mq-cake

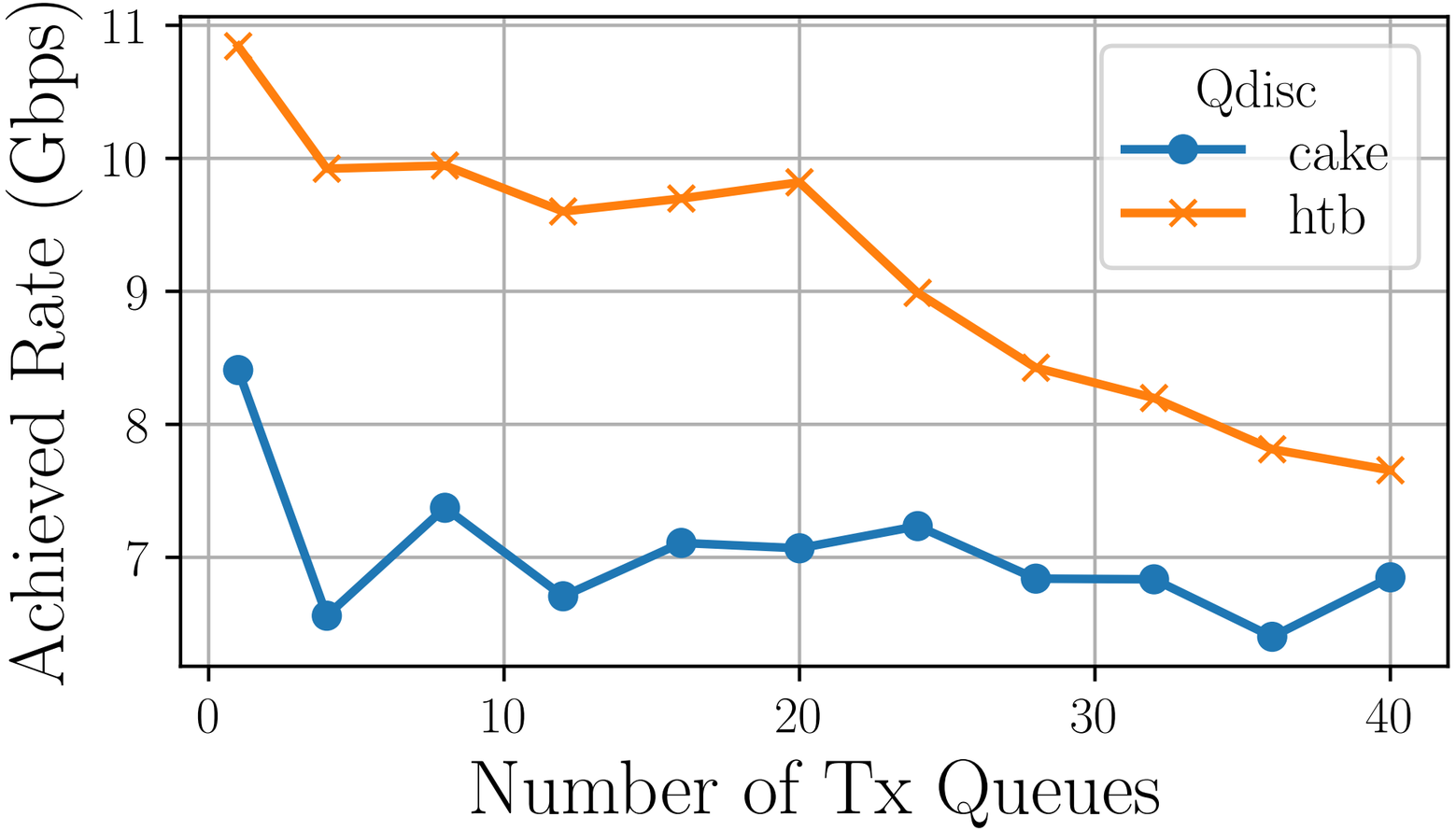
Scaling Software Rate Limiting across CPU cores

Jonas Köppeler - TU Berlin
Toke Høiland-Jørgensen - Red Hat

Motivation

- Network speeds >> CPU speeds
- Scaling software rate limiting is challenging
  - … especially enforcing a global rate limit on a network interface
- Existing qdiscs in the kernel cannot keep up with network speeds
  - … and do not scale

