|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **BÁO CÁO BÀI TẬP LỚN**  **MÔN HỆ THỐNG VIỄN THÔNG**  **ĐỀ TÀI: Tìm hiểu về TCP-CUBIC**   |  |  |  | | --- | --- | --- | | **Giảng viên hướng dẫn:** | **PGS.TS Nguyễn Thành Chuyên** | | | **Mã lớp học:** | **133399** | | | **Nhóm sinh viên** | **Nhóm 07** | | |  | **Phạm Hồng Hùng** | **20182558** | |  | **Nguyễn Đức Bình** | **20182380** | |  | **Vũ Đăng Tình**  **Đàm Khắc Trình** | **20182823**  **20182384** |   **Hà Nội, 7-2022** |

**LỜI NÓI ĐẦU**

Tài liệu này được Nhóm sinh viên soạn thảo trong quá trình làm bài tập lớn cho môn học Hệ thống viễn thông dưới sự hướng dẫn của **PGS.TS Nguyễn Thành Chuyên**. Do đây là tài liệu được soạn thảo trong quá trình học tập và tìm hiểu nên có thể có nhiều điểm sai sót, mong quý thầy cô cũng như bạn đọc thông cảm và góp ý thêm để tài liệu trở nên đúng đắn hơn.

**LỜI CAM ĐOAN**

Những người soạn thảo tài liệu này xin được cam đoan rằng tài liệu này không được sử dụng để làm bài tập lớn cũng như đồ án cho bất kỳ một môn học nào khác. Cùng với đó cũng đảm bảo tài liệu này là do chính chúng tôi soạn thảo dựa vào những tài liệu tham khảo chứ không sao chép của nhóm sinh viên khác.

**MỤC LỤC**

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT 1.1](#_Toc108130141)

[TÓM TẮT 1.2](#_Toc108130142)

[CHƯƠNG 1. TỔNG QUAN VỀ TCP 1](#_Toc108130143)

[1.1 Giới thiệu TCP 1](#_Toc108130144)

[1.2 Hoạt động của giao thức 2](#_Toc108130145)

[1.2.1 Thiết lập kết nối 4](#_Toc108130146)

[1.2.2 Truyền dữ liệu 4](#_Toc108130147)

[1.2.3 Kết thúc kết nối 6](#_Toc108130148)

[1.2.4 Cấu trúc gói tin 7](#_Toc108130149)

[1.3 Nhận xét 8](#_Toc108130150)

[CHƯƠNG 2. CUBIC - Một biến thể TCP tốc độ cao mới 12](#_Toc108130151)

[2.1 Giới thiệu CUBIC 12](#_Toc108130152)

[2.2 CUBIC kiểm soát tắc nghẽn trong TCP 12](#_Toc108130153)

[2.2.1 BIC-TCP 12](#_Toc108130154)

[2.2.2 Hàm tăng cửa sổ CUBIC 14](#_Toc108130155)

[2.2.3 Vùng thân thiện với TCP 16](#_Toc108130156)

[2.2.4 Vùng lõm 19](#_Toc108130157)

[2.2.5 Vùng lồi 19](#_Toc108130158)

[2.2.6 Giảm số nhân 19](#_Toc108130159)

[2.2.7 Hội tụ nhanh 20](#_Toc108130160)

[2.3 CUBIC TRONG LINUX KERNEL 20](#_Toc108130161)

[2.3.1 Sự phát triển của CUBIC trong Linux 20](#_Toc108130162)

[2.3.2 Mô-đun tắc nghẽn có thể lắp ráp 21](#_Toc108130163)

[2.4 Thảo luận 22](#_Toc108130164)

[2.4.1 Cân bằng với TCP tiêu chuẩn 23](#_Toc108130165)

[2.4.2 Hoạt động của CUBIC 24](#_Toc108130166)

[2.5 Thực nghiệm và mô phỏng 27](#_Toc108130167)

[2.5.1 Thiết lập thử nghiệm 27](#_Toc108130168)

[2.5.2 Mô phỏng trên NS2 29](#_Toc108130169)

[2.5.3 Coding implement bằng C 32](#_Toc108130170)

[KẾT LUẬN 33](#_Toc108130171)

[TÀI LIỆU THAM KHẢO 34](#_Toc108130172)

# DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

|  |  |
| --- | --- |
| TCP | Transmission Control Protocol |
| RTT | Round Trip Time |
| IP | Internet Protocol |
| UDP | User Datagram Protocol |
| ACK | Acknowledgement |
| BDP | Bandwidth Delay Product |
| cwnd | Congestion Window |
| BIC | Binary Increase Congestion |

TÓM TẮT

Hiện nay, có rất nhiều giao thức truyền dữ liệu trên Internet. Trong đó, phổ biến có lẽ là giao thức truyền tải TCP. TCP là viết tắt của cụm từ Transmission Control Protocol, tức là giao thức điều khiển truyền nhận. Giao thức này đóng vai trò kiểm tra và đảm bảo sự chuyển giao thông tin từ nơi nguồn tới nơi nhận một cách an toàn và đúng thứ tự.Một giao thức kiểm soát tắc nghẽn cho TCP là CUBIC.Thời gian qua chúng em đã nghiên cứu về đề tài “Tìm hiểu hoạt động của TCP CUBIC tại lớp truyền vận”. Đây là một đề tài hay và giúp chúng em tìm hiểu sâu hơn về lớp truyền vân trong TCP.Được sự hướng dẫn tận tình của thầy Nguyễn Thành Chuyên nhóm em đã phân chia công việc và hoàn thành nội dung của đề tài được giao. Do kiến thức còn hạn hẹp chưa chuyên sâu nên không tránh khỏi những sai sót, chúng em mong nhận được những nhận xét và góp ý từ phía thầy cô để giúp cho nghiên cứu của chúng em hoàn thiện hơn.

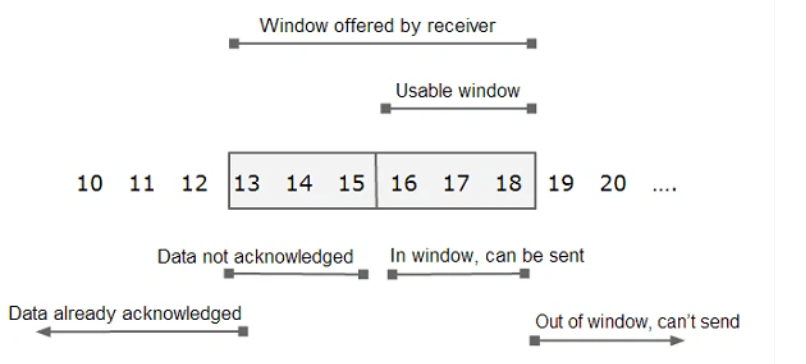
# TỔNG QUAN VỀ TCP

Trình bày cái nhìn sơ lược về TCP và cách hoạt động của TCP cũng như những khái niệm cơ bản và nêu ra những ưu, nhược điểm của TCP, từ đó đề xuất giải pháp cải tiến, phần sẽ được nói chi tiết ở chương sau.

## Giới thiệu TCP

TCP (Transmission Control Protocol) là một giao thức mạng quan trọng được sử dụng trong việc truyền dữ liệu qua một mạng nào đó. Một giao thức trong phạm vi mạng là một tập hợp các quy tắc và trình tự kiểm soát việc thực hiện truyền dữ liệu sao cho tất cả mọi người trên thế giới bất kể vị trí địa lý, bất kể ứng dụng, phần mềm họ đang sử dụng đều có thể thao tác theo cùng một phương thức giống nhau được gọi là TCP. TCP thường kết hợp với IP (Giao thức Internet) theo một cặp được gọi là TCP/IP. IP sẽ xử lý việc gán địa chỉ và chuyển tiếp các gói tin từ nguồn đến đích trong khi TCP kiểm soát độ tin cậy của truyền dẫn.

TCP có chức năng là kiểm soát mức độ tin cậy của việc truyền dữ liệu trên các mạng như Internet, dữ liệu được truyền theo dạng gói tin, các gói này là các cụm dữ liệu được truyền hoàn toàn độc lập trên mạng, được tập hợp lại với nhau khi chúng đến địa chỉ đích và sau đó trả về dữ liệu gốc. Truyền dữ liệu trên mạng được xử lý theo các lớp, mỗi một giao thức trên một lớp sẽ thực hiện công việc bổ sung cho các lớp khác. Tập hợp các lớp này được gọi là các ngăn giao thức (protocol stack). TCP và IP làm việc liên quan chặt chẽ với nhau, lớp này trên lớp kia.



Hình 1:Window congestion

TCP dán nhãn các gói tin theo dạng đánh số. TCP cũng sẽ đảm bảo rằng dữ liệu tới đích trong một thời hạn xác định (một khoảng thời gian vài trăm mili giây được gọi là thời gian chờ) và tuân theo một số quy định kỹ thuật khác. Với mỗi gói tin nhận được, thiết bị gửi sẽ được thông báo thông qua một gói được gọi là xác nhận. Sau khi hết thời gian chờ, không nhận được xác nhận, nguồn gửi sẽ gửi đi một bản sao của gói tin bị mất hoặc bị hoãn. Các gói tin không theo trình tự cũng sẽ không được xác nhận. Nhờ vậy, tất cả các gói dữ liệu sẽ luôn được tập hợp theo thứ tự, không có sơ hở, trong một khoảng thời gian chờ xác định và chấp nhận được.

Tóm lại, ta có thể hiểu rằng TCP ở đây là 1 giao thức truyền và nhận dữ liệu trong một mạng nào đó giữa các thiết bị và giao thức này đóng vai trò kiểm tra và đảm bảo sự chuyển giao thông tin từ nơi nguồn tới nơi nhận một cách an toàn và đúng thứ tự.

