IETF '87

PIE: A lightweight latency control to address the bufferbloat problem issue

Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Prabhu, Vijay Subramanian, Fred Baker and Bill Ver Steeg

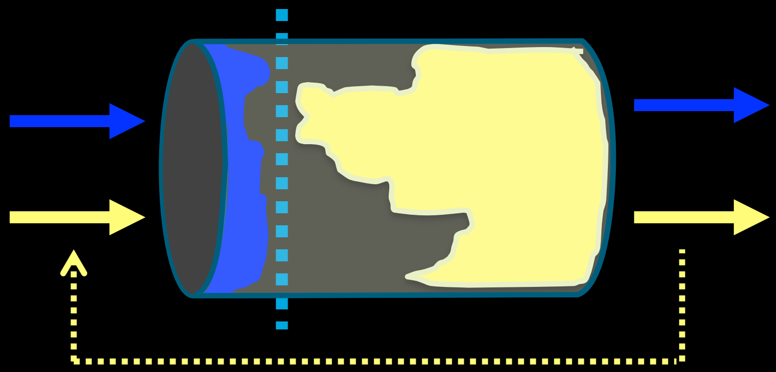
July 30, 2013

The Problem of Buffer Bloat

- Causes of the buffer bloat:
  - Sheer volume of Internet traffic: explosion of internet traffic
  - Cheap memory: customers want more memory to avoid packet drops
  - Nature of TCP: the TCP protocol can consume all buffers available
  - No efficient queue managements: no simple and effective algorithms
- Lack of a robust, consistent solution will cause:

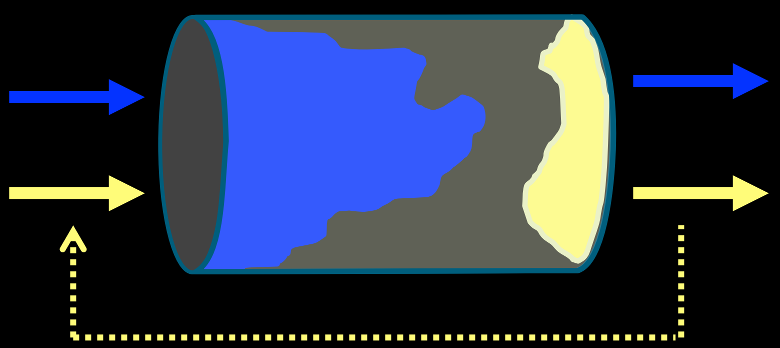
Control Average Delay and Allow Big Burst

Current Design



- Large TCP flows occupy most buffer
- Feedback signals are sent when buffer occupancy is big
- Average delay is consistently long
- Little room left for sudden burst

