ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN MẠNG MÁY TÍNH HƯỚNG AN TOÀN THÔNG TIN

Tìm hiểu về giải thuật TCP CUBIC, TCP Compound và TCP BBR

Môn học: Mạng máy tính nâng cao

Sinh viên thực hiện:

Nguyễn Ngọc Phương Uyên - 22127446

Nguyễn Nhật Hiệu - 22127116

Mục lục

Danh sách bảng

Danh sách hình vẽ

1 TCP Cubic

1.1 Lý do ra đời

Mạng Internet ngày nay gồm nhiều thành phần là Long Fat Network (LFNs). Một khái niệm cần chú ý trong mạng LFNs là Bandwidth Delay Product (BDPs). BDPs cho biết lượng dữ liệu tối đa có thể tồn tại trên một kênh truyền trong một thời điểm nhất định, giúp đánh giá hiệu suất của kết nối mạng. Các mạng LFNs có BDPs cao cho phép bên gửi có thể gửi nhiều dữ liệu hơn cùng lúc. Tuy nhiên, các giải thuật kiểm soát tắc nghẽn trước đây mất rất nhiều thời gian để đạt đến tốc độ tối ưu do cơ chế tăng kích thước cửa sổ tắc nghẽn (cwnd) chậm rãi (tăng 1 Maximum segment size (MSS)/Round-trip Time (RTT)). Giải thuật TCP Cubic ra đời năm 2005 với sự cải tiến trong giai đoạn Congestion avoidance giúp tận dụng được lượng băng thông khổng lồ của LFNs.

1.2 Giai đoạn khi khởi tạo kết nối TCP

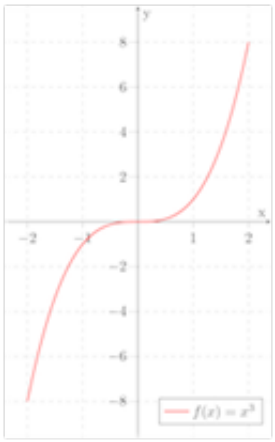
- Ở giai đoạn mới khởi tạo kết nối, cũng giống như các giải thuật TCP Tahoe và TCP Reno, TCP Cubic trải qua giai đoạn Slow start và cwnd sẽ được tăng gấp đôi so với giá trị trước đó cho tới khi gặp tắc nghẽn lần đầu.

- Khi tắc nghẽn xảy ra, giai đoạn Congestion avoidance bắt đầu, giai đoạn Slow start tạm thời dừng lại và sẽ được kích hoạt lại nếu mất gói được dự đoán xảy ra nhờ vào dấu hiệu hết thời gian timeout (TCP Cubic coi đây là một dấu hiệu nghiêm trọng của tắc nghẽn đường truyền).

1.3 Tổng quan về TCP Cubic ở giai đoạn Congestion avoidance

- **Dấu hiệu nhận biết tắc nghẽn**: TCP Cubic nhận biết tắc nghẽn dựa vào mất gói (loss-based algorithm). Nó cũng phân biệt tín hiệu mất gói bởi 3 duplicate ACKs và tín hiệu mất gói bởi hết thời gian timeout để có thể phản hồi thích hợp trong từng trường hợp.

- **Hàm dùng để thay đổi kích thước cửa sổ tắc nghẽn**: TCP Cubic dùng hàm khối để thay đổi kích thước cửa sổ tắc nghẽn.



Hình 1: Hàm khối

- Hàm khối gồm các phần concave, convex và điểm uốn. Điểm uốn tương ứng với giá trị Wmax (là kích thước của cửa sổ tắt nghẽn ở lần tắc nghẽn trước đó). Phần concave là phần dưới điểm uốn, phần này thể hiện sự thay đổi của cwnd khi giá trị chưa vượt qua Wmax. Phần convex là phần phía trên điểm uốn, phần này thể hiện sự thay đổi cwnd khi giá trị đã vượt qua Wmax.

