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

Rong Pan, Preethi Natarajan, Chiara Piglione, Mythili Prabhu,

Fred Baker and Bill Ver Steeg

November 5, 2012

The Problem of Buffer Bloat

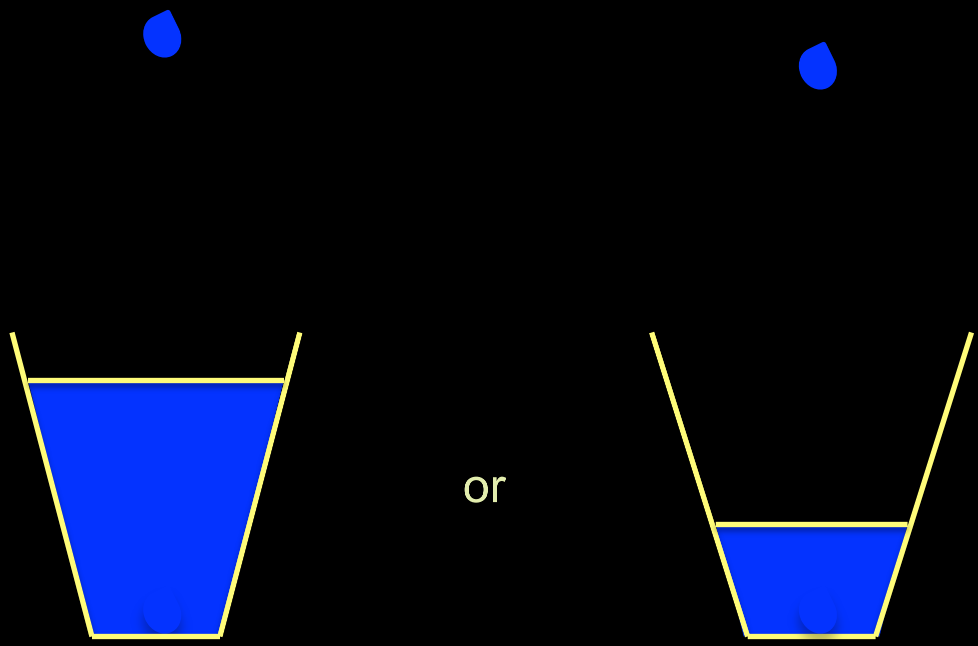
- Causes of the buffer bloat:
  - Sheer volume of Internet traffic: explosion of video traffic
  - Cheap memory: customers want more memory to avoid packet drops
  - Nature of TCP: the TCP protocol can consume all buffers available
  - No efficient queue managements: no simple and effective algorithms

- Lack of a robust, consistent solution will cause:

Delay Bloat → **Real Time Application Impairment** → **Poor Quality of Experience**

Water Level in a Leaky Bucket: An Analogy

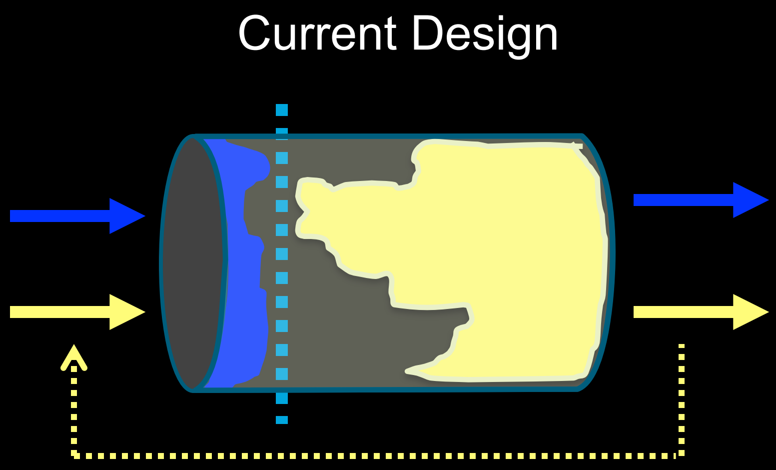
water level can
stay high
If arrival rate =
departure rate

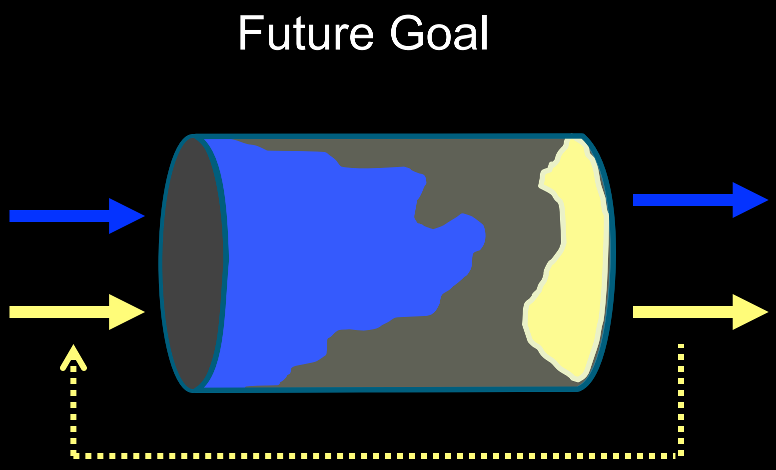


water level can
also kept low if
arrival rate =
departure rate

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

Control Average Delay and Allow Big Burst





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

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

A Brief Dive into PIE

As Easy As PIE!