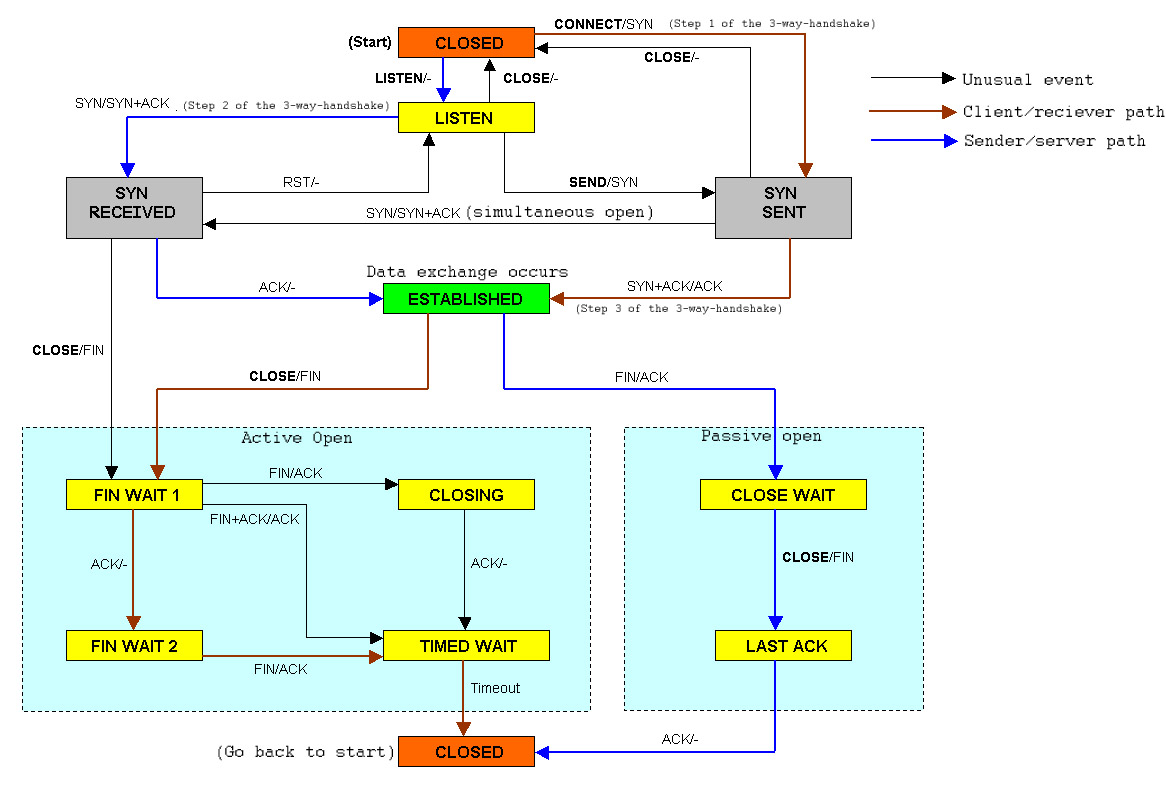
## Hoạt động của giao thức

Không như giao thức UDP - giao thức có thể lập tức gửi gói tin mà không cần thiết lập kết nối, TCP đòi hỏi thiết lập kết nối trước khi bắt đầu gửi dữ liệu và kết thúc kết nối khi việc gửi dữ liệu hoàn tất. Cụ thể, các kết nối TCP có ba pha:

1. Thiết lập kết nối
2. Truyền dữ liệu
3. Kết thúc kết nối

Trước khi miêu tả các pha này, ta cần lưu ý các trạng thái khác nhau của một socket:

* LISTEN
* SYN-SENT
* SYN-RECEIVED
* ESTABLISHED
* FIN-WAIT
* CLOSE-WAIT
* CLOSING
* LAST-ACK
* TIME-WAIT
* CLOSER



Hình 2: Trạng thái của socket

Trong đó đáng chú ý là các trạng thái sau:

* **LISTEN :** đang đợi yêu cầu kết nối từ một TCP và cổng bất kỳ ở xa (trạng thái này thường do các TCP server đặt)
* **SYN-SENT:** đang đợi TCP ở xa gửi một gói tin TCP với các cờ SYN và ACK được bật (trạng thái này thường do các TCP client đặt)
* **SYN-RECEIVED:** đang đợi TCP ở xa gửi lại một tin báo nhận sau khi đã gửi cho TCP ở xa đó một tin báo nhận kết nối (*connection acknowledgment*) (thường do TCP server đặt)
* **ESTABLISHED:** cổng đã sẵn sàng nhận/gửi dữ liệu với TCP ở xa (đặt bởi TCP client và server)
* **TIME-WAIT:** đang đợi qua đủ thời gian để chắc chắn là TCP ở xa đã nhận được tin báo nhận về yêu cầu kết thúc kết nối của nó. Theo [RFC 793](http://tools.ietf.org/html/793) (tài liệu năm 1981), một kết nối có thể ở tại trạng thái TIME-WAIT trong vòng tối đa 4 phút.

### Thiết lập kết nối

Để thiết lập một kết nối, TCP sử dụng một quy trình bắt tay 3 bước (3-way handshake). Trước khi client thử kết nối với một server, server phải đăng ký một cổng và mở cổng đó cho các kết nối: đây được gọi là mở bị động. Một khi mở bị động đã được thiết lập thì một client có thể bắt đầu mở chủ động. Để thiết lập một kết nối, quy trình bắt tay 3 bước xảy ra như sau:

1. Client yêu cầu mở cổng dịch vụ bằng cách gửi gói tin SYN (gói tin TCP) tới server, trong gói tin này, tham số sequence number được gán cho một giá trị ngẫu nhiên X.
2. Server hồi đáp bằng cách gửi lại phía client bản tin SYN-ACK, trong gói tin này, tham số acknowledgment number được gán giá trị bằng X + 1, tham số sequence number được gán ngẫu nhiên một giá trị Y
3. Để hoàn tất quá trình bắt tay ba bước, client tiếp tục gửi tới server bản tin ACK, trong bản tin này, tham số sequence number được gán cho giá trị bằng X + 1 còn tham số acknowledgment number được gán giá trị bằng Y + 1

Tại thời điểm này, cả client và server đều được xác nhận rằng, một kết nối đã được thiết lập.

### Truyền dữ liệu

Một số đặc điểm cơ bản của TCP để phân biệt với UDP:

* Truyền dữ liệu không lỗi (do có cơ chế sửa lỗi/truyền lại)
* Truyền các gói dữ liệu theo đúng thứ tự
* Truyền lại các gói dữ liệu mất trên đường truyền
* Loại bỏ các gói dữ liệu trùng lặp
* Cơ chế hạn chế tắc nghẽn đường truyền

Ở hai bước đầu tiên trong ba bước , hai máy tính trao đổi một số thứ tự gói ban đầu (Initial Sequence Number -ISN). Số này có thể chọn một cách ngẫu nhiên. Số thứ tự này được dùng để đánh dấu các khối dữ liệu gửi từ mỗi máy tính. Sau mỗi byte được truyền đi, số này lại được tăng lên. Nhờ vậy ta có thể sắp xếp lại chúng khi tới máy tính kia ,bất kể các gói tới nơi theo thứ tự thế nào.

Trên lý thuyết, mỗi byte gửi đi đều có một số thứ tự và khi nhận được thì máy tính nhận gửi lại tin báo nhận (ACK). Trong thực tế thì chỉ có byte dữ liệu đầu tiên được gán số thứ tự trong trường số thứ tự của gói tin và bên nhận sẽ gửi tin báo nhận bằng cách gửi số thứ tự của byte đang chờ.

Ví dụ: Máy tính A gửi 4 byte với số thứ tự ban đầu là 100 (theo lý thuyết thì 4 byte sẽ có thứ tự là 100, 101, 102, 103) thì bên nhận sẽ gửi tin báo nhận có nội dung là 104 vì đó là thứ tự của byte tiếp theo nó cần. Bằng cách gửi tin báo nhận là 104, bên nhận đã ngầm thông báo rằng nó đã nhận được các byte 100, 101, 102 và 103. Trong trường hợp 2 byte cuối bị lỗi thì bên nhận sẽ gửi tin báo nhận với nội dung là 102 vì 2 byte 100 và 101 đã được nhận thành công.

Giả sử ta có 10.000 byte được gửi đi trong 10 gói tin 1.000 byte và có 1 gói tin bị mất trên đường truyền. Nếu gói bị mất là gói đầu tiên thì bên gửi sẽ phải gửi lại toàn bộ 10 gói vì không có cách nào để bên nhận thông báo nó đã nhận được 9 gói kia. Vấn đề này được giải quyết trong giao thức SCTP (Stream Control Transmission Protocol - "Giao thức điều khiển truyền vận dòng") với việc bổ sung báo nhận chọn lọc.

Số thứ tự và tin báo nhận giải quyết được các vấn đề về lặp gói tin, truyền lại những gói bị hỏng/mất và các gói tin đến sai thứ tự. Để phục vụ mục đích kiểm tra, các gói tin có trường giá trị tổng kiểm (checksum).

Với trình độ hiện tại, kỹ thuật kiểm tra tổng trong TCP không đủ mạnh. Các tầng liên kết dữ liệu với xác suất lỗi bit cao có thể cần được bổ sung các khả năng phát hiện lỗi tốt hơn. Nếu như TCP được thiết kế vào thời điểm hiện tại, nhiều khả năng nó sẽ bao gồm trường kiểm tra độ dư tuần hoàn (cyclic redundancy check - CRC) với độ dài 32 bit. Điểm yếu này một phần được bù đắp bằng CRC hay những kỹ thuật khác tại tầng thứ 2 (trong mô hình 7 lớp OSI) ở bên dưới cả TCP và IP như trong các giao thức điểm-điểm (PPP) hoặc Ethernet. Tuy nhiên điều này cũng không có nghĩa là trường kiểm tra tổng của TCP là không cần thiết: thống kê cho thấy các sai sót do cả phần cứng và phần mềm gây ra giữa các điểm áp dụng kỹ thuật kiểm tra CRC là khá phổ biến và kỹ thuật kiểm tra tổng có khả năng phát hiện phần lớn các lỗi (đơn giản) này.

Điểm cuối cùng là khả năng hạn chế tắc nghẽn.

Tin báo nhận (hoặc không có tin báo nhận) là tín hiệu về tình trạng đường truyền giữa

2 máy tính. Từ đó, hai bên có thể thay đổi tốc độ truyền nhận dữ liệu phù hợp với điều kiện. Vấn đề này thường được đề cập là điều khiển lưu lượng, kiểm soát tắc nghẽn. TCP sử dụng một số cơ chế nhằm đạt được hiệu suất cao và ngăn ngừa khả năng nghẽn mạng. Các cơ chế này bao gồm: cửa sổ trượt (sliding window), thuật toán slow-start, thuật toán tránh nghẽn mạng (congestion avoidance), thuật toán truyền lại và phục hồi nhanh,...

Hiện nay, vấn đề cải tiến TCP trong môi truyền truyền dẫn tốc độ cao đang là một hướng nghiên cứu được quan tâm.

#### Kích thước cửa sổ TCP

Chuỗi số thứ tự gói và cửa sổ trong TCP hoạt động giống như một cái đồng hồ. Kích thước của cửa sổ (đo bằng byte) được thiết lập bởi khả năng tiếp nhận của máy tính nhận. Cửa sổ này được dịch đi mỗi khi máy tính nhận, nhận được dữ liệu và gửi tin báo nhận. Khi chuỗi thứ tự tăng đến tối đa thì lại quay lại về 0.

Kích thước của cửa sổ là chiều dài (byte) của khối dữ liệu có thể lưu trong bộ đệm của bên nhận. Bên gửi chỉ có thể gửi tối đa lượng thông tin chứa trong cửa sổ này trước khi nhận được tin báo nhận.

#### Dãn kích thước cửa sổ

Để tận dụng khả năng truyền dẫn của mạng thì cửa sổ dùng trong TCP cần được tăng lên. Trường điều khiển kích thước cửa sổ của gói TCP có độ dài là 2 byte và do đó kích thước tối đa của cửa sổ là 65.535 byte. Do trường điều khiển không thể thay đổi nên người ta sử dụng một hệ số dãn nào đó. Hệ số này được định nghĩa trong tài liệu RFC 1323 có thể sử dụng để tăng kích thước tối đa của cửa sổ từ 65.535 byte lên tới 1 gigabyte. Tăng kích thước cửa sổ lớn hơn nữa cũng cần thiết trong TCP Tuning.

