AN EVALUATION OF HOST BANDWIDTH MANAGER

Lawrence Brakmo

Facebook

brakmo@fb.com

Linux Plumbers Conference 2019

What is HBM?

- Host Bandwidth Manager is a BPF based framework for managing ingress and egress host network bandwidth
- It uses egress and ingress cgroup skb hooks
- Linux already supports allocating and managing many system resources such as CPU and memory.
- Bandwidth management is harder since it also involves a remote resources
- This is especially true for ingress bandwidth management since it is the remote host, the sender, that needs to change its rate

What is different from previous talk?

- Code changes:
  - BPF spinlocks are implemented (instead of a global lock)
  - No enter_cwr() helper, instead use return value to indicate congestion and call tcp_enter_cwr()
  - No support for reducing probe timer. Instead there is support for reading “tp→packets_out” and BPF program can decide how to handle case when packets_out < 2

What is different from previous talk? (2)

- Evaluation
  - Uses actual patches (instead of experimental code)
  - Tests include using multiple cgroups (i.e. multiple bw limits)
  - Tests for prevention of incast losses
  - Tests include use of fq's EDT

What other options are there?

- Traffic control (tc) allows shaping of outgoing traffic and policing of incoming traffic
  - htb qdisc is commonly used to support multiple egress bandwidths
  - Issues with qdisc root lock and large htb trees
- BPF (other than egress/ingress cgroup skb hook)
  - Google uses TC clsact egress hook and a flat HTB
  - One can use fq with EDT here, but some overhead if not done by the NIC

QUICK OVERVIEW OF HBM

Overview

- Use existing egress and ingress cgroup skb hooks.
- Egress policing/"shaping" is done through
  - ECN
  - return code to trigger TCP's congestion window reduction (CWR)
  - fq's Earliest Departure Time (EDT)
  - packet drops
- Ingress policing/"shaping" is done through
  - ECN
  - Packet drops

Overview (2)

- Policy (algorithm) is implemented in BPF program
- Can use TCP state to improve behavior (such as fairness)
  - Tp->packets_out: At least 2 to prevent delayed-acks
  - Tp->srtt: Improve fairness of short and long RTT flows

BW management

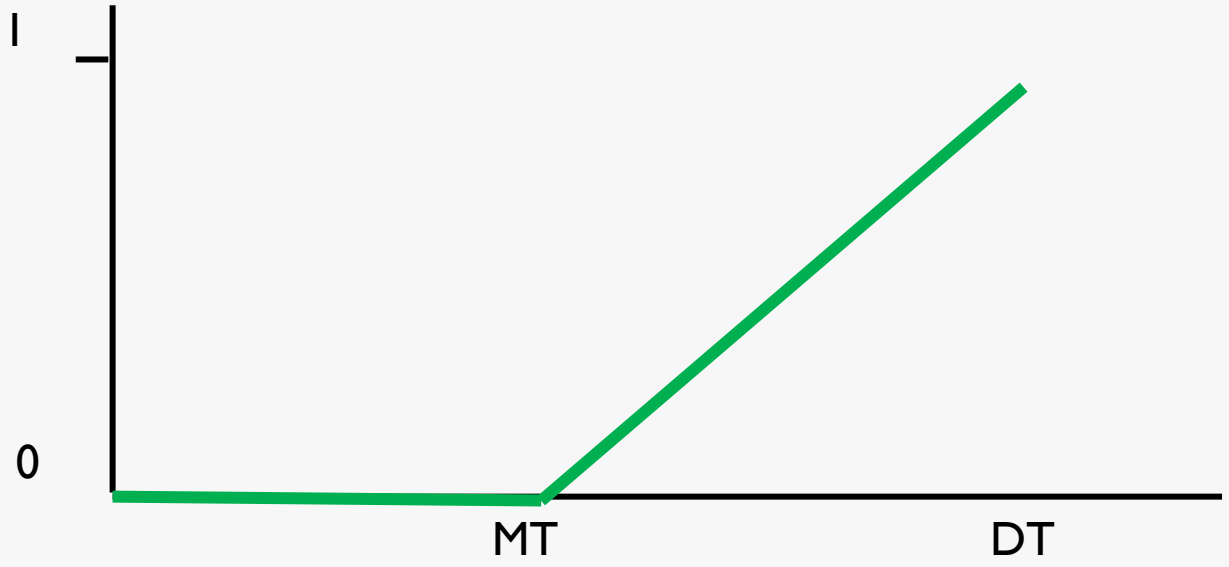
- We use a virtual queue to track bw use (per cgroup)
  - `Struct vqueue { // in cgroup local storage`
  - `struct bpf_spin_lock lock;`
  - `long long lasttime; /* in ns */`
  - `int credit; /* in bytes */`
  - `unsigned int rate; /* in bytes per NS`
  - `<< 20 */ };`
- When sending a packet:
  - Credit += 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

Current Congestion Algorithm

- If credit < Small pkt drop threshold, all packets are dropped (except some cases if unless packets_out < 2)
- If credit < Large pkt drop threshold, drop large packets
- If credit < Mark Threshold, then “mark it”
  - ECN: mark it
  - TCP – non-ECN: return “congestion” with a linear probability. The closer credit is to Drop Threshold, the more likely to return “congestion”

