Phân tích và mô phỏng giao thức TCP Hybla ở lớp truyền vận. Đánh giá so sánh hiệu suất với TCP Cubic trong truyền thông vệ tinh

Phạm Chí Vỹ
Đại học Bách Khoa Hà Nội
Trường Điện - Điện tử
Hà Nội, Việt Nam
Vy.PC214155@sis.hust.edu.vn

Lê Văn Minh
Đại học Bách Khoa Hà Nội
Trường Điện - Điện tử
Hà Nội, Việt Nam
Minh.LV214001@sis.hust.edu.vn

Đào Duy Anh
Đại học Bách Khoa Hà Nội
Trường Điện - Điện tử
Hà Nội, Việt Nam
Anh.DD210035@sis.hust.edu.vn

***Tóm tắt nội dung*—Truyền thông vệ tinh đã trở nên quan trọng trong truyền dữ liệu toàn cầu hiện nay nhờ phạm vi phủ sóng rộng lớn. Các liên kết vệ tinh bị ảnh hưởng bởi hai yếu tố chính: trễ lan truyền và tỷ lệ lỗi cao. Các biến thể kiểm soát tắc nghẽn TCP tiêu chuẩn không được thiết kế để giải quyết những vấn đề này. Chúng thường phụ thuộc vào thời gian trễ trọn vòng (RTT), và do đó bị ảnh hưởng nặng nề nếu RTT lớn. Nhiều biến thể TCP khác nhau, như Hybla, đã được đề xuất để tối ưu hóa thông lượng kênh trong các mạng không đồng nhất kết hợp các liên kết vệ tinh. Tỷ lệ lỗi của các kênh vệ tinh có thể được giảm đáng kể bằng cách sử dụng kỹ thuật sửa lỗi chuyển tiếp (FEC). Trong tình huống như vậy, theo kết quả thu được từ nghiên cứu so sánh của chúng tôi, hiệu suất của Cubic, được thiết kế cho các mạng tốc độ cao, tốt hơn Hybla trong điều kiện độ trễ cao.**

***Từ khóa*—RTT, TCP, Hybla, Cubic, sat-com**

I. Giới thiệu

Bằng thực nghiệm đã xác định rằng các biến thể TCP gặp phải những hạn chế về hiệu suất do độ trễ lan truyền và khả năng xảy ra lỗi bit. Những hạn chế này ngày càng trở nên quan trọng đối với các hệ thống giao tiếp vệ tinh (sat-com), do chúng được triển khai trên quy mô lớn để hỗ trợ các trung tâm làm việc phân bố trên toàn cầu. Thông lượng của một kênh TCP phụ thuộc vào kích thước cửa sổ tắc nghẽn (cwnd) và vào quy trình được sử dụng để quản lý kích thước cửa sổ này. Slow Start và Congestion control là hai bước chính của quy trình này. Các phiên bản chuẩn của giao thức TCP (Tahoe, Reno, New Reno) bao gồm hai giai đoạn, cụ thể là Slow Start (SS) và Congestion Avoidance (CA), trong đó CA là một phần của Congestion control theo sau là truyền lại nhanh (re-transmit) với khôi phục nhanh (fast recovery). Trong giai đoạn Slow Start, người gửi mở cửa sổ tắc nghẽn (cwnd) theo cấp số nhân, nhân đôi cwnd sau mỗi Thời gian khứ hồi (RTT) cho đến khi đạt đến ngưỡng khởi động chậm (ssthresh). Sau đó, kết nối chuyển sang CA, trong đó cwnd tăng trưởng chỉ 1 gói tin sau mỗi RTT (hoặc tuyến tính). Ssthresh ban đầu được đặt thành một giá trị mặc định tùy ý, trong phạm vi từ 4 K đến 64 K byte, tùy thuộc vào việc triển khai hệ điều hành. Bằng cách đặt ssthresh ban đầu thành một giá trị tùy ý, hiệu suất TCP có thể gặp phải hai vấn đề tiềm ẩn: (a) nếu ssthresh được đặt quá cao so với tích độ trễ băng thông mạng (BDP), sự gia tăng theo cấp số nhân của cwnd tạo ra quá nhiều gói quá nhanh, gây ra nhiều lần mất mát với sự giảm đáng kể thông lượng kết nối; (b) nếu ssthresh ban đầu được đặt quá thấp so với BDP, kết nối sẽ chuyển sang tăng trưởng cwnd tuyến tính trước thời hạn, dẫn đến việc sử dụng kênh thấp. Trong trường hợp mất gói dữ liệu đáng kể hoặc độ trễ lớn, cơ chế phục hồi của TCP hoạt động khá kém, vì tính toán sai lý do tắc nghẽn, gây ra việc kích hoạt hoàn toàn không phù hợp các cơ chế tránh tắc nghẽn. Điều này là do thuật toán truyền dựa trên cửa sổ TCP, được kích hoạt bởi các lần xác nhận (ACK), phụ thuộc vào độ trễ của mạng. RTT dài làm giảm tốc độ tăng trưởng của cửa sổ tắc nghẽn và dẫn đến suy giảm đáng kể thông lượng. Trong trường hợp này, suy giảm hiệu suất là kết quả trực tiếp của việc quản lý cửa sổ của giao thức TCP và sự phụ thuộc của nó vào RTT.

