ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

LÊ XUÂN ANH

CÁC KẾ HOẠCH QUẢN LÝ HÀNG ĐỢI ĐỘNG CHO TRUYỀN THÔNG ĐA PHƯƠNG TIỆN

Ngành: Công Nghệ Thông Tin
Chuyên ngành: Truyền dữ liệu và Mạng máy tính
Mã số:

LUẬN VĂN THẠC SĨ TRUYỀN DỮ LIỆU VÀ MẠNG MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS Nguyễn Đình Việt

Hà Nội – 2016

LỜI CAM ĐOAN

Tôi xin cam đoan Luận Văn này là của riêng tôi. Kết quả đạt được trong Luận văn là sản phẩm của riêng cá nhân tôi, không dùng bất kỳ hình thức sao chép lại nào từ các công trình của người khác. Những phần được trình bày trong nội dung Luận văn này, đều là của cá nhân hoặc được tổng hợp từ nhiều nguồn tài liệu khác nhau. Tôi xin cam đoan tất cả các tài liệu tham khảo đều có xuất xứ rõ ràng và được trích dẫn đúng quy cách, quy định. Tôi xin hoàn toàn chịu trách nhiệm và chịu mọi hình thức kỷ luật theo quy định cho lời cam đoan này của mình.

Hà Nội, 11/2016

Lê Xuân Anh

LỜI CÁM ƠN

Trước hết tôi xin gửi lời cảm ơn chân thành, sâu sắc nhất tới người hướng dẫn tôi, thầy PGS.TS. Nguyễn Đình Việt – Giảng viên khoa Công nghệ Thông tin - Trường Đại học Công nghệ - Đại học Quốc Gia Hà Nội, người đã định hướng đề tài, định hướng nghiên cứu, luôn luôn tận tình giúp đỡ, hướng dẫn và chỉ bảo tôi trong suốt quá trình thực hiện luận văn cao học này.

Tôi xin gửi lời cảm ơn chân thành tới các thầy các cô đã giảng dạy và giúp đỡ tôi trong suốt quá trình nghiên cứu và học tập tại trường Đại Học Công Nghệ - Đại Học Quốc Gia Hà Nội.

Sau cùng, tôi xin cám ơn và biết ơn tới gia đình, những người thân của tôi, những người đã ủng hộ, khuyến khích, giúp đỡ tôi rất nhiều trong quá trình học tập và thực hiện luận văn.

Do điều kiện nghiên cứu, kiến thức có hạn, nên bản luận văn không tránh khỏi sơ suất, kính mong nhận được sự góp ý của quý thầy cô, bạn bè và đồng nghiệp để bản luận văn được hoàn thiện hơn.

Hà Nội, 11/2016

Lê Xuân Anh

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

A-RED	Adaptive - Random Early Drop; Adaptive-RED
A-RIO	Adaptive – RED with In and Out bit; Adaptive-RIO
ACL	Access Control Lists
AF	Assured Forwarding
AQM	Active Queue Management
ARPANET	Advanced Research Projects Agency Network
CA	Congestion Avoidance
CBR	Constant Bit Rate
CBS	Commited Burst Size
CIR	Commited Information Rate
CP	Code Point
DS	Differentiated Services
DSCP	Differentiated Service Code Point
EBS	Excess Burst Size
ECN	Explicit Congestion Notification
EF	Expedited Forwarding
FCFS	First Come First Serve
FIFO	First In First Out
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
IETF	Internet Engineering Task Force
IntServ	Integrated Services

IP	Internet Protocol
ISP	Internet Service Provider
LAN	Local Area Network
NS	Network Simulator
PHB	Per-Hop Behavior
PIR	Peak Information Rate
PBS	Peak burst size
PQ	Priority Queue
QoS	Quality of Service
RED	Random Early Detection; Random Early Drop
RIO	RED with In and Out bit
RIO-C	RIO-Coupled
RIO-D	RIO-Decoupled
RSVP	Resource Revervation Protocol
RTT	Round Trip Time
SS	Slow Start
TCP	Transmission Control Protocol
ToS	Type of Service
TSW2CM	Time Sliding Window with Two Color Marking/Maker
TSW3CM	Time Sliding Window with Three Color Marking/Maker
srTCM	Single Rate Three ColorMaker
TSW	Time Sliding Window
UDP	User Datagram Protocol
WFQ	Weighted Fair Queuing
WRED	Weighted RED

