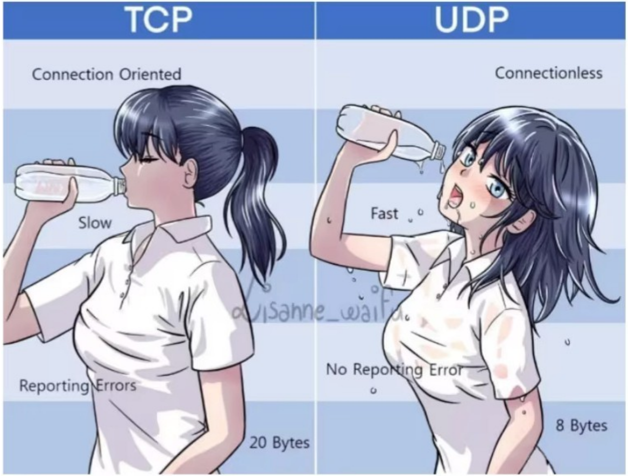
TCP BBR Fundamentals and Updates

December 2, 2024

Outline

TCP: The Reliable Workhorse of the Internet

- Connection-oriented: Establishes a dedicated connection between sender and receiver. (End to End)
- Reliable: Guarantees in-order delivery and retransmission of lost packets.
- Flow controlled: Prevents sender from overwhelming receiver.
- Congestion controlled: Avoids contributing to network congestion.
- Widely used: Foundation for many applications (HTTP, FTP, SMTP, etc.).



TCP BBR Fundamentals and Updates

TCP Transmission Mechanisms

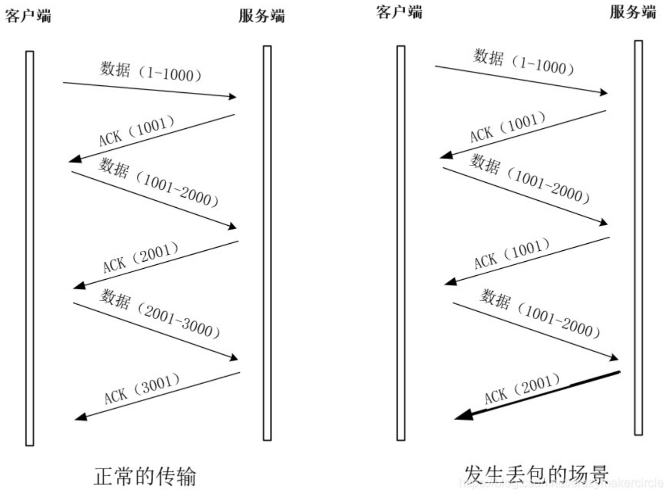
- Three-way handshake: Establishes a connection before data transfer.
- Sliding window: Controls the flow of data by limiting the amount of unacknowledged data in flight.
- Acknowledgements (ACKs): Confirm successful receipt of data packets.

Acknowledgements (ACKs)

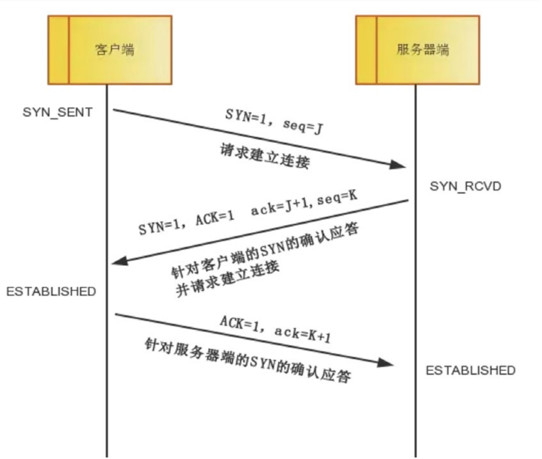
- When the data from the sender reaches the receiving host, the receiving host will return a notification that the message has been received. This message is called an acknowledgment.
- After the sender sends the data, it will wait for the acknowledgment from the other end. If there is an acknowledgment, it means that the data has successfully reached the other end. **Otherwise, there is a high possibility of data loss.**

Acknowledgements (ACKs)

- If the sender does not receive an acknowledgement within a certain period of time, it can assume that the data has been lost and resend it.
- Failure to receive an acknowledgement does not necessarily mean that the data has been lost.
- The target host must discard duplicate data packets. For this purpose, we introduce sequence numbers.