Một số cơ chế đã được đề xuất để bù đắp cho việc gán lỗi tắc nghẽn và cung cấp quản lý cửa sổ tắc nghẽn tốt hơn cũng như độc lập với RTT. Hiệu suất của Hybla được báo cáo là tốt hơn Westwood trong trường hợp kênh có độ trễ cao và lỗi cao.

Những cải tiến trong kiểm soát tắc nghẽn TCP bằng Hybla cho các mạng có độ trễ cao với tỷ lệ lỗi cao, như liên kết sat-com , đã được thiết lập. Tương tự như vậy, những lợi thế của Cubic trong các mạng tốc độ cao đã được chứng minh và kiểm chứng. Với việc sử dụng các kỹ thuật FEC mạnh mẽ được triển khai trong hầu hết các trung tâm làm việc phân tán tĩnh, tỷ lệ lỗi phát sinh do hỏng bit ở lớp truyền vận đang giảm dần.

Bài báo này trình bày một phân tích so sánh giữa Hybla và Cubic cho các mạng có độ trễ cao. Công cụ mô phỏng mạng NS-2 sử dụng bản vá TCP-Linux được sử dụng để mô phỏng Hybla và Cubic trên các mạng có dây và mạng sat-com không đồng nhất (liên kết vệ tinh + có dây). Bài báo được tổ chức như sau. Trong Phần II, trình bày các cơ chế để quản lý cửa sổ tắc nghẽn của Cubic và Hybla. Trong Phần III, cấu trúc mô hình và thiết lập mô phỏng được đưa ra. Trong Phần IV, kết quả mô phỏng và phân tích. Trong Phần V, kết luận của bài báo.

II. Biến thể TCP

A. Các thuật toán tiêu chuẩn (Tahoe, Reno, và newReno)

Khi một kết nối mới được thiết lập, phía gửi sẽ thăm dò băng thông khả dụng bằng cách tăng dần kích thước cửa sổ tắc nghẽn W. Trong giai đoạn slow start (SS), W được khởi tạo bằng một giá trị W_0 , thường bằng hoặc gấp đôi maximum segment size (MSS). Kích thước cửa sổ tắc nghẽn được tăng thêm MSS bytes sau mỗi lần nhận ACK không trùng lặp với lần trước đó. Khi kích thước sổ tắc nghẽn chạm ngưỡng slow start threshold (ssthresh), phía phát sẽ chuyển sang trạng thái congestion avoidance (CA). Ở trạng thái này, kích thước cửa sổ tắc nghẽn chỉ tăng với tốc độ MSS/W bytes sau mỗi lần nhận ACK. Trong một kênh truyền thực tế, sự tăng kích thước cửa sổ sẽ tiếp tục diễn ra cho tới khi bộ đệm phía thu đạt độ lớn tối đa hoặc một đoạn dữ liệu bị mất. Trong trường hợp này, tùy vào thuật toán TCP được sử dụng mà quá trình khôi phục lỗi đường truyền sẽ diễn ra khác nhau.

1) Thuật toán slow start và congestion avoidance

Độ lớn cửa sổ tắc nghẽn W với đơn vị là MSS có quy tắc cập nhật được biểu diễn như sau:

$$W_{i+1} = \begin{cases} W_i + 1, & \text{SS} \\ W_i + \frac{1}{W_i}, & \text{CA} \end{cases} \tag{1}$$

trong đó chỉ số i biểu thị thứ tự của gói ACK nhận được.

Để tiến hành phân tích các tác động của RTT đối với thuật toán chống tắc nghẽn của TCP, ta bắt đầu xem xét với một kênh truyền lý tưởng (không có lỗi đường truyền) và mô tả sự thay đổi của kích thước cửa sổ tắc nghẽn (congestion window – cwnd) theo thời gian.

