



CÁC GIẢI THUẬT KIỂM SOÁT TẮC NGHẼN **TCP CUBIC, **TCP COMPOUND VÀ TCP BBR****

Sinh viên:

Nguyễn Nhật Hiệu – 22127116

Nguyễn Ngọc Phương Uyên - 22127446



TCP CUBIC

1. Lý do ra đời

**2. Giai đoạn khởi tạo
kết nối TCP**

**3. Tổng quan về giai đoạn
Phòng tránh tắc nghẽn**

4. Khi xảy ra tắc nghẽn

5. Sau khi gặp tắc nghẽn

6. Ưu điểm và nhược điểm

TCP CUBIC – 1. Lý do ra đời

Mạng Internet ngày nay có nhiều thành phần là **Long Fat Network (LFNs)**. Đây là các mạng **có băng thông lớn**, cho phép bên gửi **gửi với tốc độ cao** hơn nhiều so với trước đây.

Các giải thuật kiểm soát tắc nghẽn cũ như TCP Tahoe và TCP Reno **không tận dụng được lượng băng thông** này do cơ chế tăng chậm trong giai đoạn *Phòng tránh tắc nghẽn (Congestion avoidance)* của nó.

➔ **TCP Cubic ra đời năm 2005 nhằm sử dụng tối ưu nguồn băng thông khổng lồ của các LFNs.**

TCP CUBIC – 2. Giai đoạn khởi tạo kết nối TCP

Khi mới khởi tạo kết nối, TCP Cubic trải qua giai đoạn **Khởi động chậm (Slow start)** giống như giải thuật TCP Reno.

Ở giai đoạn Khởi động chậm, với mỗi ACK nhận được trong Round-trip time, TCP Cubic sẽ tăng kích thước cửa sổ lên 1 MSS. Vì vậy, kích thước cửa sổ sẽ **tăng gấp đôi sau 1 round trip**.

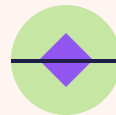
Khi phát hiện tắc nghẽn, TCP Cubic dừng giai đoạn Khởi động chậm và **chuyển sang giai đoạn Phòng tránh tắc nghẽn**.

Giai đoạn **Khởi động chậm sẽ được thực hiện lại** khi có sự kiện **hết thời gian timeout** - dấu hiệu của tắc nghẽn nghiêm trọng.

TCP CUBIC – 3. Tổng quan về giai đoạn Phòng tránh tắc nghẽn

TCP Cubic **nhận biết tắc nghẽn dựa vào mất gói**. Nó cũng **phân chia 2 trường hợp mất gói là 3 ACK trùng** và sự kiện hết thời gian timeout để có phản hồi tương ứng.

Thay vì chỉ tăng 1 MSS sau mỗi round trip, TCP Cubic sử dụng cơ chế tăng nhanh và mạnh hơn bằng cách **sử dụng hàm khối để xác định kích thước của sổ tắc nghẽn**. Đây chính là sự cải tiến của TCP Cubic so với TCP Reno và giúp nó phù hợp với các mạng LFNs.