1.4 Khi gặp tắc nghẽn

1.4.1 Tắc nghẽn được nhận biết do có 3 duplicate ACK.

Wmax được gán bằng với kích thước cửa sổ tắc nghẽn hiện tại.

$$W_{max} = cwnd$$

Slow start threshold được gán bằng cwnd * nếu cwnd * > 2 hoặc 2 nếu ngược lại.

$$ssthresh = max(2, cwnd * \beta)$$

Trong đó, β là hệ số nhân có thể điều chỉnh khi cài đặt.
Giảm kích thước cửa sổ tắc nghẽn bằng cách gán

$$cwnd = cwnd * \beta$$

1.4.2 Tắc nghẽn được nhận biết do hết thời gian timeout

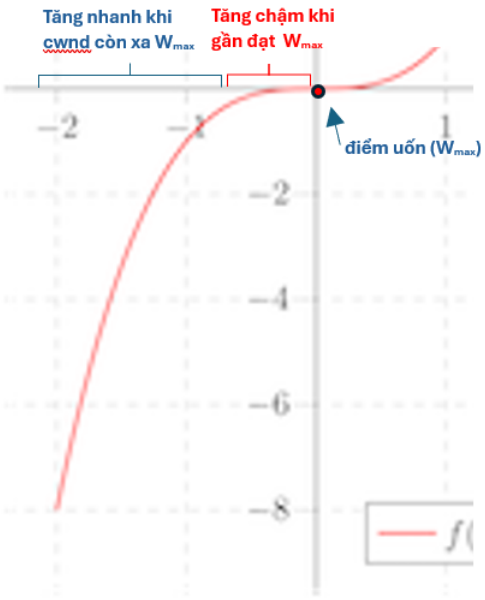
Khi tắc nghẽn được nhận biết do hết thời gian timeout, các giá trị Wmax và ssthresh cũng được gán như khi có 3 duplicate ACK. Tuy nhiên, kích thước cửa sổ tắc nghẽn sẽ được gán theo công thức dưới đây.

$$cwnd = min(cwnd, IW)$$

Trong đó, IW là kích thước cửa sổ tắc nghẽn ở lần tắc nghẽn đầu tiên
Sau đó, nó chuyển sang giai đoạn Slow start.

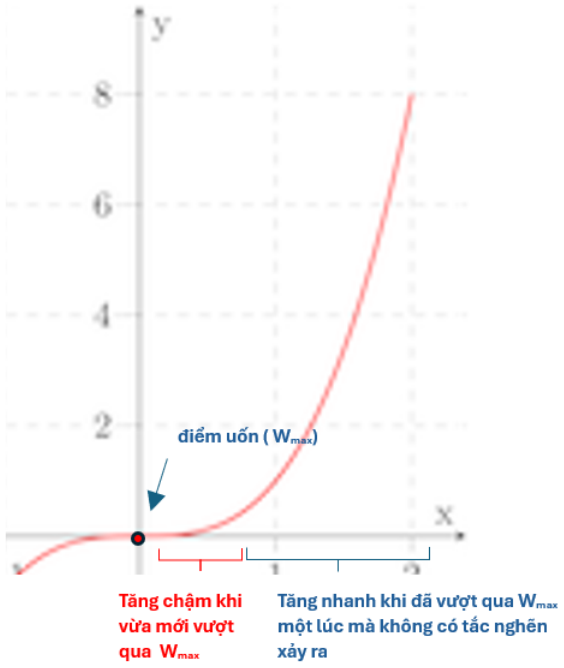
1.5 Sau khi tắc nghẽn

- Sau khi cwnd giảm đáng kể để giải quyết tình trạng tắc nghẽn, kích thước cửa sổ tắc nghẽn sẽ có xu hướng tăng lên để tìm kiếm nhiều băng thông hơn nhằm tăng tốc độ gửi tin. TCP Cubic sẽ phản hồi khác nhau tương ứng với hai dấu hiệu mất gói, cụ thể:
  - Chuyển sang giai đoạn Slow start nếu tắc nghẽn được xác định do hết thời gian timeout. Giai đoạn Slow start đã được mô tả ở mục 2. Giai đoạn khi mới khởi tạo kết nối TCP.
  - cwnd tăng theo hàm khối nếu tắc nghẽn được xác định do có 3 duplicate ACK.
- Đối với tín hiệu 3 duplicate ACK:
  - cwnd sẽ tăng theo hàm khối. cwnd sẽ tăng nhanh khi nó chưa đạt tới giá trị Wmax và tăng chậm lại khi đến gần với giá trị Wmax. Điều này giúp bên gửi sử dụng được nhiều băng thông hơn trong quá trình đạt tới Wmax so với các giải thuật TCP cũ (TCP Tahoe và TCP Reno) đồng thời vẫn thận trọng với tắc nghẽn. Khi nhìn vào đồ thị hàm khối, phần concave của đồ thị giúp trực quan hóa cơ chế tăng này.