Có thể nhận thấy, ở trạng thái slow start, quy tắc cập nhật kích thước cửa sổ cwnd theo (1) dẫn đến kết quả cwnd tăng theo cấp số nhân (gấp đôi) sau mỗi khoảng thời gian RTT. Trong khi đó, ở trạng thái congestion avoidance, cwnd tăng theo một hàm tuyến tính theo thời gian. Khi đó, ta có công thức tính kích thước cửa sổ theo thời gian

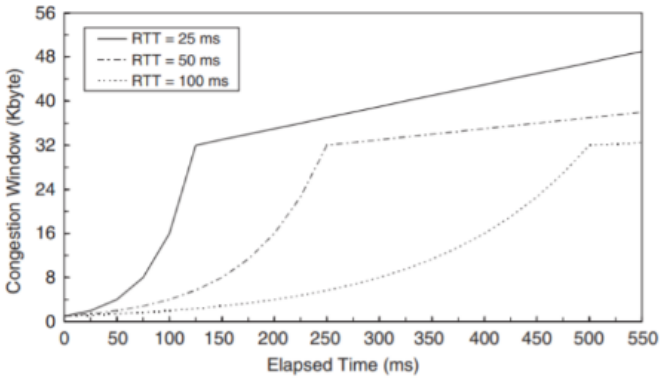
$$W(t) = \begin{cases} 2^{\frac{t}{RTT}}, & 0 \le t \le t_\gamma, \quad \text{SS} \\ \dfrac{t - t_\gamma}{RTT} + \gamma, & t \ge t_\gamma, \quad \text{CA} \end{cases} \tag{2}$$

trong đó γ là giá trị ngưỡng ssthresh mà tại đó thuật toán chuyển sang trạng thái congestion avoidance và $t_\gamma = \text{RTT} \cdot \log_2(\gamma)$ là thời gian để cửa sổ tắc nghẽn có kích thước đạt đến ngưỡng đó. Dựa vào (2), có thể thấy RTT càng nhỏ thì tốc độ tăng kích thước cửa sổ càng nhanh.

Ta có thể tính được giá trị của cwnd theo đơn vị byte - $W_B(t)$ bằng cách nhân $W(t)$ với MSS. Hình 1 biểu diễn giá trị $W_B(t)$ theo thời gian, ứng với ba giá trị RTT khác nhau. Trong đó, MSS được lấy giá trị 1024 bytes và γ bằng 32. Khi đó, với mỗi giá trị RTT khác nhau thì $W_B(t)$ cũng tăng nhanh hoặc chậm khác nhau. Cụ thể, RTT càng nhỏ thì tốc độ tăng kích thước cửa sổ cwnd càng nhanh.

Tốc độ truyền gói tin của giao thức TCP chuẩn theo đơn vị gói/s được tính bằng công thức (3).

$$B(t) = \frac{W(t)}{\text{RTT}} \tag{3}$$



Hình 1. TCP Standard: Sự tăng trưởng kích thước cửa sổ tắc nghẽn theo thời gian với một số giá trị RTT khác nhau

Để so sánh hiệu suất của các thuật toán TCP, việc tính lượng gói tin được truyền bởi một nguồn TCP chuẩn từ khi bắt đầu quá trình truyền - $T_d(t)$ là khá cần thiết. Đại lượng này có thể tính bằng công thức (4).

$$T_d(t) = \begin{cases} \frac{2^{t/\text{RTT}_0} - 1}{\ln(2)}, & 0 \le t \le t_\gamma, o\,(\text{SS}) \\ \frac{\gamma - 1}{\ln(2)} + \frac{(t - t_\gamma, o)^2}{2\,\text{RTT}_0^2} + \frac{\gamma(1 - t_\gamma, o)}{\text{RTT}_0}, & t \ge t_\gamma, o\,(\text{CA}) \end{cases} \tag{4}$$

2) Cơ chế khôi phục lỗi

Giả sử gói thứ n bị mất trên đường truyền. Khi đó, phía gửi sẽ nhận một ACK số thứ tự n (xác nhận đã nhận được gói thứ $n-1$ mà không gặp lỗi, đang chờ nhận gói thứ n), tức là giống với lần nhận trước đó. Việc này lặp lại đến lần thứ ba thì phía gửi sẽ kích hoạt quá trình khôi phục lỗi. Đầu tiên, gói thứ n sẽ được truyền lại. Sau đó, ssthresh được cập nhật bằng $1/2$ kích thước cửa sổ cwnd khi mới xuất hiện tắc nghẽn. Lúc này, cwnd cũng được cập nhật lại bằng ssthresh cộng 3. Nếu khôi phục được dữ liệu bị mất, thuật toán TCP sẽ được thực hiện một cách bình thường.