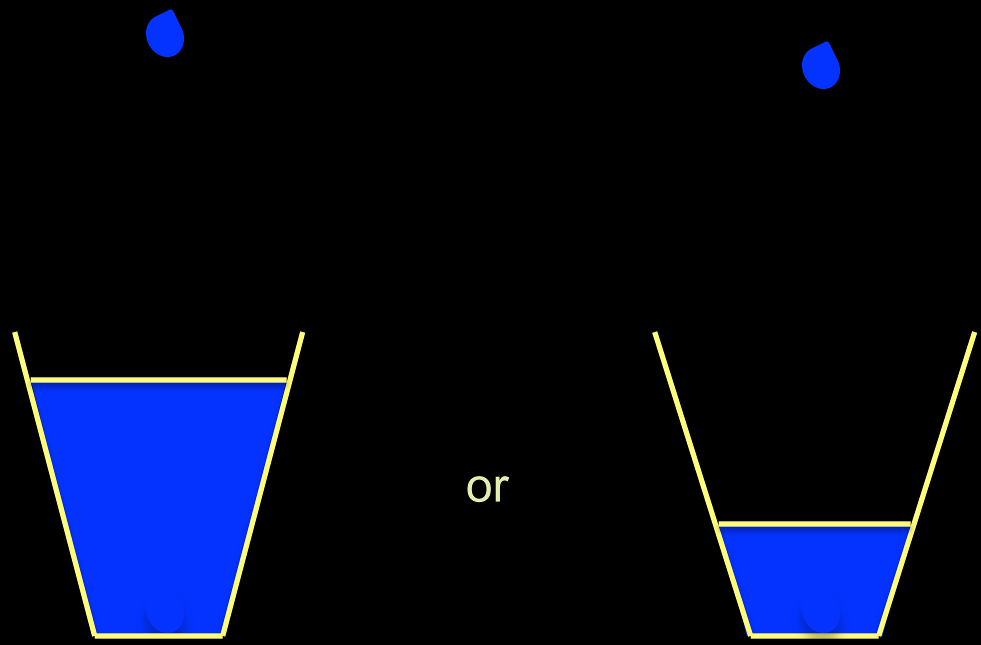
Future Goal



- Large TCP flows occupy small buffer
- Feedback signals are sent early
- Average delay is kept low
- Much room left for sudden burst

Water Level in a Leaky Bucket: An Analogy

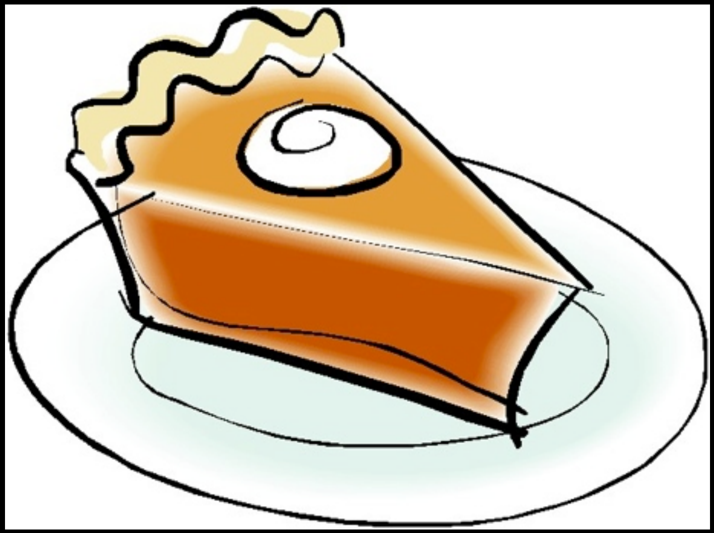
water level can
stay high
If arrival rate =
departure rate



water level can
also kept low if
arrival rate =
departure rate

big buffer (bucket size) does not have to imply
high average delay (standing water level)

Solution Maybe



As Easy As PIE!

Goal #1: Controlling Delay instead of Queue Length

- Control latency instead of queue length
  - Queue sizes change with link speed and estimation of RTT
  - Delay is the key performance factor that we want to control