Hình 2: Phần concave của đồ thị thị hàm khối

– Khi cwnd vượt qua giá trị giá trị Wmax mà không có tắc nghẽn xảy ra, TCP Cubic sẽ tăng cwnd chậm rãi. Nếu sau đó tắc nghẽn vẫn không xảy ra, TCP Cubic xem đây là dấu hiệu cho thấy tình hình đường truyền đã được cải thiện và nó sẽ tăng nhanh kích thước cửa sổ để tận dùng nguồn băng thông khả dụng. Phần convex của đồ thị hàm khối minh họa cho cơ chế này.



Hình 3: Phần convex của đồ thị hàm khối

1.6 Ưu điểm và nhược điểm

1.6.1 Ưu điểm

- **Tăng nhanh đến tốc độ gửi tối ưu**: Sử dụng hàm khối để xác định cwnd giúp đạt được tốc độ gửi tối ưu nhanh hơn.
- **Phản ứng tốt với những thay đổi của đường truyền**: TCP Cubic phản ứng khác nhau tùy theo các dấu hiệu mất gói khác nhau. Ngoài ra, nó giảm tốc độ tăng khi đang gần đến Wmax và khi mới vượt qua Wmax giúp nó thăm dò tình trạng đường truyền cẩn thận hơn từ đó giúp phản ứng kịp thời.
- **Phù hợp với các mạng LFNs**: Tận dụng được lượng băng thông lớn của các mạng LFNs nhờ cơ chế tăng cwnd của nó.

1.6.2 Nhược điểm

- **Làm tăng Round Trip Time**: Khi TCP Cubic tăng cwnd quá nhanh và nhiều, tình trạng các gói tin phải đợi ở hàng đợi của các router sẽ xảy ra và làm tăng thời gian gói tin đến đích,

từ đó làm tăng thời gian bên gửi nhận được gói tin ACK dẫn tới tăng RTT.

- **Giảm TCP Fairness**: Khi hoạt động trên đường truyền gồm nhiều cơ chế kiểm soát tắt nghẽn khác nhau, TCP Cubic có xu hướng chiếm dụng băng thông của các luồng khác. Một sự thật thú vị là TCP Cubic hoạt động tốt hơn trên kênh truyền có các luồng sử dụng cơ chế khác so với khi hoạt động 1 mình trên kênh truyền.

2 TCP Compound

2.1 Lý do ra đời

- Tương tự như TCP Cubic, TCP Compound ra đời nhằm tận dụng lượng băng thông của các mạng LFNs mà các giải pháp cũ là TCP Tahoe và TCP Reno không đáp ứng được.
- Các giải pháp kiểm soát tắc nghẽn dựa vào mất gói như HSTCP, STCP, Cubic TCP mặc dù tận dụng tốt hơn lượng băng thông nhưng chiến thuật tăng mạnh bạo của nó chiếm đi lượng băng thông khả dụng cho các luồng khác trên cùng được truyền, do đó làm giảm TCP fairness.
- Các giải pháp chỉ dựa vào độ trễ như FAST TCP mặc dù giúp hạn chế mất gói và làm tăng throughput nhưng không cạnh tranh được với các luồng khác.

$\Longrightarrow$ TCP compound ra đời năm 2005 với 3 nguyên tắc Efficiency, RTT fairness, TCP fairness giúp giải quyết các vấn đề nêu trên.

2.2 Ý tưởng của TCP Compound

- 3 nguyên tắc mà TCP Compound tuân theo:
  - Efficiency: Tăng throughput của kết nối để tận dụng băng thông đường truyền
  - RTT fairness: Đảm bảo băng thông được chia đều cho các RTT khác nhau trên cùng đường truyền
  - TCP fairness: Đảm bảo các luồng với giải thuật kiểm soát tăng nghẽn khác nhau đều sử dụng được lượng băng thông phù hợp

- Ý tưởng chính
  - Bên cạnh cửa sổ tắc nghẽn dựa vào tín hiệu mất gói (loss-based), gọi là cwnd, TCP Compound thêm một cửa sổ tắc nghẽn dựa vào độ trễ (delay-based), gọi là cwnd. Kích thước của cửa sổ thật sự, gọi là w, là tổng của cwnd và dwnd.

