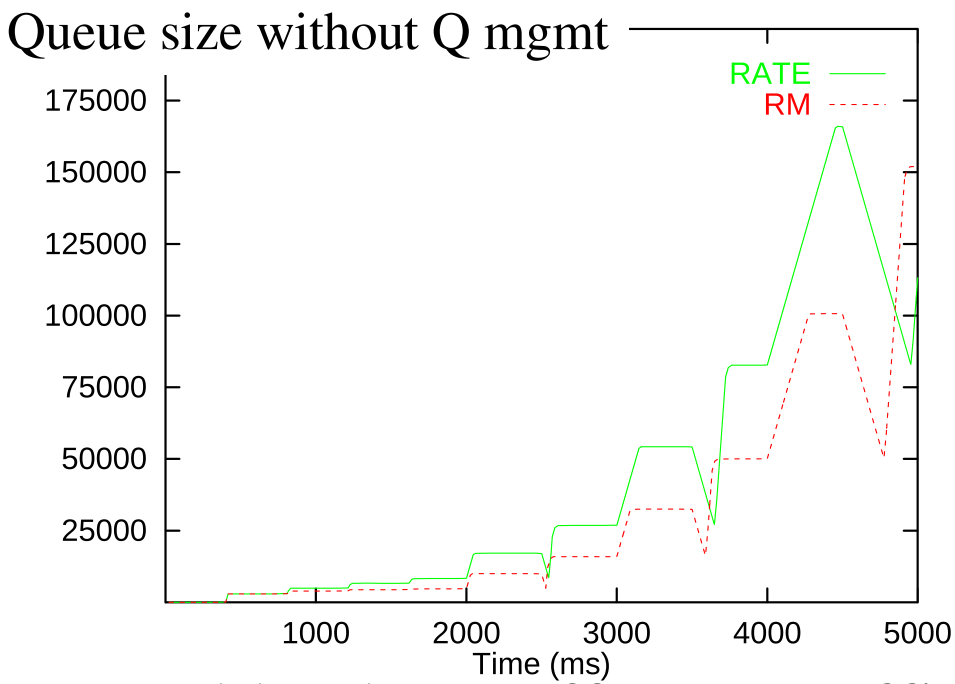
Effects of per-VC queueing

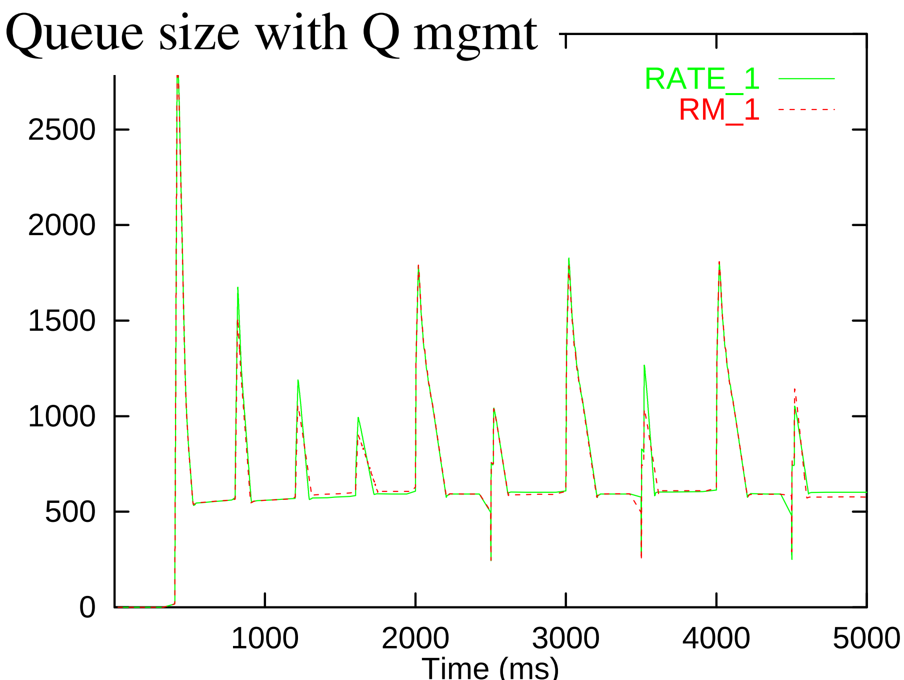




- Traditional perception: isolation => better queue behavior
- A new VC has small queue and ramps up fast
  - results in increased delay for cells already queued for other VCs
- Queue management: works well for per-VC queueing
  - maintains small queue and eliminate undesirable interaction