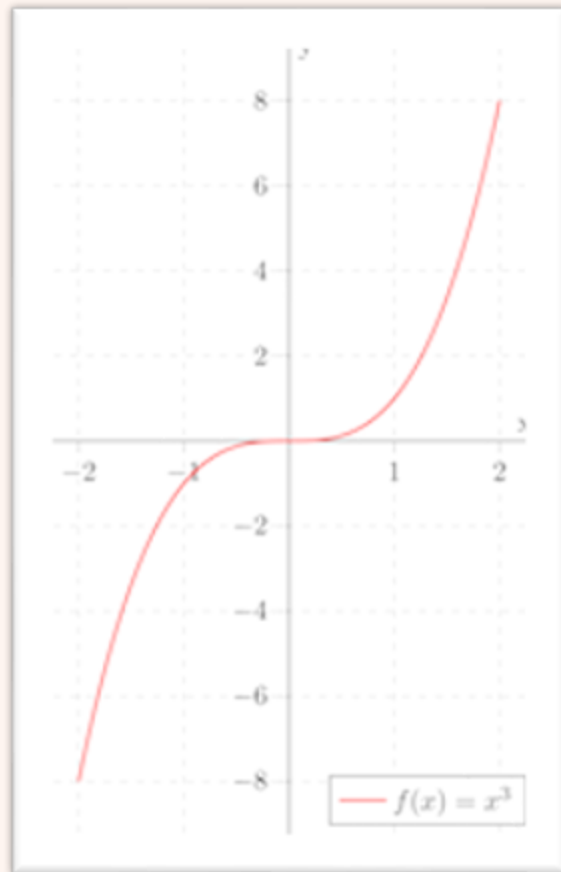


TCP CUBIC – 3. Tổng quan về giai đoạn Phòng tránh tắc nghẽn

Hàm khối có **điểm uốn** và hai phần trên, dưới điểm uốn là **concave** và **convex**.

Hàm khối **tăng nhanh khi xa** điểm uốn và **tăng chậm khi đến gần hoặc vừa vượt qua** điểm uốn.

Khi ứng dụng vào kiểm soát tắc nghẽn, ta xem điểm uốn là điểm mà lần tắc nghẽn gần nhất xảy ra, hàm khối **cho phép tốc độ gửi tăng nhanh và chậm 1 cách hợp lý**.



TCP CUBIC – 4. Khi xảy ra tắc nghẽn

4.1 Khi nhận biết tắc nghẽn do có 3 ACK trùng

Kích thước cửa sổ tại thời điểm xảy ra tắc nghẽn được gán là W_{\max} :

$$W_{\max} = \text{cwnd}$$

Slow start threshold được gán như sau:

$$\text{ssthresh} = \text{Max}(2, \text{cwnd} * \beta)$$

Trong đó, β là hệ số số nhân có thể điều chỉnh khi cài đặt.

cwnd được điều chỉnh như sau:

$$\text{cwnd} = \text{cwnd} * \beta$$



TCP CUBIC – 4. Khi xảy ra tắc nghẽn

4.1 Khi nhận biết tắc nghẽn do hết thời gian timeout

Khi tắc nghẽn được phát hiện do hết thời gian timeout, các giá trị W_{max} , $ssthresh$ cũng được gán như khi có 3 ACK trùng. Tuy nhiên, kích thước cửa sổ tắc nghẽn sẽ bị giảm như sau:

$$cwnd = \text{Min}(cwnd, IW)$$

Trong đó, IW là kích thước của cửa sổ tắc nghẽn ở thời điểm xảy ra tắc nghẽn lần đầu tiên

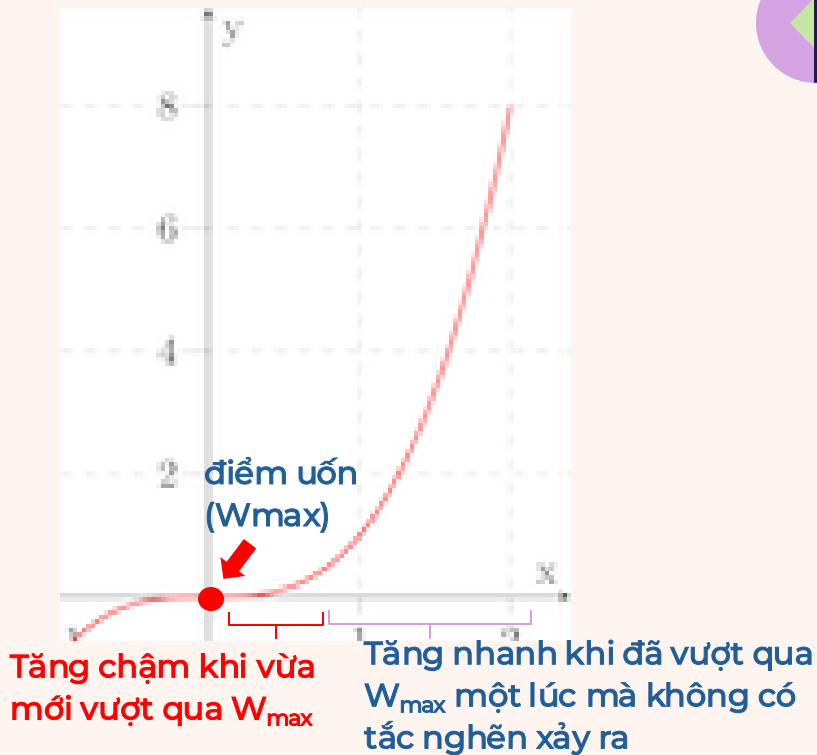
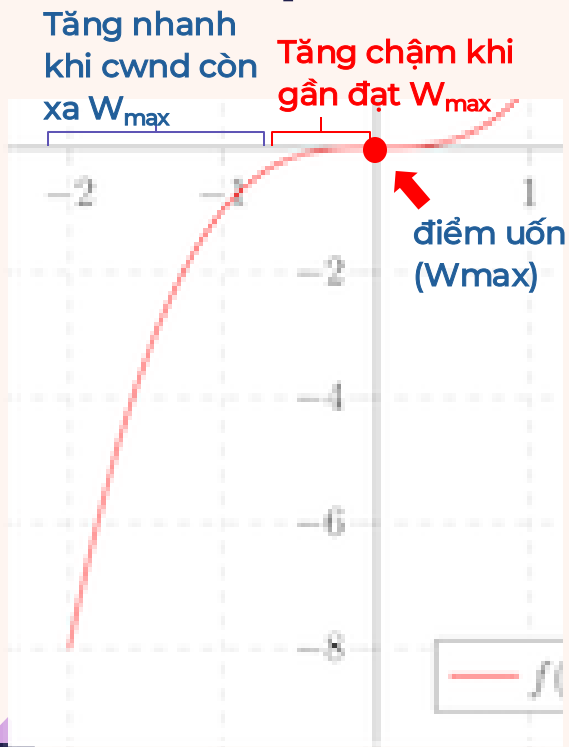
TCP CUBIC – 5. Sau khi tắc nghẽn