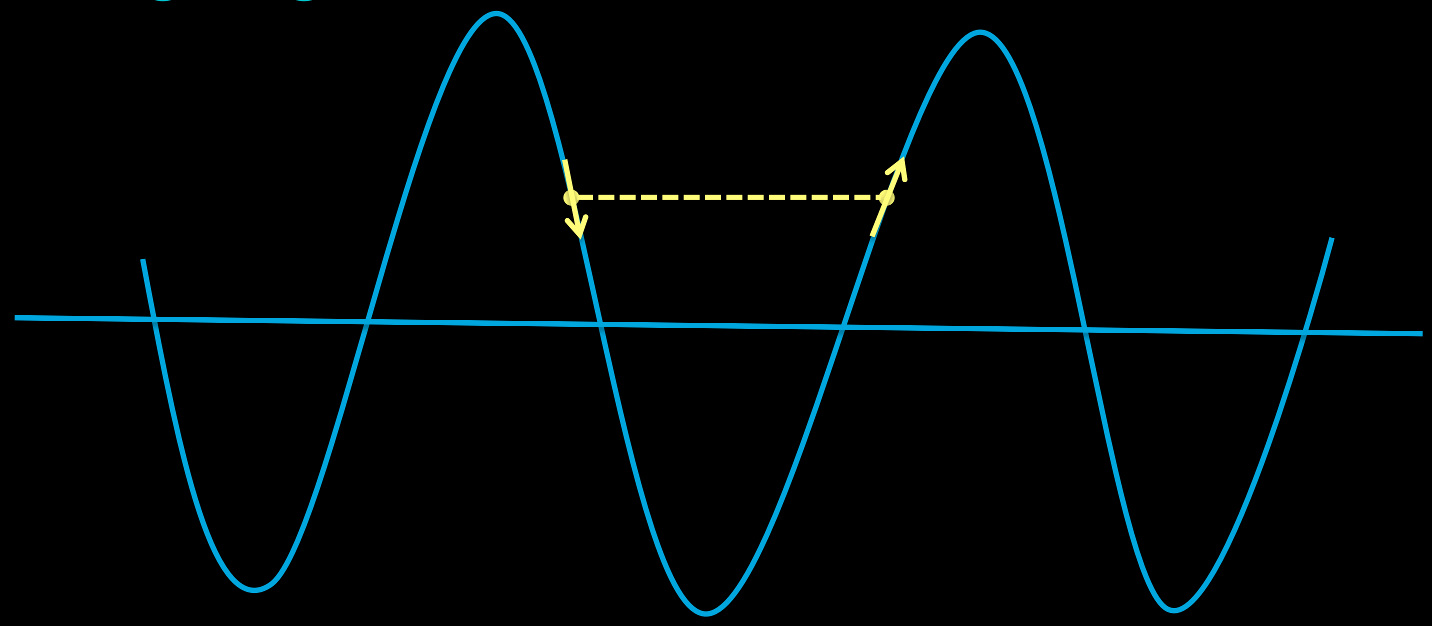
- Delay bloat is really the issue. If delay can be controlled to be reasonable, buffer bloat is not an issue. As a matter of fact, a lot of customers want MORE and MORE buffers for sudden bursts

Goal #2:
Simple Design and Low Operational Overhead

- Design a drop-at-enque algorithm like RED
  - Drops at deque are costly and waste network resources
  - Make deque timing unpredictable

- The algorithm should be simple, easily scalable in both hardware and software
  - Need to work with both UDP and TCP traffic