Việc tăng kích thước cửa sổ chỉ được dùng trong giao thức bắt tay 3 pha. Giá trị của trường co giãn cửa sổ thể hiện số bit cần được dịch trái đối với trường kích thước cửa sổ. Hệ số dãn có thể thay đổi từ 0 (không dãn) tới 14 (dãn tối đa).

### Kết thúc kết nối

Để kết thúc kết nối hai bên sử dụng quá trình bắt tay 4 bước và chiều của kết nối kết thúc độc lập với nhau. Khi một bên muốn kết thúc, nó gửi đi một gói tin FIN và bên kia gửi lại tin báo nhận ACK. Vì vậy, một quá trình kết thúc thường sẽ có 2 cặp gói tin

trao đổi. Một kết nối có thể tồn tại ở dạng "nửa mở": một bên đã kết thúc gửi dữ liệu nên chỉ nhận thông tin, bên kia vẫn tiếp tục gửi.

### Cấu trúc gói tin

Một gói tin TCP bao gồm 2 phần

* header (có độ dài 20 bytes)
* dữ liệu

Phần header có 11 trường trong đó 10 trường bắt buộc. Trường thứ 11 là tùy chọn (trong bảng minh họa có màu nền đỏ) có tên là: options

#### Header

|  |
| --- |
|  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **+** | **Bít 0 - 3** | **4 - 9** | **10 - 15** | **16 - 31** |
| **0** | Source Port | | | Destination Port |
| **32** | Sequence Number | | | |
| **64** | Acknowledgement Number | | | |
| **96** | Data Offset | Reserved | Flags | Window |
| **128** | Checksum | | | Urgent Pointer |
| **160** | Options (optional) | | | |
| **160/192+** | Data | | | |

Hình 3: cấu trúc gói tin

* **Source port:** Số hiệu của cổng tại máy tính gửi.
* **Destination port** :Số hiệu của cổng tại máy tính nhận.
* **Sequence number :**Trường này có 2 nhiệm vụ. Nếu cờ SYN bật thì nó là số thứ tự gói ban đầu và byte đầu tiên được gửi có số thứ tự này cộng thêm 1. Nếu không có cờ SYN thì đây là số thứ tự của byte đầu tiên.
* **Acknowledgement number :**Nếu cờ ACK bật thì giá trị của trường chính là số thứ tự gói tin tiếp theo mà bên nhận cần.
* **Data offset:** Trường có độ dài 4 bit quy định độ dài của phần header (tính theo đơn vị từ 32 bit). Phần header có độ dài tối thiểu là 5 từ (160 bit) và tối đa là 15 từ (480 bit).
* **Reserved :**Dành cho tương lai và có giá trị là 0.
* **Flags (hay Control bits)** Bao gồm 6 cờ:
  + URG: Cờ cho trường Urgent pointer
  + ACK :Cờ cho trường Acknowledgement
  + PSH :Hàm Push
  + RST: Thiết lập lại đường truyền
  + SYN :Đồng bộ lại số thứ tự
  + FIN :Không gửi thêm số liệu
* **Window :**Số byte có thể nhận bắt đầu từ giá trị của trường báo nhận (ACK)
* **Checksum :**16 bit kiểm tra cho cả phần header và dữ liệu. Phương pháp sử dụng được mô tả trong RFC 793: 16 bit của trường kiểm tra là bổ sung của tổng tất cả các từ 16 bit trong gói tin. Trong trường hợp số octet (khối 8 bit) của header và dữ liệu là lẻ thì octet cuối được bổ sung với các bít 0. Các bit này không được truyền. Khi tính tổng, giá trị của trường kiểm tra được thay thế bằng 0,

Nói một cách khác, tất cả các từ 16 bit được cộng với nhau. Kết quả thu được sau khi đảo giá trị từng bit được điền vào trường kiểm tra. Về mặt thuật toán, quá trình này giống với IPv4.

* **Options:** Đây là trường tùy chọn. Nếu có thì độ dài là bội số của 32 bit.
* **Data**(Dữ liệu) :Trường cuối cùng không thuộc về header. Giá trị của trường này là thông tin dành cho các tầng trên (trong mô hình 7 lớp OSI). Thông tin về giao thức của tầng trên không được chỉ rõ trong phần header mà phụ thuộc vào cổng được chọn.

## Nhận xét

Khi Internet phát triển ,bao gồm nhiều đường dẫn mạng đường dài và tốc độ rất cao, hiệu suất của TCP đã bị thách thức. Các mạng này được đặc trưng bởi băng thông lớn và tích độ trễ (BDP) thể hiện tổng số gói tin cần thiết trong quá trình bay trong khi vẫn giữ băng thông được sử dụng đầy đủ, nói cách khác, là kích thước của cửa sổ tắc nghẽn. Trong TCP tiêu chuẩn như TCP-Reno, TCPNewReno và TCP-SACK, TCP mở rộng cửa sổ của nó một lần mỗi lần khứ hồi (RTT). Điều này làm cho tốc độ vận chuyển dữ liệu của TCP được sử dụng trong tất cả các hệ điều hành chính bao gồm Windows và Linux khá chậm chạp, ít nhất là việc sử dụng mạng cực kỳ kém hiệu quả, đặc biệt nếu độ dài của các luồng ngắn hơn nhiều so với thời gian TCP phát triển các cửa sổ của nó, kích thước đầy đủ BDP của một đường dẫn. Ví dụ: nếu băng thông của đường dẫn mạng là 10 Gbps và RTT là 100 ms, với các gói 1250 byte, thì BDP của đường dẫn là khoảng 100.000 gói. Để TCP phát triển cửa sổ của nó từ điểm giữa của BDP, ví dụ 50.000, thì cần khoảng 50.000 RTT, tương đương với 5000 giây (1,4 giờ). Nếu một luồng kết thúc trước thời điểm đó, nó sử dụng kém đường dẫn một cách nghiêm trọng. Để giải quyết vấn đề sử dụng kém này của TCP, nhiều biến thể TCP “tốc độ cao” được đề xuất (ví dụ: FAST, HSTCP, STCP, HTCP, SQRT và BIC-TCP). Nhận thức được vấn đề này với TCP, cộng đồng Linux đã nhanh chóng phản hồi để triển khai phần lớn các giao thức này trong Linux và gửi chúng như một phần của hệ điều hành của nó. Sau một loạt thử nghiệm của bên thứ ba và xác nhận hiệu suất, vào năm 2004, từ phiên bản 2.6.8, nó đã chọn BIC-TCP làm thuật toán TCP mặc định và các biến thể TCP khác là tùy chọn. Điều làm cho BIC-TCP nổi bật so với các tổ hợp TCP khác là tính ổn định của nó. Nó sử dụng thuật toán tìm kiếm nhị phân trong đó cửa sổ phát triển đến điểm giữa giữa kích thước cửa sổ cuối cùng (tức là tối đa), nơi TCP bị mất gói và kích thước cửa sổ cuối cùng (tức là, tối thiểu) nó không bị mất đối với một RTT, giai đoạn = Stage. Việc “tìm kiếm” điểm giữa này trực quan có ý nghĩa vì dung lượng của đường dẫn hiện tại phải nằm giữa hai kích thước cửa sổ tối thiểu và tối đa nếu điều kiện mạng không nhanh chóng thay đổi kể từ tín hiệu tắc nghẽn cuối cùng (là lần mất gói cuối cùng ).

Sau khi cửa sổ phát triển đến điểm giữa, nếu mạng không bị mất gói, thì điều đó có nghĩa là mạng có thể xử lý nhiều lưu lượng hơn và do đó BIC-TCP đặt điểm giữa là min mới và thực hiện một “binary- tìm kiếm ”với các cửa sổ tối thiểu và tối đa. Điều này có tác dụng tăng cửa sổ thực sự nhanh khi kích thước cửa sổ hiện tại khác xa so với

dung lượng khả dụng của đường dẫn và nếu nó gần với dung lượng khả dụng(đã bị mất trước đó), nó sẽ từ từ giảm cửa sổ .Nó có gia số cửa sổ nhỏ nhất tại điểm bão hòa và lượng vượt quá của nó vượt quá điểm bão hòa, nơi tổn thất xảy ra rất nhỏ. Toàn bộ hàm tăng trưởng cửa sổ chỉ đơn giản là một hàm lõm logarit. Hàm lõm này giữ cho cửa sổ tắc nghẽn ở điểm bão hòa hoặc điểm cân bằng lâu hơn nhiều so với hàm lồi hoặc hàm tuyến tính nơi chúng có gia số cửa sổ lớn nhất tại điểm bão hòa và do đó có độ vọt lố lớn nhất tại thời điểm xảy ra mất gói. Các tính năng này làm cho BIC-TCP rất ổn định và đồng thời có khả năng mở rộng cao. BIC-TCP giao dịch tốc độ để phản ứng với những thay đổi trong băng thông khả dụng (tức là tốc độ hội tụ) để ổn định. Nếu dung lượng khả dụng đã tăng lên kể từ lần mất gói cuối cùng, cửa sổ có thể phát triển vượt quá mức tối đa mà không bị mất. Khi đó, BIC-TCP tăng cửa sổ theo cấp số nhân. Lưu ý rằng một hàm mũ (một hàm lồi) phát triển rất chậm khi bắt đầu (chậm hơn một hàm tuyến tính). Tính năng này bổ sung thêm tính ổn định của giao thức bởi vì ngay cả khi giao thức mắc lỗi khi tìm cửa sổ tối đa, nó sẽ tìm cửa sổ tối đa tiếp theo gần điểm tối đa trước đó trước, do đó sẽ ở lại điểm bão hòa trước đó lâu hơn. Nhưng hàm mũ nhanh chóng bắt kịp và gia số của nó trở nên rất lớn nếu tổn thất không xảy ra (trong trường hợp đó, điểm bão hòa đã trở nên lớn hơn nhiều so với điểm trước đó). Bởi vì nó ở gần điểm bão hòa trước đó lâu hơn so với các biến thể khác, nó có thể chậm chạp để tìm điểm bão hòa mới nếu điểm bão hòa đã tăng quá xa so với điểm cuối cùng. Tuy nhiên, BIC-TCP phản ứng nhanh một cách an toàn với dung lượng giảm vì mất gói xảy ra trước mức tối đa trước đó và nó làm giảm cửa sổ theo hệ số nhân. Sự cân bằng này là một lựa chọn thiết kế của BIC-TCP. Người ta biết rằng băng thông có sẵn trên Internet thay đổi trong một khoảng thời gian dài vài giờ. Cho rằng tổn thất gói sẽ xảy ra rất không đồng bộ và cũng tỷ lệ thuận với việc tiêu thụ băng thông của một luồng trong môi trường ghép kênh có thống kê cao, hội tụ nhanh là hệ quả tự nhiên của môi trường mạng - điều mà giao thức không bắt buộc. Do đó, mặc dù BIC-TCP có thể hội tụ chậm trong điều kiện ghép kênh thống kê thấp, nơi chỉ có một số luồng đang cạnh tranh, nhưng tốc độ hội tụ của nó không phải là vấn đề trong môi trường Internet điển hình. CUBIC là phiên bản tiếp theo của BIC-TCP. Nó đơn giản hóa đáng kể thuật toán điều chỉnh cửa sổ của BIC-TCP bằng cách thay thế các phần tăng trưởng cửa sổ lồi và lõm của BIC-TCP bằng một hàm hình khối (chứa cả hai phần lõm và lồi).

