ĐẠI HỌC CÔNG NGHỆ - ĐẠI HỌC QUỐC GIA HÀ NỘI

Khoa Điện tử Viễn thông

BÁO CÁO CUỐI MÔN HỌC

MÔN HỌC: MẠNG TRUYỀN THÔNG MÁY TÍNH 2

CHỦ ĐỀ: ỨNG DỤNG VÀ MÔ PHỎNG THUẬT TOÁN RED TRONG TRAFFIC CONTROL

Sinh viên thực hiện: 1. Nguyễn Quang Vinh - 20021601

2. Bùi Ngọc Sơn - 20021576

3. Đỗ Mạnh Toàn - 20020250

Giảng viên hướng dẫn: TS. Lâm Sinh Công

Mục lục

TÓM TẮT

Traffic Control là tên được đặt cho tập hợp các hệ thống xếp hàng và cơ chế theo đó các gói được nhận và truyền trên bộ định tuyến. Điều này bao gồm việc quyết định gói nào (và liệu có) chấp nhận ở tốc độ nào trên đầu vào của giao diện và xác định gói nào sẽ truyền theo thứ tự nào với tốc độ nào trên đầu ra của một giao diện. Khi được sử dụng đúng cách, Traffic Control sẽ dẫn đến việc sử dụng tài nguyên mạng dễ dự đoán hơn và ít hơn tranh chấp không ổn định cho các tài nguyên này. Tuy nhiên nó cũng khá phức tạp trong việc sử dụng.

GIỚI THIỆU

Kiểm soát lưu lượng gồm các hoạt động để tăng mức sử dụng tài nguyên mạng và tăng hiệu quả phân phối luồng lưu lượng. Về các tham số hiệu suất mạng, các hoạt động điều khiển lưu lượng nhằm cung cấp thông lượng mạng cao và độ trễ dòng chảy thấp.

Điều kiện của *thông lượng mạng cao* cho phép sử dụng tất cả các tài nguyên truyền dẫn trong khi điều kiện *độ trễ lưu lượng thấp* cho phép đạt được việc vận chuyển gói với độ trễ mạng yêu cầu tối thiểu. Một gói tin lý tưởng sẽ gửi đến người nhận ở tốc độ dọc theo đường dẫn mạng và với độ trễ gói bao gồm độ trễ truyền cộng với độ trễ xử lý của các phần tử trung gian. Một số chiến lược được gọi là thuật toán Quản lý hàng đợi chủ động (AQM), chẳng hạn như RED, CoDel và PIE đã được thiết kế để tương phản với sự gia tăng không kiểm soát của thời gian xếp hàng.

Các giao thức vận chuyển mạng, ví dụ như TCP, đạt được giao tiếp ở mức vận chuyển giữa các nút cuối và khai thác tất cả băng thông có sẵn. Cơ chế tránh tắc nghẽn TCP (Congestion Avoidance) cố gắng tránh tắc nghẽn mạng trong khi chia sẻ băng thông một cách công bằng giữa các luồng khác nhau. Về cơ bản, cơ chế Congestion Avoidance cố gắng khám phá tình trạng tắc nghẽn mạng do mạng bị mất gói. Do thiếu thông báo về gói bị mất, người gửi sẽ giảm tốc độ gửi để giảm bớt tình trạng tắc nghẽn mạng. Thông báo tắc nghẽn xảy ra do bộ đệm FIFO đầy làm rớt gói. Ý tưởng là dùng RED làm giảm tắc nghẽn trong hàng đợi bằng cách loại bỏ các gói để một số kết nối TCP tạm thời gửi ít gói hơn vào mạng. RED còn sử dụng độ dài hàng đợi làm thước đo trạng thái tắc nghẽn.

I. LỚP ĐIỀU KHIỂN LƯU LƯỢNG (TRAFFIC CONTROL LAYER)

Lớp kiểm soát Lưu lượng nhằm đưa một tính năng tương đương với Kiểm soát Lưu lượng của Linux vào ns-3. Lớp Kiểm soát lưu lượng nằm giữa thiết bị mạng (L2) với giao thức mạng bất kỳ. Nó chịu trách nhiệm xử lý các gói tin và hành động trên chúng như lập lịch, đánh dấu…

Lớp Điều khiển lưu lượng chặn cả các gói tin đi theo hướng đi xuống từ lớp mạng tới thiết bị mạng và các gói đến theo hướng ngược lại.

Traffic control layer gồm có rất nhiều thuật toán và dưới đây là một vài thuật toán tiêu biểu:

1. Pfifo-fast queue

Pfifo_fast là qdisc mặc định cho tất cả các giao diện trong Linux. Dựa trên một FIFO thông thường qdisc, qdisc này cũng cung cấp một số thứ tự ưu tiên. Nó cung cấp ba băng tần khác nhau (FIFO riêng lẻ) cho phân luồng giao thông. Lưu lượng ưu tiên cao nhất (luồng tương tác) được đặt vào dải 0 và luôn phục vụ đầu tiên. Tương tự như vậy, băng tần 1 luôn trống các gói đang chờ xử lý trước khi băng tần 2 bị loại bỏ.

2. CoDel queue

Được phát triển bởi Kathleen Nichols và Van Jacobson như một giải pháp cho vấn đề tràn bộ đệm, CoDel (Quản lý độ trễ có kiểm soát) là một nguyên tắc xếp hàng sử dụng thời gian lưu trú của gói (thời gian trong hàng đợi) để đưa ra quyết định về việc hủy gói.

3. FqCoDel queue

Thuật toán FlowQueue-CoDel (FQ-CoDel) là một bộ lập lịch gói kết hợp và thuật toán. FqCoDel phân loại các gói đến thành các hàng đợi khác nhau, được phục vụ theo bộ lập lịch hàng đợi Deficit Round Robin (DRR) đã sửa đổi. Mỗi hàng đợi được quản lý bởi thuật toán CoDel AQM. FqCoDel phân biệt giữa hàng đợi "mới" (không tạo thành hàng đợi thường trực) và hàng đợi "cũ", đã xếp hàng đủ dữ liệu để tồn tại trong hơn một lần lặp lại của bộ lập lịch vòng tròn.