客户端
服务端
数据（1-1000）
ACK（1001）
数据（1001-2000）
ACK（2001）
数据（2001-3000）
ACK（3001）
正常的传输
客户端
服务端
数据（1-1000）
ACK（1001）
数据（1001-2000）
ACK（1001）
数据（1001-2000）
ACK（2001）
发生丢包的场景

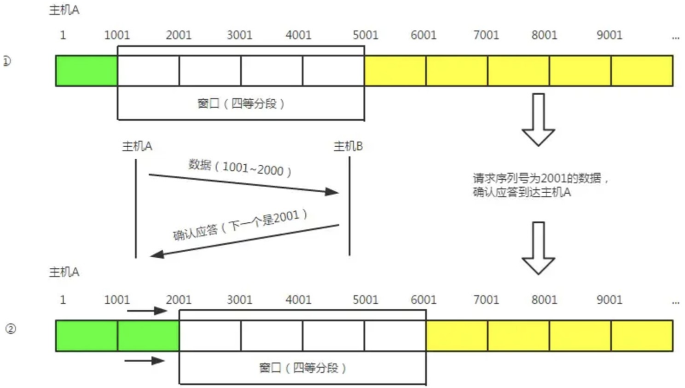
Three-way handshake



Sliding window

- TCP originally uses 1 segment as a unit, this method is slow in the long-round trip of the packet.
- To solve this problem, TCP introduced the concept of window. The confirmation response is no longer confirmed for each segment, but in larger units.
- The sending host does not need to wait for the confirmation response after sending a segment, but can continue to send.

Sliding window



在①的状态下，如果收到一个请求序列号为2001的确认应答， 那么2001之前的数据就没有必要进行重发，这部分的数据可以被过滤掉，滑动窗口成为②的样子。

Reno & Cubic

- Preset a bandwidth threshold or standard
- Gradually increasing the window size to *test* the bandwidth, modifying the preset
- Relying on packet loss as a criterion for congestion
- Large and sudden fluctuations in window size (especially reno, cubic has improved in some way)

Network Buffer Dynamics

Network Buffers:

- Temporary storage for packets at network devices
- Help handle traffic bursts

Why Buffer Size Indicates Loss:

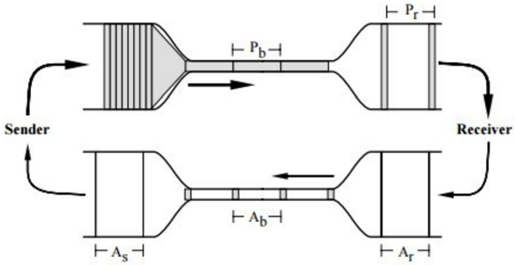
- Maximum amount of data that can be in the network
- Exceeding this leads to packet drops

Challenges of Traditional Congestion Control in Long Fat Networks (LFNs)

- **Definition of LFNs:** Networks characterized by **high bandwidth** and **high latency**, often with large Bandwidth-Delay Product (BDP).
- **Bandwidth-Delay Product (BDP):** The product of a network link's capacity and its round-trip time (RTT); represents the maximum amount of data in-flight on the link.
- **Example Scenarios:** Intercontinental data transfers, satellite communication links.

Challenges of Traditional Congestion Control in Long Fat Networks (LFNs)

> *网络上能容量的数据包的数量（水管容量） = 链路带宽（水管截面积） x 往返延迟（水管长度）*



Limitations of Loss-based Congestion Control

- **Slow Convergence:** Random losses trigger drastic rate reductions, leading to underutilization.
- **High Latency and Bufferbloat:** Loss-based algorithms cause queuing delays, as they rely on buffers for congestion detection.
- **Poor scalability:** Linear/Cubic growth struggles to adapt to large bandwidth changes.
- **Misinterpretation of loss:** Loss can occur due to factors other than congestion (e.g., wireless interference).

Real-time Graphs

We've implemented a real-time graph to help you visualize the concepts discussed. Let's take a look at the graph.
Parameters:

- **RTT**: 150 ms
- **Loss Rate**: 4%
- **BDP**: 150 ms * 200 Mbps ≈ 4 MB

Introduction to TCP BBR

What is TCP BBR?

Bottleneck **B**andwidth and **R**ound-trip propagation time (BBR) is a congestion control algorithm developed by Google.

- BBR is designed to maximize throughput without causing unnecessary packet loss, even in LFNs.