Trong thực tế, bất kỳ hàm đa thức bậc lẻ nào cũng có hình dạng này. Sự lựa chọn cho một hàm khối là ngẫu nhiên và không thuận tiện. Đặc điểm chính của CUBIC là sự tăng trưởng cửa sổ của nó chỉ phụ thuộc vào thời gian thực giữa hai sự kiện tắc nghẽn liên tiếp. Một sự kiện tắc nghẽn là thời gian TCP trải qua quá trình phục hồi nhanh chóng. Chúng tôi gọi thời gian thực này là kỷ nguyên tắc nghẽn. Do đó, tăng trưởng cửa sổ trở nên độc lập với RTT. Tính năng này cho phép các luồng CUBIC cạnh tranh trong cùng một nút cổ chai có cùng kích thước cửa sổ độc lập với RTT của chúng, đạt được sự công bằng RTT tốt. Khi RTT ngắn, vì tốc độ tăng trưởng cửa sổ được cố định, tốc độ tăng trưởng của nó có thể chậm hơn so với tiêu chuẩn TCP. Vì các tiêu chuẩn TCP (ví dụ: TCP-SACK) hoạt động tốt trong các RTT ngắn, tính năng này nâng cao tính thân thiện với TCP của giao thức. Việc triển khai CUBIC trong Linux đã trải qua một số nâng cấp. Nâng cấp đáng chú ý nhất là thực hiện hiệu quả tính toán căn bậc ba. Vì nó yêu cầu một phép toán dấu phẩy động, nên việc triển khai nó yêu cầu một số xấp xỉ số nguyên. Ban đầu nó sử dụng phương pháp phân giác và sau đó được đổi thành phương pháp Newton Raphson giúp giảm chi phí tính toán gần 10 lần. Một thay đổi khác đối với CUBIC sau khi thành lập là việc loại bỏ kẹp cửa sổ. Kẹp cửa sổ đã được giới thiệu trong BIC-TCP nơi số gia cửa sổ được kẹp với mức tăng tối đa và được kế thừa cho CUBIC cho phiên bản đầu tiên. Điều này buộc tăng trưởng cửa sổ phải tuyến tính khi điểm giữa của mục tiêu lớn hơn nhiều so với kích thước cửa sổ hiện tại. CUBIC đã thay thế BIC-TCP làm thuật toán TCP mặc định vào năm 2006 sau phiên bản 2.6.18.

# CUBIC - Một biến thể TCP tốc độ cao mới

Ở phần này, trình bày chi tiết về CUBIC cũng như cách thức hoạt động và cách thức triển khai, từ đó đưa ra những đánh giá và kết luận. Nội dung của phần này được dựa phần lớn theo nội dung của bài báo “CUBIC: A New TCP-Friendly High-Speed TCP Variant” của các nhà nghiên cứu Sangtae Ha, Injong Rhee và Lisong Xu

## Giới thiệu CUBIC

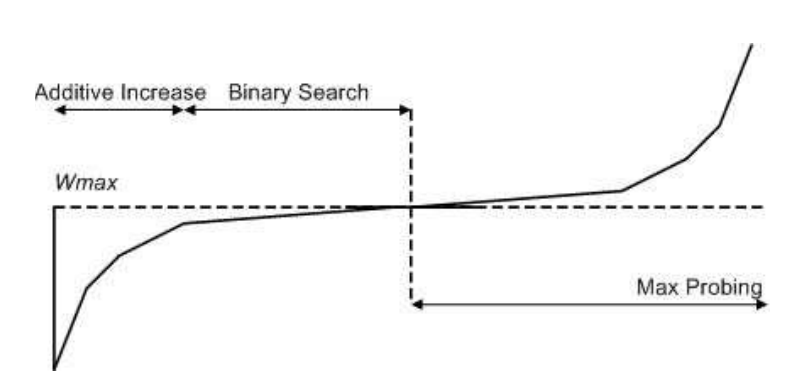
CUBIC là một giao thức kiểm soát tắc nghẽn cho TCP (giao thức điều khiển đường truyền) và thuật toán TCP mặc định hiện tại trong Linux. Giao thức sửa đổi hàm tăng trưởng cửa sổ tuyến tính của các tiêu chuẩn TCP hiện có thành một hàm khối để cải thiện khả năng mở rộng của TCP qua các mạng đường dài và nhanh. Nó cũng đạt được sự phân bổ băng thông công bằng hơn giữa các luồng có RTT khác nhau (thời gian khứ hồi) bằng cách làm cho tăng trưởng cửa sổ độc lập với RTT - do đó các luồng đó tăng cửa sổ tắc nghẽn của chúng với cùng một tốc độ. Trong trạng thái ổn định, CUBIC tăng kích thước cửa sổ một cách mạnh mẽ khi cửa sổ ở xa điểm bão hòa và càng chậm khi nó gần điểm bão hòa. Tính năng này cho phép CUBIC có khả năng mở rộng rất cao khi băng thông và sản phẩm trễ của mạng lớn, đồng thời, có độ ổn định cao và cũng công bằng với các luồng TCP tiêu chuẩn. Việc triển khai CUBIC trong Linux đã trải qua một số nâng cấp.

## CUBIC kiểm soát tắc nghẽn trong TCP

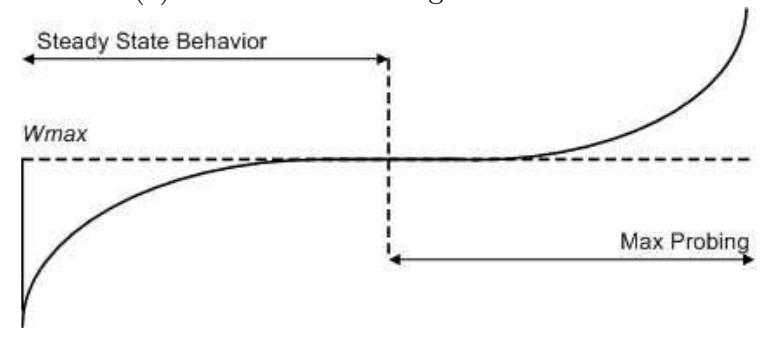
### BIC-TCP

Trong phần này, chúng tôi cung cấp một số chi tiết về **BIC-TCP** là một tiền thân của **CUBIC**. Tính năng chính của **BIC**-**TCP** là hàm tăng trưởng cửa sổ của nó như đã thảo luận trong phần giới thiệu. Hình 3 cho thấy hàm tăng trưởng của **BIC**-**TCP**. Khi gặp sự kiện mất gói, **BIC**-**TCP** giảm cửa sổ của nó bằng một hệ số nhân *β*. Kích thước cửa sổ ngay trước khi giảm được đặt thành *Wmax* tối đa và kích thước cửa sổ ngay sau

khi giảm được đặt thành *Wmin* tối thiểu. Sau đó, **BIC**-**TCP** thực hiện tìm kiếm nhị phân bằng cách sử dụng hai tham số này - bằng cách nhảy đến "điểm giữa" giữa *Wmax* và *Wmin*. Vì mất gói đã xảy ra ở *Wmax*, kích thước cửa sổ mà mạng hiện có thể xử lý mà không bị mất phải ở đâu đó giữa hai số này. Tuy nhiên, việc nhảy đến điểm giữa có thể tăng quá nhiều trong một **RTT**, vì vậy nếu khoảng cách giữa điểm giữa và điểm tối thiểu hiện tại lớn hơn một hằng số cố định, được gọi là *Smax* , gia số **BIC-TCP** cwnd bằng *Smax* (tăng tuyến tính). Nếu **BIC**-**TCP** không bị mất gói ở kích thước cửa sổ được cập nhật, thì kích thước cửa sổ đó sẽ trở thành mức tối thiểu mới. Quá trình này tiếp tục cho đến khi mức tăng cửa sổ nhỏ hơn một số hằng số nhỏ được gọi là *Smin* tại thời điểm đó, cửa sổ được đặt ở mức tối đa hiện tại. Vì vậy, hàm tăng trưởng sau khi giảm cửa sổ sẽ có nhiều khả năng là một hàm tuyến tính theo sau bởi một hàm logarit (được đánh dấu là “tăng cộng” và “tìm kiếm nhị phân” tương ứng trong Hình 3(a).) Nếu cửa sổ tăng quá mức tối đa, thì kích thước cửa sổ cân bằng phải lớn hơn mức tối đa hiện tại và phải tìm thấy mức tối đa mới. **BIC-TCP** bước vào một giai đoạn mới được gọi là “thăm dò tối đa”. Thăm dò tối đa sử dụng một hàm tăng trưởng cửa sổ đối xứng chính xác với các hàm được sử dụng trong tăng cộng và tìm kiếm nhị phân (là logarit; nghịch đảo của nó sẽ là cấp số nhân) và sau đó tăng cộng. Hình 3(a) hiển thị chức năng tăng trưởng trong quá trình thăm dò tối đa. Trong quá trình thăm dò tối đa, ban đầu cửa sổ phát triển chậm để tìm mức tối đa mới gần đó và sau một thời gian tăng chậm, nếu nó không tìm thấy mức tối đa mới (tức là mất gói), thì nó đoán mức tối đa mới ở xa hơn nên nó chuyển sang mức tăng nhanh hơn bằng cách chuyển sang mức tăng phụ gia trong đó kích thước cửa sổ được tăng lên theo mức tăng cố định lớn. Hiệu suất tốt của **BIC-TCP** đến từ sự tăng chậm xung quanh *Wmax* và tăng tuyến tính trong quá trình tăng phụ gia và thăm dò tối đa.



a) BIC-TCP



b) CUBIC

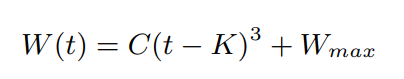
Hình 3 : Hàm tăng cửa sổ của **BIC-TCP** và CUBIC