Trong bài báo cáo này, nhóm em sẽ mô phỏng mô hình Traffic Control sử dụng thuật toán RED (Random Early Detection).

II. THUẬT TOÁN RED

Random Early Detection, hay RED, là một thuật toán quản lý hàng đợi được sử dụng để tránh hiện tượng tắc nghẽn. Cơ chế hoạt động của thuật toán là "dự đoán" thời điểm mà tắc nghẽn có thể xảy ra, từ đó sẽ giảm tốc độ truyền các gói tin. Ý tưởng ban đầu của RED là xác định điểm tắc nghẽn bằng cách đánh giấu và loại bỏ các gói tin trước khi nó xảy. Bởi vì thế, RED được dựa trên việc điều khiển kích thước trung bình của tail và đưa nó vào giữa hai mức ngưỡng cực đại và cực tiểu. Thuật toán này được xây dựng dành cho các mạng mà trong đó việc đánh dấu gói tin đủ để đưa ra tín hiệu về sự tồn tại của sự tắc nghẽn. Cơ chế điều khiển của thuật toán theo dõi và quản lý kích cỡ trung bình của từng hàng đợi đi ra, lựa chọn một cách ngẫu nhiên kết nối nào sẽ được thông báo về việc tắc nghẽn.

Thuật toán RED có có lợi thế trong việc điều khiển tắc nghẽn tại transport level và làm việc với các giao thức TCP/IP hiện đại. Tuy nhiên, nó còn tồn tại một số điểm yếu như:

Không hoạt động tốt khi mà kích cỡ quote trung bình vượt qua ngưỡng cực đại, gây nên sự suy giảm về tốc độ truyền và tăng khả năng xảy ra packet throw.

Kích cỡ trung bình của tail thay đổi dựa trên mức độ tắc nghẽn và có thể đạt tới rất gần ngưỡng cực tiểu hoặc ngưỡng cực đại.

Mục tiêu chính của thuật toán RED trong traffic control là giảm thiểu tối đa việc các gói tin bị hao hụt và trễ trong các gateway, tránh global sourcing of sources, đảm bảo hiệu suất tiêu thụ năng lượng và hạn chế những bất lợi của việc tuyền tin đột ngột. RED hiệu quả trong việc tránh tắc nghẽn tại các gateway sử dụng các giao thức truyền tin và nó là một thuật toán có thể áp dụng trong các mạng tốc độ cao.

Trái với các thuật toán quản lý hàng đợi khác khi mà gói tin bị drop tại thời điểm buffer đầy, thuật toán RED drop các gói tin tới theo xác suất xác định. Xác suất này sẽ tăng khi mà kích cỡ hàng đợi trung bình dự kiến tăng.

2.1. Tiêu chí đánh giá của RED gateway

Một số tiêu chí cần đạt được với RED gateway

- **Tránh được tắc nghẽn**: Nếu gateway drop các gói tin tới khi mà kích cỡ trung bình của hàng đợi đạt tới ngưỡng lớn nhất thì RED gateway sẽ đảm bảo được rằng giá trị tính được của kích cỡ trung bình hàng đợi sẽ không vượt quá ngưỡng lớn nhất. Nếu mà khối lượng hàng đợi w_q dành cho quy trình EWMA được điều chỉnh một cách phù hợp thì gateway sẽ quản lý giá trị thực của kích cỡ trung bình hàng đợi. Nếu RED gateway đặt một bit trên header của các packet (đánh dấu) thì khi đó kích cỡ trung bình của hàng đợi đã vượt quá ngưỡng cực đại, cho nên lúc này thay vì drop các gói tin thì gateway sẽ dựa vào sự phối hợp của các nguồn phát để quản lý được kích cỡ trung bình hàng đợi.

- **Thang thời gian phù hợp**: Sau khi thông báo cho một kết nối về việc tắc nghẽn, nó sẽ tốn ít nhất một roundtrip để gateway có thể thấy được sự suy giảm tốc độ tới của các gói tin. Trong RED gateway, thang thời gian của quá trình phát hiện tắc nghẽn đại khái sẽ phù hợp với thang thời gian cần để kết nối có thể phản ứng với tắc nghẽn. RED gateway sẽ không thông báo cho các kết nối để dẫn tới giảm kích thước cửa sổ như là một kết quả của tắc nghẽn tức thời xảy ra tại gateway.

- **Không xảy ra global synchronization**: Tốc độ mà gateway đánh dấu các gói tin sẽ phụ thuộc vào mức độ tắc nghẽn. Tại thời điểm mà mức độ tắc nghẽn thấp, tỉ lệ đánh mỗi dấu gói tin tới của gateway sẽ thấp, và khi mức độ tắc nghẽn tăng, tỉ lệ đánh dấu gói tin cũng sẽ tăng lên. RED gateway sẽ tránh global synchronization bằng cách đánh dấu các gói tin với một tốc độ thấp nhất có thể.

- **Tính đơn giản**: Tiêu chí này phải được đảm bảo để thuật toán RED có thể được áp dụng với lượng tài nguyên vừa phải trong các mạng.

- **Tối đa hoá tỉ lệ thông lượng so với trễ**: các RED gateway sẽ quản lý một cách rõ ràng kích cỡ trung bình của hàng đợi cho nên tỷ lệ của thông lượng so với trễ của của RED sẽ cao hơn nhiều so với Drop Tail.

- **Tính công bằng**: Một tiêu chí khác của việc tránh tắc nghẽn đó là tính công bằng. Đích đến của tiêu chí này không được định nghĩa rõ ràng cho nên ta có thể miêu tả một cách đơn giản hoạt động của RED gateway về vấn đề này như sau: thuật toán sẽ không phân biệt giữa những kết nối cụ thể với các lớp ở trong kết nối, số lượng của những gói tin được đánh dấu với mỗi kết nối sẽ tỉ lệ thuận với phần băng thông của kết nối. Tuy nhiên, RED gateway sẽ không đảm bảo rằng mỗi kết nối đều nhận được số thông lượng bằng nhau và sẽ không quản lý một cách rõ ràng những người dùng có hành vi sai. RED gateway sẽ cung cấp một cơ chế để có thể xác định mức độ tắc nghẽn và đồng thời nó cũng có thể được sử dụng để xác định những kết nối mà sử dụng nhiều bang thông. Nếu cần thì nhưng cơ chế khác có thể được thêm vào để giúp RED gateway điều khiển thông lượng của kết nối trong chu kỳ của tắc nghẽn.