Non-ECN TCP MARK FUNCTIONS

Probability of returning "congestion" so tcp_enter_cwr() is called



- CREDIT

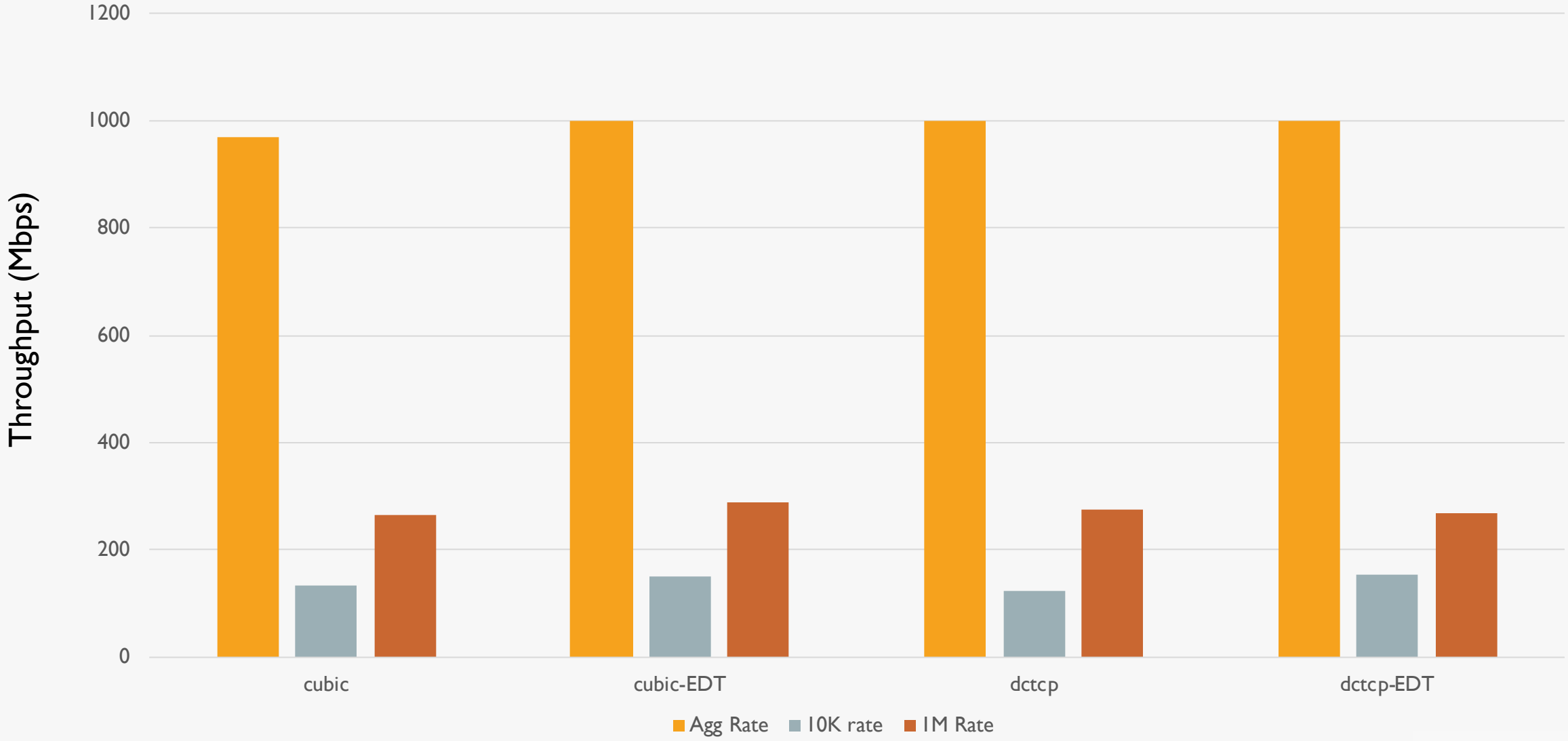
EVALUATION

experiments

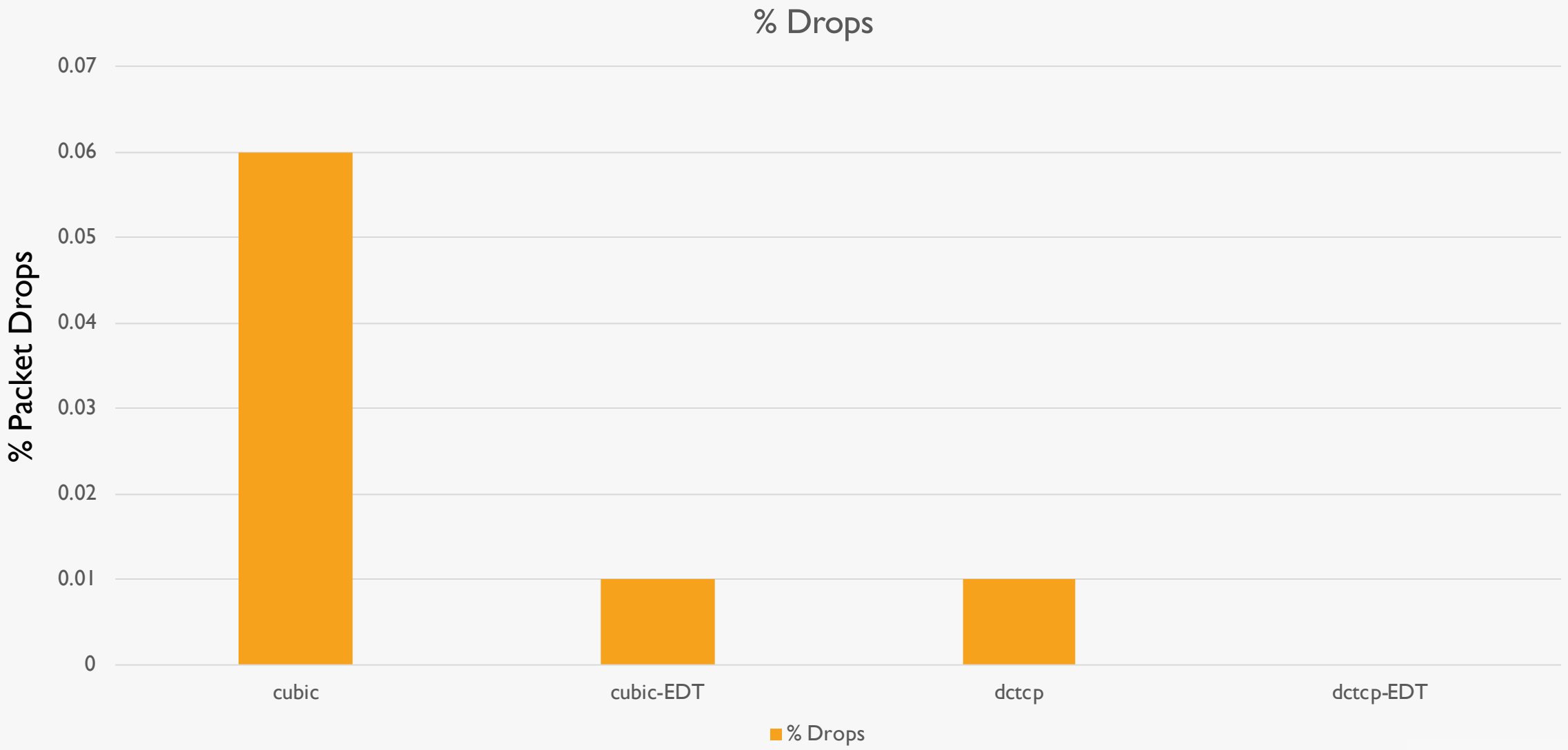
- Single cgroup egress
  - 1-10KB RPC, 3-1MB RPCs
- Multiple cgroups egress
  - 1-10KB RPC, 3-1MB RPCs
- Single cgroup ingress (preventing incast losses)
  - 1-10KB RPC, 3-1MB and 3-8MB RPCs