CAKE and HTB

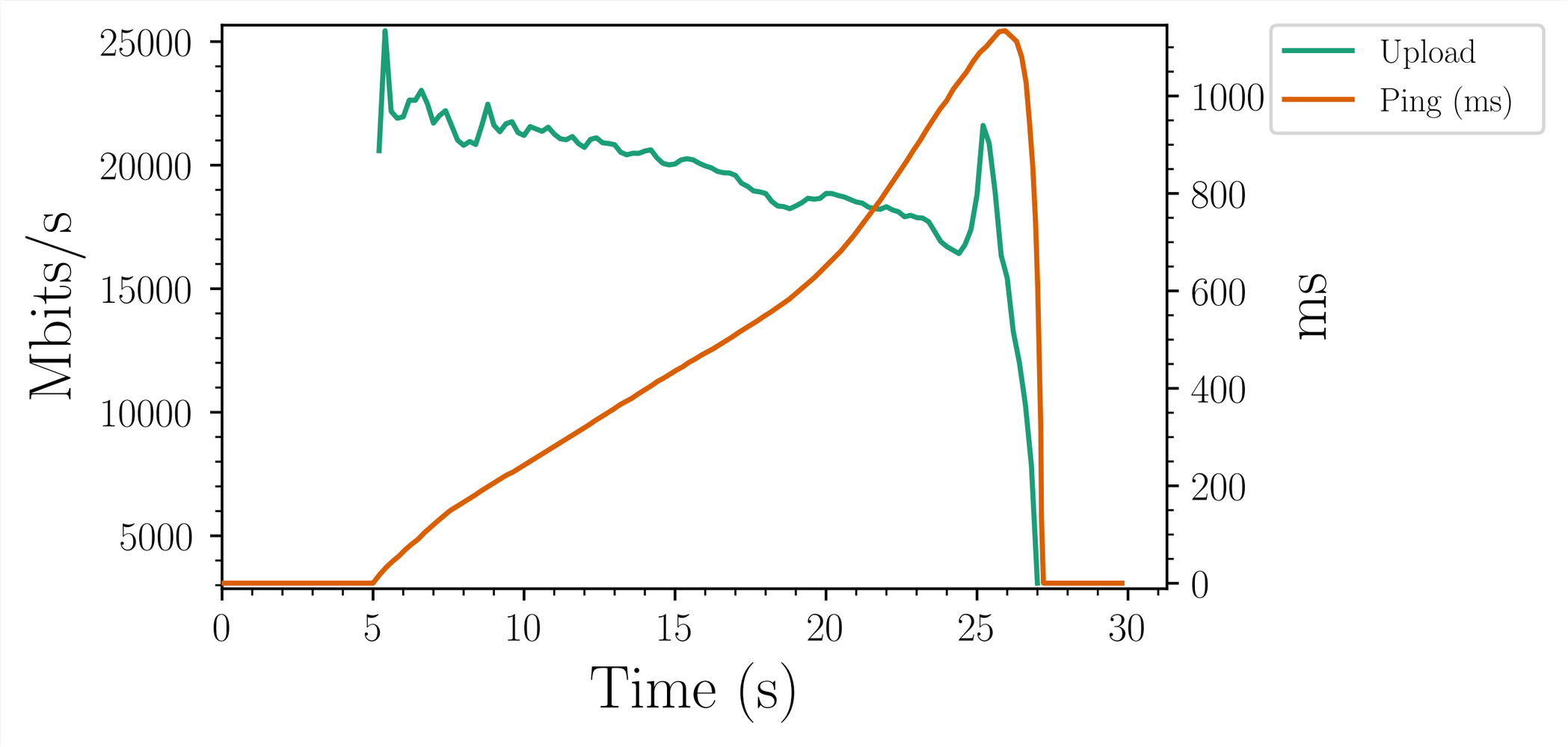


20 Gbps rate limit, UDP flood with full MTU-sized packets

Motivation (cont.)

- A solution like EDT-BPF is great for rate limiting flow aggregates on end hosts
  - … but falls short in enforcing a global rate limit
  - builds a virtual FIFO across queues
  - negative effects on latencies