DANH MỤC CÁC HÌNH VẼ

DANH MỤC CÁC ĐỒ THỊ

DANH MỤC CÁC BẢNG

MỞ ĐẦU

1. Mục đích và ý nghĩa của đề tài

Ngày nay trong thế giới số, trong xu hướng phát triển bùng nổ của thông tin, trong một thế giới phẳng, vấn đề liên lạc, thông tin được cập nhật liên tục, việc truyền tải thông tin ngày càng được quan tâm đặc biệt. Các ứng dụng thời gian thực trên Internet ngày càng được quan tâm và phát triển một cách nhanh chóng, vượt bâc. Vấn đề đặt ra là làm sao dữ liệu truyền đi một cách nhanh nhất, có được độ tin cậy cao nhất, tránh mất mát dữ liệu tốt nhất, giảm thiểu tối đa hiện tượng tắc nghẽn có thể xảy ra khi truyền tin. Ngày nay, các ứng dụng đa phương tiện đang là xu thế của công nghệ, có thể kể đến như điện thoại qua mạng (Internet telephony), hội thảo trực tuyến (video conferencing), xem video theo yêu cầu (video on demand)... và đặc biệt khoảng vài năm gần đây là các ứng dụng truyền hình trực tiếp (live stream) thời gian thực đang ngày càng được sử dụng rộng rãi, kể đến như các ứng dụng truyền hình trực tiếp trên Youtube, Facebook, các trang live stream giải trí của VTC… Đảm bảo chất lượng dịch vụ (QoS) là vấn đề quan trọng nhất trong truyền thông đa phương tiện. Chúng ta hiểu khái quát đảm bảo chất lượng dịch vụ ở đâu là:

- Đảm bảo độ trễ và biến động trễ (jitter) nhỏ
- Thông lượng đủ lớn
- Hệ số sử dụng đường truyền cao
- Tỷ lệ mất gói tin có thể chấp nhận được ở một mức độ nhất định.

Để đáp ứng được những yêu cầu đó, chúng ta cần phải tiến hành đồng thời các cơ chế điều khiển lưu lượng đối với các giao thức truyền thông kiểu end-to-end (cụ thể là TCP) và những cơ chế đặc biệt thực hiện đối với mạng, cụ thể là thực hiện ở các bộ định tuyến (router).

Hiện tượng tắc nghẽn trong mạng xảy ra khi có quá nhiều lưu lượng truyền đến, khiến các nút mạng không có đủ khả năng để phục vụ cho tất cả. Để tránh được sự tắc nghẽn trong mạng, tận dụng được tối đa băng thông của đường truyền, giao thức TCP sử dụng kỹ thuật: khởi động chậm – SS, tránh tắc nghẽn – CA và giảm tốc độ phát lại các gói tin bị mất do tắc nghẽn theo cấp số nhân. Thực thể TCP bên gửi duy trì một cửa sổ gọi là cửa sổ tắc nghẽn dùng để giới hạn lượng dữ liệu tối đa có thể gửi đi liên tiếp ở mức không vượt quá kích thước vùng đệm của nơi nhận khi xảy ra tắc nghẽn. Khi bị mất một gói tin, thực thể TCP bên gửi giảm kích thước cửa sổ tắc nghẽn đi một nửa, nếu việc mất gói tin tiếp diễn, kích thước cửa sổ tắc nghẽn lại giảm tiếp theo cách trên (cho tới khi chỉ còn bằng kích thước của một gói tin). Với những gói tin vẫn còn nằm trong cửa sổ được phép, thời gian chờ để được gửi lại sẽ được tăng lên theo hàm mũ cơ số hai sau mỗi lần phát lại.

Trong kỹ thuật truyền thống, hàng đợi được đặt một kích thước tối đa, khi các gói tin đến, sẽ được cho vào các hàng đợi đã thiết lập, khi hàng đợi đã đầy, các gói tin tiếp theo đến sẽ bị loại bỏ đến khi nào hàng đợi có chỗ (khi các gói tin trong hàng đợi được chuyển đi) thì mới được nhận tiếp vào hàng (đây là kỹ thuật FIFO hay còn gọi là FCFS). Trong bộ mô phỏng mạng NS, kỹ thuật trên được cài đặt với tên gọi là "DropTail". Với kiểu hàng đợi truyền thống FIFO này, tình trạng hàng đợi đầy xảy ra thường xuyên, dẫn đến độ trễ truyền tin lớn, tỷ lệ mất mát gói tin cao và thông lượng đường truyền là thấp, vì thế ta cần phải có các kỹ thuật khác hiệu quả hơn, đảm bảo cho mạng đạt được mục tiêu là thông lượng cao và độ trễ trung bình nhỏ,

