BPF HOST NETWORK RESOURCE MANAGEMENT

Lawrence Brakmo Alexei Starovoitov

Facebook

brakmo@fb.com ast@fb.com

Introduction

- Linux supports allocating and managing many system resources such as CPU and memory.
- Network allocation an management is harder since it is both a local and a global resource.
- Require mechanisms to allocate and manage bandwidth both locally (i.e. per cgroup) and externally (i.e. per link or per switch).
  - Ingress bandwidth management requires notifying senders to slow down.

Traffic control

- Current mechanism, traffic control (tc), allows shaping of outgoing traffic and policing of incoming traffic.
- It has been used for managing external bandwidths (e.g. Google's BwE).
- However, tc has a history of performance issues when using many htb (Hierarchical Queuing Discipline) queues.
- tc bandwidth allocation usually results in standing queues* (other issues with codel).
- Lacks the flexibility usually provided by general programming constructs.

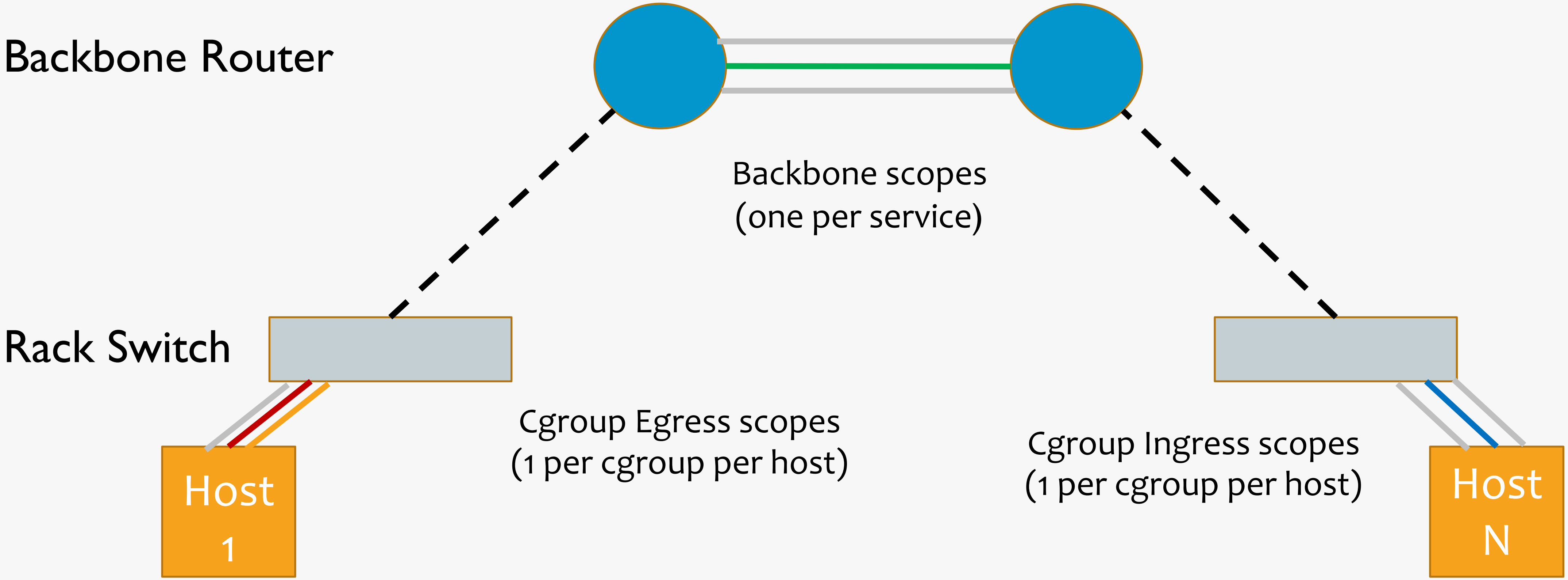
Goals

- a BPF based framework (NRM f) for efficiently supporting shaping of both egress and ingress traffic based on both local and global network allocations.
- Initial assumption that majority of traffic is TCP (or it has similar congestion control).
- Eliminate/reduce standing queues.
- Flexibility (comes for free with BPF).