5.1 Khi nhận biết tắc nghẽn do có 3 ACK trùng

Khi tắc nghẽn được phát hiện do có 3 ACK trùng, hàm khối được sử dụng để tăng kích thước cửa sổ giúp bên gửi nhanh chóng đạt tới tốc độ gửi tối ưu trong khi vẫn đề phòng tắc nghẽn xảy ra.

TCP CUBIC – 5. Sau khi tắc nghẽn

5.1 Khi nhận biết tắc nghẽn do có 3 ACK trùng





TCP CUBIC – 5. Sau khi tắc nghẽn

5.2 Khi nhận biết tắc nghẽn do hết thời gian timeout

Khi tắc nghẽn được phát hiện do hết thời gian timeout, sau khi cập nhật các giá trị W_{max} , $ssthresh$, và $cwnd$, giai đoạn Phòng tránh tắc nghẽn dừng lại và **giai đoạn Khởi động chậm bắt đầu**.



TCP CUBIC – 6. Ưu điểm và nhược điểm

Ưu điểm

Tăng nhanh đến tốc độ gửi tối ưu

Phản hồi tốt với những thay đổi của đường truyền

Phù hợp với các mạng LFNs

Nhược điểm

Làm tăng Round-trip time

Làm giảm TCP fairness

TCP COMPOUND

1. Lý do ra đời

2. Ý tưởng của TCP Compound

3. Giai đoạn khởi tạo kết nối TCP

4. Khi xảy ra tắc nghẽn

5. Sau khi xảy ra tắc nghẽn

6. Ưu điểm và nhược điểm

TCP COMPOUND – 1. Lý do ra đời

Cũng như TCP Cubic, TCP Compound ra đời nhằm **tận dụng băng thông của các mạng LFNs**.

Các cải tiến của **loss-based congestion control** mặc dù tận dụng đường truyền tốt hơn nhưng **chiếm dụng băng thông** của các luồng khác trên đường truyền.

Các giải thuật chỉ **dựa vào delay** cải thiện throughput nhưng **không cạnh tranh được** với các giải thuật loss-based trên cùng đường truyền.

➡ **Kết hợp cả loss-based và delay-based để tạo ra TCP Compound với 3 nguyên tắc chính là Efficiency, RTT fairness và TCP Fairness.**

TCP COMPOUND – 2. Ý tưởng của TCP Compound

2.1 3 nguyên tắc TCP Compound tuân theo

Efficiency: Làm tăng throughput để sử dụng hiệu quả đường truyền

RTT fairness: Đảm bảo các luồng với RTT khác nhau đều có thể sử dụng đường truyền ổn định.