AQM (Active Queue Management) là một chiến lược quản lý hàng đợi động, trong đó các thực thể đầu cuối có thể phản ứng lại tắc nghẽn khi hiện tượng này mới chớm có dấu hiệu xuất hiện (*). Theo đó, gateway sẽ quyết định cách thức loại bỏ sớm gói tin trong hàng đợi của nó trong khi tình trạng của mạng còn có thể kiểm soát được. Hai chiến lược AQM đặc trưng sẽ được trình bày trong luận văn là:

RED (Random Early Detection of Congestion; Random Early Drop) là một chiến lược AQM cơ bản, áp dụng cho mạng chuyển mạnh gói. RED thực hiện loại bỏ gói tin trong hàng đợi hoặc đánh dấu vào trường ECN trong header của các gói tin TCP để báo cho bên gửi biết là sắp có tắc nghẽn xảy ra, yêu cầu nguồn giảm phát tin để tránh tràn hàng đợi. Một khuyết điểm của RED là nó đối xử công bằng với tất cả các gói tin đến. Ưu điểm chính của RED là tính đơn giản, không yêu cầu tất cả các gateway trên Internet cùng phải sử dụng kỹ thuật này, mà có thể triển khai dần [16].

RIO (RED with In/Out bit) là một thuật toán mở rộng của RED, kế thừa lại RED và bổ xung thêm cách phân loại các gói tin đến theo cấp độ ưu tiên khác nhau. RIO là thuật toán AQM áp dụng cho kiến trúc mạng DiffServ, dùng để chuyển tiếp có phân loại các gói tin [9].

Mục tiêu chính của Luận văn là tập trung **nghiên cứu và đánh giá các kế hoạch quản lý hàng đợi động cho truyền thông đa phương tiện, nhằm đảm bảo chất lượng dịch vụ QoS. Nghiên cứu, đánh giá và so sánh giữa các chiến lược quản lý hàng đợi động cho truyền thông đa phương tiện, đánh giá sự ảnh hưởng của các luồng lưu lượng đột biết tác động lên các luồng có sẵn trong mạng, đánh giá vai trò đảm bảo chất lượng dịch vụ của mô hình mạng DiffServ, áp dụng chiến lược quản lý hàng đợi động RIO vào mô hình DiffServ.**

Với mục tiêu trên, với sự giúp đỡ của thầy PGS.TS Nguyễn Đình Việt, tôi đã dành thời tìm hiểu, nghiên cứu, mô phỏng, đánh giá các thuật toán quản lý hàng đợi động AQM dùng trong mạng truyền thống là RED và mở rộng của nó là A-RED. Sau đó với mô hình mạng DiffServ, mô hình mạng có phân loại các luồng dữ liệu đến tôi đã tìm

hiểu và nghiên cứu về RIO một thuật toán mở rộng của RED. Để cuối cùng áp dụng RIO vào DiffServ tiến hành mô phỏng và đánh giá nó hướng đến kết luận của mục tiêu đề ra.

Luận Văn với đề tài "CÁC KẾ HOẠCH QUẢN LÝ HÀNG ĐỢI ĐỘNG CHO TRUYỀN THÔNG ĐA PHƯƠNG TIỆN" của tôi sẽ được chia thành các mục chính như trình bày dưới đây:

2. Cấu trúc các chương

Xuất phát từ những mục đích trên Luận văn được chia làm 4 chương như sau:

- Chương 1: Tổng quan về Mạng Internet và các dịch vụ. Giới thiệu về truyền thông đa phương tiện trên mạng, khái niệm QoS, các phương pháp đảm bảo chất lượng dịch vụ trong truyền thông đa phương tiện. Các khái niệm về hiệu năng và các độ đo, các phương pháp đánh giá hiệu năng mạng, giới thiệu sơ lược về bộ mô phỏng NS2.35 mà chúng tôi sẽ dùng để mô phỏng và đánh giá trong luận văn.
- Chương 2: Trình bày về các chiến lược quản lý hàng đợi động trên kiến trúc mạng truyền thống: RED, A-RED. Mỗi chiến lược đều có mô phỏng (thông qua bộ mô phỏng NS2) và kết quả mô phỏng đi kèm.
- Chương 3: Trình bày về các chiến lược quản lý hàng đợi động trong kiến trúc mạng DiffServ, hướng đến mục tiêu nhằm đảm bảo chất lượng dịch vụ QoS. Tổng quan về DiffServ và trình bày chiến lược đặc trưng là RIO, RIO là một thuật toán kế thừa RED và có thêm chức năng xử lý các gói ti đến theo mức độ ưu tiên khác nhau. Áp dụng RIO vào mạng DiffServ, đánh giá so sánh giữa RIO và RED, nghiên cứu và đánh giá vai trò đảm bảo dịch vụ trong truyền thông đa phương tiện của mô hình DiffServ kết hợp với thuật toán quản lý hàng đợi động RIO, sự ảnh hưởng của các luồng lưu lượng ưu tiên và không ưu tiên gây ra trong mạng (Sử dụng bộ mô phỏng NS2).
- Chương 4: Mô phỏng và đánh giá.