Goal #1:
Controlling Delay instead of Queue Length

- From what learned from CoDel, control delay 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

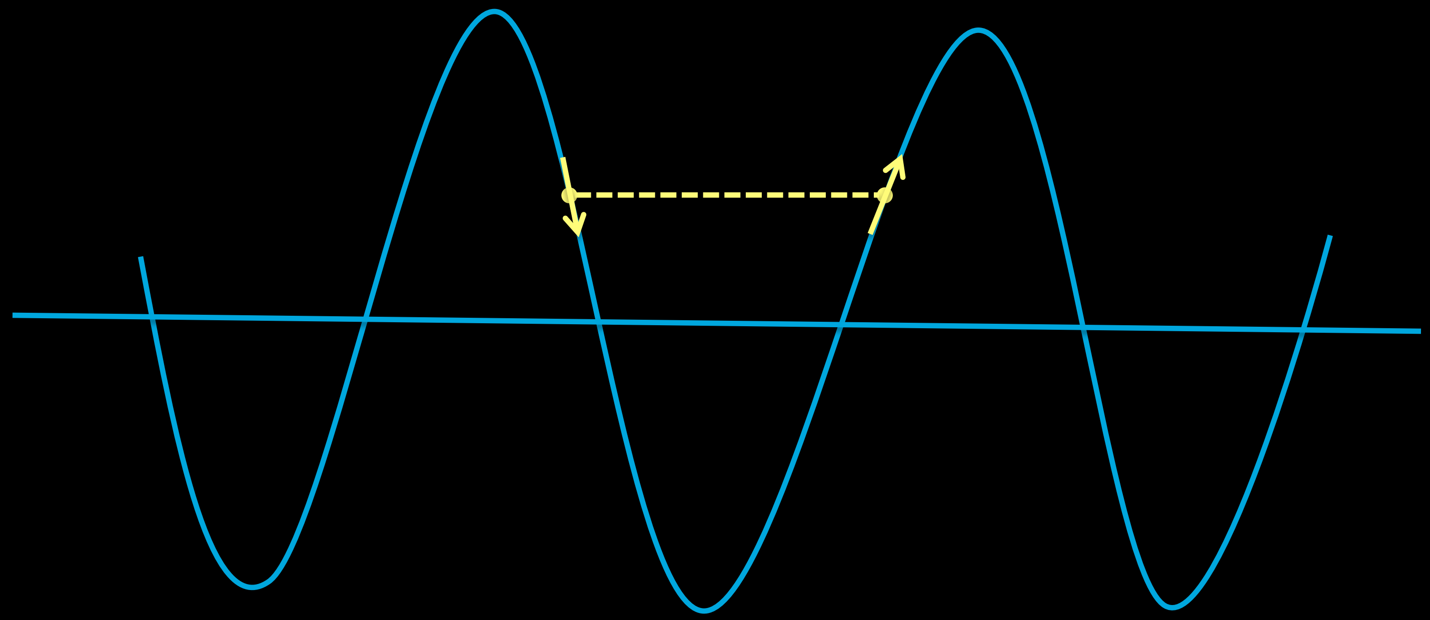
- Delay bloat is really the issue. If delay can be controlled to be reasonable, buffer bloat is not an issue. As a matter of fact, a lot of customers want MORE and MORE buffers for sudden bursts

Goal #2: Simple Design and Low Operational Overhead

- Design a drop-at-enque algorithm like RED, not drop-at-deque algorithm like CoDel
  - Drops at deque are costly and waste network resources
  - Require memory speed up: e.g. 10:1 oversubscription would require 20x bandwidth speed up to the memory

- The algorithm should be simple, easily scalable in both hardware and software
  - Need to work with both UDP and TCP traffic, no need of extra queue (which implies extra hardware cost)

Goal #3:
Achieving High Link Utilization and Maintain Stability



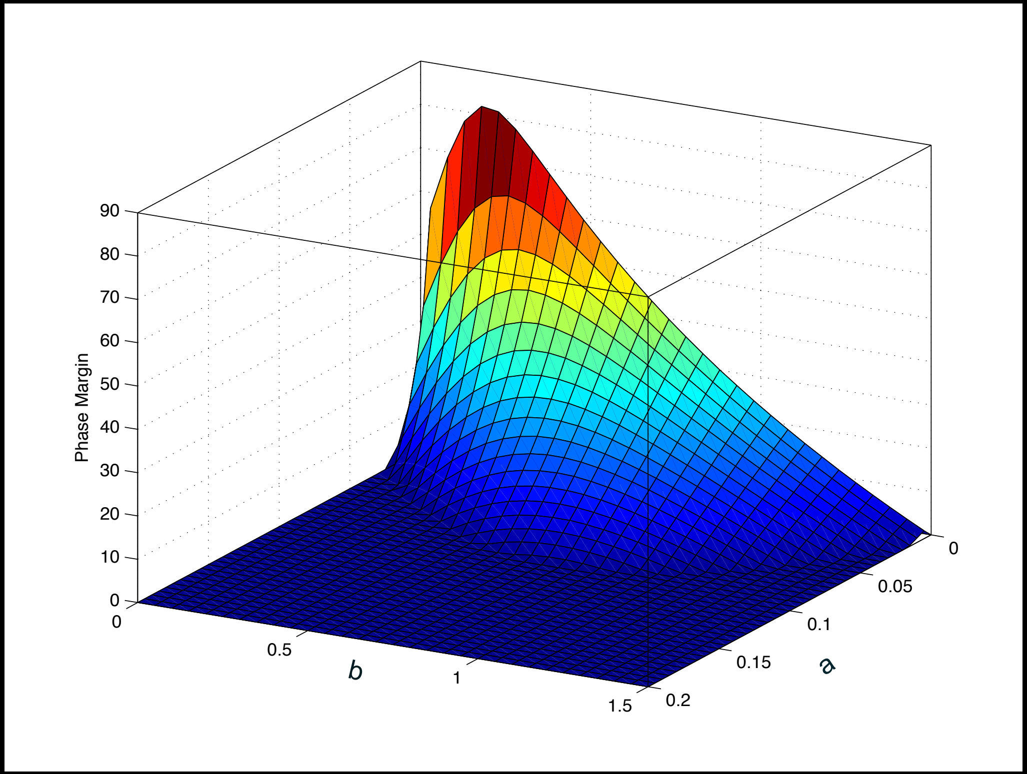
- Traditionally drops/marks increase as the queue lengths increase (longer delays), which could result in wide swing delay variation
- Knowing the direction of the changing latency, we can increase stability and modulate the drops/marks intensity to reduce latency and jitter.