### Hàm tăng cửa sổ CUBIC

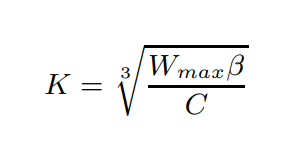
**BIC-TCP** đạt được khả năng mở rộng tốt trong các mạng tốc độ cao, công bằng giữa các luồng cạnh tranh của riêng nó và ổn định với dao động cửa sổ thấp. Tuy nhiên, chức năng tăng trưởng của **BIC-TCP** vẫn có thể quá mạnh đối với **TCP**, đặc biệt là trong các mạng **RTT** ngắn hoặc tốc độ thấp. Hơn nữa, một số giai đoạn khác nhau (tăng tìm kiếm nhị phân, thăm dò tối đa, *Smax* và *Smin)* của kiểm soát cửa sổ làm tăng thêm sự phức tạp trong việc triển khai giao thức và phân tích hiệu suất của nó. Chúng tôi đã tìm kiếm một hàm tăng trưởng cửa sổ mới mà vẫn giữ được các điểm mạnh của **BIC**-**TCP** (đặc biệt là tính ổn định và khả năng mở rộng của nó), đơn giản hóa việc kiểm soát cửa sổ và nâng cao tính thân thiện với **TCP** của nó. Chúng tôi giới thiệu một biến thể **TCP** tốc độ cao mới: **CUBIC**. Như tên của giao thức đại diện, hàm tăng trưởng cửa sổ của **CUBIC** là một hàm hình khối có hình dạng rất giống với hàm tăng trưởng của **BIC**-**TCP**. **CUBIC** sử dụng một hàm khối của thời gian đã trôi qua từ sự kiện tắc nghẽn cuối cùng. Trong khi hầu hết các thuật toán thay thế cho Standard **TCP** sử dụng hàm tăng lồi

trong đó sau một sự kiện mất mát, gia số cửa sổ luôn tăng lên, **CUBIC** sử dụng cả biên dạng lõm và lồi của một hàm khối để tăng cửa sổ. Hình 3 (b) cho thấy chức năng tăng trưởng của **CUBIC**. Các chi tiết của **CUBIC** như sau. Sau khi giảm cửa sổ sau một sự kiện mất mát, nó đăng ký *Wmax* là kích thước cửa sổ nơi sự kiện mất mát xảy ra và thực hiện giảm nhân số cửa sổ tắc nghẽn theo hệ số *β* trong đó *β* là hằng số giảm cửa sổ và khôi phục nhanh và truyền lại thường xuyên **TCP**. Sau khi nó đi vào trạng thái tránh tắc nghẽn do khôi phục nhanh, nó bắt đầu tăng cửa sổ bằng cách sử dụng biên dạng lõm của hàm khối. Hàm lập phương được thiết lập để có giá trị bình nguyên tại *Wmax* nên sự tăng lõm tiếp tục cho đến khi kích thước cửa sổ trở thành *Wmax*. Sau đó, hàm lập phương biến thành một biên dạng lồi và sự phát triển cửa sổ lồi bắt đầu. Phong cách điều chỉnh cửa sổ này (lõm và sau đó lồi) cải thiện giao thức và độ ổn định mạng trong khi vẫn duy trì hiệu suất sử dụng mạng cao. Điều này là do kích thước cửa sổ gần như không đổi, tạo thành một bình nguyên xung quanh *Wmax* nơi mức sử dụng mạng được coi là cao nhất và ở trạng thái ổn định, hầu hết các mẫu kích thước cửa sổ của **CUBIC** đều gần với *Wmax*, do đó thúc đẩy việc sử dụng mạng cao hơn và ổn định giao thức. Lưu ý rằng các giao thức có hàm tăng trưởng lồi có xu hướng có gia số cửa sổ lớn nhất xung quanh điểm bão hòa, dẫn đến một loạt tổn thất gói lớn.

Hàm tăng trưởng cửa sổ của CUBIC :

(1)

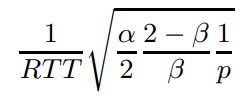
trong đó *C* là tham số **CUBIC**, *t* là thời gian trôi qua kể từ lần giảm cửa sổ cuối cùng và *K* là khoảng thời gian mà hàm trên cần để tăng *W* đến *Wmax* khi không có sự kiện mất mát nào nữa và được tính bằng cách sử dụng công thức sau:

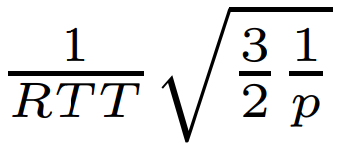
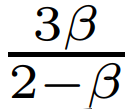
 (2)

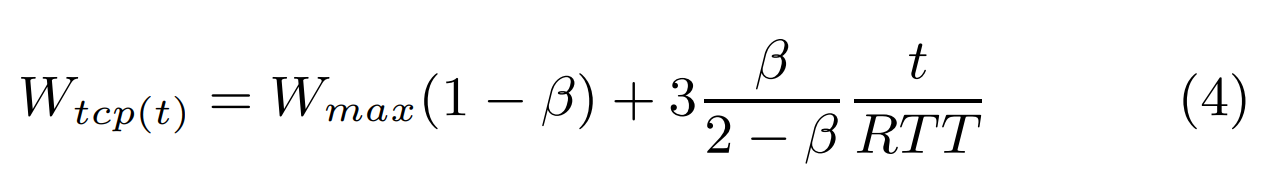
Khi nhận được **ACK** trong quá trình tránh tắc nghẽn, **CUBIC** sẽ tính toán tỷ lệ tăng trưởng cửa sổ trong khoảng thời gian **RTT** tiếp theo bằng cách sử dụng công thức (1). Nó đặt *W (t + RTT)* làm giá trị mục tiêu ứng viên của cửa sổ tắc nghẽn. Giả sử rằng kích thước cửa sổ hiện tại là *cwnd*. Tùy thuộc vào giá trị của *cwnd*, **CUBIC** chạy ở ba chế độ khác nhau. Đầu tiên, nếu *cwnd* nhỏ hơn kích thước cửa sổ (tiêu chuẩn) **TCP** sẽ đạt được tại thời điểm t sau sự kiện mất mát cuối cùng, thì **CUBIC** đang ở chế độ **TCP** (chúng tôi mô tả bên dưới cách xác định kích thước cửa sổ này của TCP tiêu chuẩn theo thời gian *t*). Ngược lại, nếu cwnd nhỏ hơn *Wmax* thì **CUBIC** nằm trong vùng lõm và nếu cwnd lớn hơn *Wmax* thì **CUBIC** nằm trong vùng lồi. Thuật toán 1 cho thấy mã giả của thuật toán điều chỉnh cửa sổ của **CUBIC** được thực hiện trong Linux.

### Vùng thân thiện với TCP

Khi nhận được **ACK** để tránh tắc nghẽn, trước tiên chúng tôi kiểm tra xem giao thức có nằm trong vùng **TCP** hay không. Điều này được thực hiện như sau. Chúng ta có thể phân tích kích thước cửa sổ của **TCP** theo thời gian đã trôi qua t. Sử dụng một phân tích đơn giản trong, chúng ta có thể tìm thấy kích thước cửa sổ trung bình của phép cộng cộng và giảm số nhân (**AIMD**) với hệ số cộng *α* và hệ số nhân *β* là hàm sau:

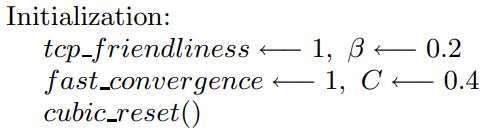
 (3)

Theo phân tích tương tự, kích thước cửa sổ trung bình của **TCP** với α = 1 và *β* = 0,5 là  .Do đó, đối với công thức 3 giống như của **TCP**, α phải bằng . Nếu **TCP** tăng cửa sổ của nó lên α trên mỗi **RTT**, chúng ta có thể nhận được kích thước cửa sổ của **TCP** theo thời gian đã trôi qua t như sau:

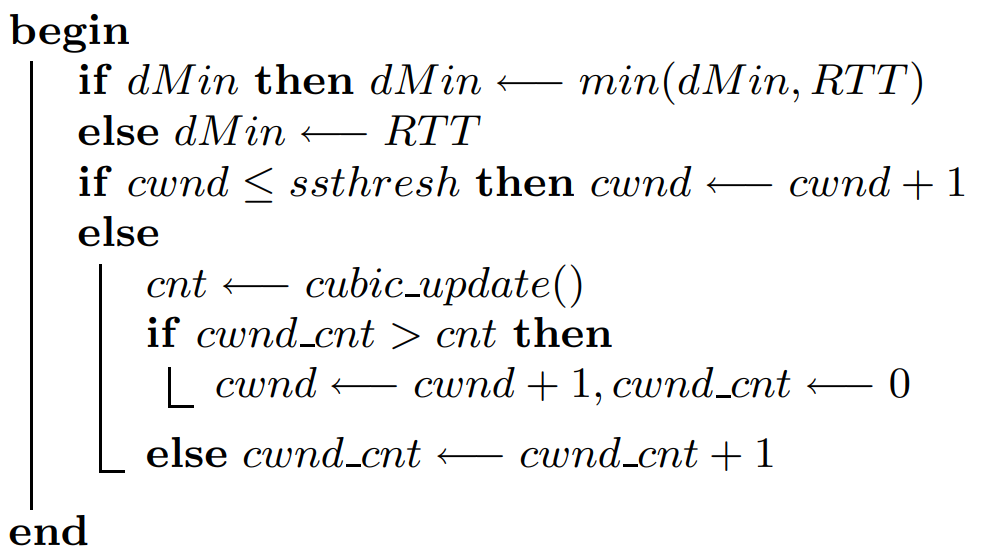


Nếu *cwnd* nhỏ hơn *Wtcp (t*) , thì giao thức nằm trong TCP chế độ và *cwnd* được đặt thành Wtcp (t) tại mỗi lần nhận **ACK**. **cubic\_tcp\_friendrity()** trong Thuật toán 1 mô tả hành vi này.

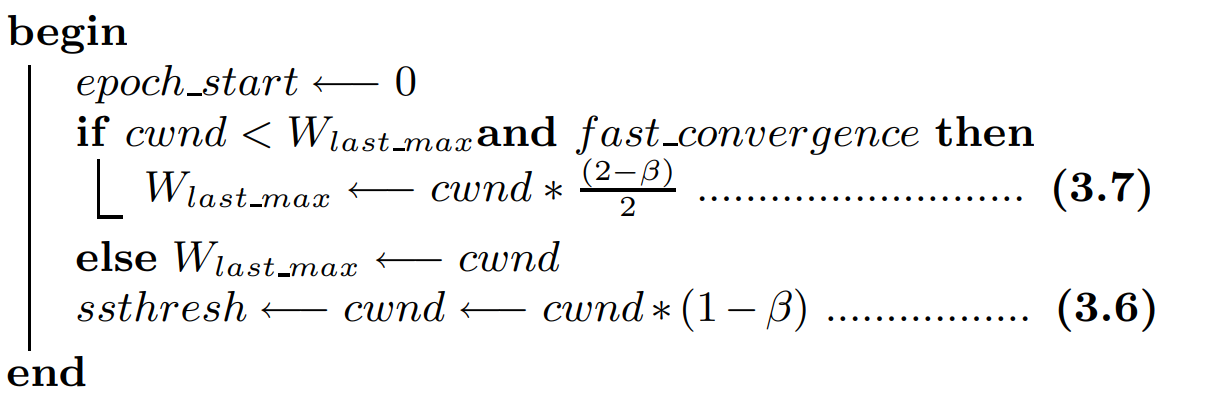
Thuật toán 1: thuật toán **CUBIC** trong Linux