B. TCP Hybla

Trong các mạng không đồng nhất, các kết nối TCP được đặc trưng bởi RTT lớn (tức là bao gồm đường dẫn qua vệ tinh) đạt hiệu suất kém so với các kết nối có dây có giá trị RTT thấp hơn. Ý tưởng cơ bản của TCP Hybla là đạt được cùng tốc độ truyền các gói tin của kết nối TCP tham chiếu (ví dụ: có dây) cho các kết nối RTT dài (ví dụ: sat-com và không dây). Các quy tắc cập nhật cửa sổ tắc nghẽn cho triển khai TCP tiêu chuẩn (ví dụ: Tahoe, Reno, New Reno) có thể được biểu thị dưới dạng phương trình (2) khi chúng ta có chỉ số i, biểu thị việc tiếp nhận ACK thứ i.

Bây giờ, mục tiêu chính đằng sau TCP Hybla là tạo ra tốc độ truyền dữ liệu độc lập với RTT. Điều này đạt được bằng cách sử dụng RTT chuẩn hóa (ρ), được định nghĩa như sau:

$$\rho = \frac{\text{RTT}}{\text{RTT}_0} \tag{5}$$

trong đó, RTT_0 là RTT của kết nối tham chiếu mà ta dùng để cân bằng hiệu suất.

Để làm cho tốc độ tăng trưởng cửa sổ bằng với liên kết tham chiếu, phần phụ thuộc vào RTT trong phương trình (2) được nhân với ρ. Điều này dẫn đến một phương trình mà tốc độ tăng trưởng của cửa sổ tắc nghẽn được xác định bởi RTT_0, giúp kết nối có khả năng tăng cửa sổ W^H nhanh như liên kết tham chiếu. Tiếp theo, phương trình thu được được nhân với ρ để bù đắp cho thực tế rằng cửa sổ chỉ có thể cập nhật mỗi RTT chứ không phải RTT_0. Áp dụng các quy tắc này, ta có:

$$W^H(t) = \begin{cases} \rho 2^{\rho \frac{t}{RTT}}, & 0 \le t < t_\gamma, \quad \text{SS} \\ \rho \left[\rho \dfrac{t - t_\gamma}{RTT} + \gamma \right], & t \ge t_\gamma, \quad \text{CA} \end{cases} \tag{6}$$

Ở đây, chỉ số trên H biểu thị tốc độ tăng trưởng cửa sổ cho Hybla. Bây giờ từ phương trình (3) và phương trình (6), chúng ta có:

$$B^H(t) = \begin{cases} \dfrac{2^{\frac{t}{RTT_0}}}{RTT_0}, & 0 \le t < t_\gamma, \ \text{SS} \\ \dfrac{1}{RTT_0} \left[\dfrac{t - t_\gamma}{RTT_0} + \gamma \right], & t \ge t_\gamma, \ \text{CA} \end{cases} \tag{7}$$

Như có thể quan sát, $B^H(t)$ được đạt được mà không phụ thuộc vào RTT. Khi viết lại công thức này thành các quy tắc cập nhật cửa sổ tắc nghẽn, các quy tắc cập nhật cửa sổ tắc nghẽn "thông thường" của phương trình (1) được thay thế bằng:

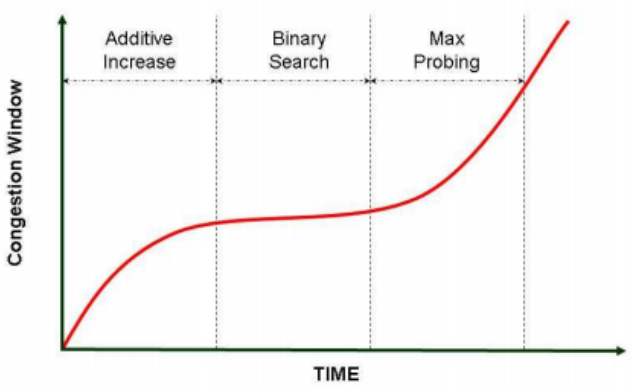
$$W_{i+1}^H = \begin{cases} W_i^H + 2^\rho - 1, & \text{SS} \\ W_i^H + \frac{\rho^2}{W_i^H}, & \text{CA} \end{cases} \tag{8}$$

Vì vậy, mục tiêu cuối cùng của Hybla là thực hiện việc truyền dữ liệu độc lập với RTT để có được thông lượng tối đa có thể đạt được bằng cách thực hiện thuật toán trên.

C. TCP Cubic

Cubic là một triển khai TCP với thuật toán kiểm soát tắc nghẽn được tối ưu hóa cho các mạng tốc độ cao. Đây là phiên bản cải tiến của BIC-TCP với khả năng kiểm soát cửa sổ tốt hơn và hiệu quả hơn trong các môi trường tốc độ cao. 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 lõm và lồi của BIC-TCP bằng một hàm bậc ba (chứa cả phần lõm và phần lồi). Trên thực tế, bất kỳ hàm đa thức bậc lẻ nào cũng có hình dạng này—việc lựa chọn hàm bậc ba là ngẫu nhiên và để phù hợp. Một tính năng chính của Cubic là sự tăng trưởng cửa sổ của nó chỉ phụ thuộc vào thời gian giữa hai sự kiện tắc nghẽn liên tiếp.

