Sponsored By

Google intel STORDIS The Open Networking Expert

Advanced Congestion & Flow Control with Programmable Switches

Jeongkeun "JK" Lee
Principal Engineer
Intel, Barefoot Division

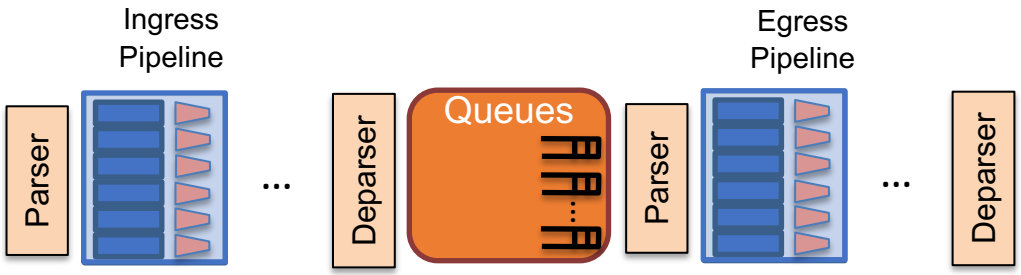
Agenda

- Programmable building blocks for congestion/flow controls
  - Egress queue congestion information @ ingress
  - Programmable packet generation
  - Dataplane advanced flow control
- Case for source quenching in datacenter networks
  - One of many use cases of the programmable building blocks
  - Source-pause and its interaction with e2e congestion control schemes

- Contributions from
  - Intel: Anurag Agrawal, Ashutosh Agrawal, Jeremias Blendin, Remy Chang, Evan Cheshire, Changhoon Kim, JK Lee, Georgios Nikolaidis, Rong Pan, Mickey Spiegel, Han Wang
  - Yale University: Robert Soule
  - University of Wisconsin, Madison: Aditya Akella, Yanfang Le, Qingkai Meng

Egress queue congestion information @ ingress pipe

- Queue info often available at
  - Post-queueing @ egress pipe
  - Or, pre-queueing but after ingress forwarding decision
- New: Tofino2 provides egress queue info at ingress MAU, prior to routing/queueing
  - Ingress has visibility of all 4 egress pipes
- Control plane configures a set of egress queues to monitor
- Q update trigger modes
  - Every change vs. color change
- Use cases
  - Congestion-aware routing
  - Source quenching
  - …



Egress queue congestion information @ ingress pipe

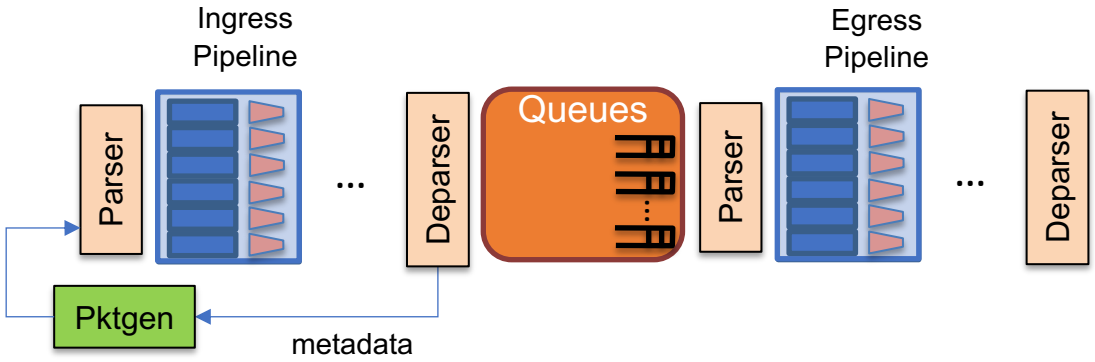
- Queue info often available at
  - Post-queueing @ egress pipe
  - Or, pre-queueing but after ingress forwarding decision
- New: Tofino2 provides egress queue info at ingress MAU, prior to routing/queueing
  - Ingress has visibility of all 4 egress pipes
- Control plane configures a set of egress queues to monitor
- Q update trigger modes
  - Every change vs. color change
- Use cases
  - Congestion-aware routing
  - Source quenching
  - …

```
// 1024 queues monitored
// q_register values are updated by a separate thread
Register<...>(1024) q_register;
control ingress(...) {
   RegisterAction<...>(q_register) read_q_reg = {
      void apply(inout bit<32> value, out bit<32> rv) {
         rv = value; }
   };
   action get_qdepth(bit<10> idx) {
      ig_md.eg_qdepth = read_q_reg.execute(idx);}
   table q_select {
      key = {
         ig_intr_tm_md.ucast_egress_port : exact;
         ig_intr_tm_md.qid : exact;
      }
      actions = {get_qdepth; NoAction;}
      size = 1024;
      default_action = NoAction();
   }
   apply {
      q_select.apply();
      // ig_md.eq_qdepth contains egress queue depth
   }
}
```