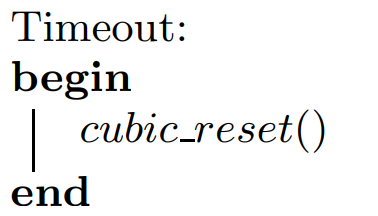


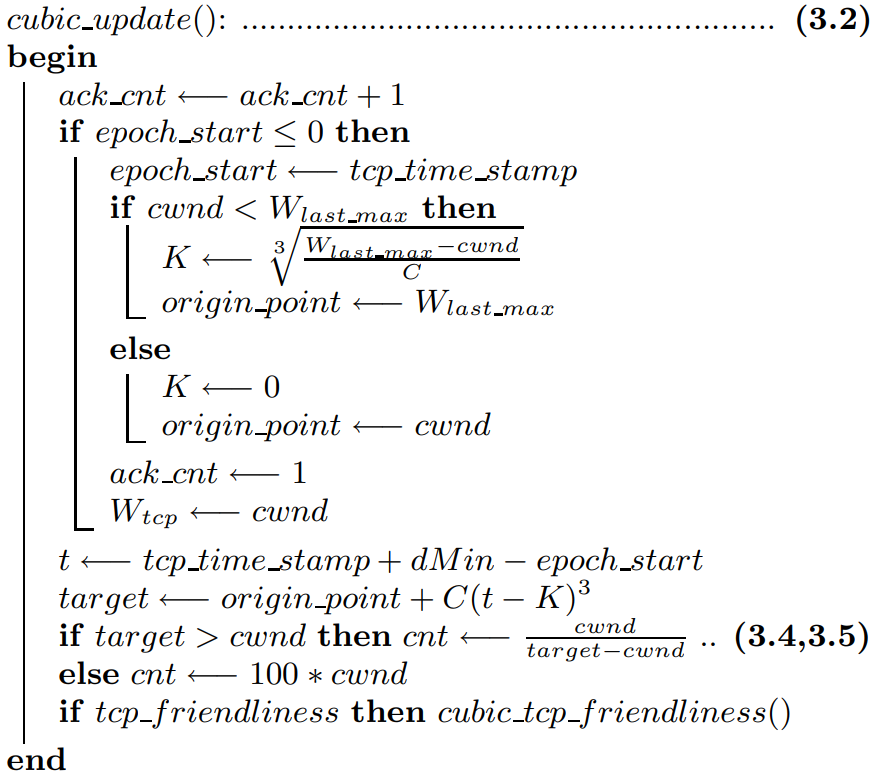
Với mỗi ACK:

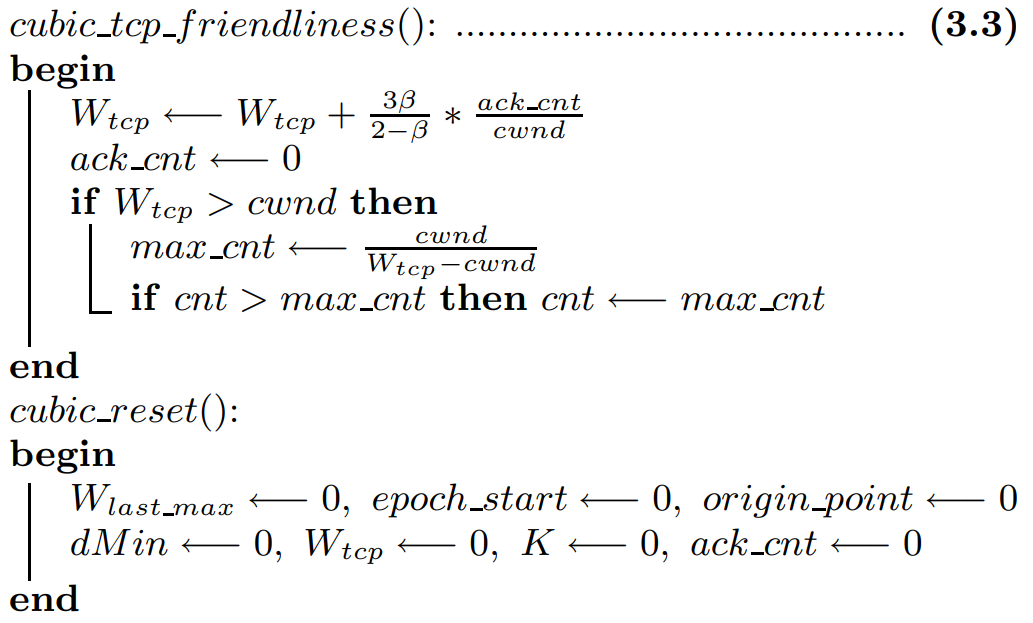


Mất gói:

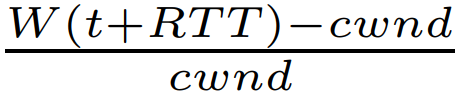








### Vùng lõm

Khi nhận **ACK** trong trường hợp tránh tắc nghẽn, nếu giao thức không ở chế độ **TCP** và *cwnd* nhỏ hơn *Wmax*, thì giao thức nằm trong vùng lõm. Trong vùng này, *cwnd* được tăng lên bởi  , được hiển thị tại (3.4) trong Thuật toán 1.

### Vùng lồi

Khi kích thước cửa sổ của **CUBIC** lớn hơn *Wmax*, nó sẽ vượt qua bình nguyên của hàm lập phương mà sau đó **CUBIC** tuân theo biên dạng lồi của hàm lập phương. Vì *cwnd* lớn hơn điểm bão hòa *Wmax* trước đó, điều này cho thấy rằng các điều kiện mạng có thể đã bị xáo trộn kể từ sự kiện mất mát cuối cùng, có thể ngụ ý băng thông khả dụng hơn sau khi một số luồng xuất phát. Vì Internet có tính không đồng bộ cao nên luôn tồn tại những biến động về băng thông khả dụng. Biên dạng lồi đảm bảo rằng cửa sổ tăng rất chậm lúc đầu và tăng dần tốc độ tăng trưởng của nó. Chúng tôi cũng gọi giai đoạn này là giai đoạn thăm dò cực đại vì **CUBIC** đang tìm kiếm *Wmax* mới. Vì chúng tôi không sửa đổi hàm tăng cửa sổ chỉ cho vùng lồi, nên chức năng tăng cửa sổ cho cả hai vùng vẫn không thay đổi. Nói một cách chính xác, nếu giao thức là vùng lồi bên ngoài chế độ **TCP**, *cwnd* được tăng lên bởi *,* được hiển thị tại (3.5) trong Thuật toán số 1.

### Giảm số nhân

Khi xảy ra mất gói, **CUBIC** giảm kích thước cửa sổ của nó bằng hệ số *β*. Chúng tôi đặt *β* là 0,2. Một tác dụng phụ của việc đặt *β* thành giá trị nhỏ hơn 0,5 là sự hội tụ chậm hơn. Chúng tôi tin rằng trong khi cài đặt thích ứng hơn của *β* có thể dẫn đến hội tụ nhanh hơn, nó sẽ làm cho việc phân tích giao thức khó hơn nhiều và cũng ảnh hưởng đến tính ổn định của giao thức. Điều chỉnh thích nghi này của *β* là một vấn đề nghiên cứu trong tương lai.

### Hội tụ nhanh

Để cải thiện tốc độ hội tụ của **CUBIC**, chúng tôi thêm phương pháp heuristic trong giao thức. Khi một luồng mới tham gia vào mạng, các luồng hiện có trong mạng cần phải từ bỏ việc chia sẻ băng thông của chúng để cho phép luồng mới có chỗ phát triển. Để tăng giải phóng băng thông này theo các luồng hiện có, chúng tôi thêm cơ chế sau được gọi là hội tụ nhanh. Với tính năng hội tụ nhanh, khi sự kiện mất mát xảy ra, trước khi giảm cửa sổ tắc nghẽn, giao thức sẽ ghi nhớ giá trị cuối cùng của *Wmax* trước khi

cập nhật *Wmax* cho sự kiện mất mát hiện tại. Chúng ta hãy gọi giá trị cuối cùng của *Wmax* là *Wlastmax*. Tại sự kiện mất mát, nếu giá trị hiện tại của *Wmax* nhỏ hơn giá trị cuối cùng của nó, *Wlastmax*, điều này cho thấy rằng điểm bão hòa mà luồng này trải qua đang bị giảm do sự thay đổi trong băng thông khả dụng. Sau đó, chúng tôi cho phép luồng này giải phóng nhiều băng thông hơn bằng cách giảm *Wmax* hơn nữa. Hành động này có hiệu quả kéo dài thời gian để dòng chảy này tăng cửa sổ của nó bởi vì *Wmax* giảm buộc dòng chảy phải bình nguyên sớm hơn. Điều này cho phép nhiều thời gian hơn để quy trình mới bắt kịp kích thước cửa sổ của nó. Mã giả cho phép toán này được hiển thị tại (3.7) trong Thuật toán 1.

## CUBIC TRONG LINUX KERNEL

### Sự phát triển của CUBIC trong Linux

Hầu hết các bản cập nhật trên **CUBIC** đều tập trung vào cải tiến hiệu suất và hiệu quả triển khai. Một trong những tối ưu hóa đáng chú ý là cải tiến về tính toán căn bậc ba. Việc thực hiện **CUBIC** yêu cầu giải quyết công thức 2, một phép tính căn bậc ba. Việc triển khai ban đầu của **CUBIC** trong Linux sử dụng phương pháp phân chia. Nhưng cộng đồng các nhà phát triển Linux đã làm việc cùng nhau để thay thế nó bằng phương pháp Newton-Rhaphson giúp cải thiện thời gian chạy trung bình hơn 10 lần (1032 đồng hồ so với 79 đồng hồ) và giảm phương sai trong thời gian chạy. CUBIC cũng đã trải qua một số thay đổi về thuật toán để có hình thức hiện tại nhằm nâng cao khả năng mở rộng, tính công bằng và tốc độ hội tụ.

### Mô-đun tắc nghẽn có thể lắp ráp

Việc đưa thêm nhiều biến thể TCP vào nhân Linux đã làm tăng đáng kể độ phức tạp của mã TCP trong hạt nhân. Mặc dù một thuật toán TCP mới đi kèm với một bản vá cho hạt nhân, quá trình này đòi hỏi phải thường xuyên biên dịch lại hạt nhân và làm trầm trọng thêm tính ổn định của mã TCP. Để loại bỏ nhu cầu biên dịch lại hạt nhân và giúp thử nghiệm một thuật toán TCP mới với Linux, Stephen Hemminger giới thiệu một kiến trúc mới, được gọi là mô-đun tắc nghẽn có thể cắm được, trong Linux 2.6.13. Nó có thể tải

động và cho phép chuyển đổi giữa các môđun thuật toán điều khiển tắc nghẽn khác nhau một cách nhanh chóng mà không cần biên dịch lại. Hình 4 cho thấy giao diện của mô-đun này, được đặt tên là các hoạt động tắc nghẽn tcp. Mỗi phương thức trong *tcp\_congestion\_ops* là một hook trong mã TCP cung cấp quyền truy cập vào mã TCP. Một thuật toán kiểm soát tắc nghẽn mới yêu cầu xác định *cong\_avoid* và *ssthresh*, nhưng các phương pháp khác là tùy chọn. Các hàm init và release được gọi để khởi tạo và kết thúc một thuật toán TCP nhất định. *ssthresh* là ngưỡng bắt đầu chậm được gọi khi TCP đã cho phát hiện mất mát. Giới hạn dưới trên cửa sổ tắc nghẽn là ngưỡng bắt đầu chậm, nhưng khi kiểm soát tắc nghẽn cần ghi đè giới hạn dưới này, *min\_cwnd* có thể được sử dụng cho mục đích đó. *cong\_avoid* được gọi bất cứ khi nào ACK đến và cửa sổ tắc nghẽn (*cwnd)* được điều chỉnh. Ví dụ, trong TCP chuẩn New-Reno, khi một ACK đến, hãy tránh tăng từng *cwnd* nếu *cwnd* hiện tại nhỏ hơn *ssthresh* (trong khi khởi động chậm). Nếu không, hãy tránh gia tăng *cwnd* bằng (trong quá trình tránh tắc nghẽn). *set\_state* được gọi khi trạng thái kiểm soát tắc nghẽn của TCP được thay đổi trong số Bình thường, Khôi phục tổn thất, Khôi phục tổn thất sau khi hết thời gian, sắp xếp lại và Giảm cửa sổ tắc nghẽn. Sự kiện *cwnd* được gọi khi các sự kiện được định nghĩa trong *tcp\_ca\_event* xảy ra. Khi một thuật toán yêu cầu xử lý một trong các sự kiện, nó có thể tạo một *hook* tới *cwnd\_event* được gọi khi sự kiện tương ứng xảy ra. *undo\_cwnd* xử lý phát hiện sai về việc mất hoặc hết thời gian. Khi TCP nhận ra sự thay đổi đối với cwnd là sai, nó sẽ quay trở lại *cwnd* ban đầu bằng cách sử dụng *undo\_cwnd. pkts\_acked* là một móc để đếm ACK; nhiều giao thức (ví dụ: BIC-TCP, CUBIC và H-TCP) cũng sử dụng móc này để lấy thông tin RTT. *get info* là một móc để cung cấp thông tin kiểm soát tắc nghẽn cho không gian người dùng. CUBIC đã được thực hiện như một trong những mô-đun kiểm soát tắc nghẽn có thể cắm được. Sau đây là các *hook* mà CUBIC sử dụng để triển khai nó.