Hình 2 cho thấy hàm tăng trưởng cửa sổ của BIC TCP đạt được khả năng mở rộng và ổn định tốt trong mạng tốc độ cao, trong khi trong mạng tốc độ thấp, hàm tăng trưởng cửa sổ vẫn còn quá mạnh. Một số giai đoạn (tăng cộng, tìm kiếm nhị phân và thăm dò tối đa) của kiểm soát cửa sổ làm tăng thêm độ phức tạp trong việc triển khai giao thức và phân tích hiệu suất của nó. Cubic, trong khi vẫn giữ được sức mạnh của BIC (đặc biệt là khả năng mở rộng và ổn định), đơn giản hóa kiểm soát cửa sổ và tối đa hóa thông lượng của nó.

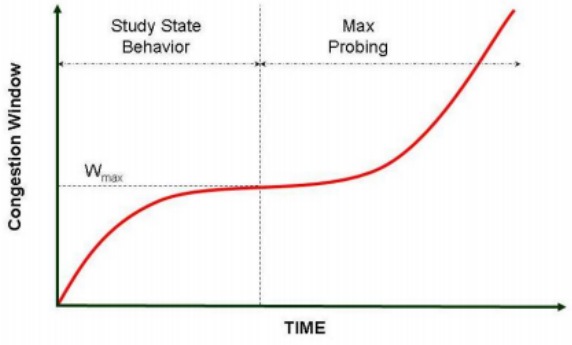


Hình 2. Hàm tăng trưởng cửa sổ của BIC

Cụ thể hơn, cửa sổ tắc nghẽn của Cubic được xác định bởi hàm sau:

$$W_{\text{cubic}} = C(t - K)^3 + W_{\max} \tag{1}$$

với C là một hệ số tỉ lệ, t là thời gian trôi qua kể từ lần giảm kích thước cửa sổ cuối cùng, $W_{\max}$ là kích thước cửa sổ ngay trước lần giảm kích thước cửa sổ cuối cùng, và $K = \sqrt[3]{\frac{W_{\max}\beta}{C}}$, trong đó β là một hệ số giảm bội áp dụng cho việc giảm kích thước cửa sổ tại thời điểm xảy ra sự kiện mất gói (tức là kích thước cửa sổ giảm xuống $\beta W_{\max}$ tại thời điểm giảm cuối cùng).



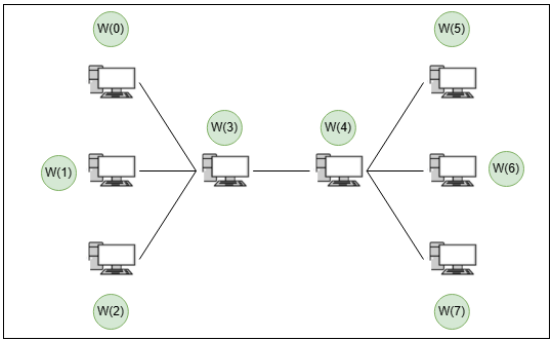
Hình 3. Hàm tăng trưởng cửa sổ của Cubic

Hình 3 cho thấy hàm tăng trưởng của Cubic với gốc tọa độ tại W_{max}. Ban đầu, cửa sổ tăng rất nhanh, nhưng khi gần đến W_{max}, tốc độ tăng trưởng chậm lại. Xung quanh W_{max}, mức tăng của cửa sổ gần như bằng không. Sau mức đó, Cubic bắt đầu thăm dò băng thông lớn hơn, trong đó cửa sổ tăng chậm ban đầu và tăng tốc độ tăng trưởng khi vượt ra khỏi W_{max}. Tốc độ tăng trưởng chậm này xung quanh W_{max} giúp tăng cường tính ổn định của giao thức và cải thiện khả năng sử dụng mạng, trong khi tốc độ tăng trưởng nhanh ra khỏi W_{max} đảm bảo khả năng mở rộng của giao thức.

III. Thiết lập mô phỏng và cấu trúc liên kết của TCP Hybla và TCP Cubic

A. Mô hình 1

Mô hình 1 xây dựng gồm 8 nút mạng và 7 liên kết có dây được mô tả ở Hình 3. Trong đó, các liên kết được tạo với băng thông 100Mbps và độ trễ 5ms (riêng kết nối 3-4 có độ trễ 10ms). Hàng đợi gắn ở các liên kết tuân theo cơ chế DropTail. Liên kết giữa nút 3-4 có độ dài hàng đợi là 100. 3 kết nối TCP Hybla được truyền từ nút 0-5, 1-6, 2-7 với RTT của từng kết nối là 25ms. Thời gian mô phỏng là 1s.