Overview

- Use existing egress and ingress cgroup skb hooks.
- For egress use ECN, calls to tcp_enter_cwr, or drops.
- For ingress use ECN (or similar) to notify sender to slow down.
- Use scopes to manage bandwidth
  - E.g. cgroup scope, particular link/switch scope, …
  - Each socket belongs to a set of scopes
  - When sending a packet we update the bw utilization of the socket's scopes
  - Congestion is determined by the most congested scope

Network Scopes Example



- Consider 2 flows from hosts 1 to host N belonging to same service
- Flow 1 could belong to scopes Red, Green & Blue
- Flow 2 belongs to scopes Orange, Green & Blue

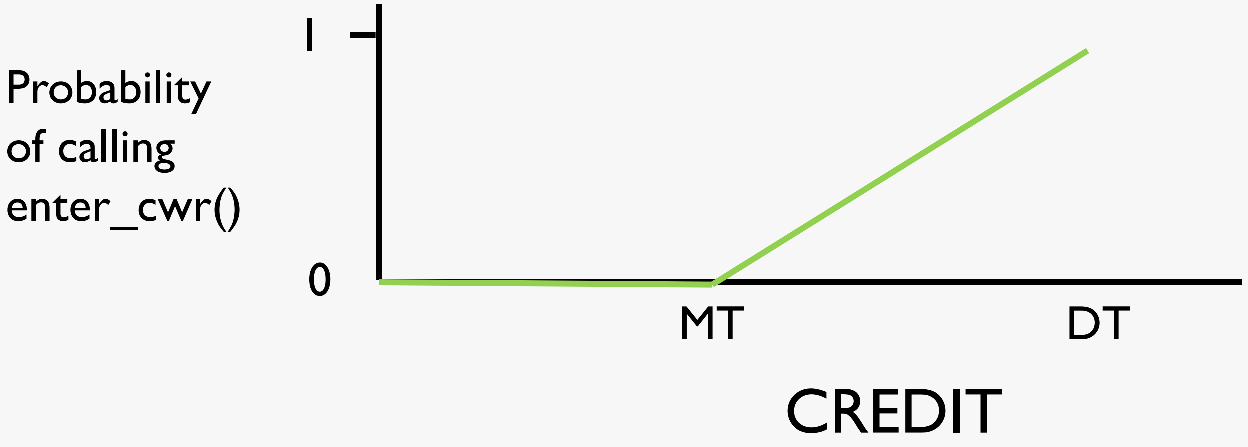
BW management

- We use a virtual queue to track bw use (per scope)
  - `Struct vqueue { int credit;                /* in bytes */`
  - `                  long long lasttime;   /* in ns */`
  - `                };`
- When sending a packet:
  - Credit += credit_per_ns(currtime – lasttime, rate); // need to bound
  - Credit -= wire_length_in_bytes(skb); // need to account for TSO
- Make decision based on credit and packet info
- Have to account for GRO and LRO packets