$$w = cwnd + dwnd$$

  - cwnd sẽ hoạt động giống như cơ chế TCP Reno. dwnd sẽ thay đổi kích thước của nó tùy thuộc vào độ trễ. Khi độ trễ dưới mức giới hạn, dwnd tiếp tục tăng kích thước của. Khi độ trễ vượt mức giới hạn, dwnd giảm kích thước. Khi mất gói xảy ra, dwnd điều chỉnh kích thước của nó theo sự thay đổi của cwnd để duy trì w ở mức mong muốn.
- Hàm xác định kích thước cửa sổ (w)
  - Kích thước cửa sổ tăng theo hàm sau:

$$w(t+1) = w(t) + \alpha win(t)^k$$

  - Kích thước cửa sổ giảm theo hàm sau:

$$w(t+1) = w(t)(1-\beta)$$

  - Các cửa sổ cấu thành là cwnd và dwnd cũng có cơ chế tăng giảm của riêng nó và các cơ chế này khi kết hợp với nhau sẽ thỏa mãn hai hàm trên.
- Làm sao để cửa sổ delay xác định được độ trễ
  - baseRTT là RTT nhỏ nhất từ khi khởi tạo kết nối tới thời điểm hiện tại.
  - RTT là RTT ở thời điểm hiện tại.
  - Ta có các giá trị sau:

$$Expected = w/baseRTT$$

$$Actual = win/RTT$$

$$Diff = Expected \breve{} Actual$$

– Sau khi tính được Diff, ta sẽ so sánh Diff với . là threshold cho biết độ giá trị độ trễ đáng báo động, giúp dự đoán sớm tắc nghẽn đường truyền.

* Khi $Diff$, TCP Compound cho rằng đường truyền đang hoạt động ổn.
* Khi $Diff \geq$, TCP Compoud cho rằng tắc nghẽn đang xảy ra.

2.3 Giai đoạn khởi tạo kết nối TCP

Trong giai đoạn khởi tạo, TCP Compound thực hiện quá trình Slow start. Lúc này cửa sổ delay chưa được kích hoạt nên giá trị của dwnd = 0. Giai đoạn Slow start đã được mô tả ở giải thuật TCP Cubic phía trên.

2.4 Khi xảy ra tắc nghẽn

2.4.1 Đối với cửa sổ loss-based

Vì cửa sổ loss-based hoạt động giống TCP Reno nên nó phát hiện tắc nghẽn sự kiện mất gói tin và phản ứng khác nhau đối với hai dấu hiệu 3 duplicate ACK và hết thời gian timeout.

- Khi nhận được 3 duplicate ACK, cwnd sẽ gán giá trị ssthresh bằng với kích thước cwnd hiện tại chia 2. Đồng thời cwnd cũng bị giảm đi một nửa.

$$ssthresh = cwnd/2$$

$$cwnd = cwnd/2$$

- Khi hết thời gian timeout, cwnd cũng gán giá trị ssthresh bằng với kích thước cwnd hiện tại chia 2. Tuy nhiên, kích thước cwnd lúc này bị giảm xuống 1 vì TCP Compound cho rằng đây là một dấu hiệu của sự tắc nghẽn nghiêm trọng.

2.4.2 Đối với cửa sổ delay-based

- Khi phát hiện tắc nghẽn sớm dựa vào độ trễ (khi Diff >=), cửa sổ delay-based giảm kích thước của nó theo hàm sau:

$$dwnd(t+1) = (dwnd - \zeta Diff)$$

ζ thể hiện mức độ giảm của dwnd

Lưu ý: Vì dwnd luôn lớn hơn hoặc bằng 0 nên kích thước cửa sổ w luôn lớn hơn hoặc bằng cwnd (w = cwnd + dwnd).

- Khi nhận được 3 duplicate ACK, kích thước cửa sổ delay-based được xác định theo hàm sau:

$$dwnd(t+1) = w(t)(1-\beta)\breve{}cwnd/2$$

Vì cwnd đã giảm đi một nửa trong trường hợp này nên dwnd cần được thay đổi để w = cwnd + dwnd thỏa mãn hàm (2).

- Khi hết thời gian timeout, TCP Compoud bước vào giai đoạn Slow start. Cửa sổ delay-based tạm thời dừng hoạt động nên dwnd lúc này bằng 0.