TCP fairness: Đảm bảo các luồng với cơ chế kiểm soát tắc nghẽn khác nhau đều có thể sử dụng đường truyền ổn định.

TCP COMPOUND – 2. Ý tưởng của TCP Compound

2.2 Ý tưởng chính

TCP Compound dự đoán tắc nghẽn dựa vào cả **2 dấu hiệu là mất gói và delay tăng**.

TCP Compound sử dụng 2 cửa sổ. Cửa sổ thứ nhất là **cwnd** nhận biết tắc nghẽn dựa vào mất gói, **hoạt động giống như TCP Reno**. Cửa sổ thứ 2 là **dwnd nhận biết tắc nghẽn sớm dựa vào độ trễ**, kết hợp với cwnd để điều chỉnh cửa sổ gửi tin (w) một cách hợp lý.

Kích thước cửa sổ gửi tin (w) được tính bằng **tổng của cwnd và dwnd**.

TCP COMPOUND – 2. Ý tưởng của TCP Compound

2.3 Hàm xác định kích thước cửa sổ gửi tin

Kích thước cửa sổ gửi tin tăng theo hàm sau:

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

Kích thước cửa sổ gửi tin giảm theo hàm sau:

$$dwnd(t + 1) = w(t)(1 - \beta) - cwnd/2 \quad (2)$$

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 sẽ thỏa mãn hàm (1), (2).

TCP COMPOUND – 2. Ý tưởng của TCP Compound

2.4 Cửa sổ delay-based xác định RTT như thế nào?

baseRTT là giá trị RTT nhỏ nhất từ lúc khởi tạo kết nối tính tới hiện tại.
RTT là RTT ở thời điểm hiện tại.

Ta có các giá trị sau:

$$\text{Expected} = w/\text{baseRTT}$$

$$\text{Actual} = w/\text{RTT}$$

$$\text{Diff} = \text{Expected} - \text{Actual}$$

Sau khi tính được Diff, ta so sánh Diff với Y để đánh giá tình trạng đường truyền. Nếu $\text{Diff} < Y$ đường truyền vẫn đang hoạt động tốt, ngược lại, đang có dấu hiệu tắc nghẽn sớm trong đường truyền.

TCP COMPOUND – 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.

TCP COMPOUND – 4. Khi xảy ra tắc nghẽn

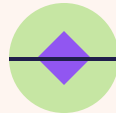
4.1 Đối với cửa sổ loss-based

Trong trường hợp nhận được 3 ACK trùng, các giá trị ssthresh và cwnd được gán theo công thức sau:

$$\text{ssthresh} = \text{cwnd}/2$$

$$\text{cwnd} = \text{cwnd}/2$$

Trong trường hợp hết thời gian timeout, ssthresh cũng được gán như trên nhưng cwnd bị giảm xuống 1 vì TCP Compound cho rằng đây là tình huống tắc nghẽn nghiêm trọng.



TCP COMPOUND – 4. Khi xảy ra tắc nghẽn

4.2 Đối với cửa sổ delay-based

Khi phát hiện tắc nghẽn sớm dựa vào độ trễ (khi $\text{Diff} \geq Y$), cửa sổ delay-based giảm kích thước của nó theo hàm sau:

$$\text{dwnd}(t + 1) = (\text{dwnd} - 3\text{Diff})$$

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

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

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

TCP COMPOUND – 5. Sau khi xảy ra tắc nghẽn

5.1 Đối với cửa sổ loss-based

Trong trường hợp có **3 ACK trùng**, sau khi cwnd giảm đi một nửa, **cwnd tăng thêm 1 sau mỗi Round trip time**.

Trong trường hợp **hết thời gian timeout**, cwnd **bước vào giai đoạn Slow start**, kích thước của nó tăng gấp đôi sau mỗi Round-trip time, tức là tăng thêm 1 MSS sau mỗi ACK nhận được trong round trip.