· *bictcp \_init*: khởi tạo các biến riêng được sử dụng cho thuật toán CUBIC. Nếu *ssthresh* ban đầu không phải là 0, thì hãy đặt ssthresh thành giá trị này. Nếu *ssthresh* ban đầu được người dùng đặt đúng cách khi không có thông tin lịch sử về đường dẫn end-to-end, nó có thể cải thiện đáng kể hành vi khởi động của CUBIC.

· *bictcp\_recalc\_ssthresh*: Nếu chế độ hội tụ nhanh được bật và *cwnd* hiện tại nhỏ hơn giá trị cực đại cuối cùng, hãy đặt giá trị cực đại cuối cùng thành *cwnd ∗ (1 – β/2).*

Nếu không, hãy đặt giá trị tối đa cuối cùng thành *cwnd ∗ (1 - β).* *ssthresh* luôn được đặt thành *cwnd ∗ (1 - β)* vì TCP cần phải lùi lại khi bị tắc nghẽn.

· *bictcp \_cong\_avoid*: tăng cwnd bằng cách tính toán sự khác biệt giữa giá trị cwnd hiện tại và giá trị dự kiến của vòng RTT tiếp theo thu được bằng phép tính căn bậc hai.

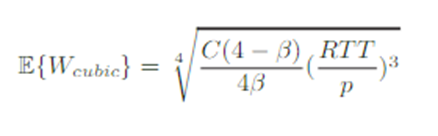
· *bictcp\_set\_state*: đặt lại tất cả các biến khi thời gian chờ xảy ra.

· *bictcp\_undo\_cwnd:* trả về giá trị lớn nhất giữa giá trị *cwnd* hiện tại và *last\_max* (là cửa sổ tắc nghẽn trước khi thả).

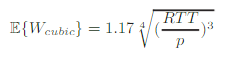
· *bictcp\_acked*: duy trì độ trễ tối thiểu được quan sát cho đến nay. Độ trễ tối thiểu được đặt lại khi hết thời gian chờ xảy ra.

## Thảo luận

Với mô hình tổn thất xác định trong đó số gói tin giữa hai sự kiện tổn thất liên tiếp luôn là 1 / p, CUBIC luôn hoạt động với cấu hình cửa sổ lõm giúp đơn giản hóa đáng kể việc phân tích hiệu suất của CUBIC. Kích thước cửa sổ trung bình của CUBIC có thể được lấy bằng hàm sau:



Để đảm bảo tính công bằng cho Standard TCP dựa trên lập luận của chúng tôi trong phần giới thiệu, chúng tôi đặt *C* thành 0,4. Chúng tôi thấy rằng giá trị này của *C* cho phép kích thước của vùng thân thiện với TCP đủ lớn để bao gồm hầu hết các môi trường mà Standard TCP hoạt động tốt trong khi vẫn bảo toàn khả năng mở rộng của chức năng phát triển cửa sổ. Với *β* đặt thành 0,2, công thức trên được rút gọn thành hàm sau:



Nó được sử dụng để tranh luận về tính công bằng của CUBIC đối với TCP Chuẩn và sự an toàn của nó khi triển khai bên dưới.

### Cân bằng với TCP tiêu chuẩn

Trong môi trường mà TCP tiêu chuẩn có thể sử dụng hợp lý băng thông có sẵn, CUBIC không thay đổi đáng kể trạng thái này.

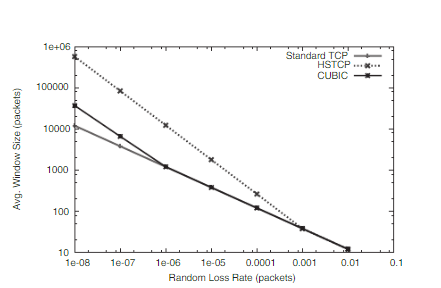
TCP chuẩn hoạt động tốt trong hai loại mạng sau:

1. mạng có sản phẩm độ trễ băng thông nhỏ (BDP).

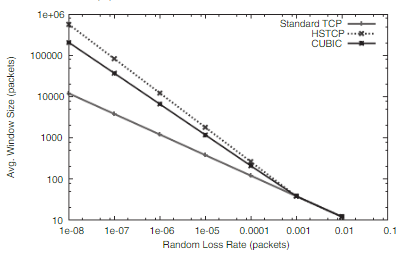
2. mạng có RTT ngắn, nhưng không nhất thiết phải là BDP nhỏ.

CUBIC được thiết kế để hoạt động rất giống với TCP tiêu chuẩn trong hai loại mạng trên. Hình 4 cho thấy chức năng phản hồi (kích thước cửa sổ trung bình) của TCP, HSTCP và CUBIC tiêu chuẩn. Kích thước cửa sổ trung bình của TCP và HSTCP tiêu chuẩn là từ [15]. Kích thước cửa sổ trung bình của CUBIC được tính bằng cách sử dụng (6) và phương trình thân thiện với CUBIC TCP trong (4). Hình 4 cho thấy CUBIC thân thiện với TCP hơn HSTCP, đặc biệt là trong các mạng có RTT ngắn mà TCP hoạt động khá tốt. Ví dụ, trong mạng có *RTT = 10ms* và *p =* , TCP có một cửa sổ trung bình là 1200 gói. Nếu kích thước gói là 1500 byte, thì TCP có thể đạt được tốc độ trung bình là 1,44 Gbps. Trong trường hợp

này, CUBIC đạt được tốc độ giống hệt như Standard TCP, trong khi HSTCP mạnh hơn khoảng mười lần so với Standard TCP



(a).Mạng có RTT 10ms.



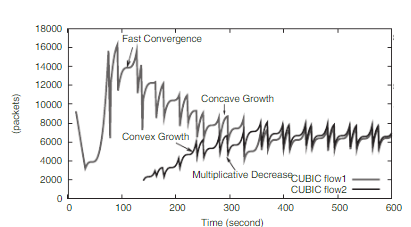
(b).Mạng có RTT 100 ms.

Hình 4: Chức năng đáp ứng của TCP, HSTCP và CUBIC tiêu chuẩn trong các mạng có RTT lần lượt là 10ms (a) và 100ms (b).

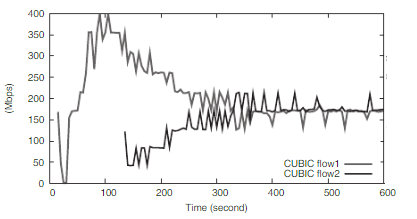
### Hoạt động của CUBIC

Hình 5 cho thấy đường cong cửa sổ của CUBIC theo thời gian chạy. Biểu đồ này thu được bằng cách chạy thử nghiệm thử nghiệm trên cấu hình mạng hình quả tạ với lưu lượng truy cập nền đáng kể theo cả hai hướng. Dung lượng nút cổ chai là 400Mbps và

RTT được đặt thành 240ms. Các bộ định tuyến đuôi thả được sử dụng. Có hai luồng CUBIC, và chúng có cùng RTT và nút cổ chai. Lưu ý rằng các đường cong có cao nguyên xung quanh Wmax là kích thước cửa sổ tại thời điểm xảy ra sự kiện mất gói cuối cùng. Chúng tôi quan sát thấy rằng hai luồng sử dụng tất cả các giai đoạn của các chức năng CUBIC trong thời gian chạy và hai luồng hội tụ thành một phần công bằng trong vòng 200 giây



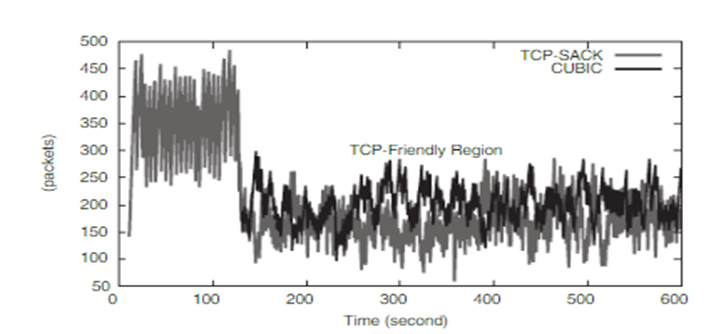
1. Đường cong cửa sổ CUBIC



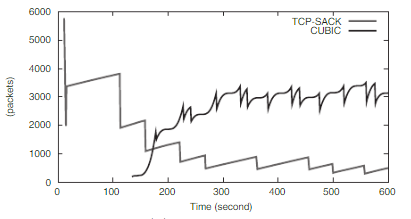
1. Thông lượng của hai luồng CUBIC

Hình 5: Hai luồng CUBIC với 246ms RTT

Hình 6 cho thấy sự thân thiện của CUBIC đối với TCP-SACK. Trong thử nghiệm này, chúng tôi chạy một luồng CUBIC với một luồng TCP-SACK trên đường dẫn mạng RTT ngắn (8ms) và đường dẫn mạng RTT dài (82ms), tương ứng. Trong mạng RTT ngắn (8ms) mà ngay cả TCP-SACK cũng có thể sử dụng toàn bộ băng thông của đường dẫn, CUBIC hoạt động ở chế độ thân thiện với TCP. Hình 6(a) xác nhận rằng một luồng CUBIC chạy ở chế độ thân thiện với TCP và chia sẻ băng thông công bằng với luồng TCP-SACK khác bằng cách duy trì cửa sổ tắc nghẽn của CUBIC tương tự như của TCP-SACK. Trong mạng RTT dài (82ms), nơi Standard TCP có vấn đề sử dụng kém, CUBIC sử dụng một hàm khối để có thể mở rộng cho môi trường này. Hình 4(b) xác nhận rằng luồng CUBIC chạy một chức năng tăng trưởng cửa sổ khối không giống như trường hợp với mạng RTT ngắn trong đó CUBIC không thể phân biệt được với TCP-SACK.