Fundamental Network Metrics

RTT (Round-Trip Time):

- Time taken for a packet to:
  1. Travel from sender to receiver
  2. And back to the sender

Delivery Rate:

- Amount of data successfully delivered per unit time
- Measured in bits/second or packets/second
- Limited by network bottleneck capacity

Fundamental Network Metrics

BDP (Bandwidth-Delay Product):

- **Definition**: Amount of data that can be in transit before receiving first acknowledgment
- **Formula**: $BDP = \text{Bandwidth} \times \min RTT$
- Represents optimal amount of "in-flight" data
- The "sweet spot" for network utilization

BDP (cont'd)

- **Under BDP**:
  - Network underutilized
  - Low latency, but suboptimal throughput
- **At BDP**:
  - Optimal operation point
  - Maximum throughput with minimal latency
- **Over BDP**:
  - Increased latency
  - Buffer bloat in deep buffers
  - Packet loss in shallow buffers

BBR's Approach to Finding Optimal Point

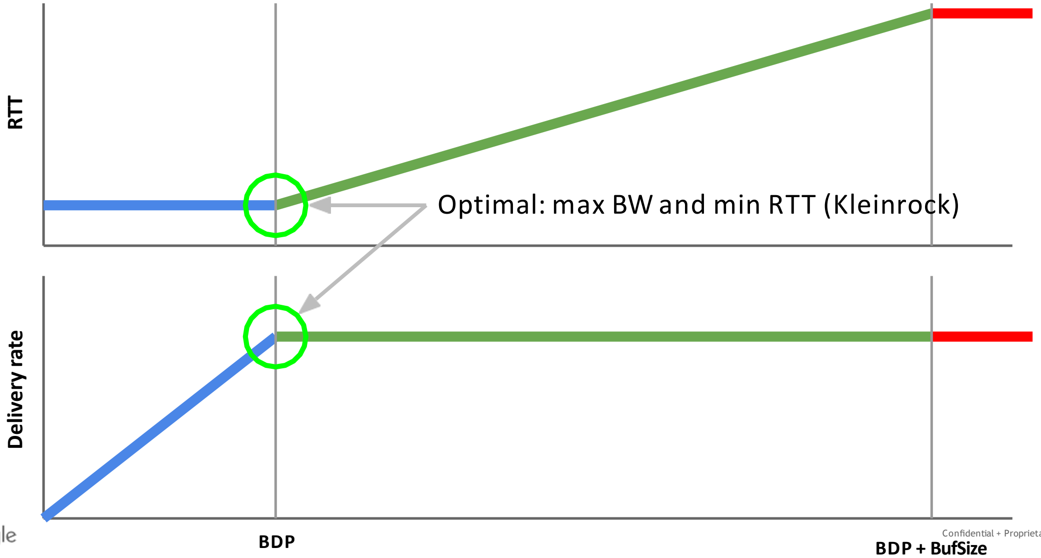
Why These Metrics Matter:

- RTT measurements indicate queuing
- Delivery rate shows available bandwidth
- BDP helps find optimal operating point

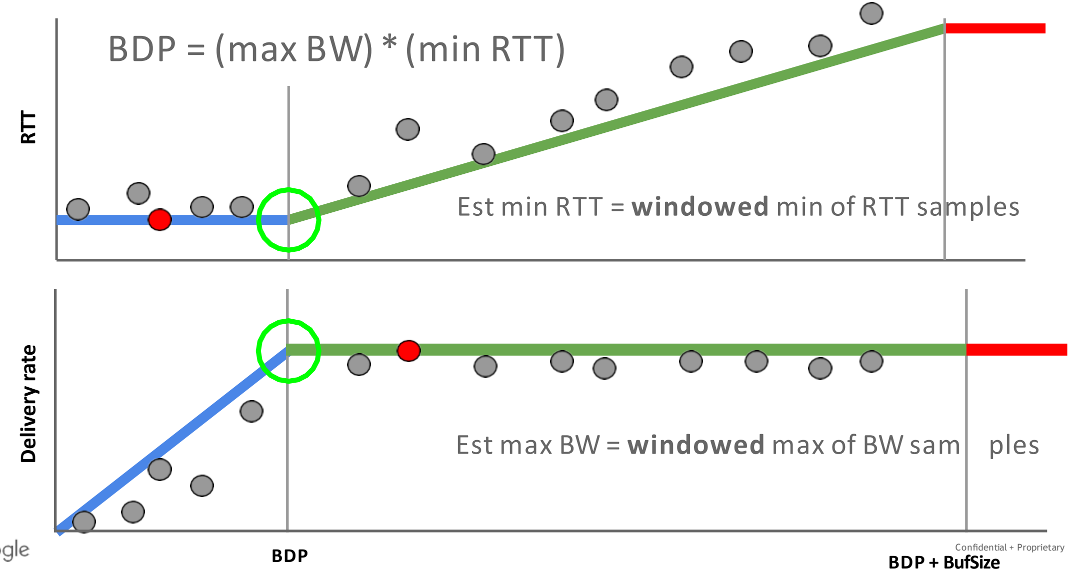
Great Ideas:

1. Measure RTT when network underutilized
2. Probe for max bandwidth when RTT rises
3. Calculate $BDP = \max \text{Bandwidth} \times \min RTT$
4. Control sending rate

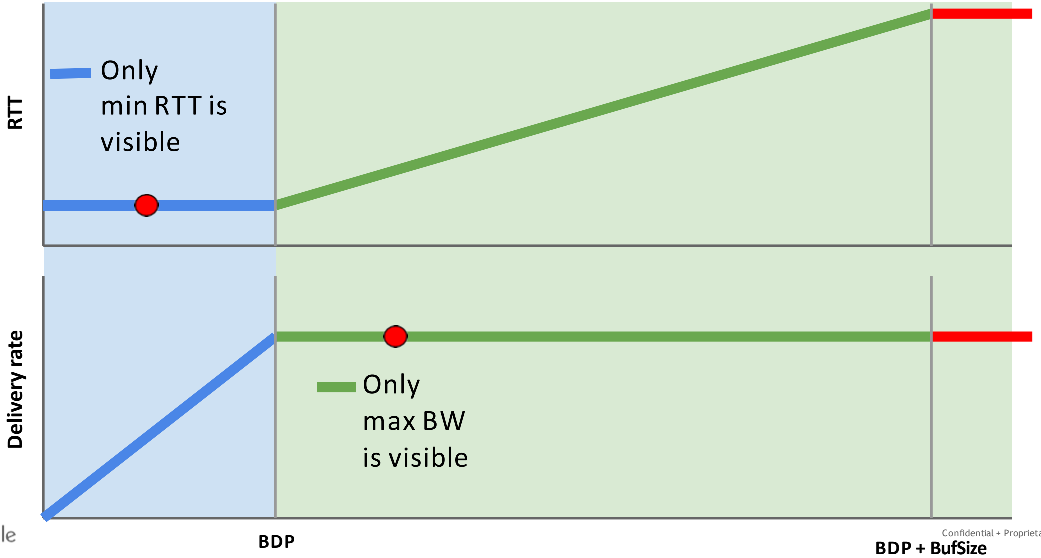
Optimal operating point



Estimating optimal point (max BW, min RTT)



To see max BW, min RTT: probe both sides of BDP



Real-time Graphs

We've implemented a real-time graph to help you visualize the concepts discussed. Let's take a look at the graph.

Parameters:

- **RTT**: 150 ms
- **Loss Rate**: 4%
- **BDP**: 150 ms * 200 Mbps ≈ 4 MB

BBR Control Loop

Key Components:

1. **Pacing Rate Control**:
   - Paces packets near estimated bandwidth
   - Prevents burst-induced queuing
2. **Window Control**:
   - Maintains inflight data near BDP
   - Prevents queue buildup
3. **Gain Cycling**:
   - Alternates between probing and draining phases
   - Helps maintain accurate BW and RTT estimates

BBRv1 Core Algorithm

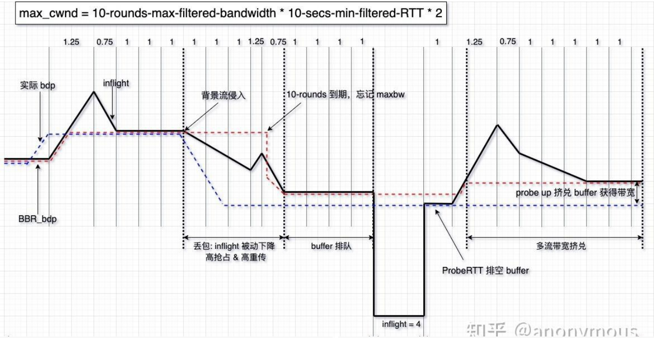
- Two main functions that work together:
  - `onAck()`: Updates RTprop and BtlBw estimates
  - `send()`: Paces packets to match bottleneck rate
- Uses max/min filters to track network constraints
- Maintains inflight data near BDP