2.5 Sau khi xảy ra tắc nghẽn

2.5.1 Đối với cửa sổ loss-based

- **Phát hiện tắc nghẽn nhờ vào 3 duplicate ACK**: Sau khi xảy ra tắc nghẽn do 3 duplicate ACK và giảm kích thước đi 1 nửa, cửa sổ loss-based tăng kích thước của nó lên 1 sau mỗi Round trip time.

- **Phát hiện tắc nghẽn nhờ vào hết thời gian timout**: cwnd bươc vào giai đoạn Slow start, kích thước của nó sẽ được tăng gấp đôi sau mỗi Round trip time, tức là tăng thêm 1 tương ứng với mỗi gói tin ACK nhận được.

2.5.2 Đối với cửa sổ delay-based

Kích thước của cửa sổ delay-based sẽ tăng theo hàm sau $Diff$:

$$dwnd(t+1) = dwnd(t) + \alpha win(t)^k - 1$$

Vì trong trường hợp này, cwnd đã tăng kích thước thêm 1 nên dwnd sẽ tăng một lượng là $\alpha win(t)^k - 1$ để w = cwnd + dwnd thỏa mãn hàm (1)

2.6 Ưu điểm và nhược điểm

2.6.1 Ưu điểm

- **Phát hiện và phản ứng sớm với tắc nghẽn**: Nhận biệt tắc nghẽn nhờ vào độ trễ mà không đợi đến khi mất gói xảy ra nhờ vào cửa sổ delay-based.
- **Làm tăng RTT fairness và TCP fairness**: Cửa sổ delay-based giúp giảm dần tốc độ gửi khi đường truyền đang hoạt động gần hết công suất. Điều này khiến cho bên gửi không chiếm dụng băng thông của các luồng khác trên cùng đường truyền
- **Sử dụng hiệu quả đường truyền**: Kích thước cửa sổ được giảm đúng lúc và hợp lý làm hạn chế mất gói và tăng throuhgput của kết nối.

2.6.2 Nhược điểm

- **Phụ thuộc vào độ chính xác của RTT**: Cửa sổ delay-based tính độ trễ dựa vào RTT và tùy vào đó mà điều chỉnh tốc độ gửi. Nếu giá trị RTT thiếu chính xác sẽ dẫn tới đánh giá sai tình trạng tắc nghẽn làm giảm trầm trọng hiệu quả trong việc sử dụng đường truyền.
- **Phức tạp để cài đặt**: Việc duy trì 2 cửa sổ và các cách xử lí khác nhau cho từng trường hợp khiến TCP Compound trở nên phức tạp khi cài đặt hơn so với các giải thuật kiểm soát tắc nghẽn khác.

3 TCP BBR

3.1 Lý do ra đời

- Hiện nay, tốc độ Internet không đáp ứng đủ nhu cầu của người dùng. Nhiều người gặp phải tình trạng trễ từ vài giây đến vài phút, đặc biệt là khi sử dụng Wi-Fi ở sân bay hoặc hội trường. Các nhà nghiên cứu vật lý và khí tượng, cần trao đổi hàng petabyte dữ liệu toàn cầu, cũng gặp khó khăn khi cơ sở hạ tầng chỉ đạt tốc độ vài Mbps thay vì hàng Gbps như mong đợi.

- Nguyên nhân chính của những vấn đề này là hệ thống quản lý tắt nghẽn dựa trên gói tin bị mất (loss-based congestion control), ngay cả với phiên bản tốt nhất hiện nay là CUBIC. Khi bộ đệm bottleneck lớn, hệ thống này giữ cho nó luôn đầy, dẫn đến hiện tượng tràn bộ đệm. Khi bộ đệm bottleneck nhỏ, hệ thống dịch sai tín hiệu tắt nghẽn, làm giảm băng thông.

- Giải pháp cho vấn đề này là sử dụng hệ thống quản lý tắt nghẽn dựa trên băng thông và thời gian trễ (BBR - Bottleneck Bandwidth and Round-trip propagation time). BBR không dựa vào gói tin bị mất để xác định tắt nghẽn mà dựa vào băng thông và thời gian trễ để điều chỉnh tốc độ truyền dữ liệu, giúp tối ưu hóa băng thông và giảm độ trễ.