EDT-BPF



20 Gbps rate limit, 1024 TCP streams

We present: mq-cake

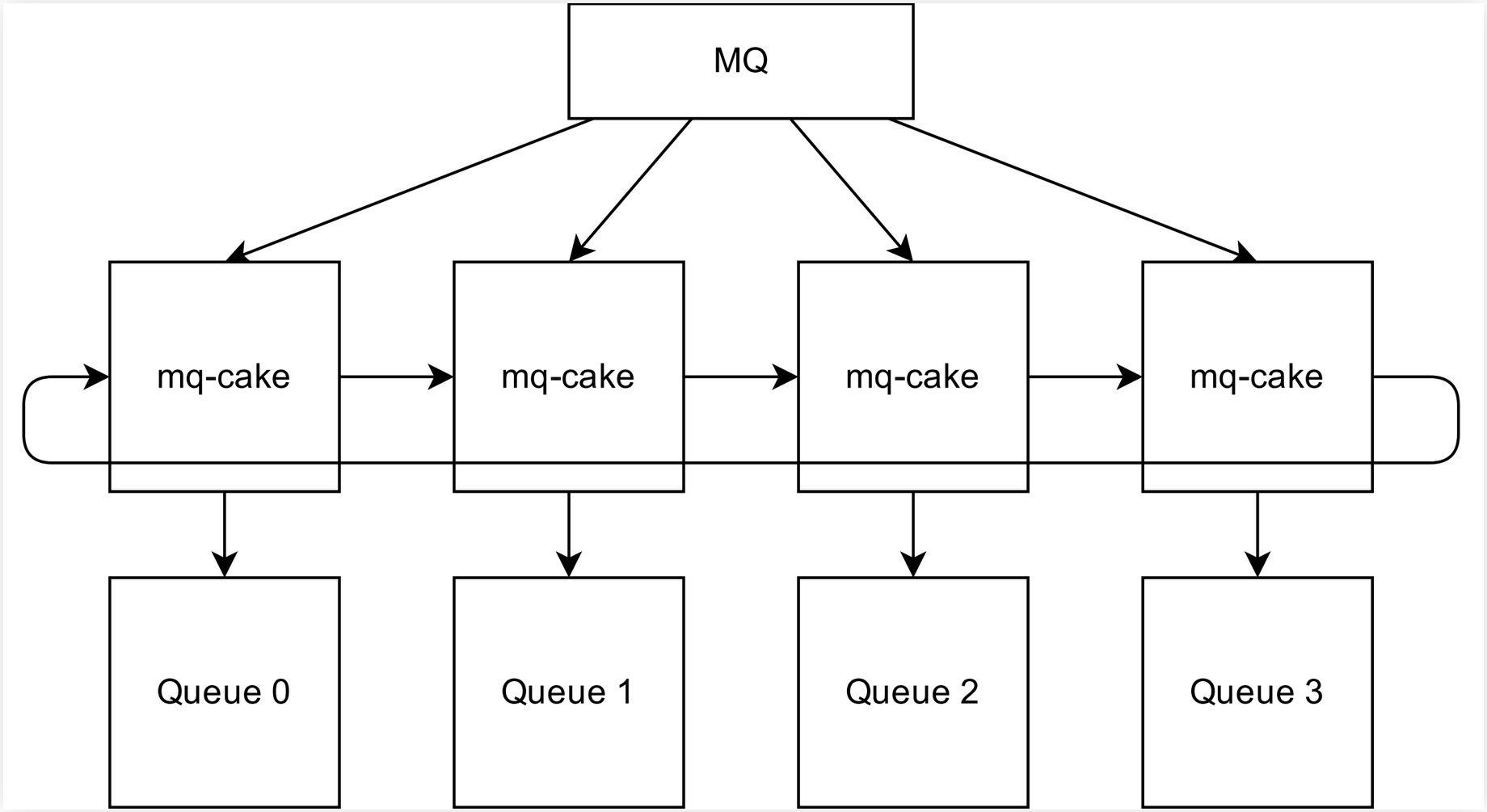
- Multi-queue compatible variant of sch_cake
- Enforce high rate limits while keeping queues smart

Outline

- Design
- Evaluation
- Outlook

Design

Architecture



Four hardware queues, mq-cake instances connected via linked list