The design of PIE

- Upon every packet departure
  - depart_count += deque_packet_size;
- Every T_{update} interval
  - estimated_delay, est_del = queue_length/depart_count*Tupdate
  - drop_prob += a*(est_del – target_delay) + b* (est_del – est_del_old)
  - est_del_old = est_del;
  - depart_count = 0;
- Upon every packet arrival
  - randomly drop a packet based on drop_prob

a and b are chosen via control analysis

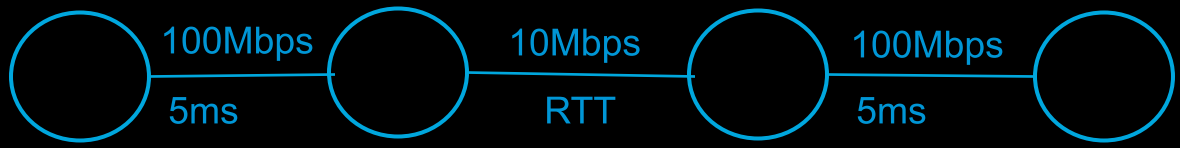
Parameters are Chosen using Feedback Control Analysis to Ensure Stability



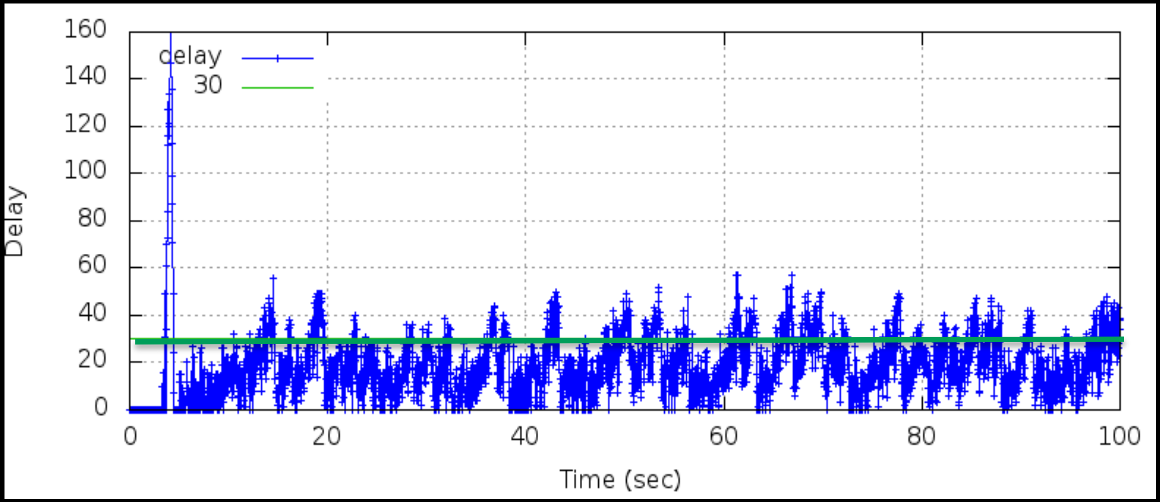
Parameters are self-tuning, no configuration required

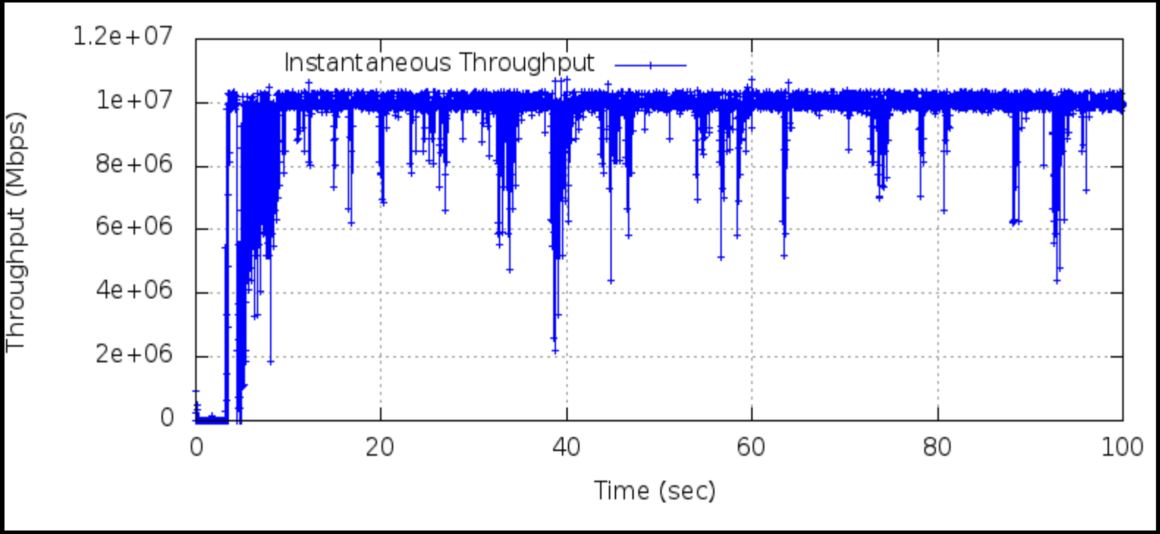
Linux Lab Test Topology Setup

- Congestested Link Bw: 10Mbps
- Packet Size 1.5KB
- TCP: Sack1, RTT: 100ms
- ECN is not enabled in all tests
- Target_Del: 30ms
- Linux Version: 3.6