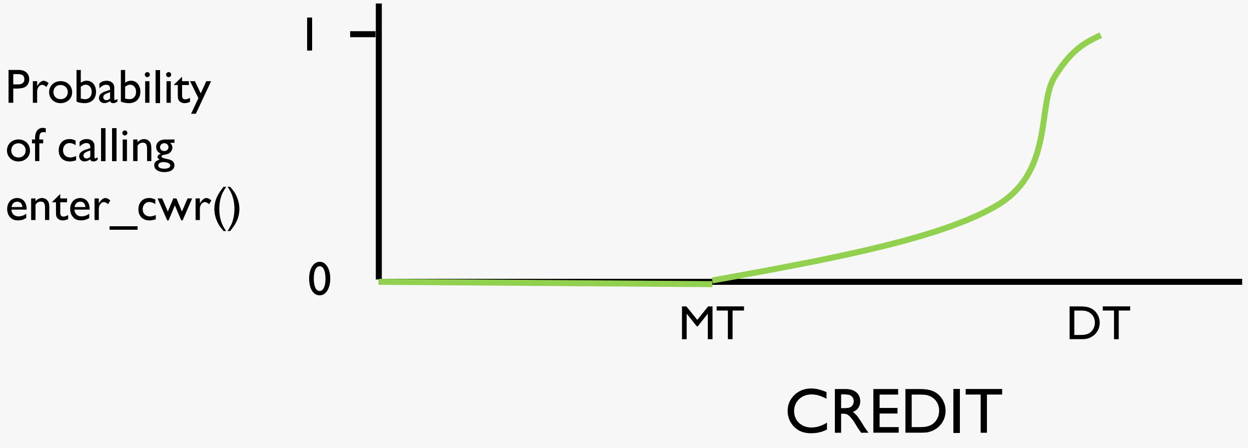
Current Congestion Algorithm

- If credit < Drop Threshold, then drop it (small packet buffer)
- If credit < Mark Threshold, then "mark it"
  - ECN: mark it
  - TCP – non-ECN: call tcp_cwr_enter() with a linear probability. The closer credit is to Drop Threshold, the more likely to call cwr
- Virtual buffer to absorb bursts
- Drop threshold: 600pkts (reserved space for small packets)
  - Mark threshold: *120pkts*

CUBIC MARK FUNCTIONS



Current



Explore other response functions

Issues

- Packets dropped by cgroup skb BPF program do not trigger call to enter_cwr (cwnd reduction).
  - Solution: helper BPF function to call tcp_enter_cwr
  - Advantage: can make better decisions (probabilistic reductions)

- High tail latencies due to dropping packet and no more packets in transit (packets_out = 0).
  - For example, 1Gbps bound and 9 flows within rack => cwnd should be less than one.
  - When no more ACKs to trigger new packets, TCP depends on probe0 timer to resend. Default >= 200ms
  - Solution: reduce probe timer to 20ms in these cases

Issues (2)

- Update of Credit and Lasttime is a critical section
  - Needs to be protected
  - Currently we do not have spinlocks in BPF programs
  - Hack: spinlock the whole BPF program
  - Fix 1: work on bpf_spinlock support is happening in parallel
  - Fix 2: use data structure not requiring locks