Algorithm

- Every instance loops over linked list in certain intervals
  - Interval called synctime
- Determine if other mq-cake instances are active
- Calculate new local rate:

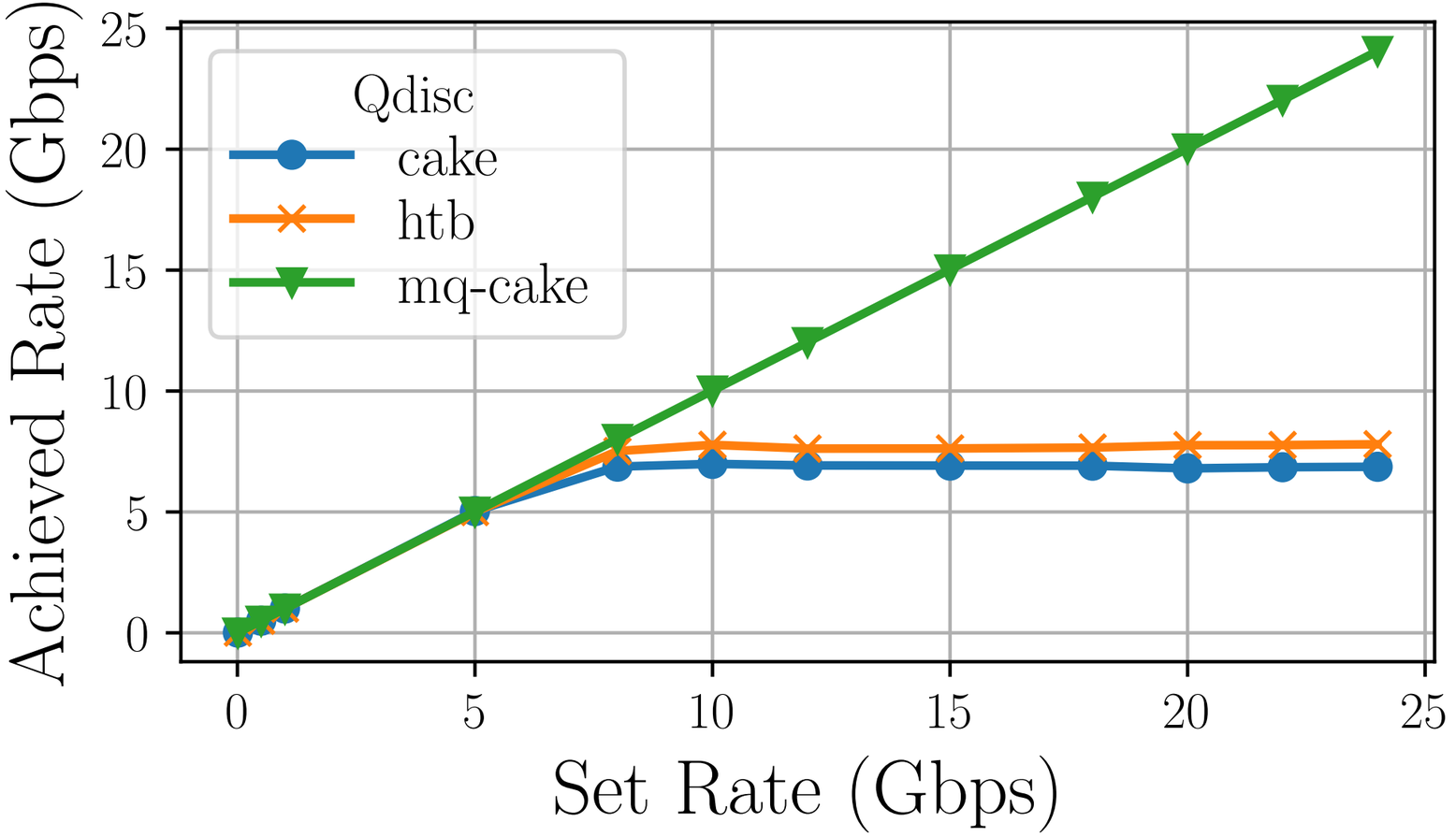
$$rate_{local} = \frac{rate_{global}}{\#\ \text{active qdiscs}}$$