3.2 Ý tưởng của TCP BBR

- Bất kể khi nào, một kết nối TCP sẽ luôn có đúng một liên kết chậm nhất (nút cổ chai) ở cả hai chiều. Một nút cổ chai rất quan trọng bởi vì:
  - Nó cho biết tốc độ truyền dữ liệu tối đa của một kết nối.
  - Đó là nơi mà những hàng chờ được hình thành. Những hàng chờ đó chỉ co lại khi tốc độ truyền đi của liên kết có nút cổ chai lớn hơn tốc độ truyền đến. Để một kết nối có thể chạy ở tốc độ truyền tối đa, mọi liên kết ở trước nút cổ chai có tốc độ truyền đi nhanh hơn để những hàng chờ có thể di chuyển gần hơn tới nút cổ chai.

- Tóm lại, một kết nối phức tạp bất kì có hoạt động như là một liên kết với một RTT (round-trip time – tổng thời gian cả hai chiều) và tốc độ tại nút cổ chai cố định. Hai đại lượng RTprop (thời gian truyền) và BtlBw (băng thông tại nút cổ chai) định nghĩa cho tốc độ truyền tải.

(Giải thích dễ hiểu, giả sử đường dẫn là một cái ống thì RTprop là độ dài và BtlBw là đường kính.)

- Ở những thuật toán TCP thông thường, những gói tin sẽ bị thất lạc khi lượng dữ liệu vào gần đầy buffer tại nút cổ chai nhưng tắt nghẽn lại được phát hiện chỉ khi nút cổ chai bị quá tải dữ liệu. Từ đó dẫn đến một khoảng trì hoãn lớn.

- BDP (Bandwidth-Delay Product) là lượng dữ liệu tối đa có thể truyền qua mạng mà không cần chờ xác nhận. Khi lượng dữ liệu truyền đi bằng với BDP, hiệu suất đạt tối đa. Nếu vượt quá BDP, sẽ tạo ra hàng chờ và tăng độ trễ. Cả hai thuật toán dựa trên độ trễ và mất gói tin đều hoạt động khi lượng dữ liệu vượt qua điểm tối ưu Kleinrock.

- Từ đó, BBR ra đời để đảm bảo rằng nút cổ chai luôn ở trạng thái đầy nhưng không nghẽn. BBR sẽ tính toán băng thông với lưu lượng truyền tối dữ liệu tối đa BtlBw và thời gian truyền với RTT tối thiểu. Hai giá trị này phải được tính toán tách rời nhau bởi vì đẩy cao băng thông làm tăng thời gian trì hoãn và ngược lại.

3.3 Giai đoạn khởi tạo kết nối

Ở giai đoạn mới khởi tạo kết nối, cũng giống như giải thuật TCP Cubic, TCP BBR trải qua giai đoạn Startup và cwnd sẽ được tăng gấp đôi so với giá trị trước đó nhưng khác ở chỗ TCP BBR sẽ tự kết luận rằng nó đã đạt được băng thông tối đa là BtlBw khi băng thông không tăng nữa. Khi đó, một hàng chờ đã được khởi tạo tại nút cổ chai và TCP BBR sẽ làm giảm bớt hàng chờ đó bằng cách tạm thời giảm khoảng tăng thêm của tốc độ gửi.

3.4 Giai đoạn cân chỉnh

- Sau giai đoạn Drain, BBR sẽ tiến vào giai đoạn Probe Bandwidth để tìm nhiều băng thông hơn. Giai đoạn này bao gồm tám giai đoạn nhỏ, mỗi đoạn dài một khoảng bằng RTprop.

- Đầu tiên, tốc độ gửi sẽ được tăng lên 1.25 lần để tìm nhiều băng thông hơn, tiếp sau đó sẽ giảm 0.75 lần để loại bỏ hàng chờ được tạo ra. Sáu khoảng sau đó tốc độ gửi sẽ được đưa về như cũ. BBR sẽ liên tục đo băng thông và sử dụng giá trị lớn nhất để ước lượng BtlBw, giá trị BtlBw này sẽ hợp lệ trong khoản mười RTprop.

- Khi không đo được giá trị RTprop nào mới trong 10 giây, BBR sẽ dừng giai đoạn Probe Bandwidth để chuyển sang giai đoạn Probe RTT. Trong suốt giai đoạn này, băng thông sẽ bị giảm còn 4 gói tin để loại bỏ toàn bộ những hàng chờ và tính toán RTT. Giai đoạn này kéo dài 200ms và 1 RTT. Nếu một giá trị RTT nhỏ nhất được tìm thấy, RTprop sẽ được cập nhập và sử dụng trong giai đoạn này trong 10 giây tiếp theo.

