A PCC-Vivace Kernel Module

PRESENTED BY TOMER GILAD
Internet Congestion Control

Senders

Data

Acks

The Internet

Receivers
Internet Congestion Control

- Senders
  - Choose when to send
- The Internet
- Receivers
  - Data
  - Acks
Internet Congestion Control

Senders
Choose when to send

Data
The Internet

Acks

Receivers
Pretty passive, send acks
Internet Congestion Control

Senders
Choose when to send

Data

The Internet
Tries to get packets through but...

Acks

 Receivers
Pretty passive, send acks
Internet Congestion Control

Senders

Choose when to send

Data

The Internet

Tries to get packets through but...

small buffers
large buffers
random loss
competing flows
WiFi links
LTE links

Acks

Receivers

Pretty passive, send acks
Internet Congestion Control

Senders
Choose when to send

Data
ACKs

The Internet
Tries to get packets through but...
small buffers
large buffers
random loss
competing flows
WiFi links
LTE links

CUBIC:
Backs off on a single loss

Receivers
Pretty passive, send acks
Internet Congestion Control

Senders
Choose when to send

The Internet
Tries to get packets through but...
- small buffers
- large buffers
- random loss
- competing flows
- WiFi links
- LTE links

CUBIC:
- Backs off on a single loss
- Fills buffers

Receivers
Pretty passive, send acks
Internet Congestion Control

Choose when to send

Pretty passive, send acks

Tries to get packets through but...
- small buffers
- large buffers
- random loss
- competing flows
- WiFi links
- LTE links

BBR:
Explicit model, better performance
Internet Congestion Control

Senders
Choose when to send

Data
The Internet
Tries to get packets through but...
small buffers
large buffers
random loss
competing flows
WiFi links
LTE links

Acks

receivers
Pretty passive, send acks

BBR:
Explicit model, better performance
Fills to limited extent, drains periodically
High loss rate
Internet Congestion Control

**Senders**
- Choose when to send

**Receivers**
- Pretty passive, send acks

**The Internet**
- Tries to get packets through but...
  - small buffers
  - large buffers
  - random loss
  - competing flows
  - WiFi links
  - LTE links

**PCC:**
1. Monitor performance at various rates
2. Adapt rate in the utility-maximizing direction
PCC Utility Framework

PCC uses monitor intervals

Rate $r_1$  Rate $r_2$  Rate $r_3$

1 RTT
PCC Utility Framework

- **Utility Function**: $U(Rate, ...)$
- **Observed Statistics**: Throughput, Latency, Latency change, Loss Rate
- **Network**: Input Rate

The image illustrates the relationship between network throughput, latency, and latency change, and their impact on the utility function.
PCC Utility Framework

Observed Statistics
- Throughput
- Latency
- Latency change
- Loss Rate

Utility Function $U(Rate, ...)$

Causal relation
PCC Utility Framework

Observed
Statistics

Throughput
Latency
Latency change
Loss Rate

Utility Function

$U(Rate, \ldots)$

Causal relation
PCC Utility Framework

Example Utility Graph

Utility increases with throughput, no negative effects

Utility decreases due to latency or loss

Link Capacity
PCC Flexibility

We give two utility functions, Allegro and Vivace
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_A(r) = \frac{r}{1 + e^{100L}} - rL \]
PCC Flexibility

We give two utility functions, Allegro and Vivace

Positive reward diminishes with loss rate.

\[ U_A(r) = \frac{r}{1+e^{100L}} - rL \]
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_A(r) = \frac{r}{1+e^{100L}} - r\hat{L} \]

Positive reward diminishes with loss rate.

Penalty factor for loss.
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_V(r) = r \times (1 - \alpha \frac{dRTT}{dt} - \beta L) \]
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_V(r) = r \times (1 - \alpha \frac{dRTT}{dt} - \beta L) \]

Reward or penalty based on rate (will give a nice gradient)
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_V(r) = r \times (1 - \alpha \frac{dRTT}{dt} - \beta L) \]

Unit reward for sending

Reward or penalty based on rate (will give a nice gradient)
PCC Flexibility

We give two utility functions, Allegro and Vivace

\[ U_V(r) = r \times \left( 1 - \alpha \frac{dRTT}{dt} - \beta L \right) \]

- Unit reward for sending
- Reward or penalty based on rate (will give a nice gradient)
- Penalty factor for latency inflation. Can be extremely high to react quickly.
PCC Flexibility

We give two utility functions, Allegro and **Vivace**

\[ U_V(r) = r \times \left( 1 - \alpha \frac{dRTT}{dt} - \beta L \right) \]

- **Unit reward for sending**
- **Reward or penalty based on rate (will give a nice gradient)**
- **Penalty factor for loss. Determines maximum random loss allowed.**
- **Penalty factor for latency inflation. Can be extremely high to react quickly.**
PCC Flexibility

Other functions may work with other features:

- Functions based on jitter may work as scavengers
- Using latency directly on paths with known low-latency may give latency guarantees
- Maybe using latency directly to keep queues slightly full
PCC Rate Control