- **Phù hợp để sử dụng trong nhiều các hệ thống khác nhau**: cơ chế ngẫu nhiên của thuật toán trong việc đánh dấu các gói tin tương thích với những mạng với kết nối có nhiều roundtrip time và thông lượng, đồng thời, nó còn phù hợp với số lượng lớn các kết nối hoạt động trong một thời điểm. Sự thay đổi trong truyền tải được xác định thông qua sự thay đổi của kích cỡ trung bình hàng đợi và tốc độ các gói tin được đánh dấu sẽ được điều chỉnh tương ứng.

Trong mạng mà RED gateway thông báo về tắc nghẽn bằng cách drop các gói tin được đánh dấu, có những trường hợp mà trong một mạng TCP/IP thì những gói tin bị drop không gây nên bất cứ sự suy giảm nào trong lưu lượng đường truyền. Nếu mà gateway drop một gói tin trong kết nối TCP thì gói tin đó sẽ có khả năng sẽ bị xoá bởi nguồn sau quá trình truyền lại. Nếu gateway drop một gói tin ACK trong kết nối TCP hoặc một gói tin từ kết nối không phải kết nối TCP, quá trình drop gói tin nãy sẽ không được ghi nhận bởi nguồn. Tuy nhiên, với một mạng bị tắc nghẽn có kết cấu bao gồm nhiều kết nối TCP

ngắn hoặc bởi các kết nối không phải TCP thì RED gateway vẫn có thể quản lý kích cỡ trung bính của hàng đợi bằng cách drop tất cả cả gói tin tới khi mà kích cỡ trung bình đó vượt quá giá trị của ngưỡng cực đại.

2.2. Nguyên lý hoạt động của thuật toán RED

Thuật toán RED sẽ tính toán kích cỡ trung bình của hàng đợi sử dụng một bộ lọc thông thấp (với EWMA) sau đó so sánh với hai mức ngưỡng cực tiểu và cực đại. Khi mà kích cỡ trung bình hàng đợi nhỏ hơn ngưỡng cực tiểu thì sẽ không có gói tin nào được đánh dấu. Khi mà kích cỡ trung bình hàng đợi lớn hơn ngưỡng cực đại thì tất cả các gói tin đều được đánh dấu. Nếu mà những gói tin được đánh dấu bị drop hoặc nếu all source nodes are cooperative (cái này chưa biết dịch ntn), điều này đảm bảo kích cỡ trung bình của hàng đợi không không vượt quá ngưỡng cực đại một cách quá đáng kể. Còn trong trường hợp kích cỡ trung bình của hàng đợi nằm giữa hai ngưỡng cực tiểu và cực đại, mỗi gói tin đi tới sẽ được đánh dấu với xác suất p_a, với p_a là hàm của kích cỡ trung bình hàng đợi *avg*. Mỗi khi mà một gói tin được đánh dấu, xác suất được đánh dấu từ một kết nối cụ thể sẽ tỉ lệ thuận với băng thông chia sẻ của kết nối đó tại gateway. Thuật toán chung của RED như sau:

for each packet arrival

calculate the average queue size $\boldsymbol{avg}$

if $\boldsymbol{min_{th} \leq avg < max_{th}}$

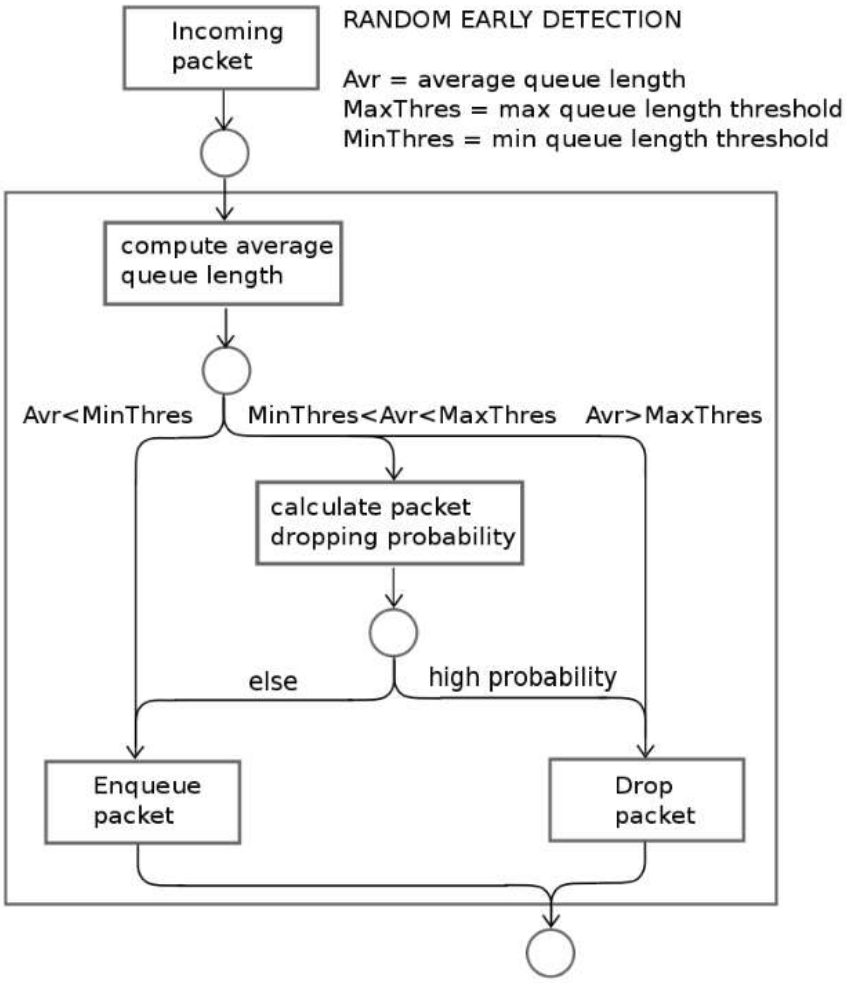
calculate probability p_a

with probability p_a

mark the arriving packet

else if $\boldsymbol{max_{th} \leq avg}$ **->** *mark the arriving packet*

Sơ đồ minh hoạ cho cơ chế của thuật toán RED:



Từ đây ta thấy RED bao gồm 2 thuật toán con riêng biệt. Một thuật toán dùng để tính toán kích cỡ trung bình hàng đợi nhằm xác định độ giãn nở cho phép trong các hàng đợi. Thuật toán còn lại sẽ tính toán xác suất đánh dấu để xác định mức độ thường xuyên mà các gói tin được đánh dấu, từ đó đưa ra những dữ liệu về mức độ tắc nghẽn. Mục tiêu của thuật toán RED là đánh dấu các gói tin tại các thời điểm cách đều nhau để ngăn chặn sự bão hào và global synchronization, đồng thời đánh dấu các gói tin đủ thường xuyên nhằm kiểm soát tốt kích cỡ trung bình của hàng đợi.

2.2.1. Kích cỡ trung bình hàng đợi

Thuật toán RED sử dụng một bộ lọc thông thấp (với EWMA) để tính toán kích cỡ trung bình của hàng đợi. Việc tính toán được thực hiện tại thời điểm các gói tin tới thay vì là tại một khoảng thời gian cố định, quá trình tính được điều chỉnh khi mà một gói tin

đi tới một hàng đợi trống tại gateway. Sau khi gói tin tới nơi thì thuật toán sẽ tính số lượng m gói tin có thể được truyền trong lúc hàng đợi trống. Kích cỡ trung bình của hàng đợi được tính toán nếu *m* gói tin tới được gateway với kích cỡ hàng đợi bằng 0. Quá trình tính toán như sau:

$m = (time - q\_time)/s$

q_time là thời điểm hàng chờ bắt đầu nghỉ

s là thời gian truyền chung của các gói tin nhỏ

$avg \leftarrow (1 - w_q)^m avg$ *; khi hàng đợi trống*

$avg \leftarrow (1 - w_q) avg + w_q\ q$ *; khi hàng đợi không trống*

w_q: Khối lượng của hàng đợi, xác định hằng số thời gian của bộ lọc thông thấp

2.2.2. Giới hạn trên của w_q

Khi mà w_q quá lớn thì tắc nghẽn tức thời sẽ không bị lọc ra. Giả sử hàng đợi ban đầu trống, kích cỡ trung bình bằng 0 và số lượng gói tin trong hàng đợi tăng từ 0 tới L. Sau khi gói tin tới lần thứ L tại gateway, kích thước trung bình của hàng đợi (avg_L) là:

$$avg_L = \sum_{i=1}^{L} i\ w_q (1 - w_q)^{L-i}$$

$$= w_q (q - w_q)^L \sum_{i=1}^{L} i\ w_q (1 - w_q)^{L-i}$$

$$= L + 1 + \frac{(1-w_q)^{L+1} - 1}{w_q}$$

Cho ngưỡng nhỏ nhất min_{th} và L gói tin đang đi tới hàng đợi, w_q được chọn để thoả mãn bất đẳng thức $avg_L < min_{th}$ hay $L + 1 + \frac{(1-w_q)^{L+1} - 1}{w_q} < min_{th}$

2.2.3. Thiết lập giá trị ngưỡng

Giá trị của min_{th} và max_{th} phụ thuộc vào kích cỡ trung bình của hàng chờ. Với Bursty traffic thì giá trị min_{th} cần phải lớn để cho phép link utilization duy trì ở môt mức cao có thể chấp nhận được. Giá trị tối ưu cho cho max_{th} phụ thuộc vào độ trễ trung bình lớn nhất đươc cho phép bởi RED gateway. Thuật toán sẽ hiệu quả nhất khi $max_{th} - min_{th}$ lớn hơn độ tăng của kích cỡ trung hình hàng đợi trong một RTT (Round-trip Time). Bởi vì lý do trên, ta thường đặt $max_{th} \geq 2min_{th}$.

2.2.4. Xác suất đánh dấu

Khi giá trị của *avg* lớn hơn max_{th}, toàn bộ các gói tin tới sẽ được đánh dấu.

Khi giá trị *avg* thay đổi trong khoảng min_{th} và max_{th}, xác suất đánh dấu gói tin p_b sẽ thay đổi tuyến tính trong từ 0 tới max_p:

$$p_b \leftarrow max_p\ (avg - min_{th})/(\ max_{th} - min_{th}).$$

Xác suất đánh dấu cuối cùng p_a tăng từ từ khi mà *count* tăng sau khi gói tin cuối cùng được đánh dấu.

$$p_a \leftarrow p_b/(1-count.p_b)$$

Với phương pháp như trên, ta có thể đảm bảo gateway không phải đợi quá lâu trước khi đánh dấu một gói tin.