SINGLE CGROUP EGRESS

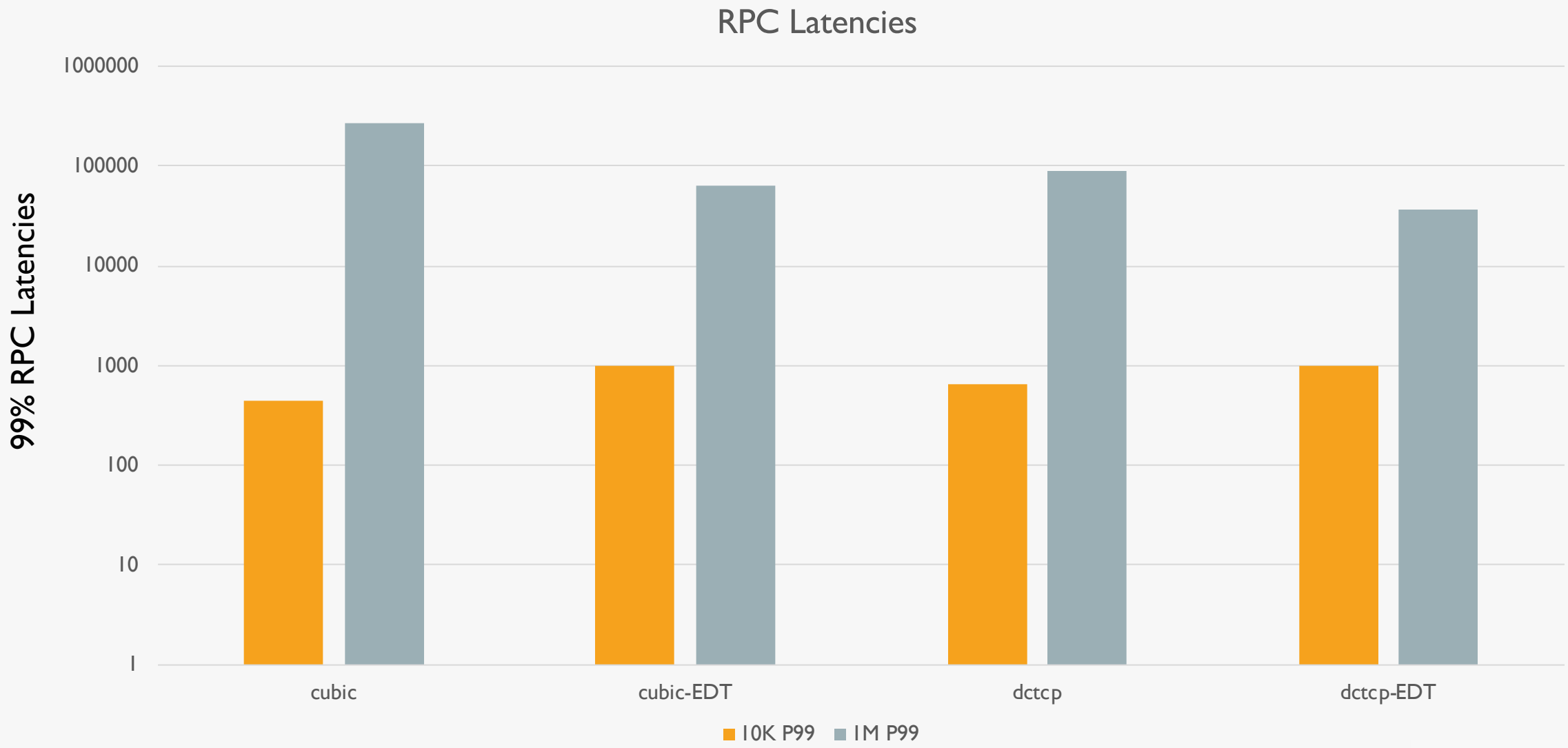
One cgroup, 1G Limit



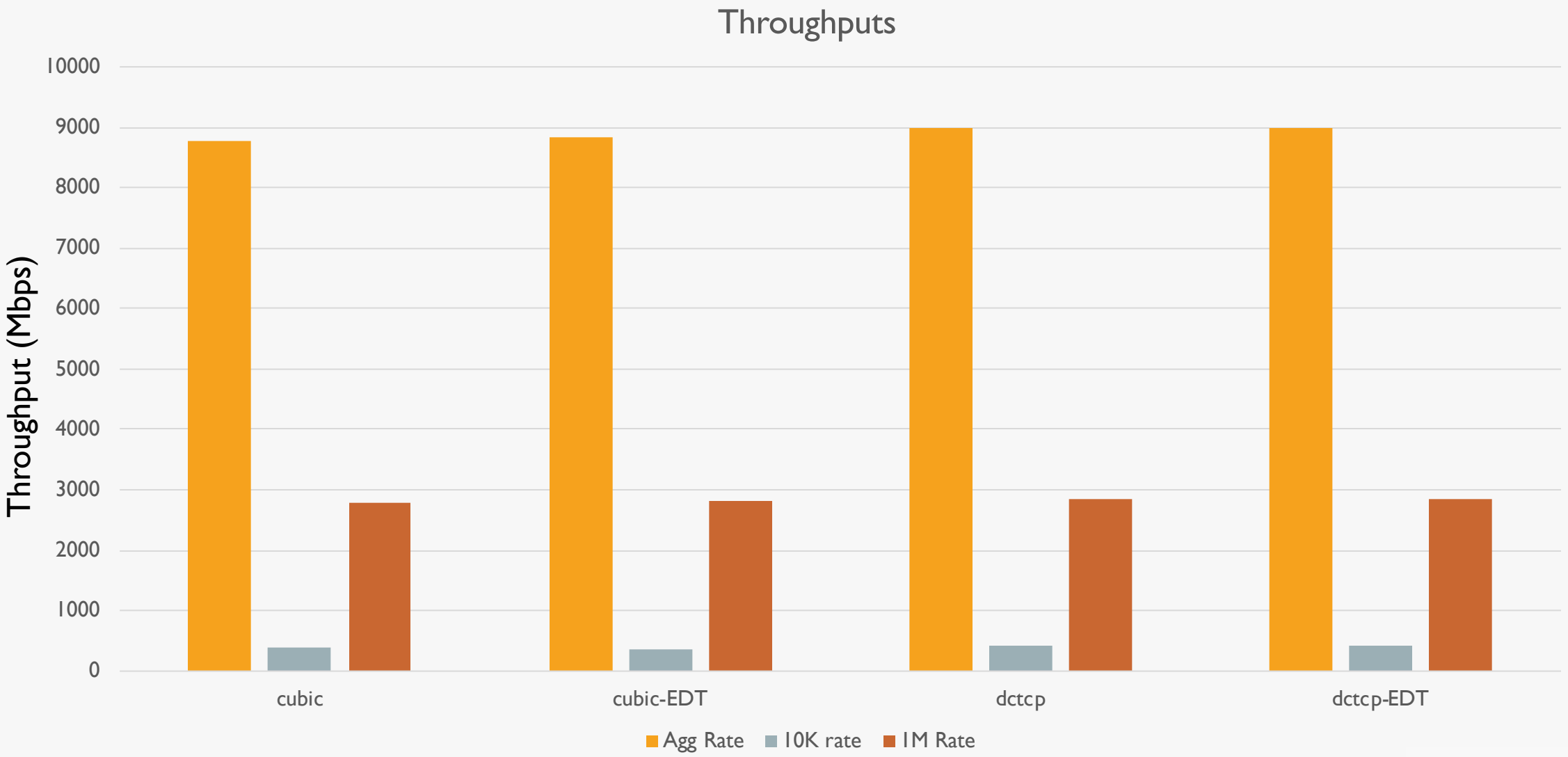
One cgroup, 1G Limit



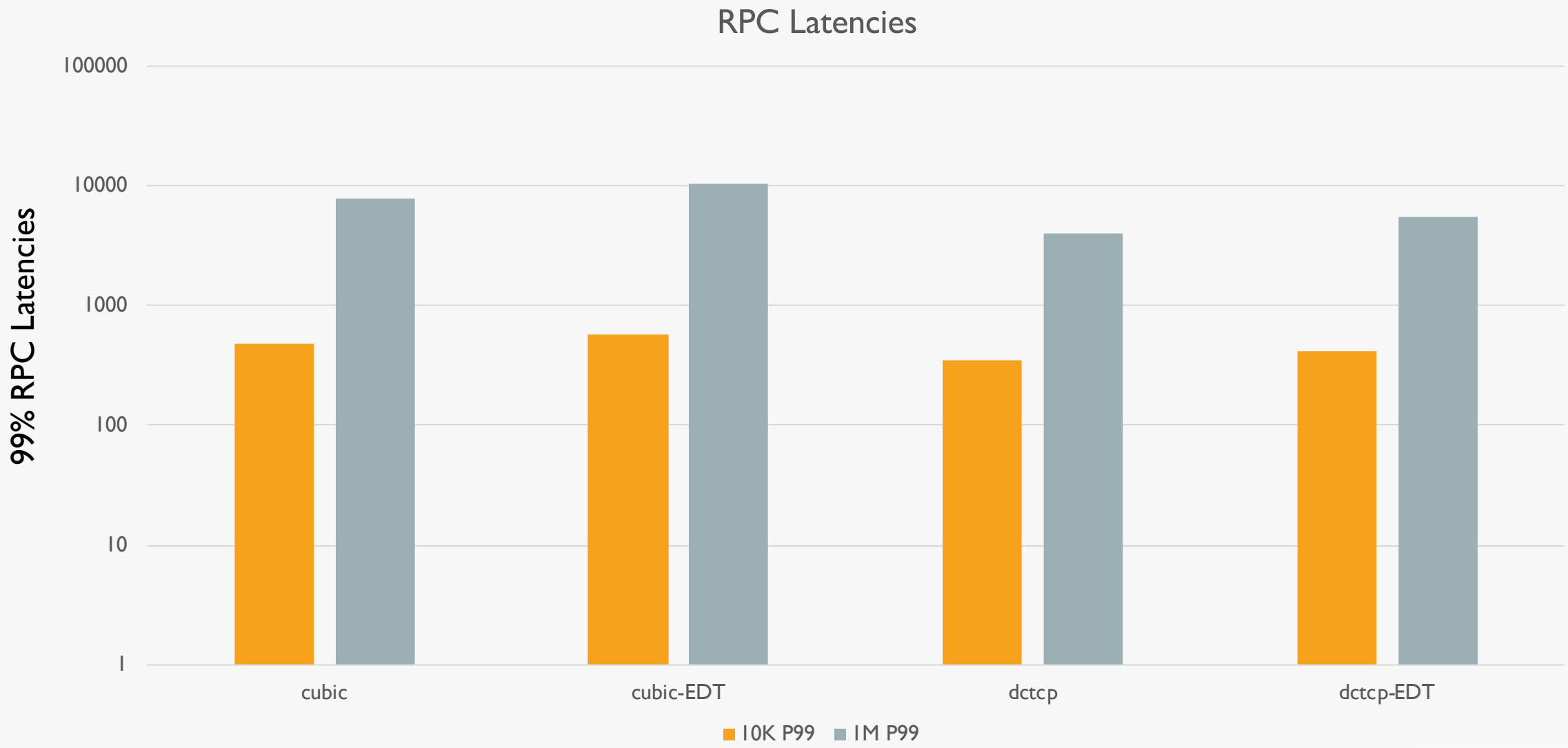