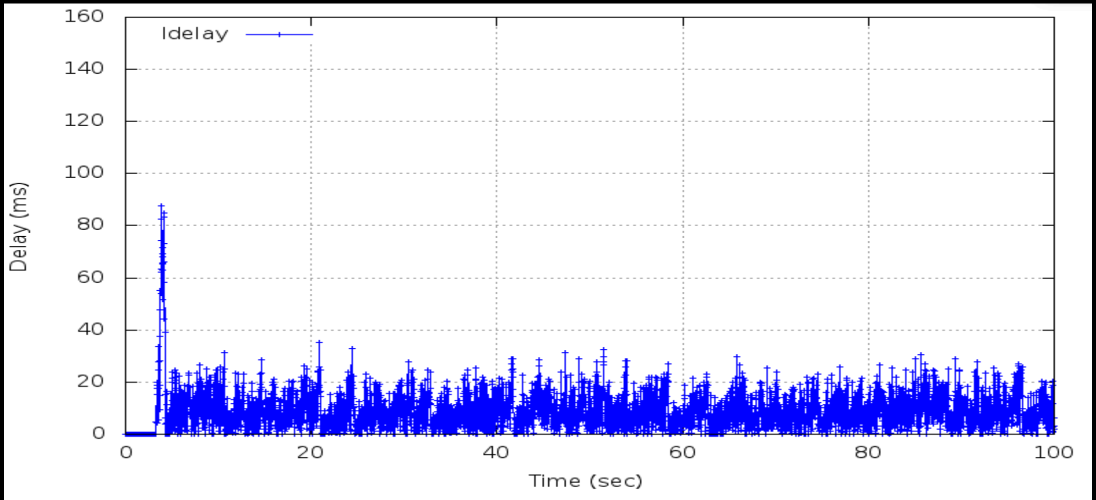
PIE – Low Load, 5 TCP Flows

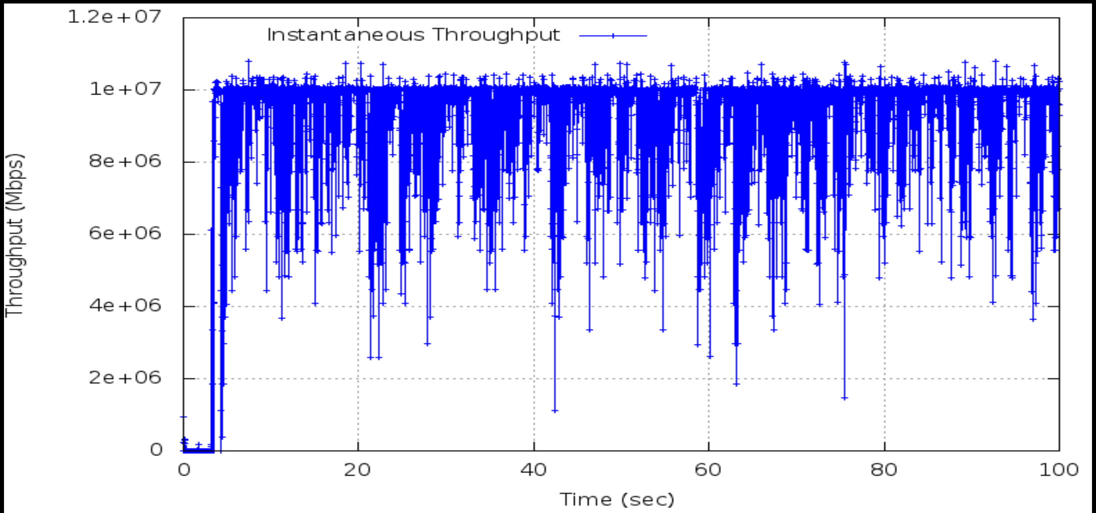




Under low load case, TCP Sawtooth is observable. However, PIE can regulate TCP flows so well that the link is close to full capacity while maintaining low latency