BBR onAck() Function

```
function onAck(packet):
    rtt = now - packet.sendtime
    update_min_filter(RTpropFilter, rtt)

    delivered += packet.size  delivered_time = now
    deliveryRate =  (delivered - packet.delivered) /
                    (delivered_time - packet.delivered_time)

    if (deliveryRate >  BtlBwFilter.currentMax ||
        !packet.app_limited):
        update_max_filter(BtlBwFilter , deliveryRate)

    if (app_limited_until > 0):
        app_limited_until -= packet.size
```

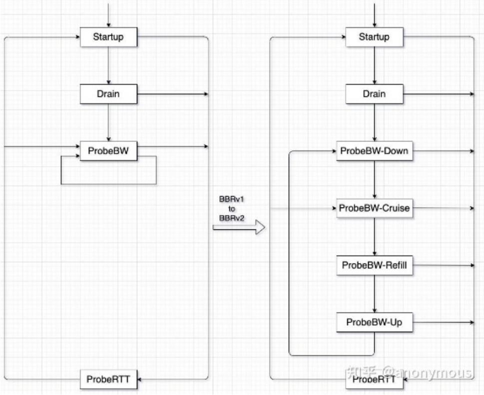
BBRv1 Challenges

- Aggressive bandwidth probing leads to high inflight packets
- Poor coexistence with loss-based congestion control (Reno, CUBIC)
- Slow reaction to network changes due to 10-round persistence
- High queue buildup and packet loss - contrary to BBR goals



BBRv2 Improvements

- More precise delivery rate measurement
- ProbeBW split into 4 sub-states for better control
- Exponential bandwidth probing (slow start then fast)
- More conservative and fair behavior



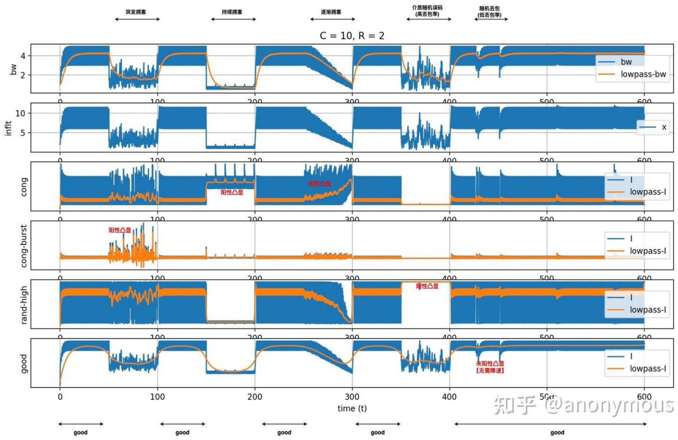
BBR Network Diagnosis Model

Condition	Delay	Delay Var	BW	BW Var
Stable Congestion	High	Low	N/A	N/A
Burst Congestion	High	High	N/A	N/A
High Random Loss	Low	Low	Low	High
Low Random Loss	Low	Low	High	Low

BBR Network Diagnosis Model (cont'd)

```
if n > 500 and n <= 1000:
    # Random burst congestion
    Buff = random.randint(0, 20)
elif n > 1500 and n <= 2000:
    # Sustained congestion
    Buff = 20
elif n > 2500 and n <= 3000:
    # Gradual congestion
    Buff += 0.04
elif n > 3500 and n <= 4000:
    # High random loss (no congestion)
    Buff = 0
    if random.random() < drate:
        wx[n] = wx[n]/2
```

Network Scenarios



BBR Today

- Widely deployed within Google's infrastructure (B4 WAN, datacenters, Google.com, YouTube).
- Available in Linux kernel (v4.9 onwards).
- Significant improvements in throughput and latency observed in real-world deployments.
- Ongoing research and development for further enhancements.

BBR's Future: Research and Development

- BBRv3 and beyond: Addressing remaining challenges and **fine-tuning**(?) performance.
- Exploring new probing techniques for more accurate network modeling.
- Integration with emerging network technologies (e.g., programmable data planes).
- Application to other transport protocols beyond TCP (e.g., QUIC).
- Addressing fairness and coexistence challenges in heterogeneous network environments.

References

- Cardwell N, Cheng Y, Gunn CS, Yeganeh SH, Jacobson V. *BBR: Congestion-based Congestion Control*. Communications of the ACM, 2017; 60(2):58-66.
- Google Cloud Blog: TCP BBR on GCP (Website)
- BBR Fundamentals and Updates (2023) (PDF Document)
- Cloudflare Blog: Optimizing TCP (Website)

Q&A