$\Longrightarrow$ Tóm lại, trong toàn bộ quá trình, BBR sẽ trải qua 4 giai đoạn chính:

  - Giai đoạn Startup: tăng 2 lần tốc độ gửi cho đến khi BtlBw được tính toán không tăng nữa.
  - Giai đoạn Drain: tạm thời giảm tốc độ gửi để giảm kích thước hàng chờ.
  - Giai đoạn Probe Bandwidth: liên tục thay đổi tốc độ gửi để tìm ra giá trị lớn nhất mà băng thông có thể đạt được và tính BtlBw.
  - Giai đoạn Probe RTT: giảm lượng gói tin truyền đi để loại bỏ hàng chờ và liện tục tìm giá trị RTprop nhỏ nhất.

3.5 Ưu điểm và nhược điểm

3.5.1 Ưu điểm

- Hoạt động tốt với những kết nối với mật độ mất gói tin không quá nhiều hoặc không xảy ra mất gói tin.

- Phù hợp với các mạng thường trì hoãn việc gửi ACK. TCP BBR không phụ thuộc vào ACK để điều chỉnh tốc độ gửi mà dựa vào đo lường băng thông và thời gian trễ. Điều này giúp BBR duy trì hiệu suất truyền dữ liệu ổn định ngay cả khi ACK bị trì hoãn.

3.5.2 Nhược điểm

- BBR có thể gặp khó khăn trong các mạng có tỷ lệ mất gói tin cao, vì nó không điều chỉnh tốc độ dựa trên mất gói tin như các thuật toán truyền thống.

- Không công bằng trong việc chia sẻ băng thông, đẩy những kết nối TCP khác ra ngoài.

4 So sánh các thuật toán TCP

TCP Compound	TCP Cubic	TCP BBR
Phát hiệu tắc nghẽn dựa vào mất gói và trễ	Phát hiện tắc nghẽn dựa vào mất gói	Hoạt động tùy theo tình trạng mạng hiện tại và không phụ thuộc vào ACK
2 Phát hiện tắc nghẽn sớm và giảm kích thước cửa sổ kịp thời giúp làm tăng thoughput. Không phản ứng tốt với các tình huống bottleneck buffer	Tăng kích thước cửa sổ gửi theo hàm khối giúp nhanh chóng đạt tốc độ gửi tối ưu. Không phản ứng tốt với các tình huống bottleneck buffer	Hạn chế tạo ra hàng đợi, giúp giảm tình trạng bufferbloat
Làm tăng TCP fairness	Làm giảm TCP fairness	Làm giảm TCP fairness
Không làm tăng latency	Làm tăng latency	Làm giảm latency

Bảng 1: So sánh các giải thuật kiểm soát tắt nghẽn TCP

.

5 Tài liệu tham khảo

- Geeksforgeeks - "TCP Congestion Control Algorithms: Reno, New Reno, BIC, CUBIC" - last updated: 31/01/2024 - link: TCP Congestion Control Algorithms: Reno, New Reno, BIC, CUBIC - GeeksforGeeks
- James F. Kurose, Keith W. Ross - "Computer Networking – A-Topdown-Approach" - 8th edition
- Sid Shanker - "Intro to Congestion control" - 18/7/2018 - link:Intro to Congestion Control
- Sid Shanker - "Congestion Control II: CUBIC" - 01/08/2018 - link:Congestion Control II: CUBIC
- Sangtae Ha, Injong Rhee, Lisong Xu – "CUBIC: A NewTCP-Friendly High-Speed TCP Variant" – link:CUBIC: A NewTCP-Friendly High-Speed TCP Variant
- Kun Tan, Jingmin Song, Qian Zhang, Murari Sridharan – "A Compound TCP Approach for High-speed and Long Distance Networks" – link: A Compound TCP Approach for High-Speed

and Long Distance Networks | Semantic Scholar

- Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson - "BBR Congestion Control" - 11/2016 - link: BBR Congestion Control
- Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson - "TCP BBR congestion control comes to GCP – your Internet just got faster" - 21/7/2017 - link: TCP BBR congestion control comes to GCP – your Internet just got faster
- Dominik Scholz, Benedikt Jaeger, Lukas Schwaighofer, Daniel Raumer, Fabien Geyer, and Georg Carle - "Towards a Deeper Understanding of TCP BBR Congestion Control" - link: Towards a Deeper Understanding of TCP BBR Congestion Control