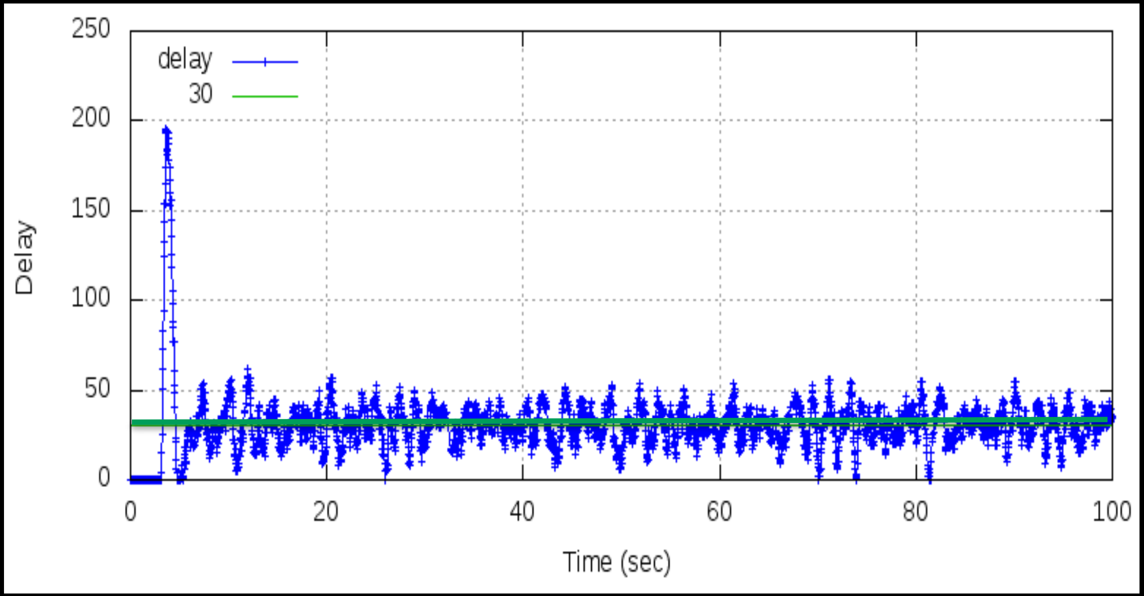
CoDel – Low Load, 5 TCP Flows

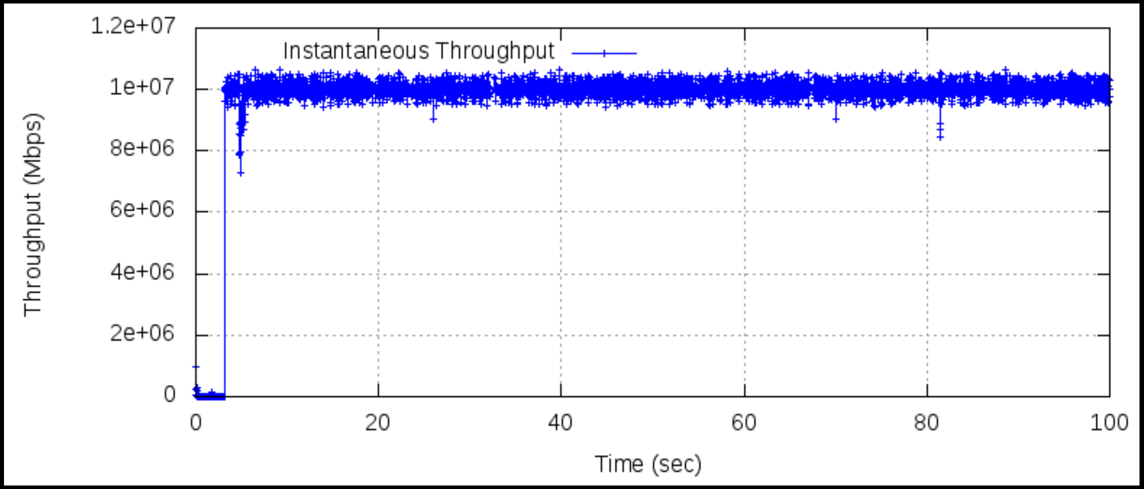




CoDel is also able to control latency. However, low latency is achieved at the expense of losing throughput

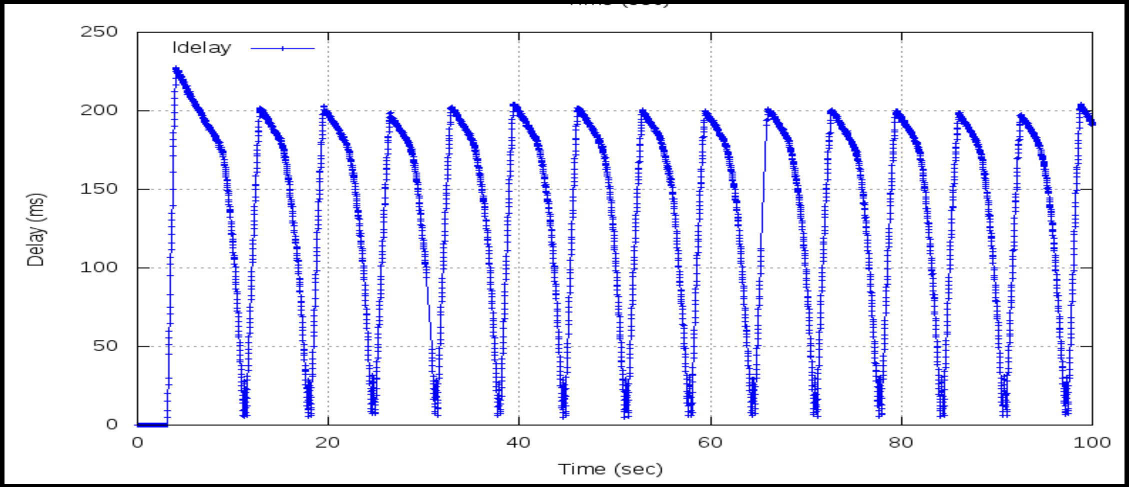
PIE – Mix Traffic, 5 TCP Flows + 2 UDP Flows (UDP 6Mbps each)

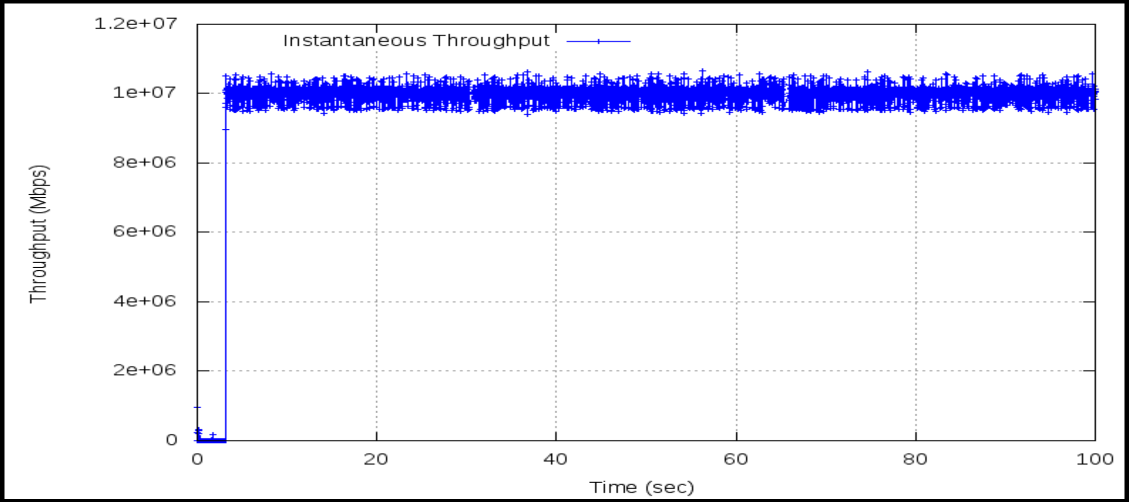




Even if UDP traffic is added, PIE handles the situation smoothly. Latency varies around the desired target value.