Active qdisc

- A qdisc is considered active if:
  - packets are backlogged, or
  - packets were sent since last scan

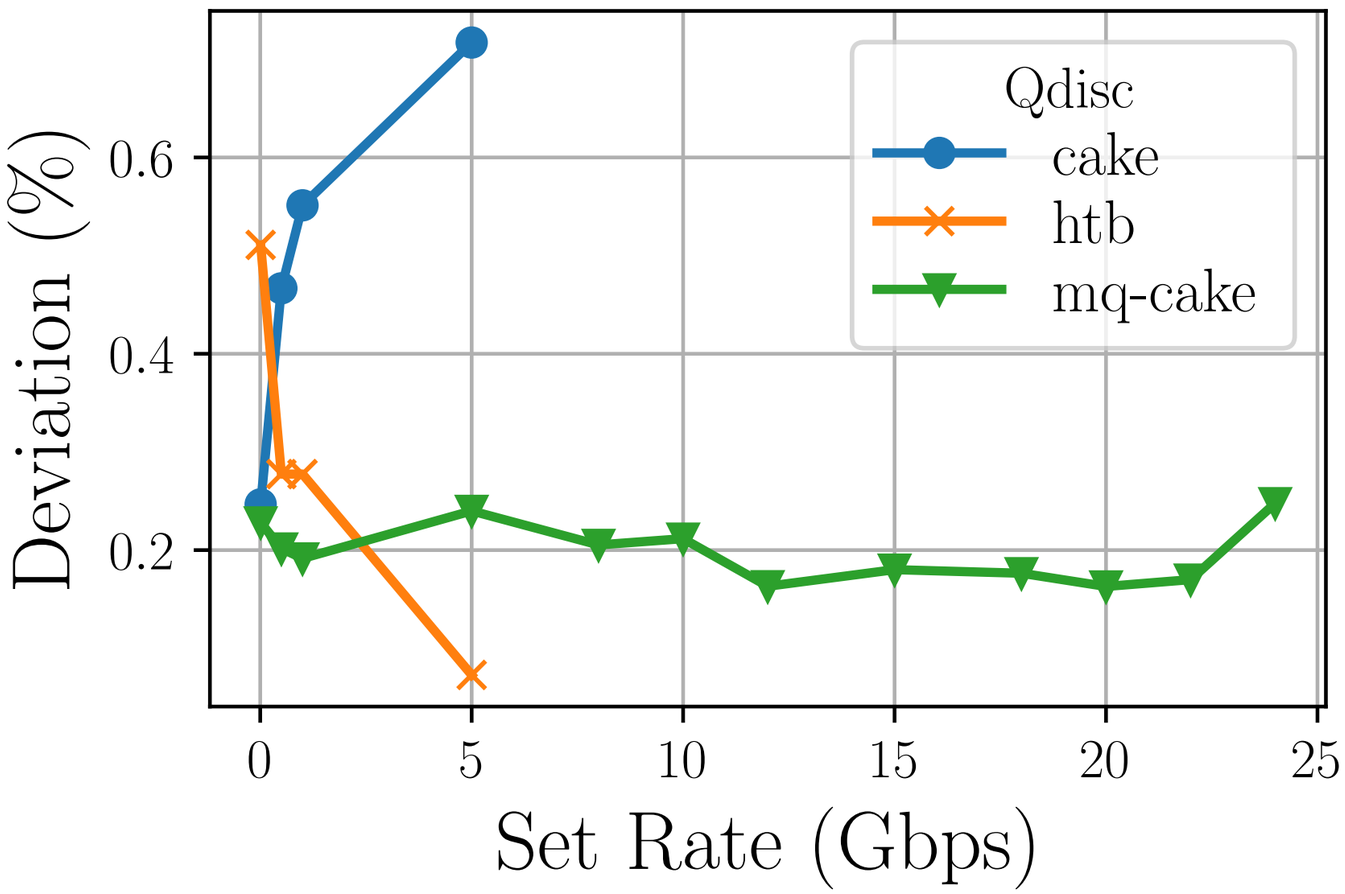
Evaluation

Rate Conformance



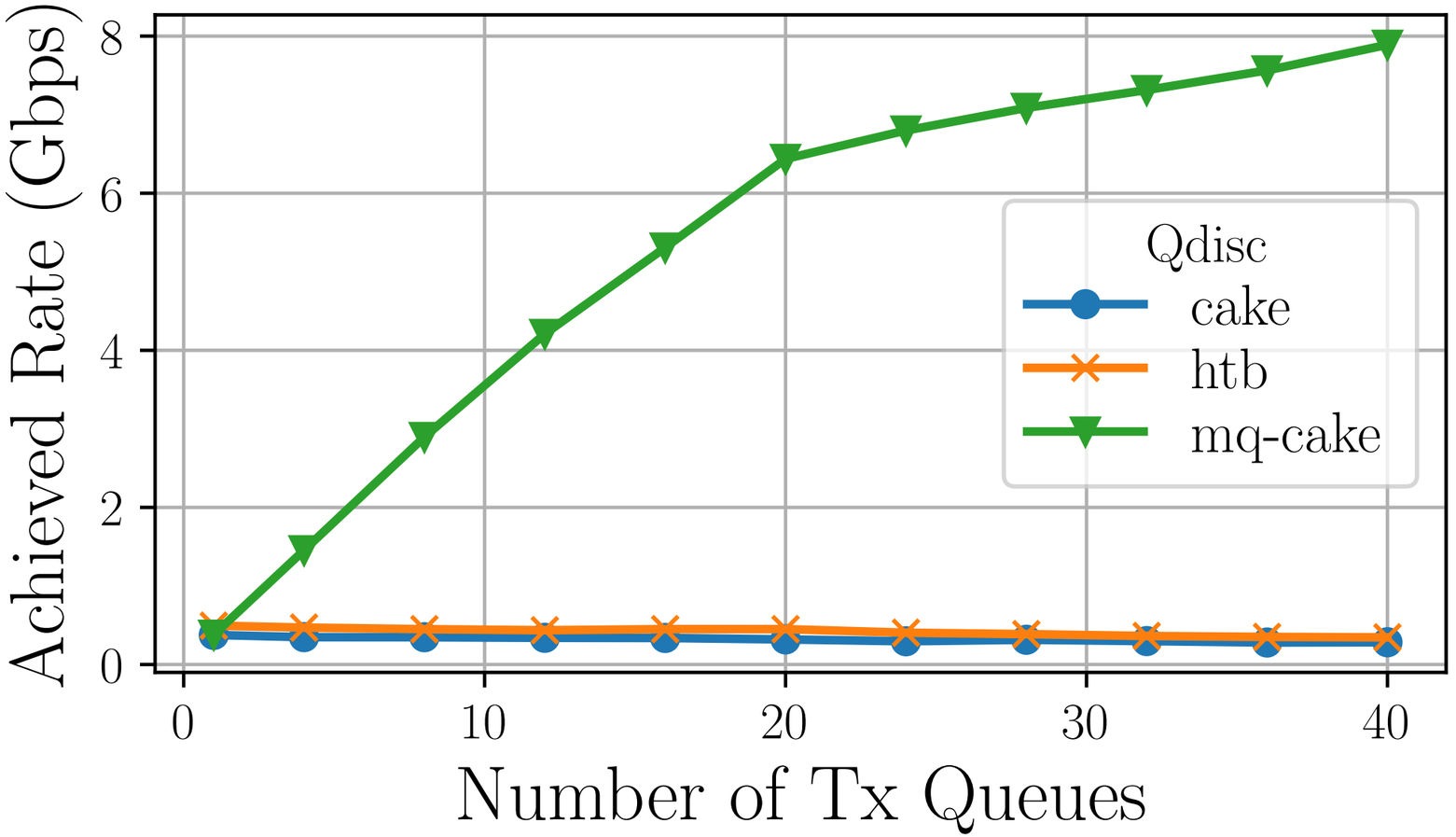
40 queues, UDP flood with full MTU-sized packets