One cgroup, 1G Limit



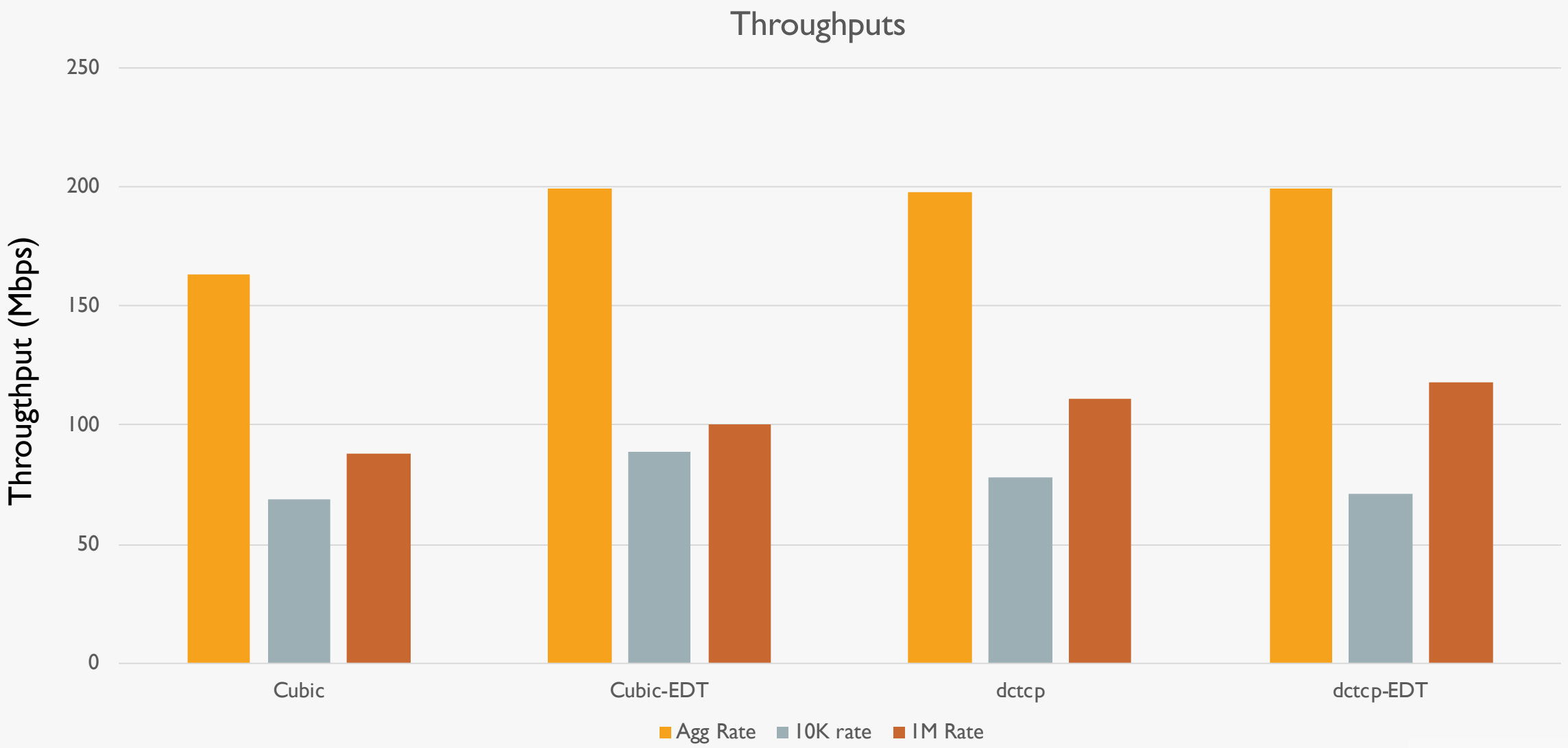
One cgroup, 9G Limit



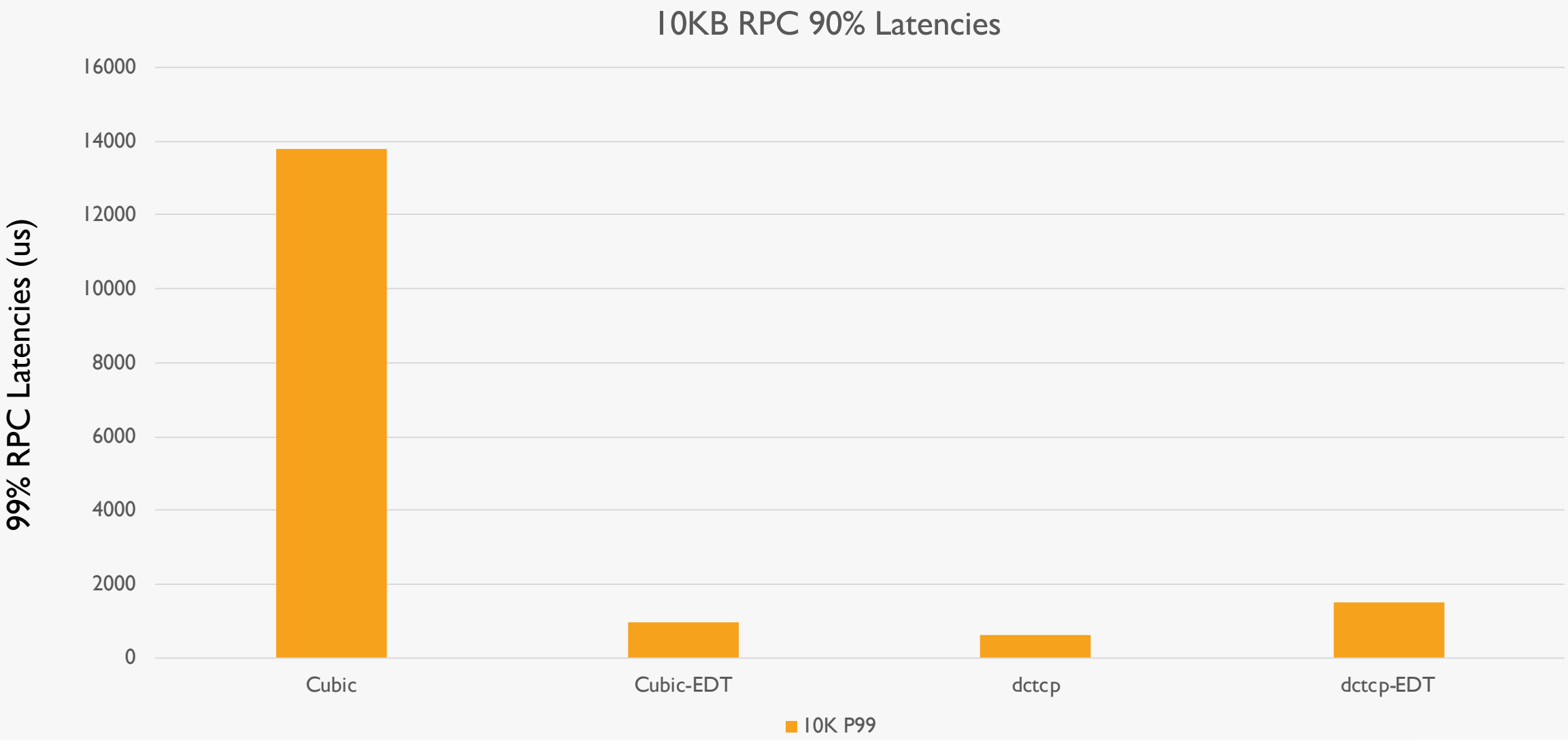
One cgroup, 9G Limit



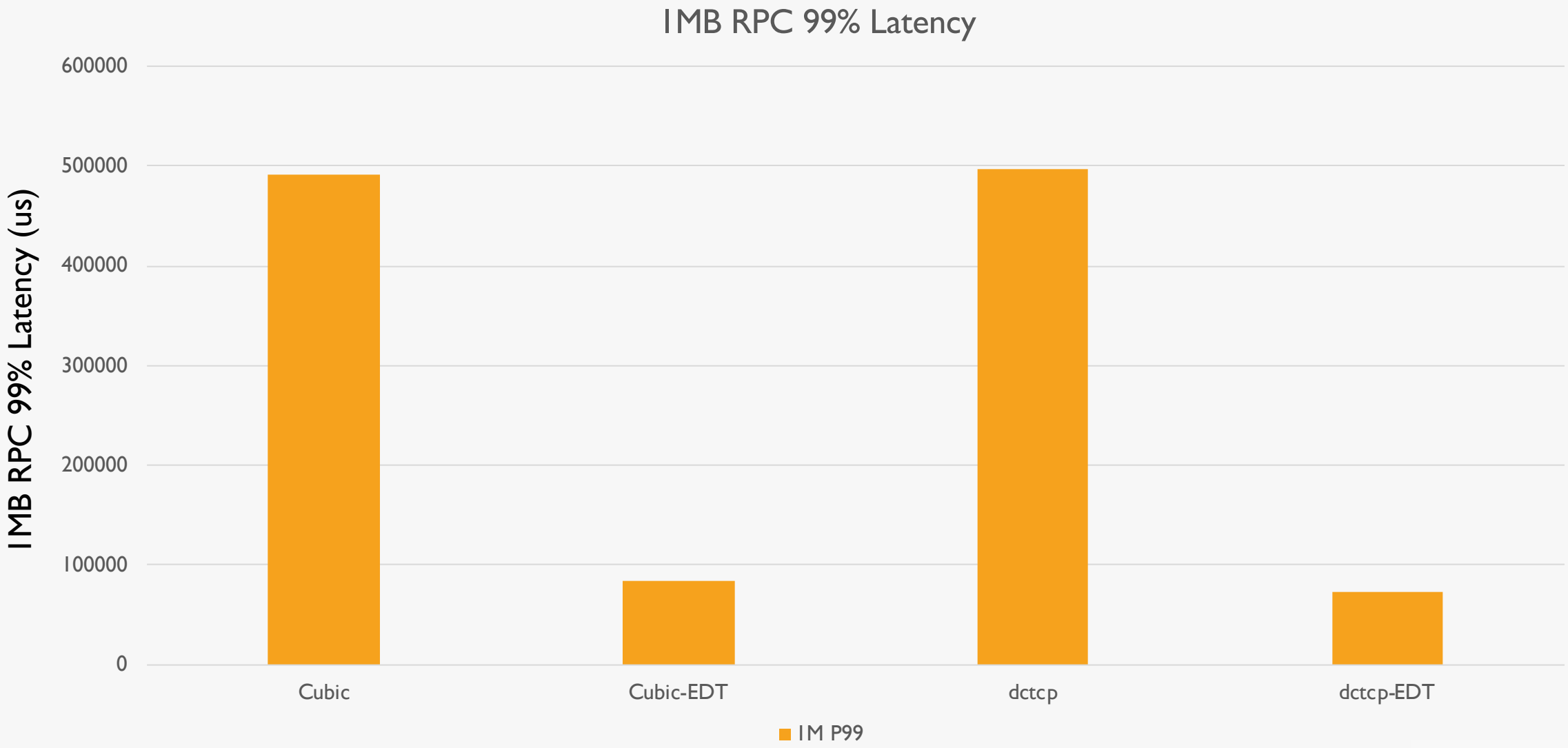