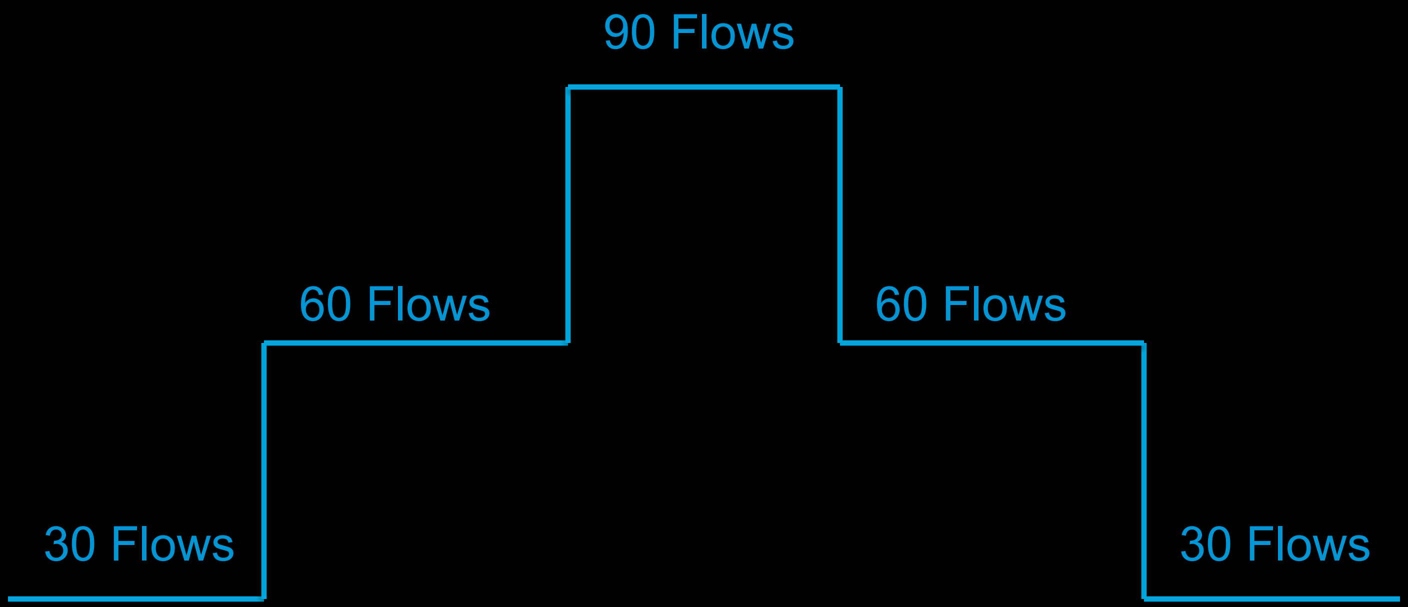
CoDel – Mix Traffic, 5 TCP Flows + 2 UDP (UDP 6Mbps each)



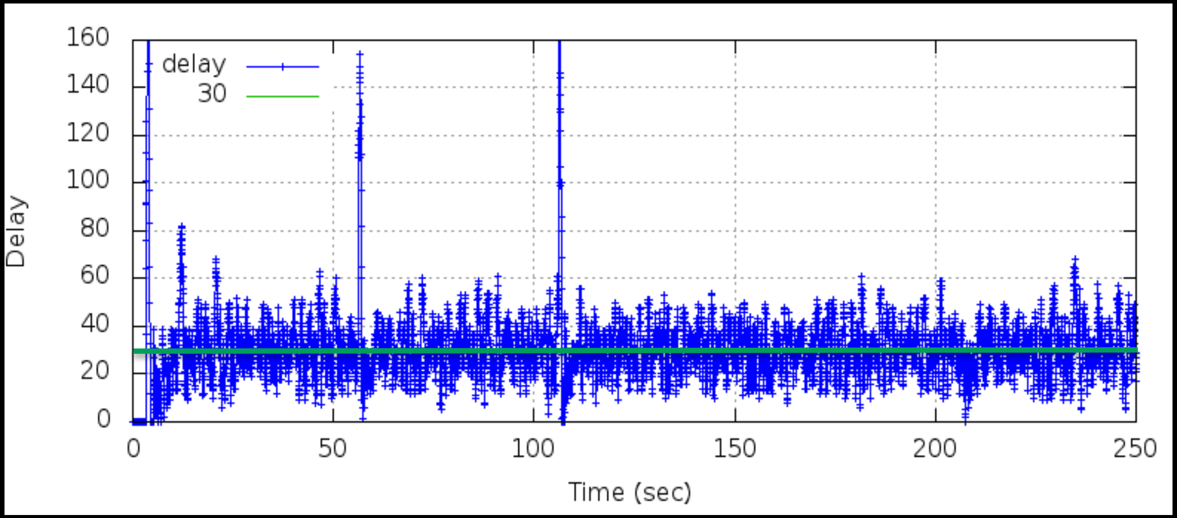


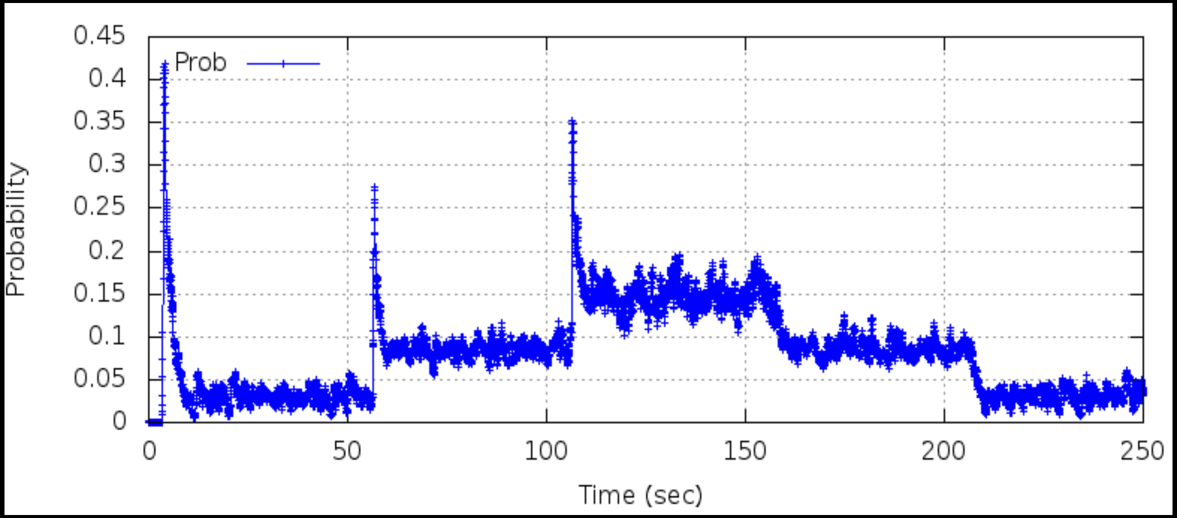
CoDel can't control the mixture of traffic. There is a suggestion to use a separate queue to handle UDP, i.e. fq_codel. We believe that, if one algorithm can handle both cases, there is no reason to add extra cost of another queue.