Deviation



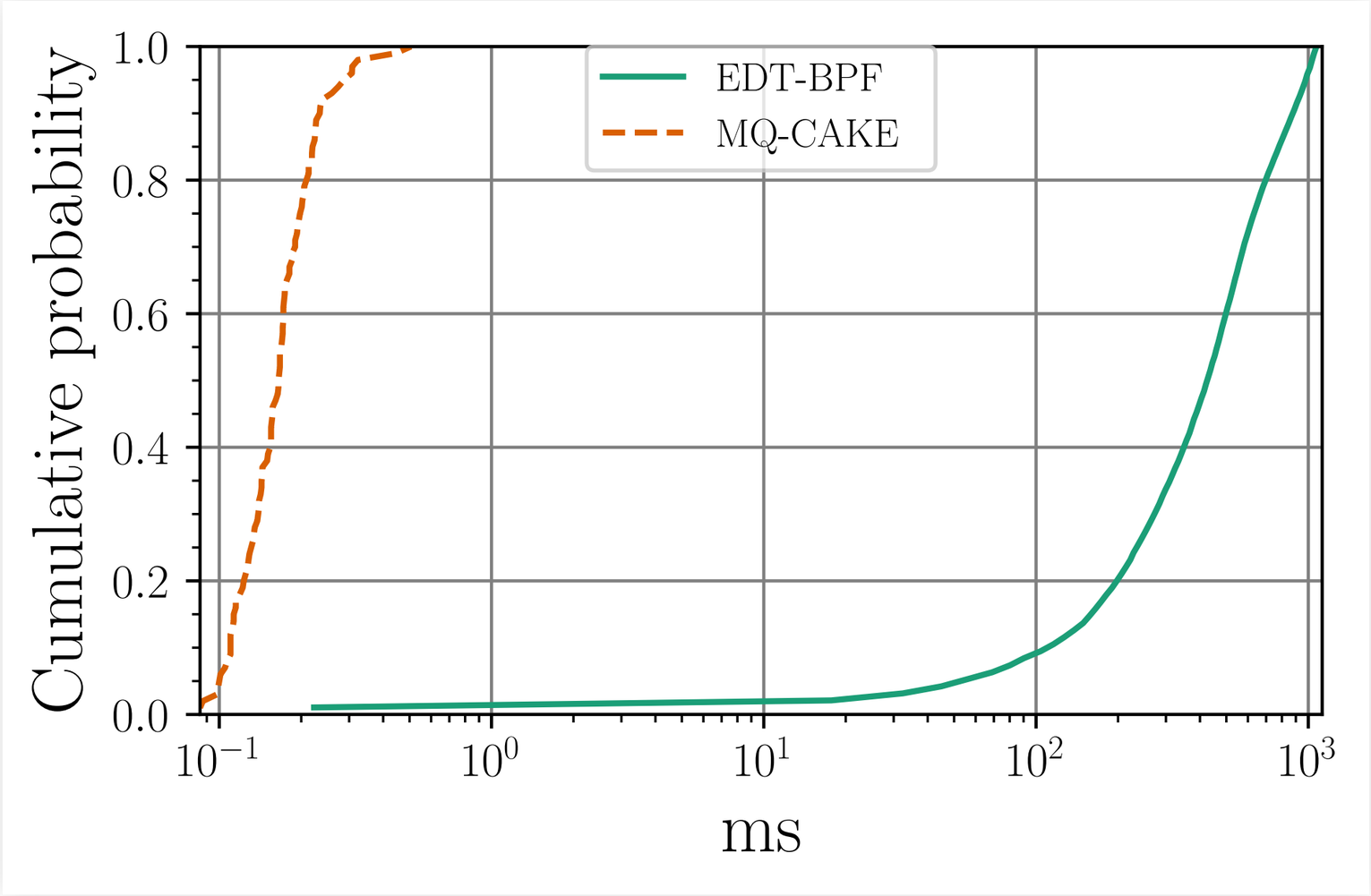
40 queues, UDP flood with full MTU-sized packets