experiments

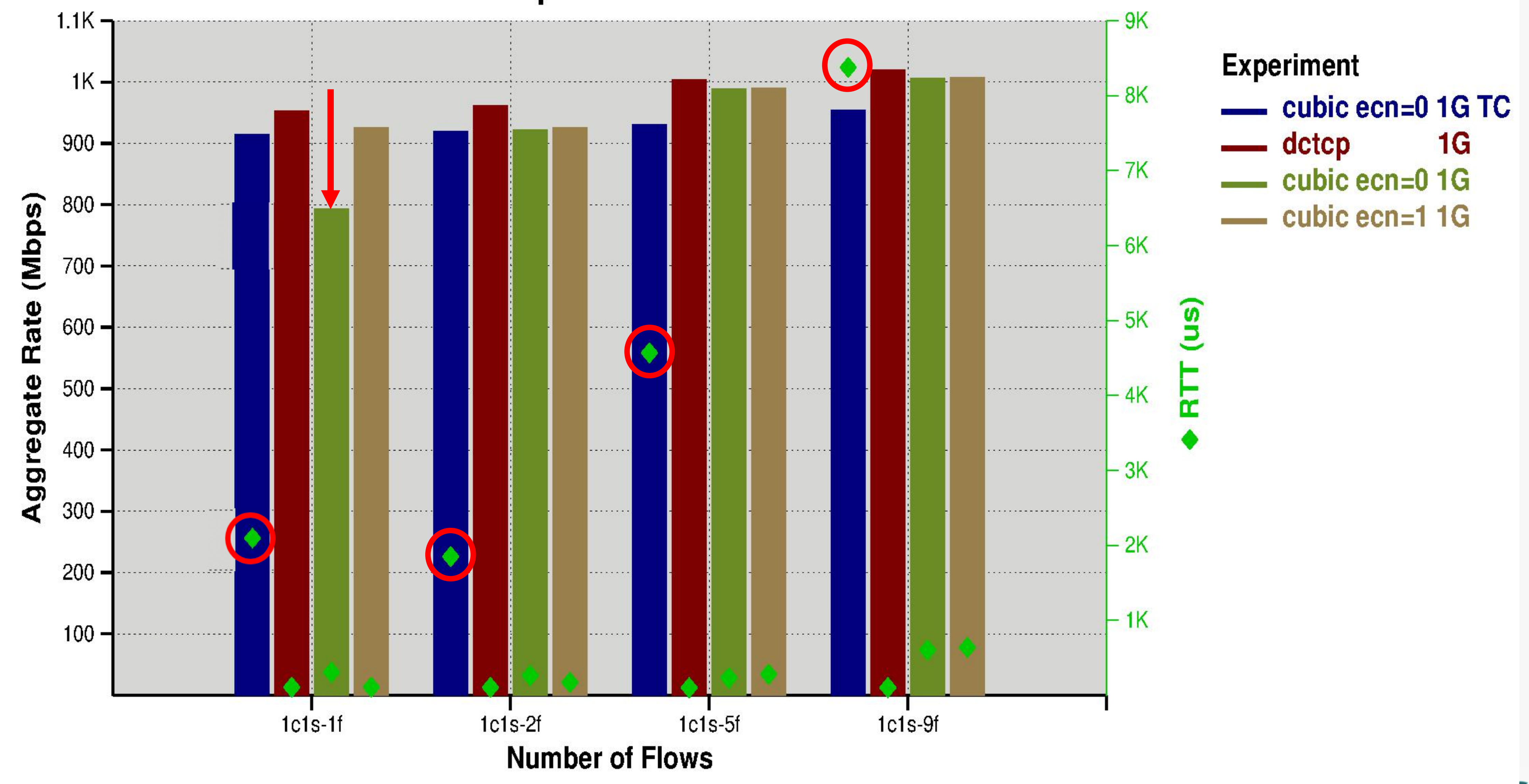
- Only 1 scope (belonging to 1 cgroup)
- One hosts sends to another host in a rack
- One or more 1MB and 10KB RPCs
  - 1 - 1MB
  - 1 - 1MB and 1 - 10KB
  - 4 - 1MB and 1 - 10KB
  - 8 - 1MB and 1 - 10KB
- Limit rate by either by NRM or TC (htb)
- Introduce latency by netem on receiving host