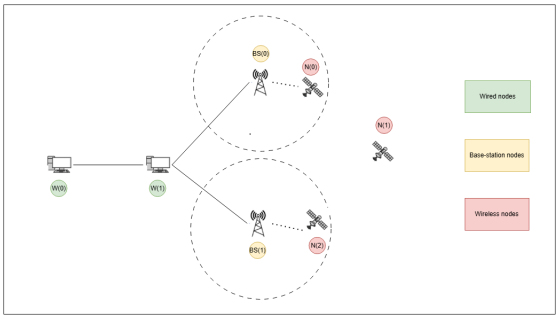


Hình 4. Mô hình mô phỏng kết nối có dây

B. Mô hình 2

Mô hình 2 xây dựng gồm 2 nút mạng có dây và 5 nút mạng không dây mô tả ở hình 5. Trong đó, liên kết có dây được tạo với băng thông 100Mbps và độ trễ 2.5ms. Trong đó, liên kết không dây được tạo với băng thông 34368Mbps. Hàng đợi gắn ở các liên kết tuân theo cơ chế DropTail. Kết nối TCP được truyền từ nút W1 đến N(2), N(0) đến W(0). RTT của kết nối không dây là 600ms, RTT của kết nối có dây là 25ms. Thời gian mô phỏng là 500s.

Để đánh giá hiệu năng của kết nối TCP Hybla và TCP Cubic trong truyền thông vệ tinh, ta thiết lập 2 kết nối TCP Hybla và 2 kết nối TCP Cubic, sau đó phân tích kích thước cửa sổ tắc nghẽn cũng như throughput và goodput của các kết nối để so sánh.



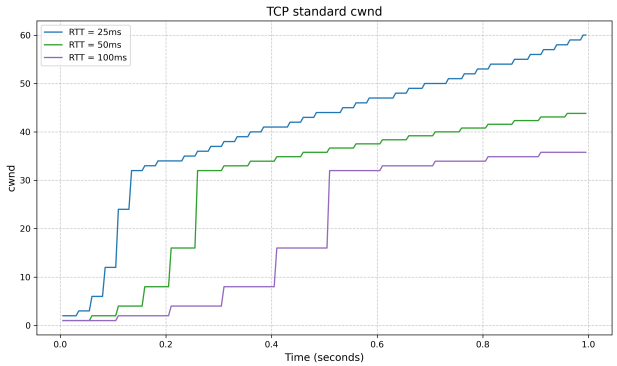
Hình 5. Mô hình mô phỏng kết nối không đồng nhất

IV. Biểu đồ đánh giá

A. Mô hình 1

Dưới đây là kết quả cửa sổ tắc nghẽn (cwnd) với các RTT khác nhau của 3 giao thức TCP Standard, TCP Hybla, TCP Cubic.

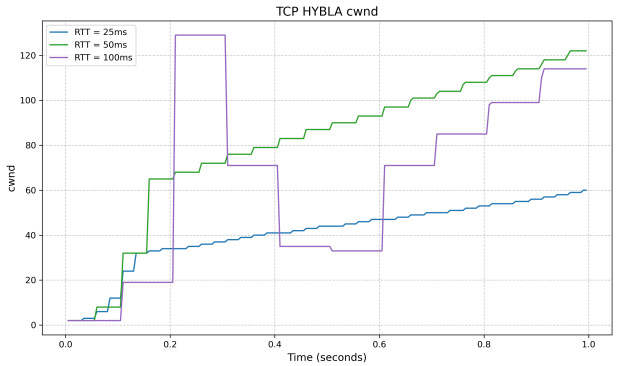
1) TCP Standard



Hình 6. Đồ thị TCP Standard với RTT khác nhau

Với tần suất lấy mẫu là 5ms, từ Hình 6 thấy rằng khi RTT tăng lên, cửa sổ tắc nghẽn (cwnd) mất nhiều thời gian hơn để đạt đến ngưỡng ssthresh = 32. Điều này cho thấy nhược điểm của TCP Standard, khi nó không tối ưu trong việc xử lý các kết nối có độ trễ cao, dẫn đến hiệu suất giảm sút trong các mạng có RTT lớn.