Varying TCP Traffic Intensity on 10Mbps Link



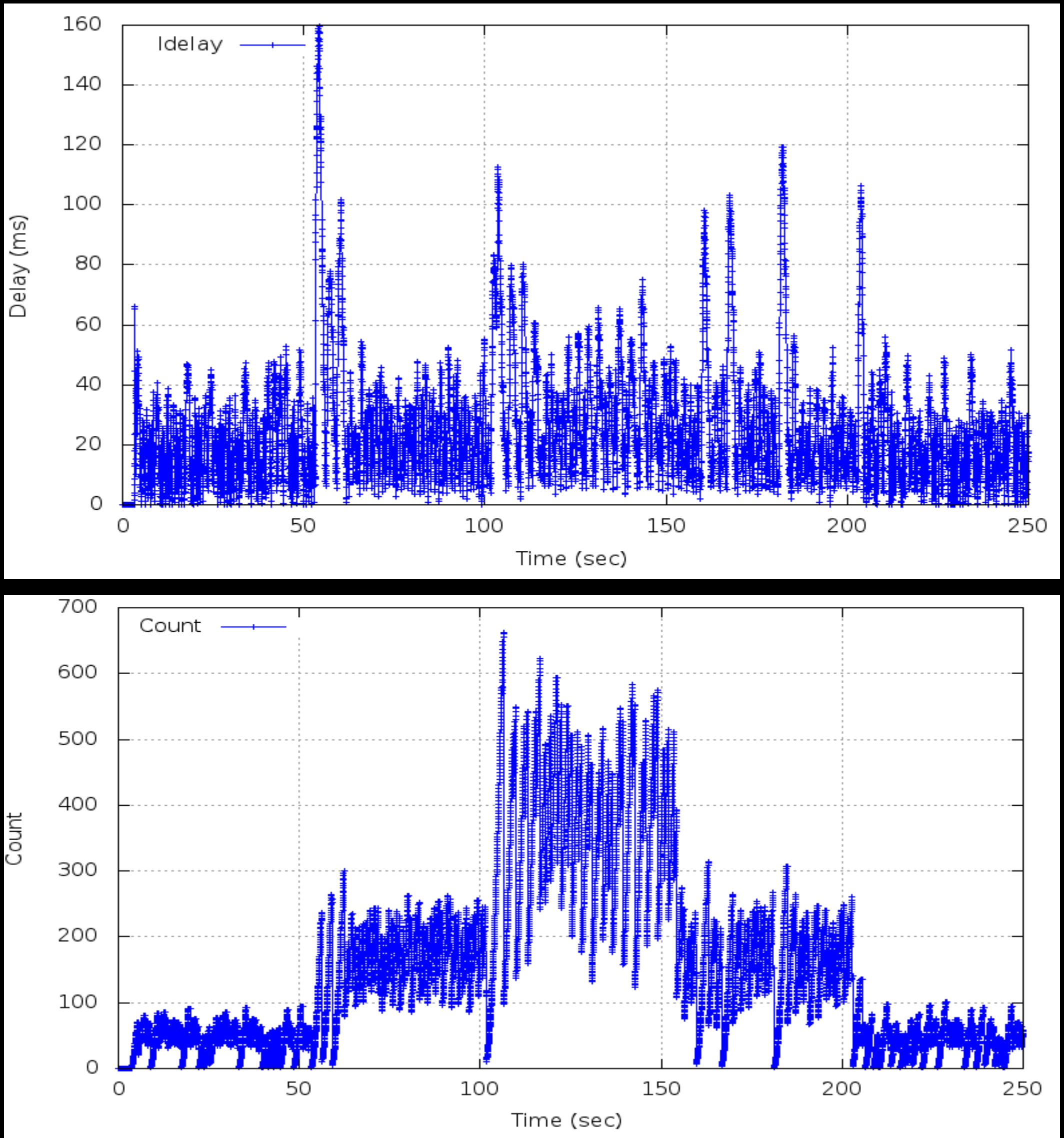
PIE – Able to Control Latency Consistently





Regardless of traffic intensity, PIE keeps latency around the target value by adjusting drop probability accordingly. The feedback loop is in tight control.