Experiment 1Gbps rate
Test smaller probe timer, 1MB RPC Latency

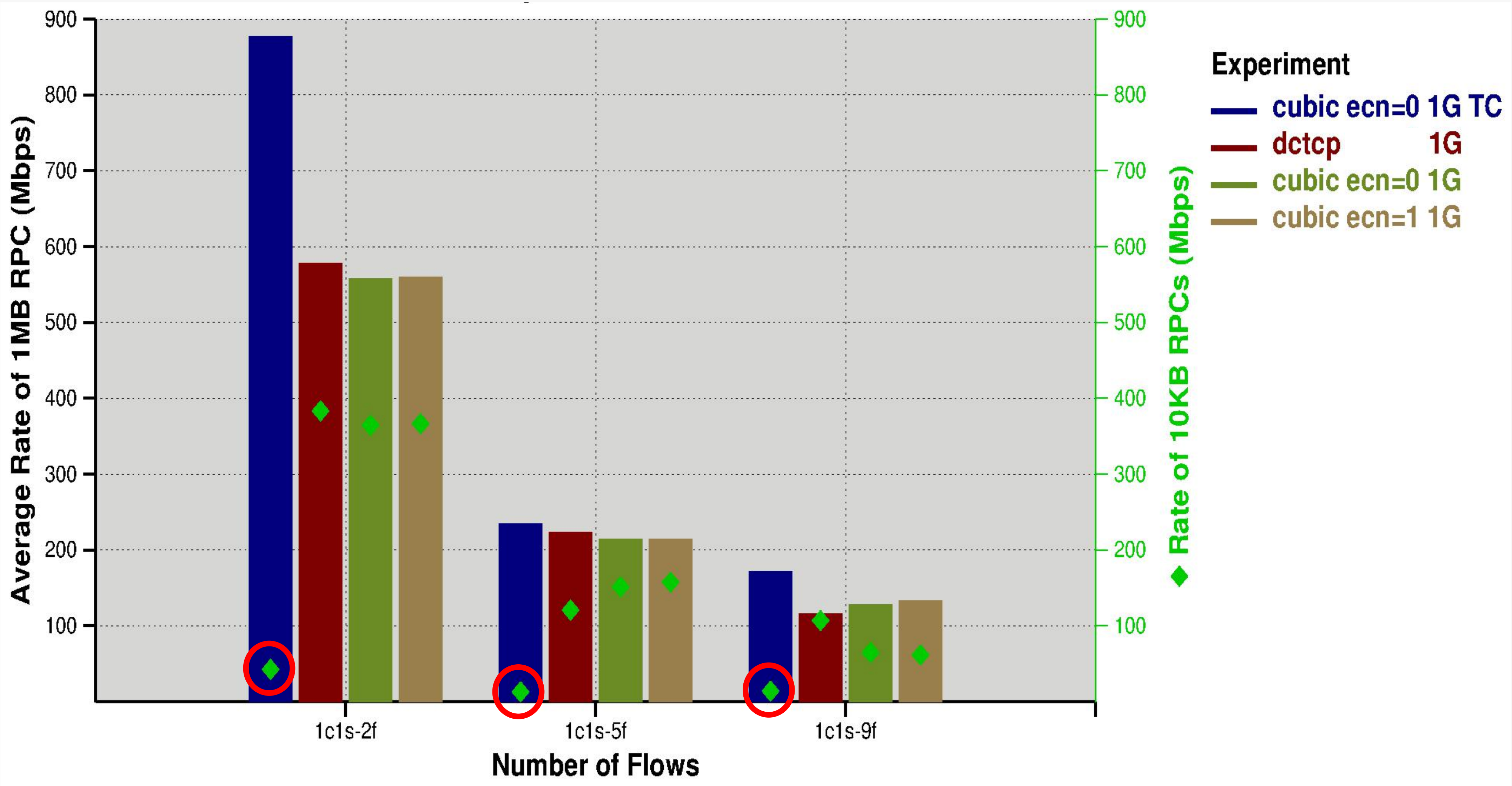
# Flows	Cubic Aggr BW		Cubic 99.9% Lat		DC-TCP Aggr BW		DC-TCP 99.9% Lat	
		< timer		< timer		< timer		< timer
1	496M	794M	250ms	84ms	953M	953M	9ms	9ms
2	856M	922M	260ms	44ms	962M	962M	22ms	21ms
5	935M	989M	465ms	92ms	1003M	1004M	52ms	48ms
9	999M	1006M	600ms	345ms	1029M	1020M	308ms	78ms

- Reducing probe timer to 20ms reduces tail latency significantly!
- From now on assume reduced probe timer