Disclaimer: P4 representation of the new features may change in the future

Programmable packet generation

- Tofino has ingress packet generator
  - p4 extern: Pktgen()
- Trigger conditions
  - Timer (one-time, periodic)
  - Link-down event
  - Packet recirculation
- New: Tofino2 allows any ingress pkt to trigger
  - P4 can define an arbitrary event (stateful/stateless) and trigger Pktgen
  - w/ per-pkt metadata
- Usecase: generate control packets such as
  - Source quench
  - RoCEv2 CNP (Congestion Notification Packet)
  - Hop-by-hop flow control
- Benefit
  - Reuse ingress tables for routing & multi-pathing of control packets
  - w/o recirculation → minimal feedback delay



Programmable packet generation

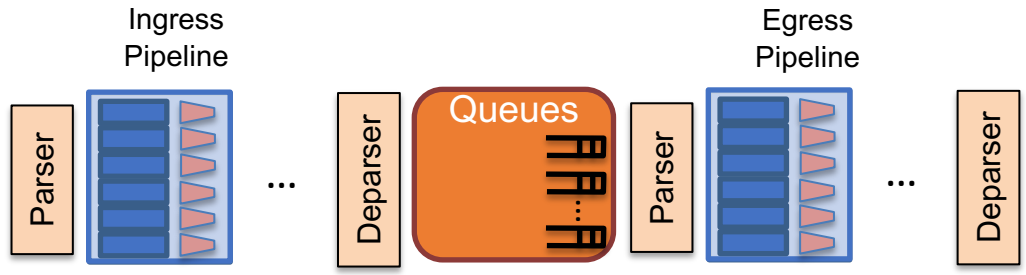
- Tofino has ingress packet generator
  - p4 extern: Pktgen()
- Trigger conditions
  - Timer (one-time, periodic)
  - Link-down event
  - Packet recirculation
- New: Tofino2 allows any ingress pkt to trigger
  - P4 can define an arbitrary event (stateful/stateless) and trigger Pktgen
  - w/ per-pkt metadata
- Usecase: generate control packets such as
  - Source quench
  - RoCEv2 CNP (Congestion Notification Packet)
  - Hop-by-hop flow control
- Benefit
  - Reuse ingress tables for routing & multi-pathing of control packets
  - w/o recirculation → minimal feedback delay

```
control ingress(...) {
   // assume q_get and q_select from the previous slide
   action set_pktgen_pre() {
      // prefix hdr to carry per-pkt metadata
      // ip addrs to swap for generating quench packet
      hdr.pktgen_pre.data[31:0] = hdr.ipv4.src_addr;
      hdr.pktgen_pre.data[63:32] = hdr.ipv4.dst_addr;
      ig_intr_dprsr_md.pktgen = 1;
   }
   apply {
      q_select.apply(); // get eg_qdepth
      if (ig_md.eq_qdepth > 100) {
         set_pktgen_pre();
      }
   }
}
control iDprsr(...) {
   Pktgen() pgen;
   apply {
      if (ig_dprsr_md.pktgen == 1) {
         pgen.emit(hdr.pktgen_pre); //provide prefix hdr
      }
      packet.emit(hdr.ethernet);
   }
}
```

Disclaimer: P4 representation of the new features may change in the future

Dataplane advanced flow control

- Tofino allows per-pkt Q selection for pkt's queueing
- New: Tofino2 allows per-pkt control of any queue, =/= pkt's queue
  - Ingress pkt controls any Q in all 4 pipes
  - Egress pkt controls any Q in the same egress pipe
- Two AFC modes (per-q CPU config)
  - Xon/Xoff (resume/pause) target queue
    - Similar to PFC but control by P4
  - add/subtract queue shaper credit
    - BPS or PPS based on shaper config
- Use cases
  - Rotating Strict Priority scheduler for
    - Approximate Fair Queue @ NSDI'18
    - Calendar Queue @ NSDI'20
  - Dataplane rate control




```
control ingress(...) {
   apply {
      // Rotating Strict Priority scheduler example:
      // Xoff (pause) a high-priority Q once empty
      q_select.apply(); // get depth of target q
      if (target_q.eg_qdepth == 0) {
         ig_intr_dprsr_md.qfc_mode = 1; // 1 is for AFC
                                        // 0 is for PFC
         ig_intr_dprsr_md.port_id = target_q.port_id,
         ig_intr_dprsr_md.queue_id = target_q.queue_id,
         ig_intr_dprsr_md.credit = 1; // 1 is for Xoff
                                      // 0 is for Xon
                                      // integer for credit
      }
   }
}
```