One cgroup, 200M Limit



One cgroup, 200M Limit

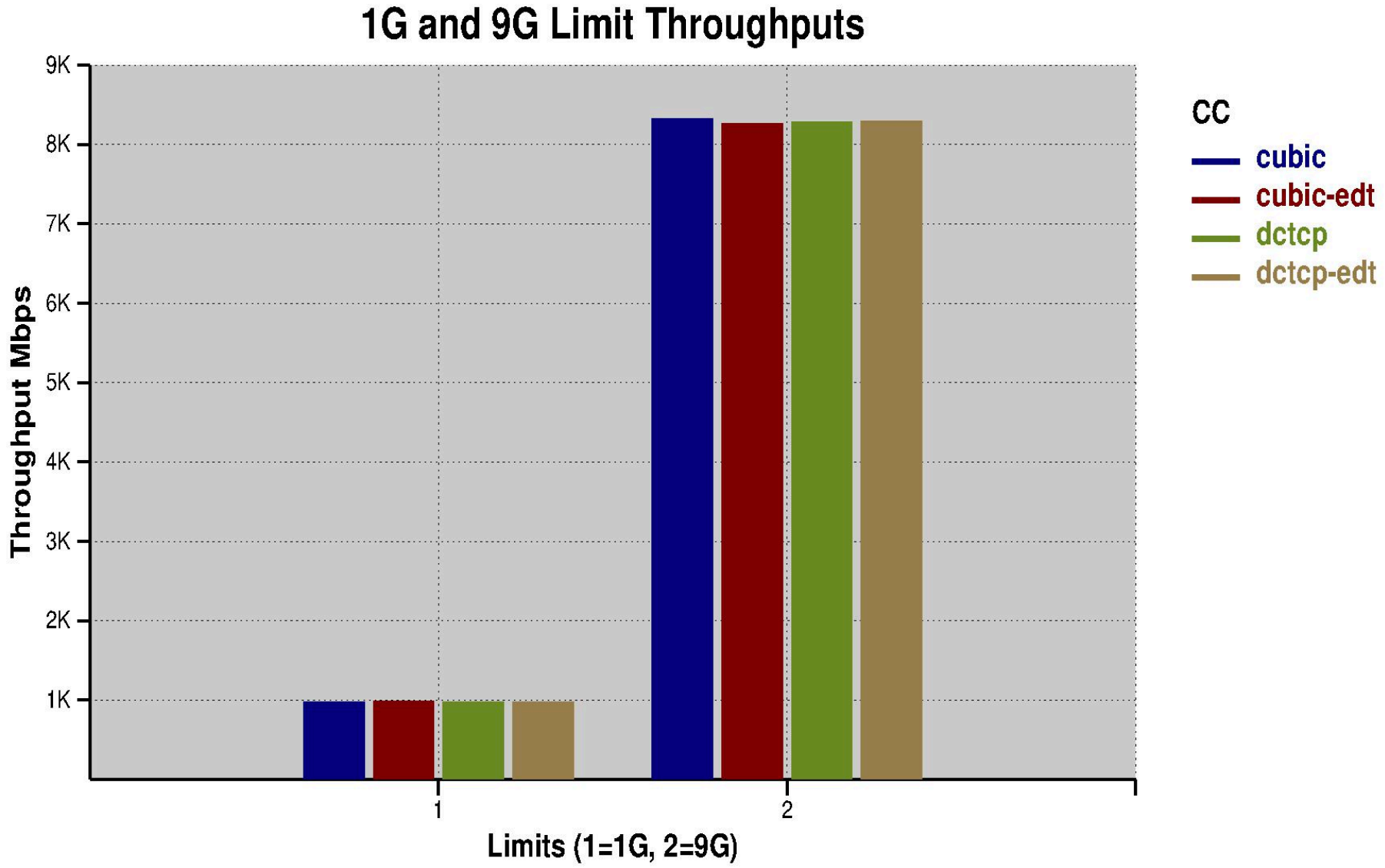


One cgroup, 200M Limit

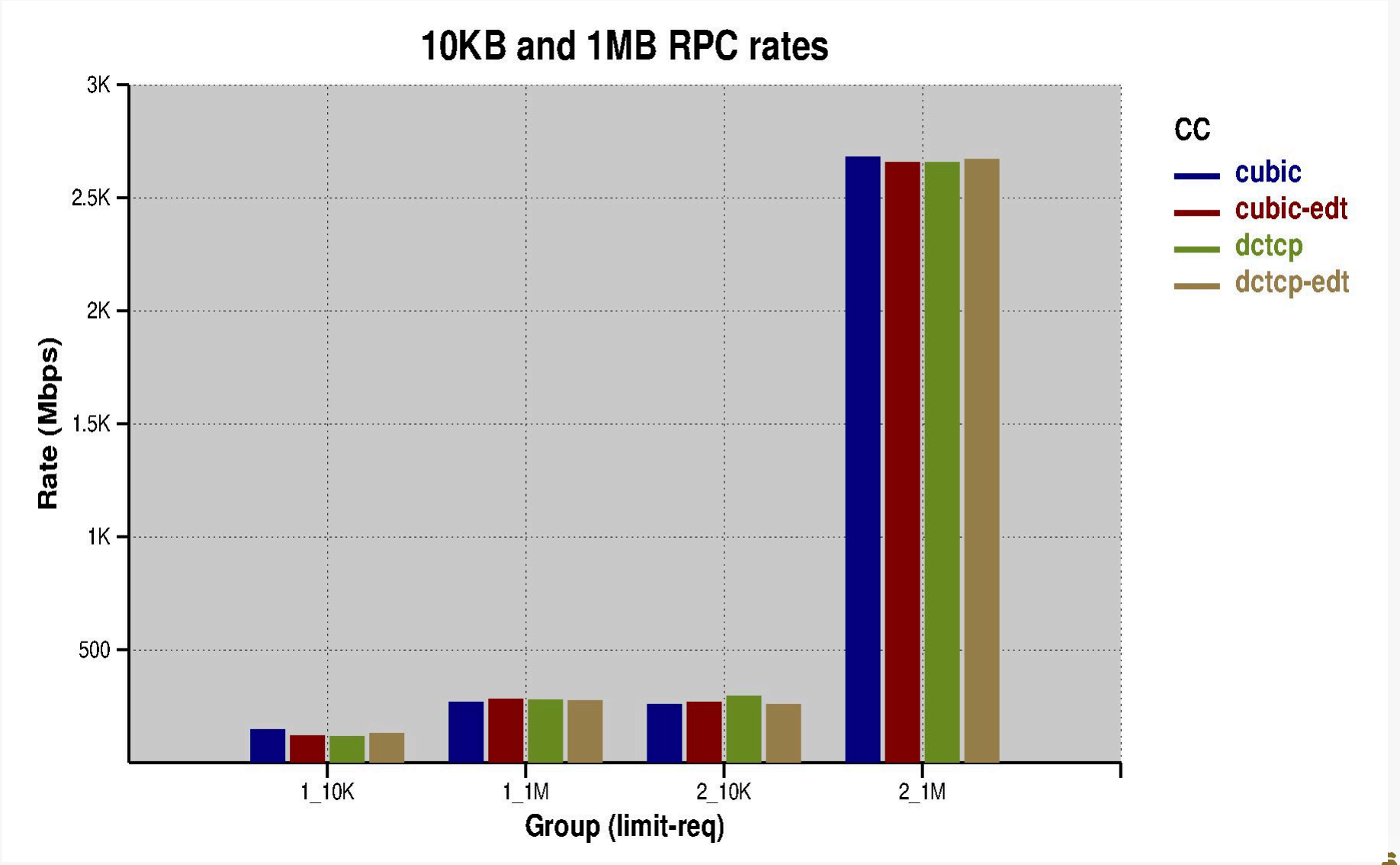


MULTIPLE CGROUP EGRESS