Một lựa chọn khác cho RED gateway đó là đo hàng đợi dựa theo bytes thay vì theo các gói tin. Khi đó, kích cỡ trung bình của hàng đợi sẽ phản ánh chính xác độ trễ trung bình tại gateway. Thuật toán trong trường hợp này cần phải chỉnh sửa để đảm bảo rằng tỉ lệ một gói tin được đánh dấu sẽ tỉ lệ thuận với kích cỡ của gói tin tính theo bytes:

$$p_b \leftarrow max_p\ (avg - min_{th})/(\ max_{th} - min_{th}).$$

$$p_b \leftarrow p_b\ \text{PacketSize/MaximumPacketSize}$$

$$p_a \leftarrow p_b/(1-count.p_b)$$

Thuật toán chi tiết của RED:

Khởi tạo:

$avg \leftarrow 0$

$count \leftarrow -1$

for each packet arrival

calculate new avg. queue size avg:

if the queue is nonempty

$avg \leftarrow (1 - w_q)avg + w_q q$

else

$m = f(time - q\_time)$

$avg \leftarrow (1 - w_q)^m avg$

if $min_{th} \leq avg < max_{th}$

increment count

calculate probability p_a *:*

$p_b \leftarrow max_p(avg - min_{th})/(max_{th} - min_{th})$

$p_a \leftarrow p_b/(1 - count.p_b)$

with probability p_a *:*

mark the moving packet

$count \leftarrow 0$

else if $math_{th} \leq avg$

mark the arriving packet

$count \leftarrow 0$

else $count \leftarrow -1$

when queue becomes empty

$q\_time \leftarrow time$

Saved Variables:

+)avg : average queue size

+)q_time : start of the queue idle time

+)$count$: packet since last mark pkt

Fixed parameters:

+)w_q : queue weight

+)min_{th}: minimum threshold for queue

+)max_{th} : maximum threshold for queue

+)max_p : maximum value for p_b

Other:

+)p_a : current pkt-marking probability

+)q : current queue size

+)$time$: current time

+)$f(t)$: a linear function of the time t

III. MÔ PHỎNG

1. Xây dựng mô hình Traffic control sử dụng thuật toán RED (Random Early Detection)

Trong phần mô phỏng lại mô hình Traffic control sử dụng thuật toán RED, ở đây nhóm em chọn tệp Red-tests.cc với địa chỉ như sau src/traffic-control/examples

Những thư viện được sử dụng:

```
#include "ns3/applications-module.h"
#include "ns3/core-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/internet-module.h"
#include "ns3/network-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/traffic-control-module.h"
```

Hình minh họa cấu trúc liên kết mạng trong tệp mặc định như sau:

```
/** Network topology
 *
 *    10Mb/s, 2ms                                10Mb/s, 4ms
 * n0--------------|                          |---------------n4
 *                 |   1.5Mbps/s, 20ms        |
 *                n2-------------------------n3
 *    10Mb/s, 3ms  |                          |    10Mb/s, 5ms
 * n1--------------|                          |---------------n5
 *
 *
 */
```

Trong cấu trúc này chúng ta có thể thấy rằng Node 2 và Node 3 đóng vai trò là gate way. Các nodes còn lại lần lượt liên kết point to point đến các gateways. Ta có thể thấy lần lượt Node 0,1 liên kết p2p đến node 2; Node 4,5 liên kết p2p đến Node 3. Mỗi liên kết đều có tốc độ truyền dữ liệu là 10Mb/s và độ trễ của mỗi liên kết là khác nhau.

Chương trình bắt đầu bằng đặt một số tham số như tốc độ truyền dữ liệu của 2 gateway N2 và N3 bằng 1.5Mb/s với độ trễ là 20ms và dòng lệnh để bật hoặc tắt các thành phần ghi nhật ký như Pcap (Quan sát bằng cách sử dụng wireshark) , Flow Monitor và một số thông số cho biểu diễn trên đồ thị.

```
int
main(int argc, char* argv[])
{
    LogComponentEnable("RedQueueDisc", LOG_LEVEL_INFO);

    uint32_t redTest;
    std::string redLinkDataRate = "1.5Mbps";
    std::string redLinkDelay = "20ms";

    std::string pathOut;
    bool writeForPlot = true;
    bool writePcap = true;
   // bool flowMonitor = false;

    bool printRedStats = true;
```

Khởi tạo các tham số của thuật toán Random Early Detection:

```
NS_LOG_INFO("Set RED params");
Config::SetDefault("ns3::RedQueueDisc::MaxSize", StringValue("1000p"));
Config::SetDefault("ns3::RedQueueDisc::MeanPktSize", UintegerValue(meanPktSize));
Config::SetDefault("ns3::RedQueueDisc::Wait", BooleanValue(true));
Config::SetDefault("ns3::RedQueueDisc::Gentle", BooleanValue(true));
Config::SetDefault("ns3::RedQueueDisc::QW", DoubleValue(0.002));
Config::SetDefault("ns3::RedQueueDisc::MinTh", DoubleValue(5));
Config::SetDefault("ns3::RedQueueDisc::MaxTh", DoubleValue(15));
```

Bước tiếp theo là khởi tạo các Nodes:

```
NS_LOG_INFO("Create nodes");
NodeContainer c;
c.Create(6);
Names::Add("N0", c.Get(0));
Names::Add("N1", c.Get(1));
Names::Add("N2", c.Get(2));
Names::Add("N3", c.Get(3));
Names::Add("N4", c.Get(4));
Names::Add("N5", c.Get(5));
n0n2 = NodeContainer(c.Get(0), c.Get(2));
n1n2 = NodeContainer(c.Get(1), c.Get(2));
n2n3 = NodeContainer(c.Get(2), c.Get(3));
n3n4 = NodeContainer(c.Get(3), c.Get(4));
n3n5 = NodeContainer(c.Get(3), c.Get(5));
```

Ta có các NodeContainer thể hiện cho liên kết point to point giữa các Nodes.

Hiện tại, ta đã tạo các Nodes, nhưng không có ngăn xếp giao thức nào. Sử dụng InternetStackHelper để cài đặt các ngăn xếp này.

```
InternetStackHelper internet;
internet.Install(c);
```