Scaling



20 Gbps rate limit, UDP flood with 64 byte packets

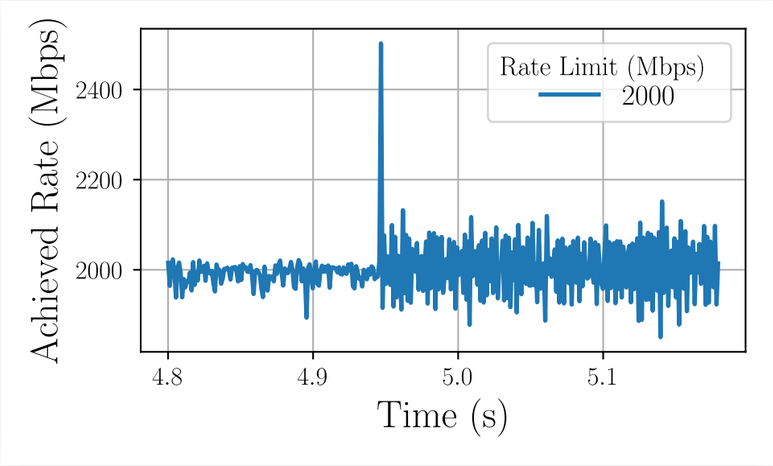
Latencies

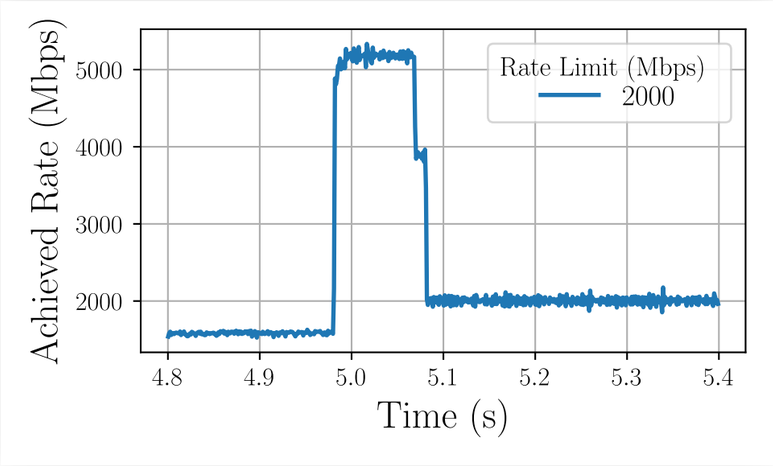


1024 TCP streams, 2s drop horizon for EDT-BPF

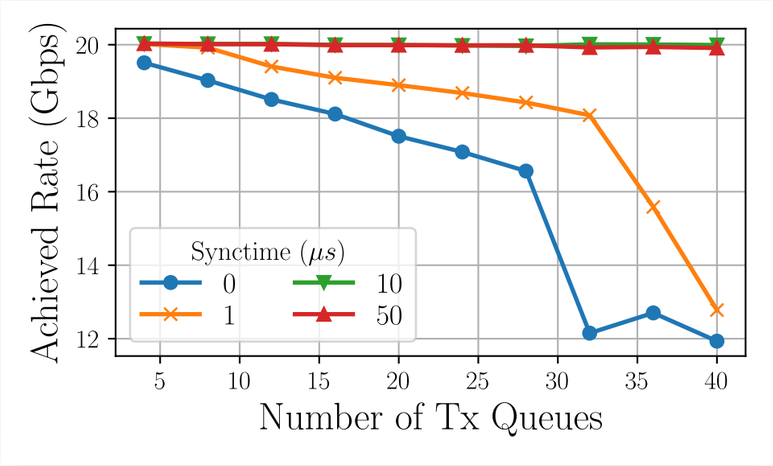
Current Limitations

Impact of synchronisation time



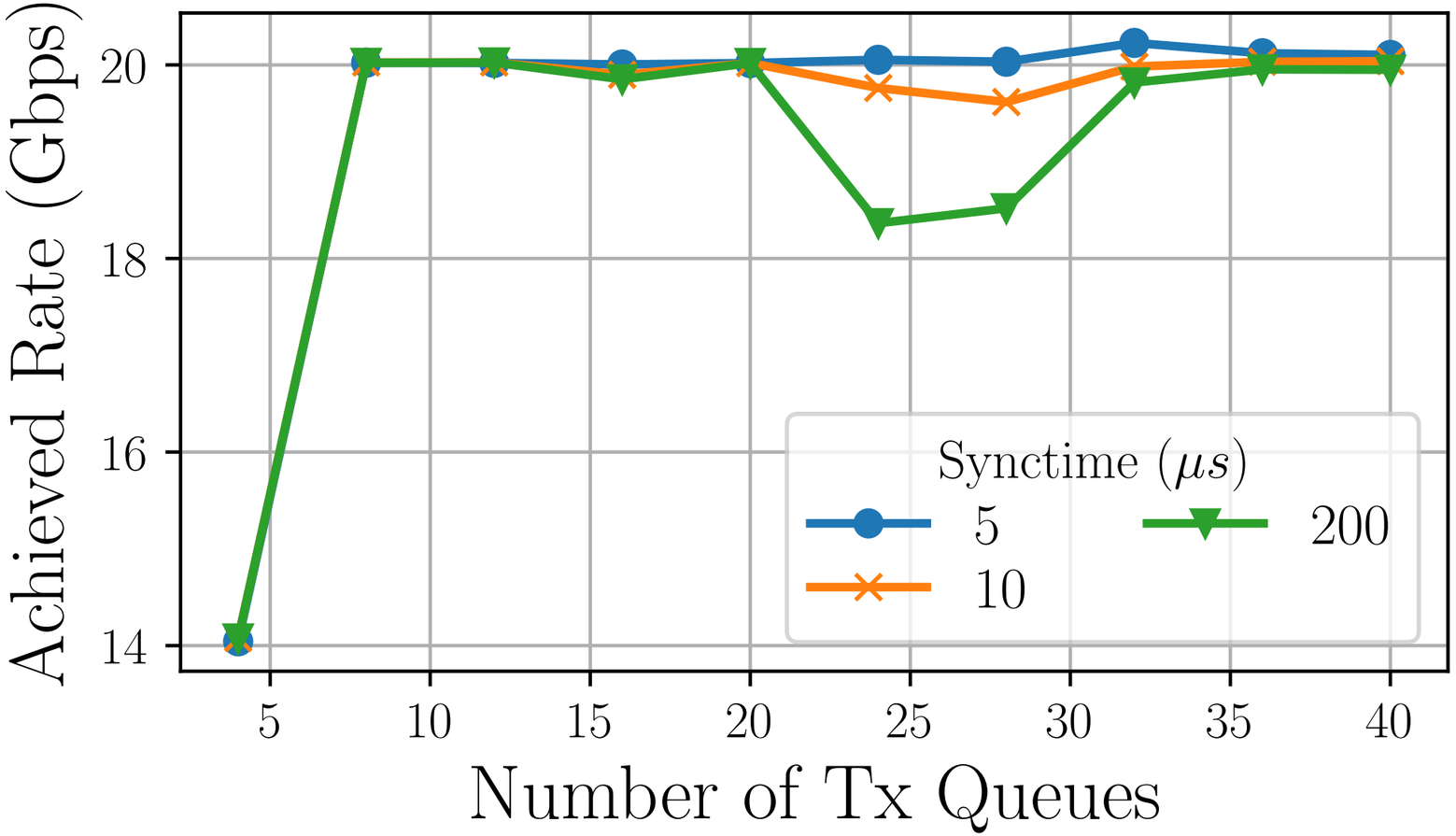


Sync time of 100 us (top) and 100 ms



Overhead of different sync times

Imbalances



Imbalance in traffic load between qdiscs

Outlook

Future Work

- Tackle imbalance issue
- Mitigate overshoot during switching events
- Test under real-world conditions
- Automatically determine best synctime value

Upstreaming Plans

Current implementation:

- sch_cake walks qdisc tree
- build data structure across sibling qdiscs

For upstreaming, we propose a “shared qdisc state API”

Shared qdisc state API

```
--- a/include/net/sch_generic.h
+++ b/include/net/sch_generic.h
@@ -70,6 +70,14 @@ struct qdisc_skb_head {
        spinlock_t      lock;
 };

+struct qdisc_shared_data {
+       struct list_head         head;
+       const struct Qdisc_ops  *owner;
+       struct rcu_head  rcu;
+       refcount_t               refs;
+       u8                       data[];
+};
+
 struct Qdisc {
        int                     (*enqueue)(struct sk_buff *skb,
                                           struct Qdisc *sch,
@@ -109,7 +117,7 @@ struct Qdisc {
        struct gnet_stats_queue __percpu *cpu_qstats;
        int                     pad;
        refcount_t              refcnt;
-
+       struct list_head        shared_state;
        /*
         * For performance sake on SMP, we put highly modified fields at the end
         */
@@ -289,6 +297,7 @@ struct Qdisc_ops {
        const struct Qdisc_class_ops    *cl_ops;