TCP COMPOUND – 5. Sau khi xảy ra tắc nghẽn

5.2 Đối với cửa sổ delay-based

Kích thước của cửa sổ delay-based sẽ tăng theo hàm sau ($\text{Diff} < \gamma$):

$$\text{dwnd}(t + 1) = \text{dwnd}(t) + \alpha \cdot \text{win}(t)^k - 1$$

TCP COMPOUND – 6. Ưu điểm và nhược điểm

Ưu điểm

Phát hiện và phản ứng sớm với tắc nghẽn

Cải thiện RTT fairness và TCP fairness

Sử dụng hiệu quả đường truyền

Nhược điểm

Phụ thuộc vào độ chính xác của RTT

Phức tạp để cài đặt

TCP BBR

1. Lý do ra đời

2. Ý tưởng của TCP BBR

3. Giai đoạn khởi tạo kết nối

4. Giai đoạn Drain

5. Giai đoạn Bandwidth Probe

6. Giai đoạn RTT Probe

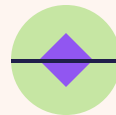
7. Ưu điểm và nhược điểm

TCP BBR – 1. Lý do ra đời

Internet ngày nay đang **không hoạt động đủ nhanh**. Trải nghiệm của đa số những người dùng mạng đang bị trễ từ vài giây cho tới vài phút.

Nguyên nhân chính của những vấn đề đó là hệ thống quản lý tắc nghẽn dựa trên những gói tin bị mất (**loss-based congestion control**) gây ra những hàng đợi dài, **dẫn tới bufferbloat** (khi các thiết bị mạng lưu trữ quá nhiều dữ liệu) và **làm tăng latency**.

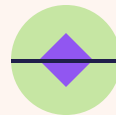
➡ **TCP BBR ra đời giúp giải quyết vấn đề bufferbloat**



TCP BBR – 2. Ý tưởng của TCP BBR

Cố gắng tìm ra và duy trì **tốc độ gửi bằng với BDP** của đường truyền bằng cách BBR sẽ liên tục tính toán băng thông với **lưu lượng truyền dữ liệu tối đa BtlBw (Bottleneck Bandwidth)** và **thời gian truyền với RTT tối thiểu**.

TCP BBR gồm 4 giai đoạn: Startup, Drain, Bandwidth Probe, Round-trip time Probe.



TCP BBR – 3. Giai đoạn khởi tạo kết nối TCP

TCP BBR trải qua giai đoạn Startup và cwnd sẽ được tăng gấp đôi so với giá trị trước đó.

Nó sẽ tự kết luận rằng đã đạt được băng thông tối đa là $BtlBw$ khi băng thông được tính toán không tăng nữa.

TCP BBR – 4. Giai đoạn Drain

Sau khi đạt được giá trị $BtlBw$, TCP BBR kết luận rằng đã đạt được băng thông tối đa.

Khi đó, **1 hàng đợi được tạo ra** ở Bottleneck, TCP BBR sẽ **giảm tốc độ gửi** để triệt tiêu hàng đợi này.

TCP BBR – 5. Giai đoạn Bandwidth Probe

Ta có RTprop (Round trip propagation time) là giá trị RTT nhỏ nhất đo được.

Bandwidth Probe gồm 8 giai đoạn nhỏ, mỗi giai đoạn bằng một RTprop.

Trong suốt quá trình đó, TCP BBR sẽ liên tục **sử dụng lượng băng thông lớn nhất** mà nó tìm ra **để tính BtlBw**. Sau đó sử dụng BtlBW trong 10 RTprop tiếp theo.

TCP BBR – 6. Giai đoạn RTT Probe

Sau 10 giây nếu không tìm được R_{tprop} nào nhỏ hơn giá trị trước đó, TCP BBR sẽ đi vào giai đoạn RTT Probe.

Khi đó, TCP BBR sẽ **giảm cwnd xuống còn 4 để triệt tiêu mọi hàng đợi**. Việc này diễn ra trong vòng $200ms + 1 RTT$.

Nếu một giá trị **R_{tprop} mới được tìm thấy**, TCP BBR **sử dụng nó** cho việc tính toán **trong 10 giây tiếp theo**.

TCP BBR – 7. Ưu điểm và nhược điểm

Ưu điểm

Hoạt động tốt với những kết nối với tỉ lệ mất gói tin không quá cao hoặc không xảy ra mất gói tin

Phù hợp với các mạng thường trì hoãn việc gửi ACK

Nhược điểm

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

So sánh 3 giải thuật TCP Cubic, TCP Compound và TCP BBR

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