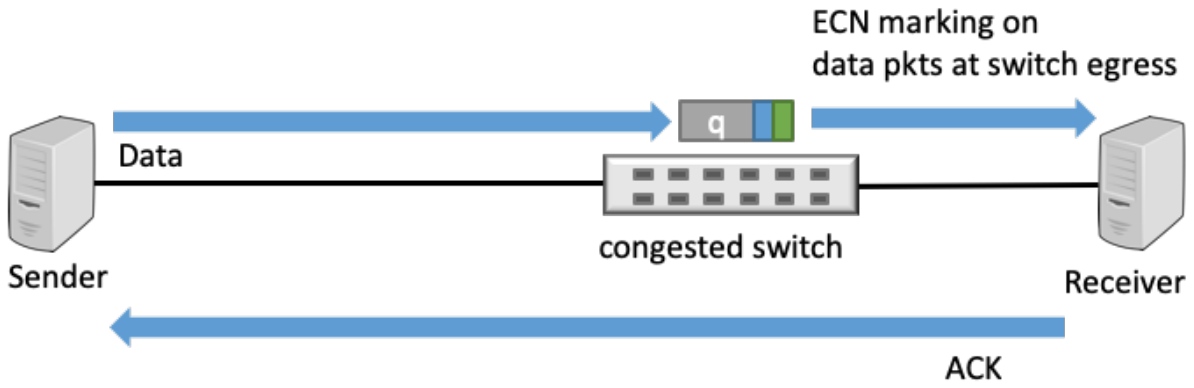
Disclaimer: P4 representation of the new features may change in the future

Agenda

- Programmable building blocks for adv. congestion/flow control @ switches
  - Egress queue congestion information @ ingress
  - Programmable packet generation
  - Dataplane queue flow control
- Case for source quenching in datacenter networks
  - One of many use cases of the programmable building blocks

Queueing delay coupled in feedback delay

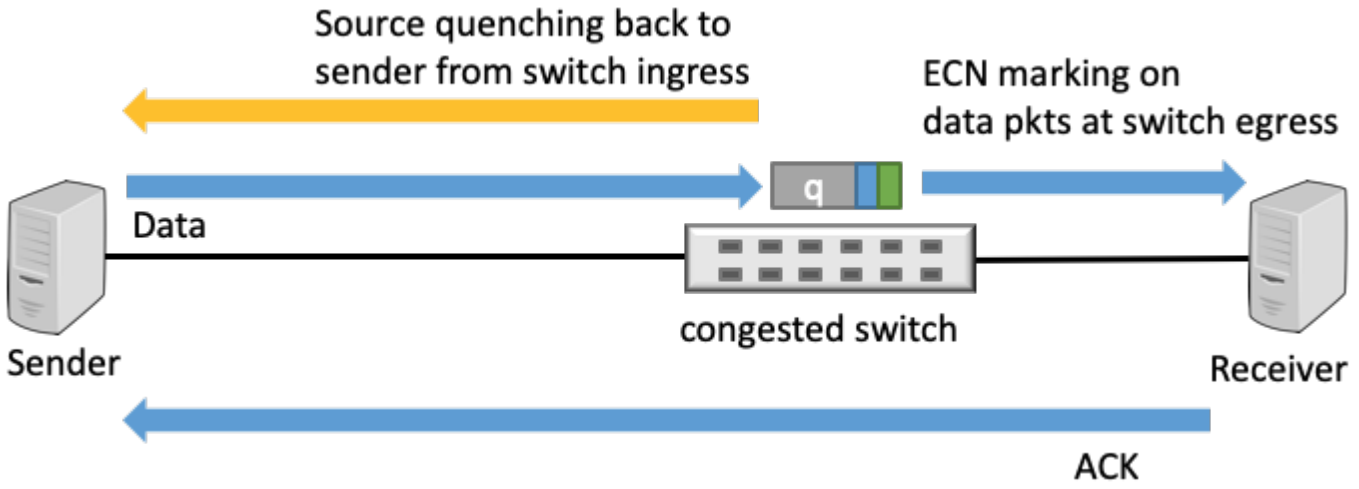
- ECN, conventional congestion feedback suffer from large queuing delay
  - New flows starting in the middle of congestion experience large queueing delay
  - Datacenter congestion-free RTT: 10~20 us
  - Incast queuing delay: up to 1-2ms
    - Time for the 1st pkt of a new flow to reach Receiver and trigger CNP



Case for source quench

- Source quench didn’t take off in Internet
  - Due to lack of trust and inter-operability
- Time to revisit in datacenter
  - Can trust a quench notification, like ECN
  - New building blocks such as
  - Congestion information @ ingress
  - Programmable packet generation

- Fastest possible feedback
  - Complete decoupling of feedback path from on-going congestion



Source quench when/where/what

- When to generate quench?
  - Qdepth above threshold
  - Tail drop → NACK
- Where?
  - At switch ingress for fast detection and notification
- What to do with quench notification?
  - DCQCN CNP → reduce rate
  - NACK → fast retx
  - Source pause, as opposed to PFC that causes HoL blocking
  - Carry addition info: INT for HPCC, fair rate for RCP, # of competing flows

Case for source *pause*

- Switch can easily compute time to drain the queue down to target depth
  - E.g., target depth = ECN threshold used by congestion control
  - Avoid queue underrun
- While sources are paused, the queue will drain and the in-flight packets deliver congestion info to the sources via ACKs/CNPs
  - Underlying congestion control decides the TX rate/window to use upon resume

- Source pause can co-exist with underlying transport & congestion control w/ minimal interactions
  - ECN-aware pause time computation @ switch
  - Pause-aware retx timer config @ NIC

Sponsored By

Google intel STORDIS The Open Networking Expert

Thank You

Contact: jk.lee@intel.com