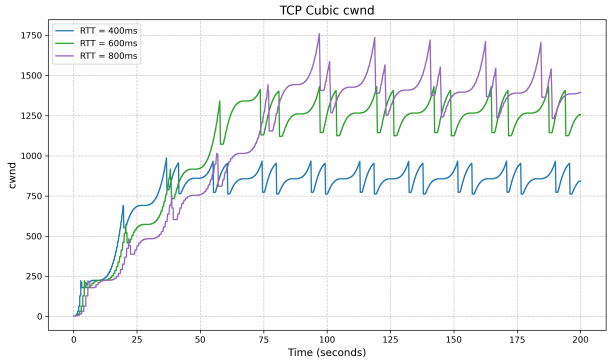
2) TCP Hybla



Hình 7. Đồ thị TCP Hybla với RTT khác nhau

Từ hình 7 TCP Hybla đã giải quyết vấn đề RTT lớn bằng cách tăng mạnh kích thước cửa sổ tắc nghẽn, giúp cải thiện hiệu suất truyền tải dữ liệu trong các mạng có độ trễ cao.

3) TCP Cubic

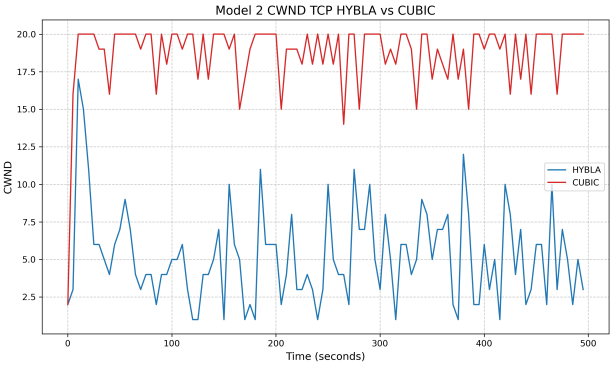


Hình 8. Đồ thị TCP Cubic với RTT khác nhau

Tương tự TCP Hybla, TCP Cubic cũng giải quyết vấn đề RTT lớn bằng cách tăng tốc độ mở rộng cửa sổ khi RTT tăng. Đặc biệt, hàm bậc ba của Cubic giúp tăng nhanh để tận dụng băng thông, chậm lại để dò tìm tránh nghẽn mạng, và tiếp tục tăng trưởng nhanh chóng sau đó.

B. Mô hình 2

1) Cửa sổ tắc nghẽn

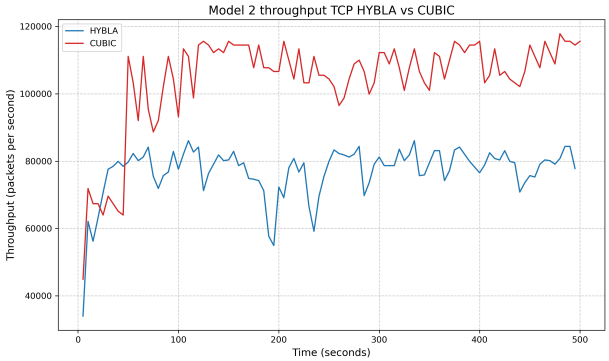


Hình 9. Đồ thị cwnd của TCP Hybla và TCP Cubic

Hình 9 đồ thị miêu tả kích thước cửa sổ tắc nghẽn của thuật toán TCP Hybla và TCP Cubic. Có thể thấy, với TCP Hybla, tại ngay thời điểm thiết lập kênh truyền, kích thước cửa sổ tăng nhanh theo hàm mũ, sau đó đã xử lý nhanh chóng khi phát hiện tắc nghẽn bằng cách giảm kích thước cửa sổ xuống gấp đôi. Trong khi đó TCP Cubic, khi bắt đầu với kích thước cửa sổ tối thiểu, sau đó từ từ gửi gói tin đến khi đạt ngưỡng ssthresh.

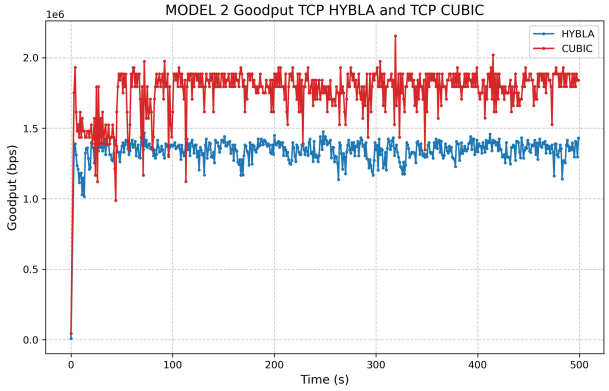
2) Thông lượng

Kết quả mô phỏng throughput của kết nối không dây được mô tả như Hình 10 được tính theo đơn vị Bps. Dựa vào kết quả mô phỏng, có thể thấy ở giai đoạn đầu thông lượng của kết nối không dây sử dụng thuật toán TCP Hybla lớn hơn so với Cubic nhưng đến giai đoạn sau thuật toán Cubic cải thiện hơn nhiều so với Hybla. Thông lượng được chiếm dụng khá nhanh bởi thuật toán và tăng theo hàm số mũ. Thông lượng có thời điểm giảm xuống là lúc thuật toán đang làm giảm việc tăng nghẽn của kênh truyền.



