**Yêu cầu:**

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

**- So sánh 3 giải thuật trên.**

**- Cho biết ưu, khuyết của 3 giải thuật này.**

**Lưu ý: Tập trung vào các nội dung sau:**

**1. Lý do ra đời của giải thuật ( vấn đề các giải thuật cũ, giải pháp...)**

**2. Giai đoạn khi mới khởi tạo kết nối TCP thì hoạt động kiểm soát tắc nghẽn thế nào.**

**3. Khi gặp tắc nghẽn thì thế nào ?**

**4. Giai đoạn sau tắc nghẽn diễn ra thế nào ?**

——————————————————————————————————————————

**TCP CUBIC** là một giải thuật kiểm soát tắc nghẽn được phát triển để cải thiện hiệu suất truyền tải trong các mạng có băng thông cao và độ trễ lớn (high Bandwidth-Delay Product - BDP). Được giới thiệu lần đầu vào năm 2005, TCP CUBIC đã trở thành giải thuật mặc định trong nhiều hệ điều hành hiện đại, như Linux. Dưới đây là phân tích chi tiết về TCP CUBIC, tập trung vào các khía cạnh quan trọng:

——————————————————————————————————————————

**1. Lý do ra đời của TCP CUBIC**

*- Hạn chế của các giải thuật cũ (như TCP Reno và TCP NewReno):*

- Tăng tốc độ tuyến tính: Các giải thuật này tăng kích thước cửa sổ tắc nghẽn một cách tuyến tính trong giai đoạn tránh tắc nghẽn (Congestion Avoidance). Điều này dẫn đến việc không tận dụng được hết băng thông trong các mạng có băng thông cao và độ trễ lớn, làm chậm tốc độ phục hồi sau tắc nghẽn.

- Quá phụ thuộc vào tín hiệu mất gói: TCP Reno dựa chủ yếu vào việc phát hiện mất gói để điều chỉnh tốc độ truyền. Điều này làm cho nó phản ứng chậm khi băng thông thay đổi và không tối ưu trong môi trường mạng phức tạp.

*- Mục tiêu của TCP CUBIC:*

- Tăng tốc độ truyền phi tuyến tính: Sử dụng hàm CUBIC để điều chỉnh tốc độ tăng trưởng của cửa sổ tắc nghẽn, giúp đạt đến tốc độ truyền tối đa nhanh hơn và sử dụng băng thông một cách hiệu quả hơn.

- Tăng cường hiệu suất trong mạng có BDP cao: TCP CUBIC có thể tận dụng tốt hơn các mạng với BDP cao nhờ vào cách điều chỉnh cửa sổ linh hoạt, giúp duy trì tốc độ truyền ngay cả khi độ trễ tăng.

——————————————————————————————————————————

**2. Hoạt động kiểm soát tắc nghẽn khi khởi tạo kết nối**

- Giai đoạn khởi tạo (Slow Start):

- CUBIC khởi động giống các giải thuật TCP truyền thống với giai đoạn Slow Start: bắt đầu bằng cách tăng gấp đôi kích thước cửa sổ tắc nghẽn (Congestion Window - CWND) sau mỗi chu kỳ RTT (Round-Trip Time) cho đến khi xảy ra hiện tượng mất gói hoặc đạt ngưỡng ssthresh (slow start threshold).

- Khi mất gói hoặc đạt ngưỡng, nó sẽ chuyển sang giai đoạn Congestion Avoidance, sử dụng hàm CUBIC để kiểm soát kích thước cửa sổ tắc nghẽn.

——————————————————————————————————————————

**3. Hoạt động khi gặp tắc nghẽn**

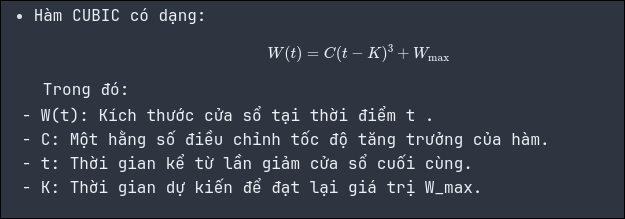
- Phản ứng với tín hiệu tắc nghẽn:

- TCP CUBIC sử dụng hàm CUBIC dựa trên phương trình bậc ba để điều chỉnh tốc độ truyền sau khi phát hiện tắc nghẽn. Cụ thể, nó xác định một điểm inflection (W\_max), là kích thước cửa sổ tại thời điểm xảy ra tắc nghẽn gần nhất.