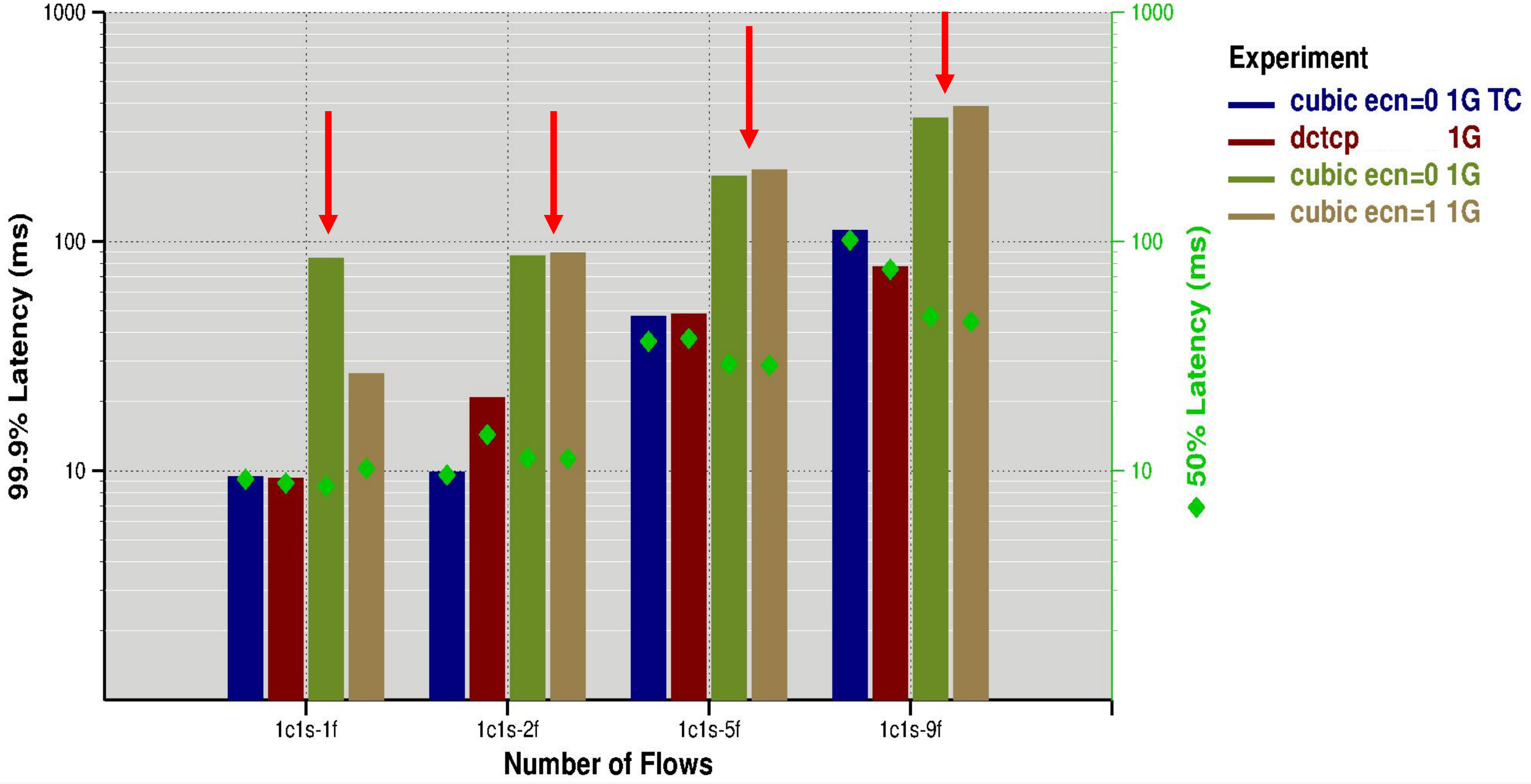
1Gbps Limit: Aggregate Rate and RTT



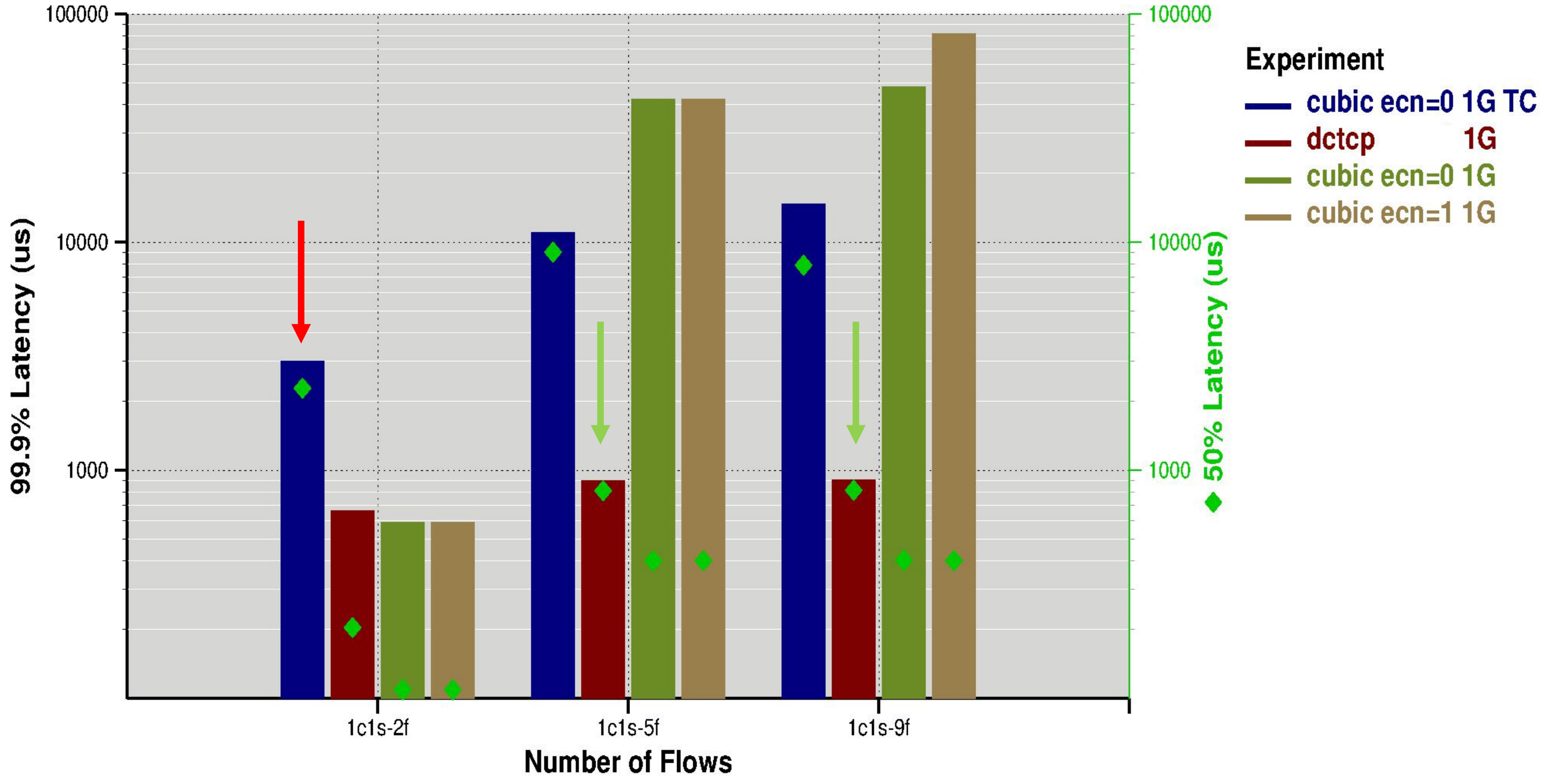
1Gbps Limit: 1MB and 10KB Rates



1Gbps Limit: 1MB RPC Latencies



1Gbps Limit: 10KB RPC Latencies



1 Gbps Limit Conclusions

- Similar aggregate rate (except for 1 flow Cubic)
- High RTTs when using TC (standing queue)
  - Default output_limit_bytes is 260KB, at 1Gbps => 2ms
- TC is size unfair, 10KB RPCs get up to 20x less rate
- Cubic and Cubic-ecn have higher 1MB RPC tail latencies
- DC-TCP has much better 10KB tail latency (10x to 80x lower)

1 Gbps Limit & 10ms

Cong Control	qdisc	Rate Control	1-flow Rate (Mbps)	9-flow Aggr Rate (Mbps)	1-flow 1-MB 99.9% Lat (ms)	9-flow 1-MB 99.9% Lat (ms)
Cubic	HTB – fq	HTB	441	858	58	85
Cubic	HTB	HTB	437	945	20	120
Cubic	mq – fq	NRM-BPF	410	915	46	141
Cubic	mq – fqc	NRM-BPF	754	944	12	218
DC-TCP	mq – fq	NRM-BPF	666	947	13	143