Hình 10. Đồ thị throughput của TCP Hybla và TCP Cubic

3) Goodput



Hình 11. Đồ thị goodput của TCP Hybla và TCP Cubic

Ngoài ra, ta cũng có thể so sánh goodput của kết nối không dây. Kết quả mô phỏng goodput của kết nối không dây khi sử dụng hai thuật toán TCP Hybla và TCP Cubic được mô tả như Hình 11 Tương tự như througput, ở kết nối vệ tinh với RTT cao (600ms), thuật toán TCP Hybla cho hiệu suất cao hơn ở giai đoạn đầu nhưng ở giai đoạn sau thuật toán TCP Cubic cho thấy hiệu quả tốt hơn.

V. Kết luận

Qua bài báo này, chúng ta đã đánh giá được sự thay đổi kích thước cửa sổ tắc nghẽn của các giao thức TCP Standard, TCP Hybla, và TCP Cubic khi áp dụng trên các giá trị RTT khác nhau. Kết quả cho thấy, TCP Hybla duy trì được kích thước cửa sổ ổn định và phù hợp với các kết nối có độ trễ cao, trong khi TCP Standard bị ảnh hưởng đáng kể bởi RTT lớn, làm giảm hiệu suất truyền dữ liệu. Mặt khác, TCP Cubic thể hiện ưu thế vượt trội với khả năng mở rộng và thích nghi tốt hơn ở các mạng không đồng nhất, đặc biệt trong việc duy trì cửa sổ tắc nghẽn và cải thiện throughput cũng như goodput so với TCP Hybla. Điều này khẳng định TCP Cubic là một lựa chọn tối ưu cho các mạng phức tạp và đa dạng về RTT.

LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn PGS.TS Nguyễn Thành Chuyên đã tận tình hướng dẫn, đóng góp ý kiến quý báu và hỗ trợ chúng em trong suốt quá trình giảng dạy và thực hiện bài báo này. Sự giúp đỡ và góp ý của thầy là nguồn động lực để chúng em hoàn thành công việc nghiên cứu và đưa ra kết quả tốt nhất.

Chúng tôi xin chân thành cảm ơn các tác giả S. Ahmad, G. Giambene, F. Vacirca, C. Caini, R. Firrincieli, D. Lacamera, L. Xu, K. Harfoush, I. Rhee, A. Islam, T. Ahmed, S. S. Alam, M. Kim, và J. Kim vì những đóng góp quý báu của họ trong lĩnh vực nghiên cứu giao thức TCP. Những kết quả và phân tích từ các bài báo của quý vị đã cung cấp nền tảng quan trọng giúp chúng tôi thực hiện và hoàn thiện nghiên cứu này.

TÀI LIỆU

[1] S. Ahmad, G. Giambene, and F. Vacirca, "Comparative performance evaluation of TCP Hybla and TCP Cubic for satellite communication under low error conditions,"in IEEE Transactions on Aerospace and Electronic Systems, vol. 50, no. 3, pp. 2077-2091, July 2014. doi: 10.1109/TAES.2014.120047.

[2] C. Caini, R. Firrincieli, and D. Lacamera, "TCP Hybla: a TCP enhancement for heterogeneous networks,"in International Journal of Satellite Communications and Networking, vol. 22, no. 5, pp. 547-566, Sept. 2004. doi: 10.1002/sat.799.

[3] L. Xu, K. Harfoush, and I. Rhee, "CUBIC: A new TCP-friendly high-speed TCP variant,"in ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64-74, July 2008. doi: 10.1145/1400097.1400107.

[4] E. Caini and R. Firrincieli, "Comparison of TCP congestion control performance over a satellite network,"in 2005 IEEE Military Communications Conference, Atlantic City, NJ, USA, 2005, pp. 1-7. doi: 10.1109/MILCOM.2005.1605746.

[5] A. Islam, T. Ahmed, and S. S. Alam, "Competing TCP congestion control algorithms over a satellite network,"in 2010 International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 2010, pp. 233-238. doi: 10.1109/ICCITECHN.2010.5723875.

[6] M. Kim, J. Kim, and I. Rhee, "NS-2 TCP-Linux: An NS-2 TCP implementation with congestion control algorithms from Linux,"in ACM SIGCOMM Computer Communication Review, vol. 36, no. 3, pp. 83-96, July 2006. doi: 10.1145/1140086.1140100.