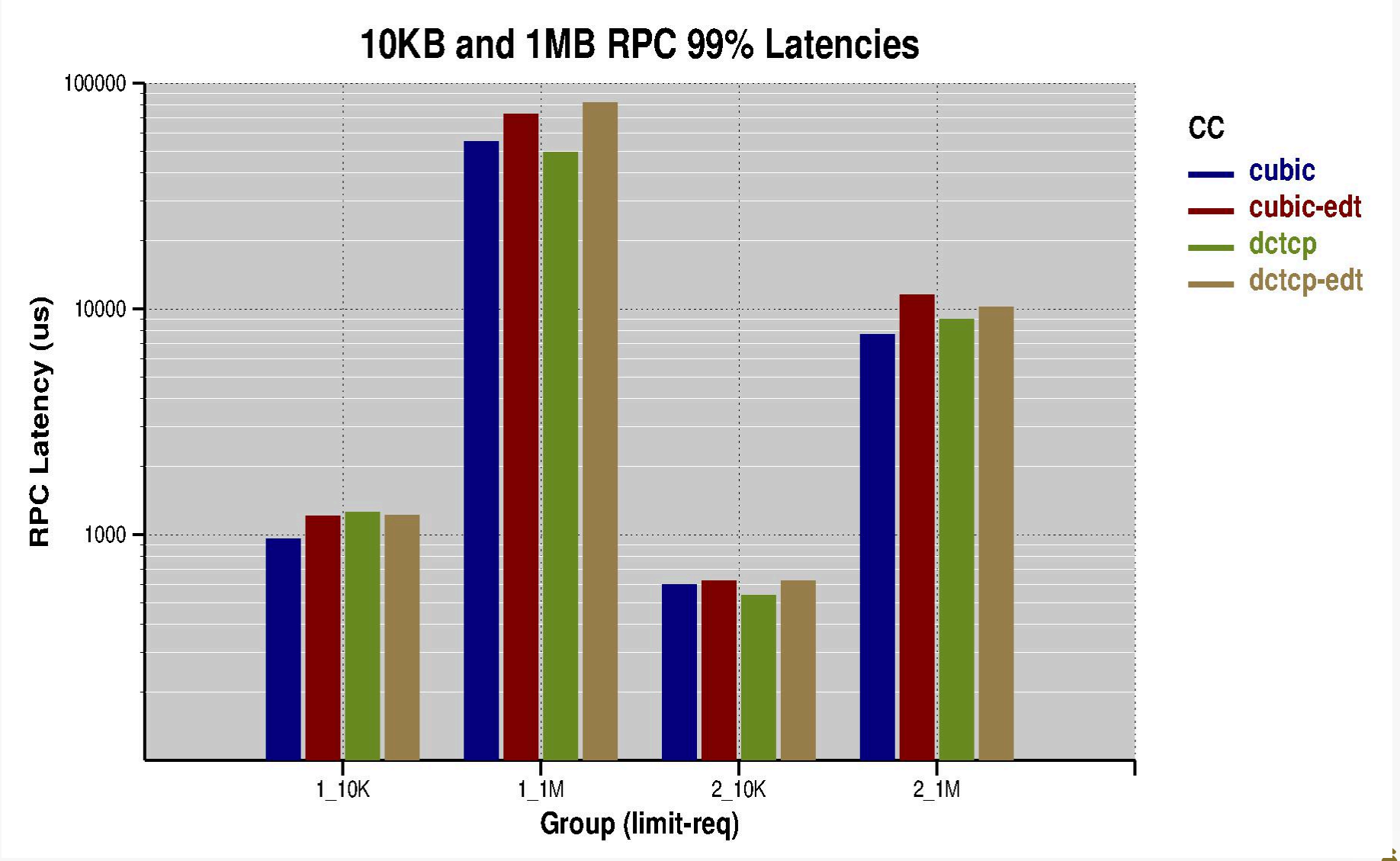
1G and 9G Limits



1G and 9G Limits

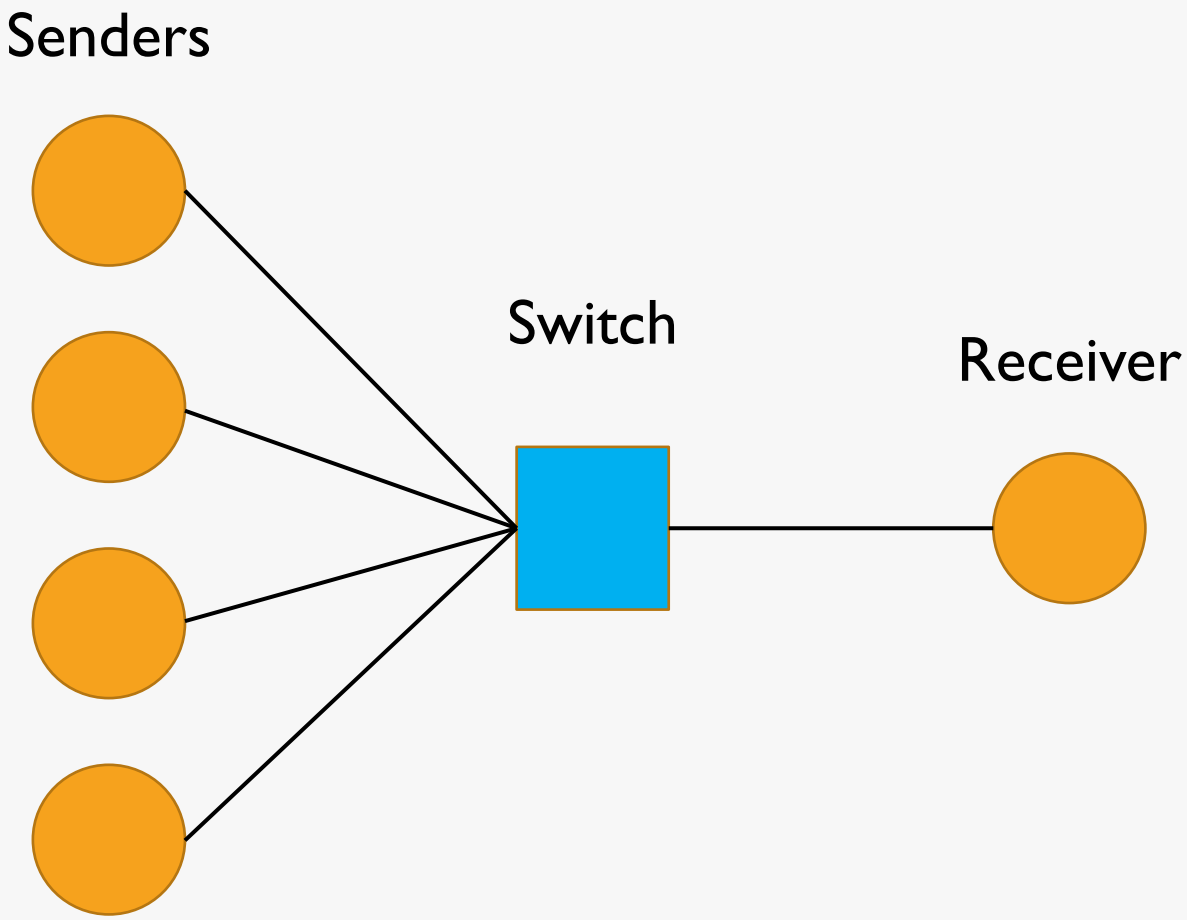


1G and 9G Limits



INGRESS

Experimental Setup



Each sender is doing:

1-10KB RCP

1-1MB RPC

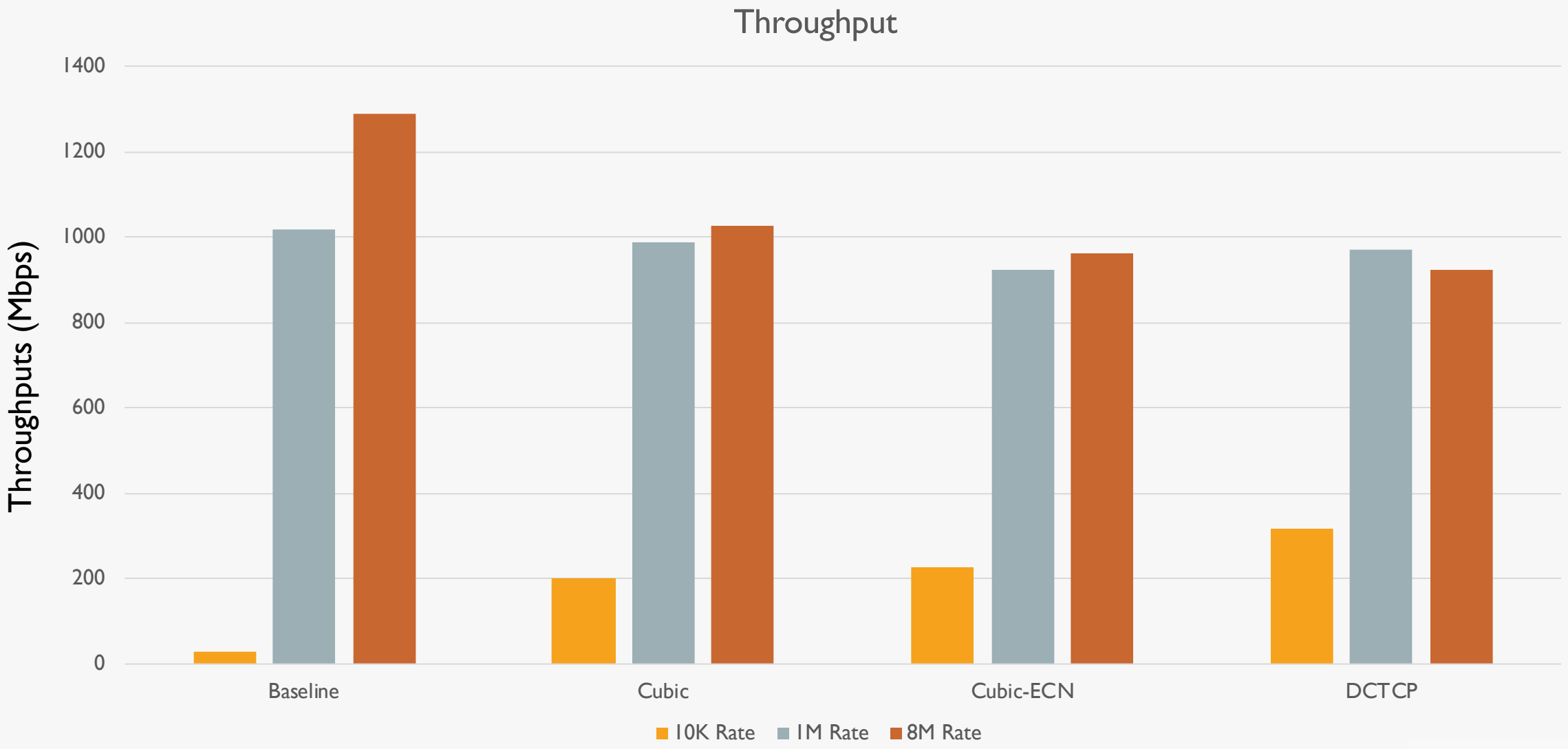
1-8MB RPC

Algorithm for preventing incast losses

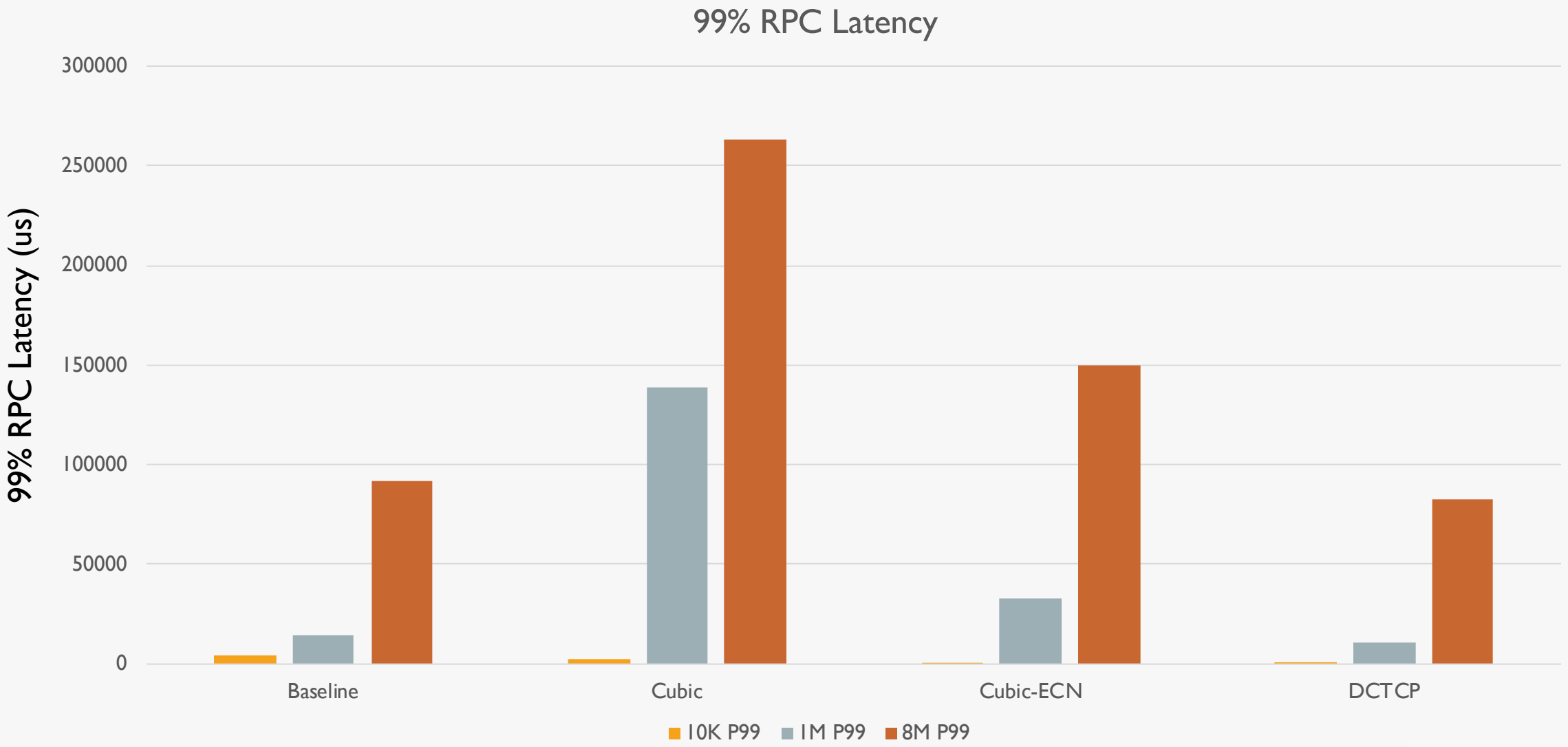
- Apply limit to root cgroup
- We need to impose a limit below link bandwidth since we are never going to see it above link bandwidth
- The further down it is, the more space we have to absorb bursts before the switch buffers start dropping
- When used to protect from incast, no need to drop packets if we are marking them (let switch drop them)