Goal #3:
Achieving High Link Utilization and Maintain Stability



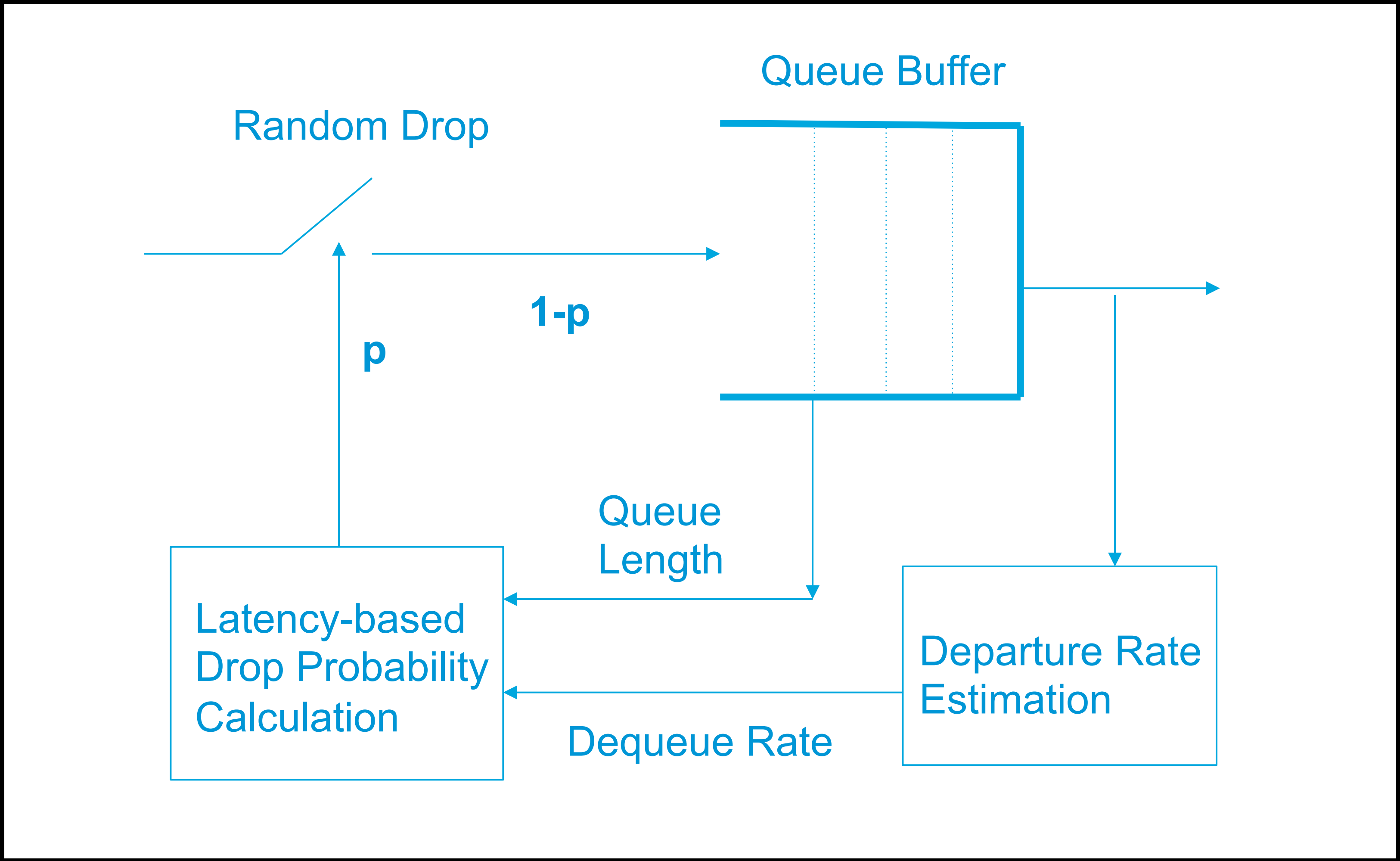
- Traditionally drops/marks increase as the queue lengths increase (longer delays), which could result in wide swing delay variation
- Crucial! Knowing the direction of the changing latency, we can increase stability and modulate the drops/marks intensity to reduce latency and jitter.

The design of PIE

a and b are chosen via control analysis

- Upon every packet arrival
  - randomly drop a packet based on drop_prob calculated below
- Every T_{update} interval
  - estimated_delay, est_del = queue_length/depart_rate
  - drop_prob += a*(est_del – target_delay) + b* (est_del – est_del_old)
  - est_del_old = est_del;
  - depart_count = 0;
- In a measurement cycle
  - Upon a packet's departure: depart_count += deque_packet_size;
  - if dq_count > deq_threshold then
  -     depart_rate = deqart_count/(now-start);
  -     dq_count = 0; start = now;

The block diagram of PIE



PIE's Work Update (based on community's feedback)

- Fixed PIE's initialization issue with very slow speed links
  - in slow speed links, original PIE may not get a valid rate measurement in one update interval, fixed the bug
- Fixed the initialization behavior of One single TCP with long RTT
  - original PIE is fragile in the above scenario. Added robustness into the design

PIE's Summary and Future Work

- Simulation results, testbed experiments and theoretical analysis show that PIE is able to
  - Control low latency across different applications
  - Simple to implement
  - Achieve high link utilization and maintain stability across various traffic scenarios
  - Self tune its parameters

- Future work: further simplify PIE, class-based PIE

Thank you.