Quy định kích cỡ của hàng đợi dành cho việc drop packet, mỗi packet sẽ được xử lý giống hệt nhau và khi hàng đợi đạt đến dung lượng tối đa, các gói mới đến sẽ bị loại bỏ cho đến khi hàng đợi có đủ không gian để chấp nhận lưu lượng đến.

```
p2p.SetQueue("ns3::DropTailQueue");
p2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("2ms"));
NetDeviceContainer devn0n2 = p2p.Install(n0n2);
tchPfifo.Install(devn0n2);

p2p.SetQueue("ns3::DropTailQueue");
p2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("3ms"));
NetDeviceContainer devn1n2 = p2p.Install(n1n2);
tchPfifo.Install(devn1n2);

p2p.SetQueue("ns3::DropTailQueue");
p2p.SetDeviceAttribute("DataRate", StringValue(redLinkDataRate));
p2p.SetChannelAttribute("Delay", StringValue(redLinkDelay));
NetDeviceContainer devn2n3 = p2p.Install(n2n3);
QueueDiscContainer queueDiscs = tchRed.Install(devn2n3);

p2p.SetQueue("ns3::DropTailQueue");
p2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("4ms"));
NetDeviceContainer devn3n4 = p2p.Install(n3n4);
tchPfifo.Install(devn3n4);

p2p.SetQueue("ns3::DropTailQueue");
p2p.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
p2p.SetChannelAttribute("Delay", StringValue("5ms"));
NetDeviceContainer devn3n5 = p2p.Install(n3n5);
tchPfifo.Install(devn3n5);
```

Tiếp theo chúng ta sẽ sử dụng Ipv4AddressHelper để gán địa chỉ IP và Ipv4GlobalRoutingHelper để thiết lập định tuyến.

```
ipv4.SetBase("10.1.1.0", "255.255.255.0");
i0i2 = ipv4.Assign(devn0n2);

ipv4.SetBase("10.1.2.0", "255.255.255.0");
i1i2 = ipv4.Assign(devn1n2);

ipv4.SetBase("10.1.3.0", "255.255.255.0");
i2i3 = ipv4.Assign(devn2n3);

ipv4.SetBase("10.1.4.0", "255.255.255.0");
i3i4 = ipv4.Assign(devn3n4);

ipv4.SetBase("10.1.5.0", "255.255.255.0");
i3i5 = ipv4.Assign(devn3n5);
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

2.Khởi chạy mô phỏng

Kết quả mô phỏng:

```
bytesInQueue  50690    Qavg 15.1333
packetsInQueue  91     Qavg 15.1333
Dropping due to Prob Mark 15.1333
bytesInQueue  0        Qavg 0
packetsInQueue  0      Qavg 0
bytesInQueue  49680    Qavg 15.2831
packetsInQueue  90     Qavg 15.2831
bytesInQueue  50690    Qavg 15.4345
packetsInQueue  91     Qavg 15.4345
bytesInQueue  0        Qavg 0
packetsInQueue  0      Qavg 0
bytesInQueue  50690    Qavg 15.5856
packetsInQueue  91     Qavg 15.5856
bytesInQueue  51700    Qavg 15.7385
packetsInQueue  92     Qavg 15.7385
bytesInQueue  0        Qavg 0
packetsInQueue  0      Qavg 0
bytesInQueue  50784    Qavg 15.891
packetsInQueue  92     Qavg 15.891
```

Ta có thể thấy khi các gói tin được truyền vào hàng đợi thì chương trình sẽ tính toán lại giá trị trung bình của hàng đợi, nếu Avr < MinThres thì gói tin sẽ được enqueuer còn nếu giá trị Avr nằm giữa 2 giá trị ngưỡng MinThres và MaxThres thì chương trình sẽ tính toán xác suất Drop, nếu xác suất lớn thì gói tin sẽ bị đánh dấu (marked) và cho drop.

Đây là số liệu được tổng hợp lại sau khi chương trình chạy xong:

Sau khi chạy, Ở Node 2 nhận được 2310 Packets/Bytes trong đó có 2083 Packets/Bytes thực hiện thao tác enqueued và dequeued và số Packets/Bytes được gửi đi là 2083.

Số Packets/Bytes bị drop là 227 với Forced Drop là 111 và Unforced drop là 116. Ở đây ta có 3 lí do chính làm cho packet bị drop trước khi enqueue là:

+) Hàng đợi nội bộ đã đầy

+) Child queue disc dropped packets. Phương thức DropB BeforeEnqueue của queue disc này được gọi tự động vì QueueDisc::AddQueueDiscClass (ở thư viện QueueDisc.h) đặt lệnh trace callback.

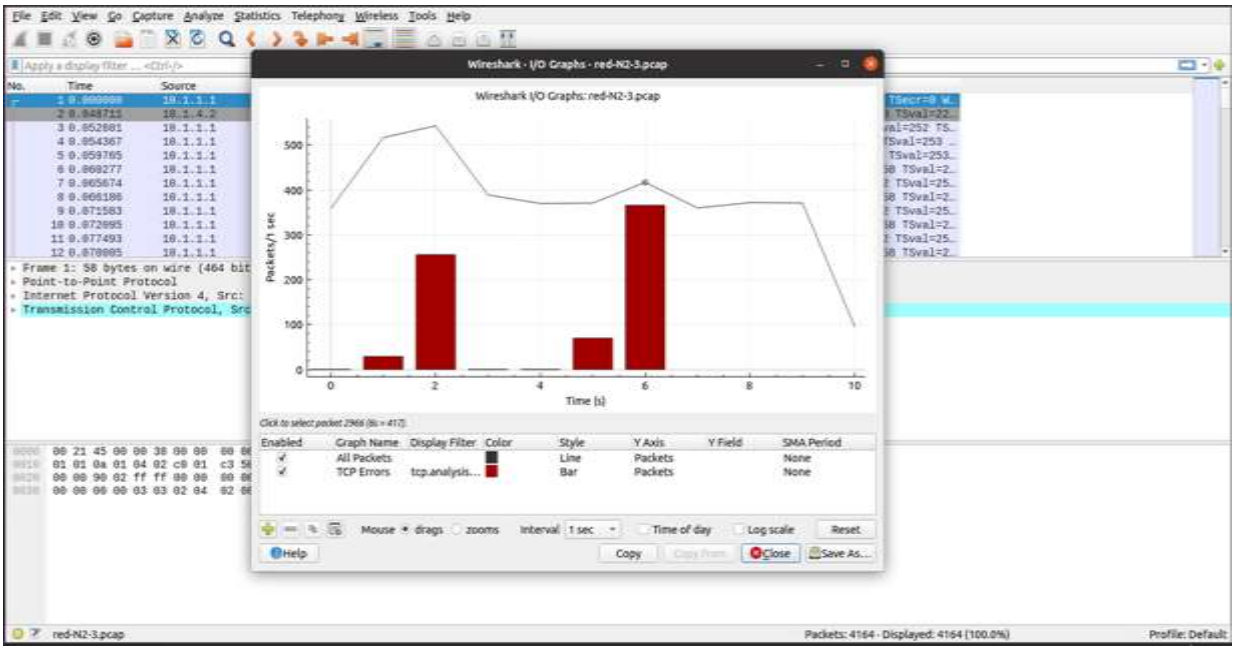
+) Các gói tin bị drop trong quá trình gửi. Khác với các thuật toán quản lý hảng đợi khác khi mà các gói tin bị drop tại thời điểm buffer đầy, thuật toán RED drop các gói tin tới theo một xác suất xác định. Xác suất sẽ tăng khi kích cỡ trung bình dự kiến của queue tăng.

Vậy số gói tin bị buộc phải Drop (Force Drop) có thể do hàng đợi nội đã đầy hoặc là do Child queue disc ép chúng phải drop. Và các gói tin bị drop mặc dù không bị tác động (Unforce Drop) có thể do xác suất của thuật toán RED.

Node 2 giao tiếp với Node 3 theo phương thức Point to Point, ở trên ta thấy Node 2 đã gửi đi 2083 Packets/Bytes đến Node 3. Do sau khi vào Node 2 thì các gói tin đã được tính toán trước xác suất Drop và những gói tin có xác suất Dropped cao đã bị drop trước khi Node 2, nên sau khi truyền Packets/Bytes từ N2 sang N3 thì sẽ không xuất hiện tình trạng drop packet do xác suất nữa. Và do RED Gateways có thể kiểm soát kích thước hàng đợi trung bình trong khi hỗ trợ tắc nghẽn tạm thời nên khi truyền giữa 2 gateway sẽ ít hoặc không xảy ra hiện tượng hàng đợi bị đầy dẫn đến các gói tin bị ép phải drop (force drop).