- In this experiments we are dropping

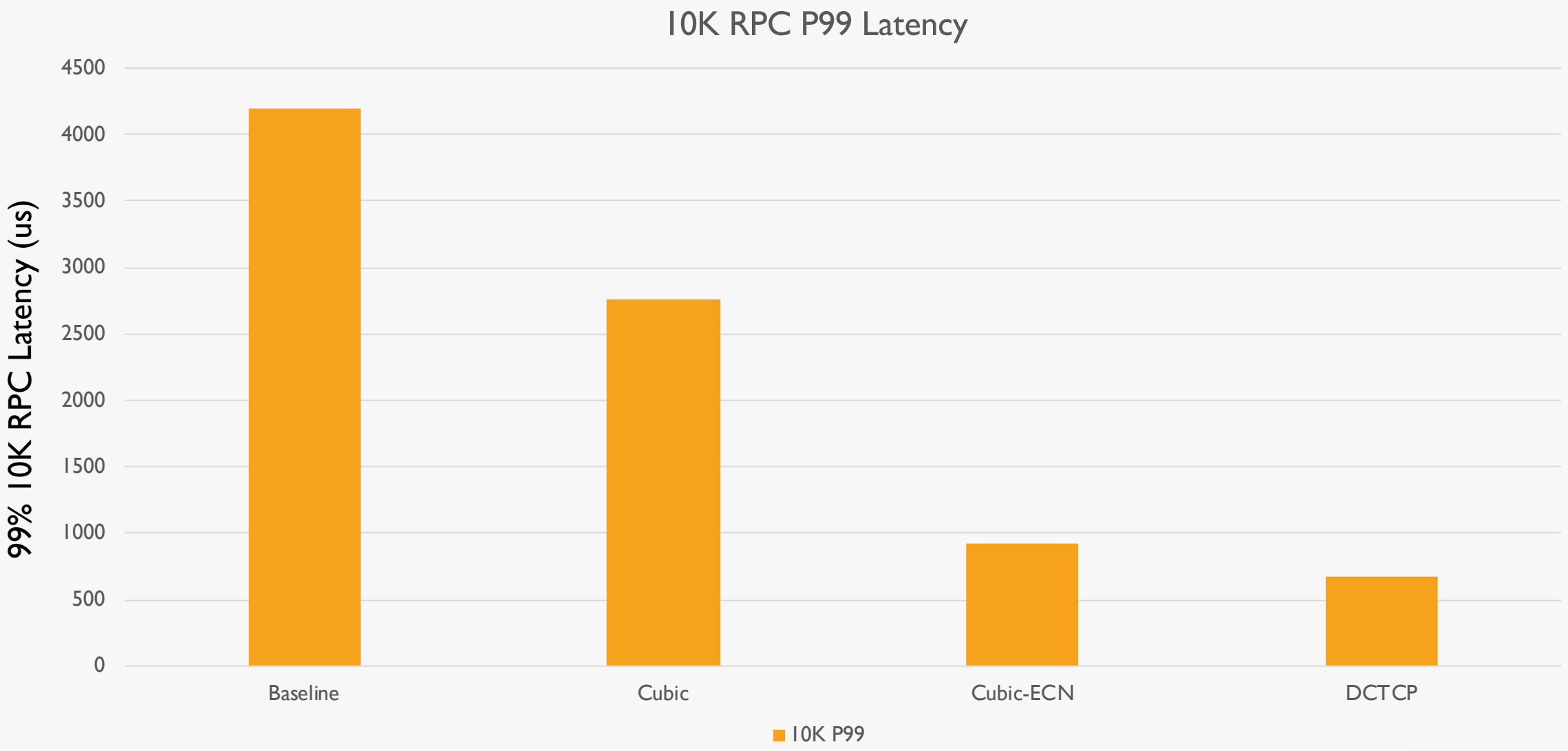
Ingress 9G Limit



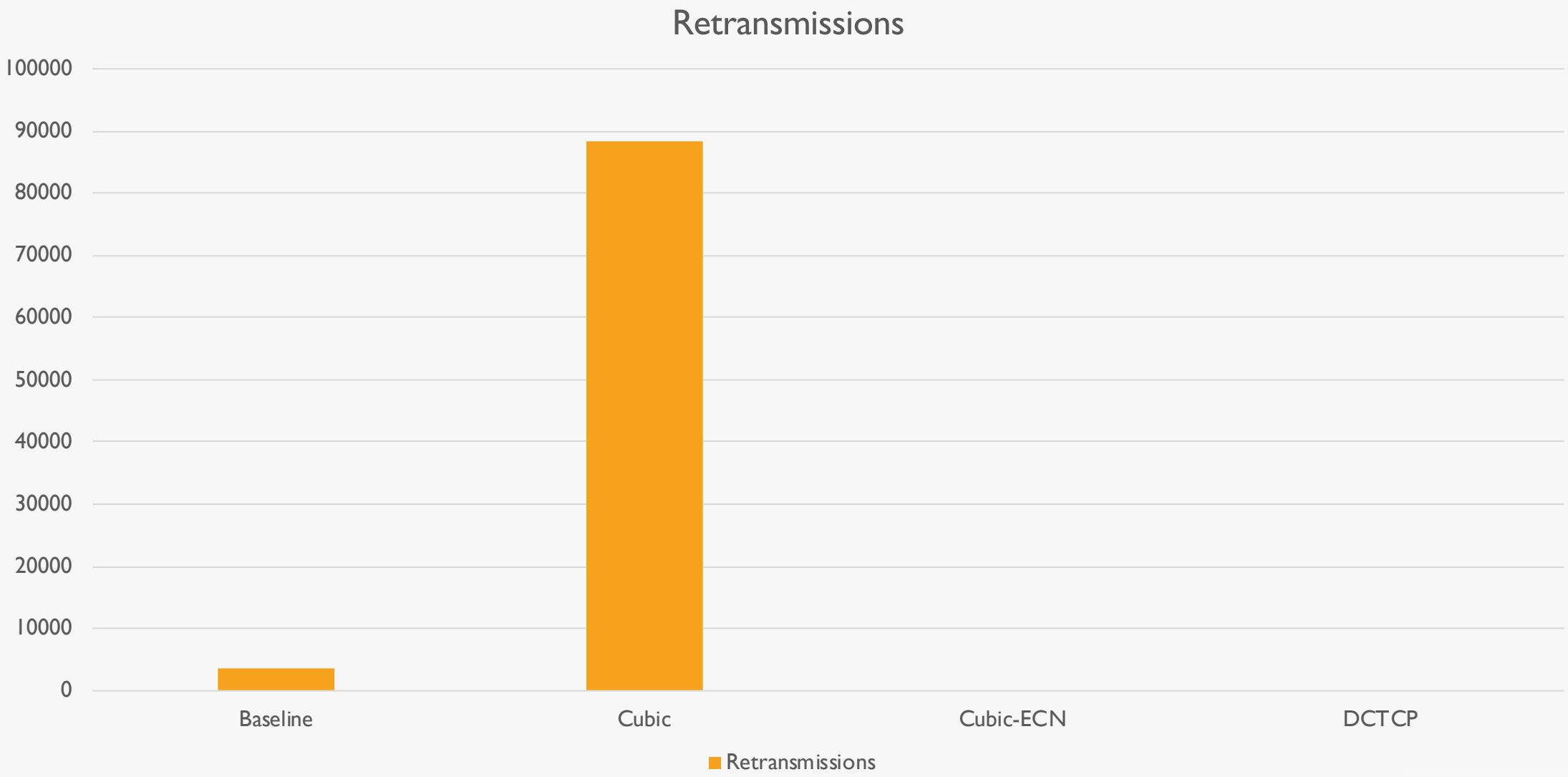
Ingress 9G Limit



Ingress 9G Limit



Ingress 9G Limit



Conclusions

- Can effectively allocate egress bandwidth per cgroup
  - Explored algorithms for egress achieve similar performance, DCTCP best
- Can also use for ingress limiting per cgroup
  - Can be used for reducing losses due to incast traffic
  - Improves fairness of small RPC traffic
  - 10KB RPC 99% latency reduced by 6x compared to baseline