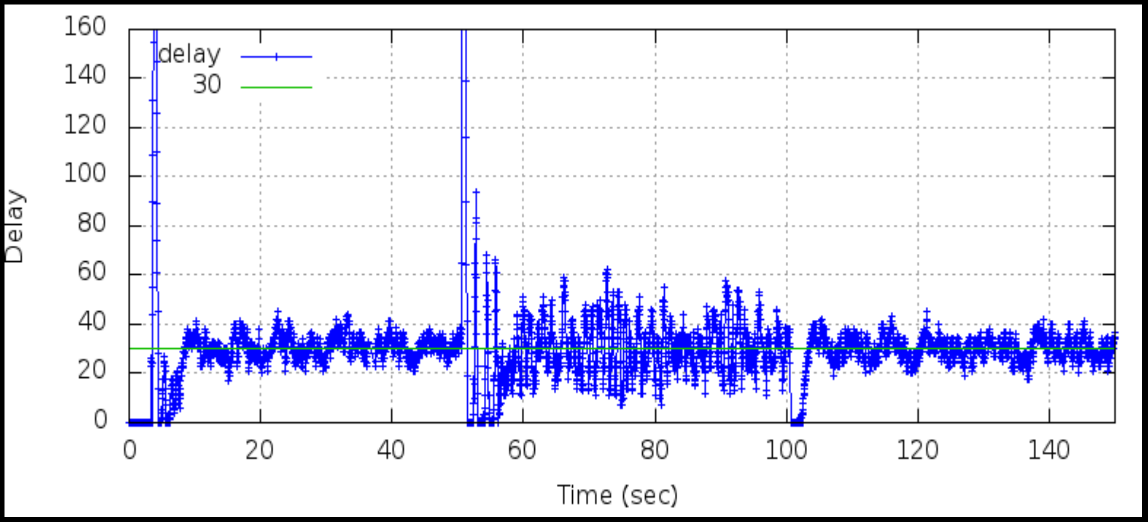
CoDel – Varying Traffic Intensity

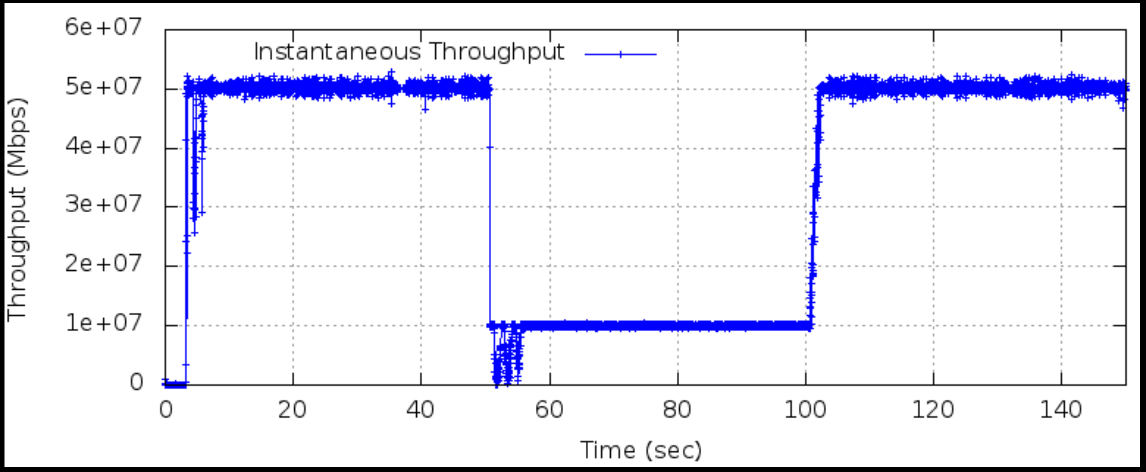


CoDel does not work well under heavy load. Their control parameter, count, oscillates widely under those situations. The feedback loop is not stable.

Varying Link Capacity, 100 TCP Flows

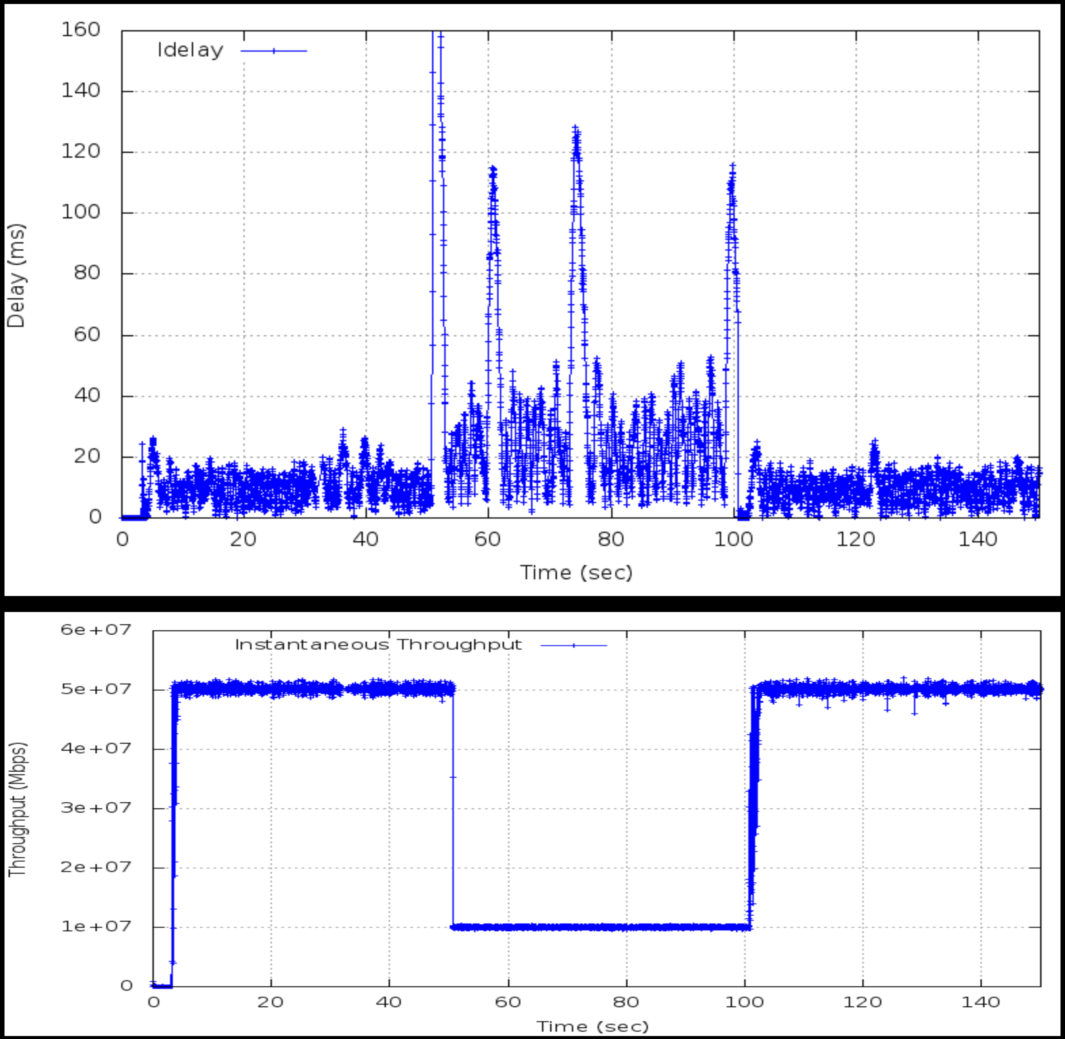
PIE – Good Control of Latency





When the queue draining rate dips from 50Mbps to 10Mbps, PIE is able to keep the latency low throughout the process

CoDel – Lose Control during the Process



When the queue draining rate dips from 50Mbps to 10Mbps, CoDel can't keep the latency low throughout the process

Summary & Future Work

- Simulation, theoretical analysis and lab results show that PIE is able to
  - Ensure low latency under various congestion situations
  - Achieve high link utilization and maintain stability consistently
  - A light-weight, enque-based algorithm that works with both TCP and UDP traffic. No memory speed up required
  - Self tune its parameters

- More PIE extensive evaluations and release to the community

Thank you and Questions?