        char                    id[IFNAMSIZ];
        int                     priv_size;
+       int                     shared_size;
        unsigned int            static_flags;

        int                     (*enqueue)(struct sk_buff *skb,
@@ -319,6 +328,9 @@ struct Qdisc_ops {
        u32                     (*ingress_block_get)(struct Qdisc *sch);
        u32                     (*egress_block_get)(struct Qdisc *sch);

+       void                    (*shared_init)(void *shared);
+       void                    (*shared_assign)(struct Qdisc *, void *shared);
+
        struct module           *owner;
 };
```

Shared qdisc state API (cont.)

```
--- a/net/sched/sch_mq.c
+++ b/net/sched/sch_mq.c
@@ -178,6 +189,15 @@ static int mq_graft(struct Qdisc *sch, unsigned long cl, struct Qdisc *new,
        struct netdev_queue *dev_queue = mq_queue_get(sch, cl);
        struct tc_mq_qopt_offload graft_offload;
        struct net_device *dev = qdisc_dev(sch);
+       struct qdisc_shared_data *shared;
+
+       if (new && new->ops->shared_size) {
+               shared = qdisc_shared_get(sch, new->ops);
+               if (!shared)
+                       return -ENOMEM;
+
+               new->ops->shared_assign(new, &shared->data);
+       }
@@ -185,6 +205,12 @@ static int mq_graft(struct Qdisc *sch, unsigned long cl, struct Qdisc *new,
        *old = dev_graft_qdisc(dev_queue, new);
        if (new)
                new->flags |= TCQ_F_ONETXQUEUE | TCQ_F_NOPARENT;
+
+       if (*old && (*old)->ops->shared size) {
+               (*old)->ops->shared_assign(*old, NULL);
+               qdisc_shared_put(sch, (*old)->ops);
+       }
+
        if (dev->flags & IFF_UP)
                dev_activate(dev);
```

Full patch:

https://git.kernel.org/toke/l/mq-cake-api-experiments

Questions?