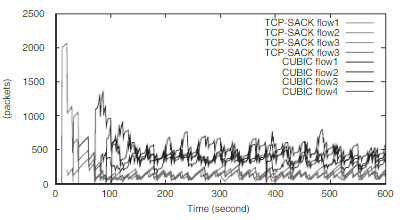
1. RTT 8ms



1. RTT 82ms.

Hình 6: Một luồng CUBIC và một luồng TCP-SACK. Băng thông được đặt là 400Mbps

Hình 7 cho thấy thử nghiệm với bốn luồng TCP-SACK và bốn luồng CUBIC. Đối với thử nghiệm này, chúng tôi đặt băng thông thành 400Mbps, RTT thành 40ms và kích thước bộ đệm là 100% BDP của một luồng. Chúng tôi quan sát thấy rằng bốn dòng chảy của CUBIC hội tụ thành thị phần công bằng một cách độc đáo trong một khoảng thời gian ngắn. Các đường cong cwnd của chúng rất mượt mà và không gây ra nhiều xáo trộn cho các luồng TCP cạnh tranh. Trong thử nghiệm này, tổng mức sử dụng mạng là khoảng 95%: bốn luồng CUBIC chiếm khoảng 72% tổng băng thông, bốn luồng TCP chiếm 23%.



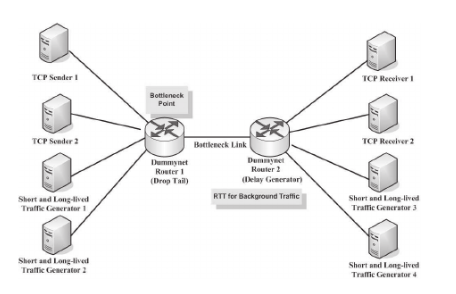
Hình 7: Bốn luồng TCP-SACK và bốn luồng CUBIC trên 40ms RTT

## Thực nghiệm và mô phỏng

### Thiết lập thử nghiệm

Chúng tôi xây dựng một cấu trúc liên kết hình quả tạ được hiển thị trong Hình 8 trong đó hai bộ định tuyến dummynet nằm ở nút cổ chai giữa hai điểm cuối. Mỗi điểm cuối bao gồm một tập hợp các máy chủ Dell Linux dành riêng cho các luồng biến thể TCP tốc độ cao và lưu lượng truy cập nền. Lưu lượng truy cập nền được tạo ra bằng cách sử dụng một sửa đổi của trình tạo lưu lượng web, được gọi là Surge [9] và Iperf [2]. Chúng tôi đã sửa đổi Surge để tạo ra nhiều kích thước luồng hơn nhằm tăng tính thay đổi trong lưu lượng chéo vì luồng có kích thước trung bình có xu hướng thực hiện đầy đủ khi bắt đầu chậm và tăng sự thay đổi trong băng thông khả dụng. RTT của mỗi lưu lượng truy cập nền được chọn ngẫu nhiên từ phân phối hàm mũ được tìm thấy trong [7]. Kích thước bộ đệm ổ cắm của các máy lưu lượng nền được cố định mặc định là 64KB trong khi các máy TCP tốc độ cao được cấu hình để có bộ đệm rất lớn để tốc độ truyền của các luồng tốc độ cao chỉ bị giới hạn bởi thuật toán điều khiển tắc nghẽn. Hai bộ định tuyến dummynet và bốn máy TCP tốc độ cao được điều chỉnh để tạo hoặc chuyển tiếp lưu lượng gần 1Gbps. Chi tiết điều chỉnh hệ thống cho cả hệ thống Linux và FreeBSD được trình bày trong [5]. Lưu ý rằng Netem [22] trong Linux cung cấp cùng một chức năng với phần mềm dummynet trong FreeBSD. Đối với thử nghiệm này, giá trị tối đa băng thông của bộ định tuyến nút cổ chai được đặt thành 400Mbps. Kích thước bộ đệm nút cổ chai được đặt thành 100% BDP nếu nó không được chỉ định rõ ràng.Lượng lưu lượng nền tương đương với khoảng 15% băng thông nút cổ chai được

đẩy vào các hướng tiến và lùi của quả tạ. Chúng tôi sử dụng bộ định tuyến đuôi thả ở nút cổ chai.



Hình 8: Thử nghiệm

### Mô phỏng trên NS2

Ảnh có chứa văn bản

Mô tả được tạo tự động

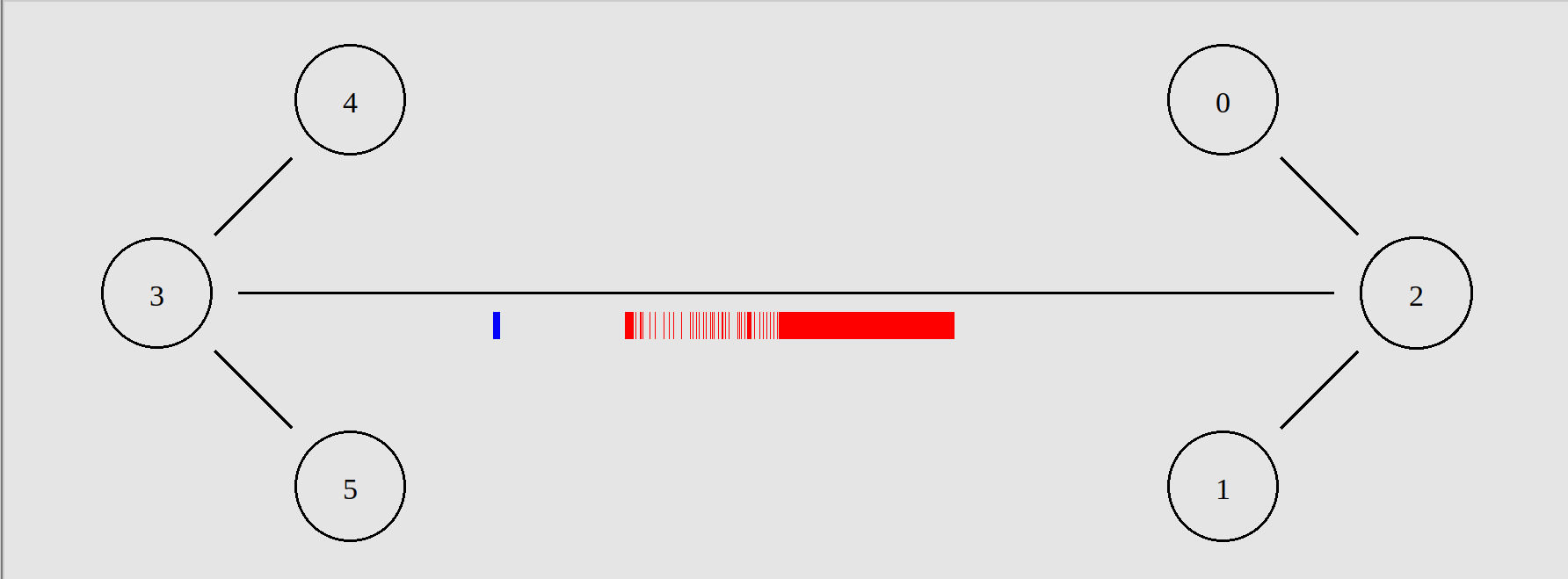
Ảnh có chứa văn bản

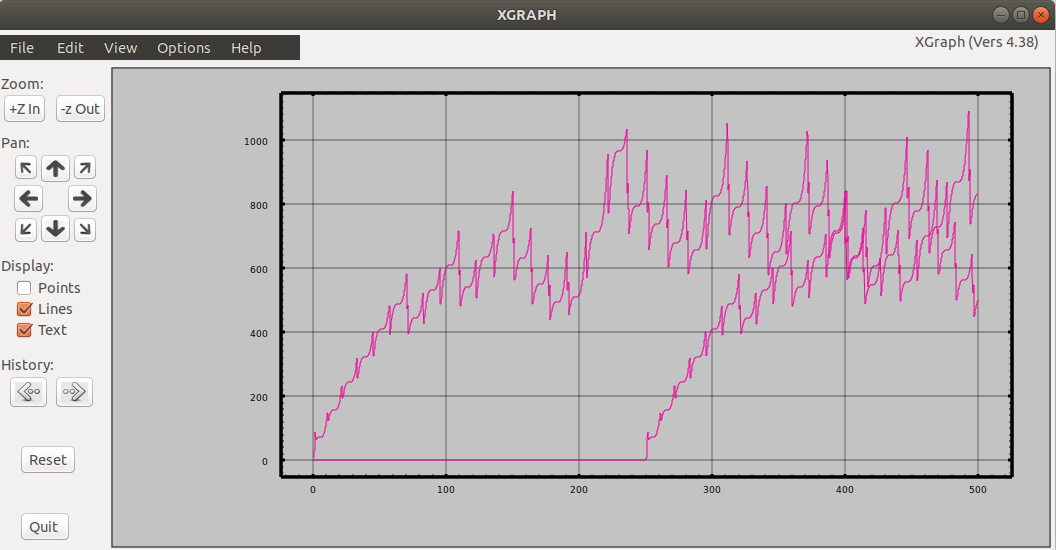
Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

* Kết quả thu được:





### Coding implement bằng C

Nội dung code có thể được tìm thấy tại đường dẫn: <https://elixir.bootlin.com/linux/latest/source/net/ipv4/tcp_cubic.c>

# KẾT LUẬN

Chúng tôi đề xuất một biến thể TCP mới, được gọi là CUBIC, cho các mạng đường dài và nhanh. CUBIC là phiên bản nâng cao của BIC-TCP. Nó đơn giản hóa việc kiểm soát cửa sổ BIC-TCP và cải thiện tính thân thiện với TCP và tính công bằng của RTT. CUBIC sử dụng hàm tăng khối về thời gian đã trôi qua kể từ sự kiện thua lỗ cuối cùng. Để cung cấp sự công bằng cho Standard TCP, CUBIC cũng hoạt động giống như Standard TCP khi chức năng tăng trưởng cửa sổ khối chậm hơn Standard TCP. Hơn nữa, bản chất thời gian thực của giao thức giữ cho tốc độ tăng trưởng cửa sổ không phụ thuộc vào RTT, điều này giúp giao thức TCP thân thiện với cả hai con đường RTT ngắn và dài. Chúng tôi hiển thị chi tiết về thuật toán CUBIC của Linux và cách triển khai. Thông qua quá trình thử nghiệm rộng rãi, chúng tôi xác nhận rằng CUBIC đã khắc phục được những thiếu sót của BIC-TCP và đạt được sự công bằng giữa giao thức nội bộ khá tốt, tính công bằng và tính thân thiện với TCP.

TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | CUBIC: A New TCP-Friendly High-Speed TCP Variant ∗ by Sangtae Ha, Injong Rhee and Lisong Xu |
| [2] | https://en.wikipedia.org/wiki/CUBIC\_TCP |
| [3] | https://en.wikipedia.org/wiki/Transmission\_Control\_Protocol |