1 Gbps Limit & 10ms RTT Conclusions

- Best rate is achieved with Cubic, mq-fq_codel and NRM-BPF for rate control
  - However, 9-flow tail latency is higher at 218ms
- Using Cubic with HTB for rate control reduces tail latency (85 or 120ms)
  - However, 1-flow rate decreases (to 441 from 760Mbps) and also increases 1-flow tail latency (to 20 or 58ms from 12ms).
- Using DC-TCP with NRM-BPF for rate control produces results between the previous 2: 666Mbps and 13/120ms tail latencies
  - Note: There seems to be an issue with DC-TCP that may increase latencies.

f@fb

5Gbps Limit

Cong Control	qdisc	Rate Control	Aggr Rate (Mbps)		1-MB 99.9% Lat (ms)		10-KB	
			1-flow	9-flow	1-flow	9-flow	Rate (Mbps)	99.9% Lat (ms)
Cubic	HTB	HTB	4.5	4.8	2.0	18	35	3.7
Cubic	mq-fqc	NRM-BPF	4.2	4.7	6.5	113	243	0.8
Cubic-ECN	mq-fqc	NRM-BPF	4.4	4.2	4.7	227	196	1.1
DC-TCP	mq-fqc	NRM-BPF	4.6	4.9	4	27	295	0.8

5Gbps Limit Conclusions

- Cubic with HTB for rate control produce the best 1MB results at the cost of 10KB results
  - 9-flow 1MB: 4.8Gbps and 18ms tail latency
  - 9-flow 10KB: 35Mbps and 3.7ms tail latency
- NRM-BPF produced much better 10KB results, but worst 1MB results
  - Cubic 9-flow 1MB: 4.2Gbps and 113ms tail latency
  - Cubic 9-flow 10KB: 243Mbps, 0.8ms tail latency
  - DC-TCP 9-flow 1MB: 4.6Gbps, 27ms tail latency
  - DC-TCP 9-flow 10KB: 295Mbps, 0.8ms tail latency

f@fb

NRM Ingress

- Similar idea: use a virtual queue to track usage
- Want a mechanism to notify sender
  - Otherwise need to depend on packet drops (bad)
- Options
  - DC-TCP – uses ECN to notify sender
  - Cubic – use a side channel to notify sender
    - Use ECN markings (maybe ECT1 if ECT0 is default as in Linux)
    - BPF program on other side does cwr
- Incast prevention does not drop, only mark so it can use switch buffers

f@fb

DCTCP Ingress

Limit (Mbps)	# Flows	Aggregate Rate (Mbps)	Retrans	99.9% Latency (ms)		50% Latency (ms)	
				1MB RPC	10KB RPC	1MB RPC	10KB RPC
1000	1	925	0	9.5		9.0	
1000	2	922	0	19.0	0.7	13.0	0.2
1000	5	931	0	47.9	0.9	43.0	0.5
1000	9	945	1493	336.0	207.0	54.0	0.8
5000	1	4600	0	4.1		1.7	
5000	2	4600	0	4.7	0.6	1.9	0.2
5000	5	4670	0	12.4	1.0	7.7	0.2
5000	9	4630	0	18.5	0.8	15.5	0.2

Conclusions

- Egress NRM prevents standing queues (i.e. smaller RTT) as long as host average BW utilization is smaller than NIC rate.
  - As a result smaller RPCs have smaller latencies
- Using BPF provides great flexibility and is a great platform for experimentation.

Future work

- Explore different marking algorithms (response functions)
- Explore using connection RTT in marking algorithm
- Test multiple scopes
  - Multiple cgroups (each flow only has one scope)
  - Multiple scopes per flow
- Test concurrent flows with different RTTs
- Test concurrent flows with different TCP variants
- Ingress NRM with sender notifications