Observed Statistics
- Throughput
- Latency
- Latency change
- Loss Rate

Utility Function
- $U(Rate_1)$
- $U(Rate_2)$
- $U(Rate)$

Unknown Network

Gradient Ascent

Rate
PCC Rate Control

- **Startup**
  - Double sending rate each RTT.
  - Utility continues to increase

- **Probing**
  - A 4-RTT test. Tries 2 rates, one above and one below the current rate for 2 RTTs each.
  - Utility increases

- **Moving**
  - Change rate toward higher utility.
  - Utility increases

**Quickly reach within 50% of link capacity**

**Determine direction of increasing utility**

**Quickly move toward greater utility**
PCC Rate Control

**Startup**
Double sending rate each RTT.

**Utility**
- When utility decreases, return to last rate.

**Probing**
- A 4-RTT test. Tries 2 rates, one above and one below the current rate for 2 RTTs each.

**Moving**
- Change rate toward higher utility.

**Diagram**
- Utility continues to increase.
- Utility decreases.
- Double rate each RTT.
- When utility decreases, return to last rate.
PCC Rate Control

**Startup**
Double sending rate each RTT.

**Utility decreases**
Utility continues to increase

**Probing**
A 4-RTT test. Tries 2 rates, one above and one below the current rate for 2 RTTs each.

**Inconclusive Test**

**Moving**
Change rate toward higher utility.

**Utility decreases**

**Gradient**

**Rate**

**Utility(r)**
PCC Rate Control

**Startup**
Double sending rate each RTT.

**Probing**
A 4-RTT test. Tries 2 rates, one above and one below the current rate for 2 RTTs each.

**Moving**
Change rate toward higher utility.

Utility continues to increase

---

Utility decreases

---

Inconclusive Test

---

Utility increases

---

Utility decreases

---

Gradient

---

Utility(r)

---

Rate
Kernel Challenge: Packet-Rate Associations

User-space: Unique packet IDs, per-packet acks

PCC-Kernel: Approximate packet-rate association

- Unique packet IDs in acks
  - Result: Easy to know the rate at which packets were sent

- Acks aggregated, packets do not have unique IDs
  - Result: Hard to know which interval a packet was sent in, so rate may not be known.

Uncertainty bound, at most 20% of packets
Why not rate_samples?

Introduced with BBR

```c
struct rate_sample {
    u64 prior_mstamp; /* starting timestamp for interval */
    u32 prior_delivered;  /* tp->delivered at "prior_mstamp" */
    s32 delivered;       /* number of packets delivered over interval */
    long interval_us;    /* time for tp->delivered to incr "delivered" */
    long rtt_us;         /* RTT of last (S)ACKed packet (or -1) */
    int losses;          /* number of packets marked lost upon ACK */
    u32 acked_sacked;    /* number of packets newly (S)ACKed upon ACK */
    u32 prior_in_flight; /* in flight before this ACK */
    bool is_app_limited; /* is sample from packet with bubble in pipe? */
    bool is_retrans;     /* is sample from retransmission? */
    bool is_ack_delayed; /* is this (likely) a delayed ACK? */
};
```
Why not rate_samples?

The data overlaps
- a single packet’s result appears in multiple samples

Cannot configure timing
- Short samples would make it easier to group them into intervals
- Configurable-length samples could be used directly.

Additional information/configuration could make them more general:
- Includes no data about pacing rate (some algorithm’s actions)
- Lost and delivered packets may not be from the same timeframe (loss can be learned about later)
Kernel Challenge: Dealing with Approximations

The PCC kernel implementation makes more approximations:
- Packet-interval association
- Calculating the change in latency

Result: Unstable gradients
Set minimum rate change to 2%
Performance Results

Preliminary results from Pantheon
- Loss Resilience
- Buffer Bloat
- Loss at Convergence

Compared against:
- The userspace versions of Allegro and Vivace
- CUBIC
- BBR
High Loss Resilience

100Mbps, 30ms rtt, 750KB buffer

BBR is resilient up to 10% loss and continues to perform well at 15% loss

PCC-Kernel is resilient up to 5% loss

It’s CUBIC, what did you expect?
Low Buffer Bloat

BBR and CUBIC both fill buffers up to 1000KB.

The PCC variants have about 1ms of self-inflicted latency

100Mbps, 30ms rtt, 0% random loss
Loss at Convergence

- BBR converges to about 15% loss rate.
- For 10 or fewer flows, PCC variants have less than 5% loss rate, but they grow to about 10%.
- TCP maintains very low loss rate for many flows.

100Mbps, 30ms rtt, 750KB buffer
Conclusion

Promising initial results

We aren’t done yet:
  ◦ Still in early stages
  ◦ Improving sampling in the kernel
  ◦ Exposing utility function parameters to the application

Code is available on Github: https://github.com/PCCproject/PCC-Kernel
For more detailed information on PCC: http://www.pccproject.net