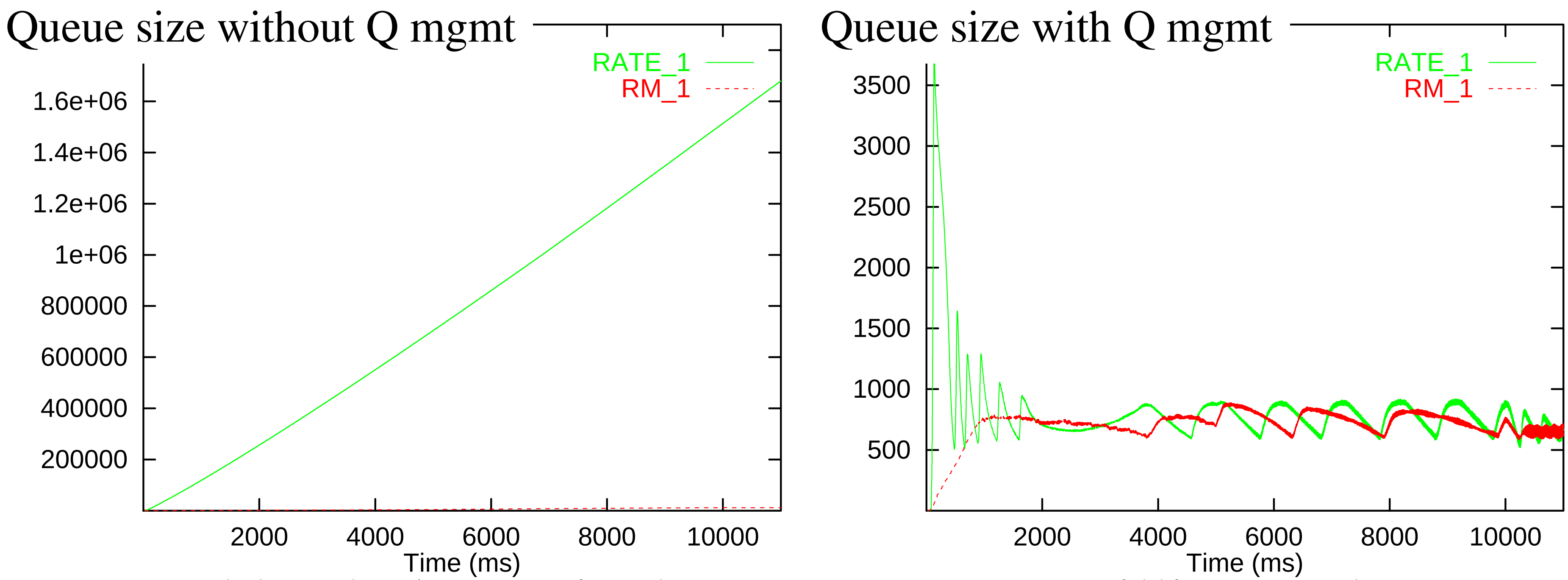
Presence of high priority traffic





- Workload: on-off CBR traffic uses 50% link capacity
- Long VCs respond slowly to reduced ABR capacity
- Queue management:
  - prevents substantial queue builtup when ABR capacity reduced
  - drains any queue that may build up

Scalability: large number of VCs



- Workload: 500 arrivals, one every 20 milliseconds
  - first 250 long VCs, then 250 short VCs
- Queue builtup: 1.6 million for RATE and 12K for RM
  - similar with standard ICR/RIF cases
- Queue management: control queue near the setpoint

Related work

- [Mishra & Kanakia 92]
  - used queue management in hop-by-hop rate control
  - primarily for manage per-VC queue
  - idea of setpoint is similar
  - but queue size is fedback
- [Jain, et al. 95]
  - similar goal
  - used for managing the queue for ABR service
  - capacity reduction based on instantaneous queue
  - reduction applied until the queue is reduced to 0

Conclusions

- Explicit rate schemes need queue management
  - incremental rate allocation can result in queue builtup
  - dynamic changes in capacity requires a way to drain the queue
- Our queue reduction mechanism is based on
  - computing a virtual queue to drive capacity reduction
  - a smooth function for capacity reduction
  - selective reduction of allocation to bottleneck VCs
  - gradual release of capacity when queue drains
- Works well
  - scales to large number of VCs
  - responsive to dynamic changes in capacity
  - allows very aggressive startup