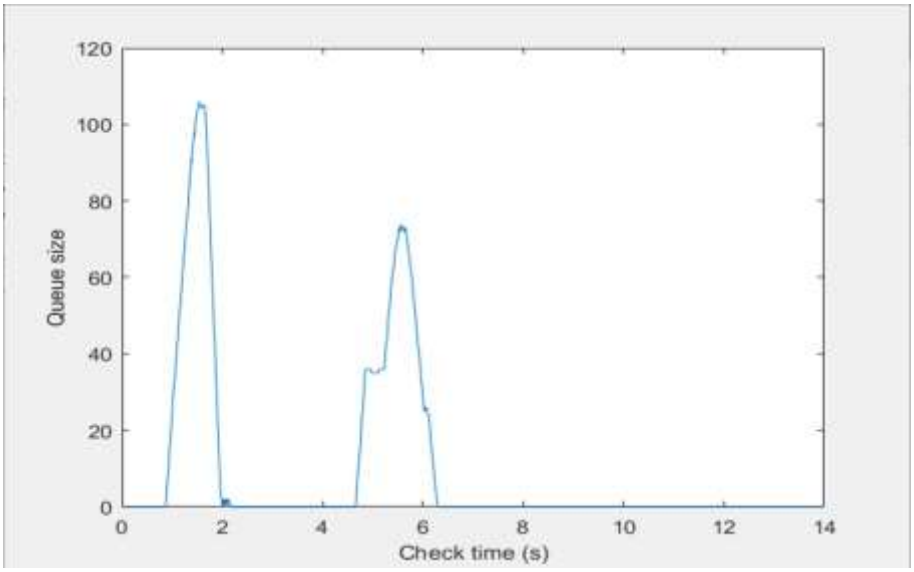
Sau khi kích hoạt các thành phần ghi nhật kí ta được file pcap (Wireshark), .xml(Flow Monitor) và 2 file ghi lại số liệu tính toán Queue size (red-queue.plotme) và Average Queue size (red-queue_avg.plotme)

Với file .pcap sau khi chạy thì ta được:

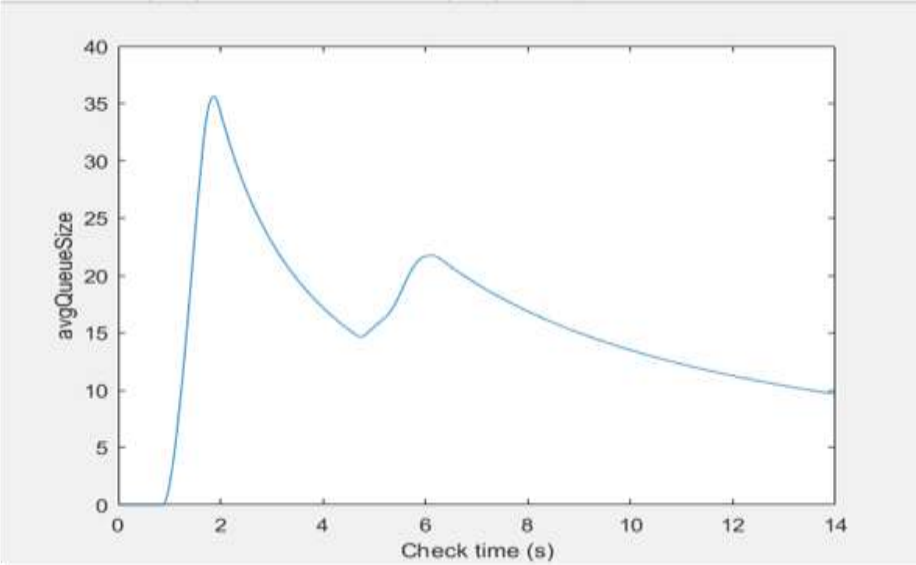


Ta có thể quan sát được số gói tin được truyền đến theo từng giây (Biểu đồ đường) và số gói tin truyền lỗi (Dropped) trong khoảng thời gian truyền dữ liệu từ Node 2 sang Node 3.

+)Với 2 file red-queue.plotme và red-queue_avg.plotme, sau khi mô phỏng trên Matlab ta được:



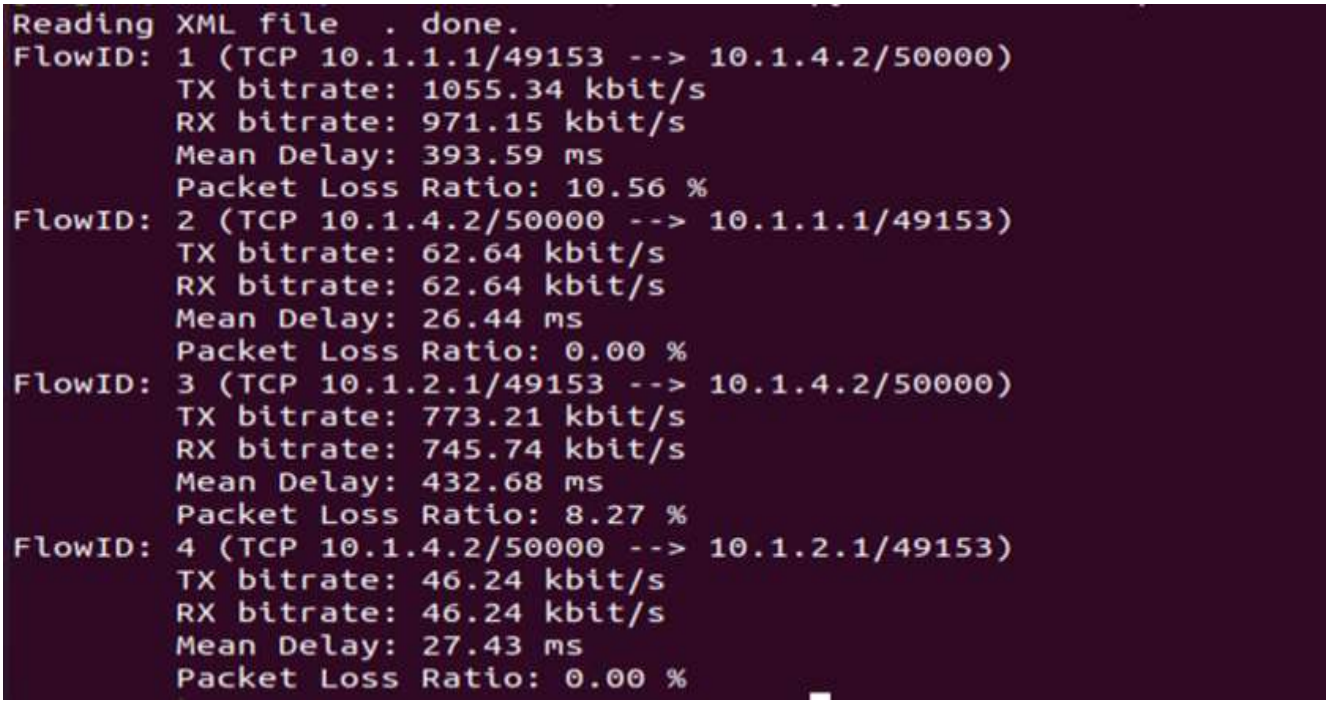
Hình 1:Đồ thị biểu diễn Queue size sau mỗi check time



Hình 2: Biểu đồ thể hiện Queue size trung bình sau mỗi check time

Chú thích: Chương trình sẽ kiểm tra kích cỡ của hàng đợi (queue size) sau mỗi 1/100 giây

+) Với file.xml ta cần thêm file flowmon-parse-results.py ở địa chỉ src/flow-monitor/examples. Ta chạy file .xml với lệnh sau: python3 flowmon-parse-results.py Red.xml. Kết quả thu được:



Ta có thể quan sát được tốc độ truyền dữ liệu truyền và nhận (TX và RX). Độ trễ trung bình (Mean Delay) và tỉ lệ Packet Loss.

IV. KẾT LUẬN

RED là một cơ chế hiệu quả để kiểm soát lưu lượng tại gateway và kết hợp tốt với các giao thức truyền tải mạng. Kết quả mô phỏng đã mô tả đúng cơ chế của thuật toán, RED hoạt động trong thời gian 'enqueue'. RED chạy trên Gateway, nhưng trong thời gian 'enqueue'! RED hoạt động 'khi có gói tin đến'. Do đó, không có khoảng thời gian định kỳ khi RED được gọi. Nếu các gói đến với tốc độ thấp hơn thì RED được gọi ở tốc độ thấp hơn. Nếu các gói không đến, thuật toán RED không được gọi. RED quyết định xem gói tin đến sẽ được xếp vào hàng đợi hay bị loại bỏ. Khi một gói mới đến, RED đưa ra quyết định; cho dù gói này nên được enqueued hoặc drop. Nó đưa ra quyết định này ngay cả khi có khoảng trống trong hàng đợi. Nó đưa ra quyết định này bởi vì nếu việc xếp gói vào

hàng đợi sẽ làm tăng độ trễ tổng thể hoặc làm đầy hàng đợi thì sẽ gây ra tắc nghẽn. Nếu RED loại bỏ gói trước khi hàng đợi đầy thì người gửi sẽ biết về nó một cách gián tiếp và nó sẽ giảm kích thước cửa sổ tắc nghẽn của nó và do đó tránh tắc nghẽn xảy ra trong tương lai.

V. TÀI LIỆU THAM KHẢO

[1] https://www.icir.org/floyd/papers/early.twocolumn.pdf?fbclid=IwAR0DAoj23mo_laXKA-pZQNAjRBMy4DCIhMEfOAVQ-YpKIv3FFRzI6YrgMJM

[2] https://www.nsnam.org/docs/models/html/traffic-control.html?fbclid=IwAR3TAvE6-da9bReger0yJh-Z2 SRMeQwOZbAC_FITjMNGZIrS_XTvwautts

[3] https://en.wikipedia.org/wiki/Random_early_detection

[4] https://www.geeksforgeeks.org/random-early-detection-red-queue-discipline/