- Khi có tín hiệu mất gói, kích thước cửa sổ giảm xuống một mức nhất định, nhưng sau đó sẽ tăng trở lại dựa trên hàm CUBIC.

- Hàm CUBIC:

- Hàm CUBIC được thiết kế để tăng kích thước cửa sổ với tốc độ nhanh hơn trong các giai đoạn ban đầu (gần điểm inflection W\_max) và chậm lại khi gần đạt đến W\_max. Sau đó, nó tiếp tục tăng tốc độ với một nhịp độ ổn định để tìm kiếm thêm băng thông.



——————————————————————————————————————————

**4. Ứng xử sau tắc nghẽn**

- Giai đoạn sau tắc nghẽn:

- Sau khi giảm kích thước cửa sổ do mất gói, TCP CUBIC sẽ tăng kích thước cửa sổ một cách phi tuyến tính, tăng nhanh hơn khi còn cách xa W\_max và chậm lại khi gần đến W\_max.

- Khi vượt qua điểm W\_max, tốc độ tăng của kích thước cửa sổ sẽ chậm hơn, giúp tránh hiện tượng tắc nghẽn lặp lại. Nếu không có tín hiệu tắc nghẽn, cửa sổ tiếp tục tăng dần theo phương trình CUBIC, khai thác thêm băng thông có sẵn.

- Điều chỉnh tốc độ truyền sau tắc nghẽn:

- TCP CUBIC có thể phục hồi từ sự sụt giảm băng thông một cách nhanh chóng nhờ vào cơ chế tăng phi tuyến. Điều này giúp nó đạt được tốc độ truyền ổn định sau khi gặp tắc nghẽn mà không cần phải mất thời gian quá lâu để phục hồi, như các giải thuật TCP tuyến tính khác.

——————————————————————————————————————————

**5. Ưu điểm và nhược điểm của TCP CUBIC**

| Ưu điểm | Nhược điểm |
| --- | --- |
| - Hiệu suất cao trong mạng có BDP lớn: TCP CUBIC tận dụng tốt băng thông trên các kết nối có băng thông cao và độ trễ lớn, giúp đạt được tốc độ truyền cao hơn. | - Hiệu suất không tối ưu trong mạng có độ trễ thấp: Trong mạng có độ trễ thấp, CUBIC có thể hoạt động không hiệu quả bằng các giải thuật khác như TCP Reno. |
| - Phục hồi sau tắc nghẽn nhanh: Việc tăng tốc độ phi tuyến giúp CUBIC phục hồi nhanh hơn sau tắc nghẽn. | Sự phức tạp của hàm CUBIC: So với các giải thuật đơn giản như TCP Reno, việc sử dụng hàm CUBIC bậc ba phức tạp hơn trong việc điều chỉnh. |
| Tương thích với các giải thuật TCP khác: CUBIC duy trì tính công bằng với các kết nối TCP khác bằng cách tăng tốc độ truyền một cách thận trọng khi gần điểm W\_max. | - Phản ứng chậm với sự thay đổi đột ngột: Khi băng thông thay đổi đột ngột, TCP CUBIC có thể mất một khoảng thời gian để tự điều chỉnh đến tốc độ truyền mới. |

——————————————————————————————————————————

**6. Kết luận**

TCP CUBIC là một giải pháp hiệu quả cho vấn đề tận dụng băng thông trên các mạng hiện đại có BDP cao. Với cách tiếp cận phi tuyến và việc sử dụng hàm CUBIC bậc ba để kiểm soát tắc nghẽn, CUBIC mang lại hiệu suất truyền tải vượt trội so với các giải thuật cũ như TCP Reno. Tuy nhiên, trong các mạng có độ trễ thấp, CUBIC có thể không hoạt động tối ưu như mong đợi. Do đó, việc lựa chọn TCP CUBIC phụ thuộc vào đặc điểm của mạng và yêu cầu cụ thể về hiệu suất truyền tải.

**References**

1. https://en.wikipedia.org/wiki/CUBIC\_TCP

2. https://www.cs.princeton.edu/courses/csarchive/fall16/cos561/papers/Cubic08.pdf

3.

| Ưu điểm | Nhược điểm |
| --- | --- |
| Hiệu suất cao trong mạng có BDP lớn: | Hiệu suất không tối ưu trong mạng có độ trễ thấp |
| Phục hồi sau tắc nghẽn nhanh | Sự phức tạp của hàm CUBIC |
| Tương thích với các giải thuật TCP khác | Phản ứng chậm với sự thay đổi đột ngột |