CHƯƠNG 1. GIỚI THIỆU

1.1 Mạng Internet và giao thức TCP/IP

1.1.1 Mạng Internet

[7] Tiền thân của mạng Internet là ARPANET, xuất phát từ một mạng thí nghiệm được Robert L.G đề xuất vào năm 1967. Cơ quan quản lý dự án nghiên cứu phát triển ARPA thuộc Bộ Quốc phòng Mỹ đã liên kết mạng tại 4 địa điểm đầu tiên vào tháng 7 năm 1968 bao gồm: Viện nghiên cứu Stanford, Đại học tổng hợp California ở Los Angeles, Đại học tổng hợp Utah và Đại học tổng hợp California ở Santa Barbara (UCSB). Đó chính là mạng liên khu vực (WAN) đầu tiên được xây dựng.

Thuật ngữ "Internet" xuất hiện lần đầu vào khoảng năm 1974. Lúc đó mạng vẫn được gọi là ARPANET.

Năm 1983, giao thức TCP/IP chính thức được coi như một chuẩn đối với ngành quân sự Mỹ và tất cả các máy tính nối với ARPANET phải sử dụng chuẩn mới này.

Năm 1984, ARPANET được chia ra thành hai phần: phần thứ nhất vẫn được gọi là ARPANET, dành cho việc nghiên cứu và phát triển; phần thứ hai được gọi là MILNET, là mạng dùng cho các mục đích quân sự. Giao thức TCP/IP ngày càng thể hiện rõ các điểm mạnh của nó, quan trọng nhất là khả năng liên kết các mạng khác với nhau một cách dễ dàng. Chính điều này cùng với các chính sách mở cửa đã cho phép các mạng dùng cho nghiên cứu và thương mại kết nối được với ARPANET, thúc đẩy việc tạo ra một siêu mạng (SuperNetwork).

Năm 1980, ARPANET được đánh giá là mạng trụ cột của Internet. Mốc lịch sử quan trọng của Internet được xác lập vào giữa thập niên 1980 khi tổ chức khoa học quốc gia Mỹ NSF thành lập mạng liên kết các trung tâm máy tính lớn với nhau gọi là NSFNET. Nhiều doanh nghiệp đã chuyển từ ARPANET sang NSFNET và do đó sau gần 20 năm hoạt động, ARPANET không còn hiệu quả đã ngừng hoạt động vào khoảng năm 1990.

Sự hình thành mạng xương sống của NSFNET và những mạng vùng khác đã tạo ra một môi trường thuận lợi cho sự phát triển của Internet.

Tới năm 1995, NSFNET thu lại thành một mạng nghiên cứu còn Internet thì vẫn tiếp tục phát triển. Với khả năng kết nối mở như vậy, Internet đã trở thành một mạng lớn nhất trên thế giới, mạng của các mạng, xuất hiện trong mọi lĩnh vực thương mại, chính trị, quân sự, nghiên cứu, giáo dục, văn hoá, xã hội... Cũng từ đó, các dịch vụ trên Internet không ngừng phát triển tạo ra cho nhân loại một thời kỳ mới: kỷ nguyên thương mại điện tử trên Internet.

1.1.2 Giao thức tầng giao vận: TCP và UDP

TCP (Transmission Control Protocol - "Giao thức điều khiển truyền vận") là một trong các giao thức cốt lõi của bộ giao thức TCP/IP. Với TCP, các ứng dụng trên các máy chủ được nối mạng có thể tạo các "kết nối" với nhau, từ đó các máy có thể gửi các gói tin cho nhau. Giao thức TCP đảm bảo dữ liệu được chuyển tới nơi nhận một cách đáng tin cậy và đúng thứ tự. TCP còn phân biệt giữa dữ liệu của nhiều ứng dụng (chẳng hạn, dịch vụ Web và dịch vụ thư điện tử) đồng thời chạy trên cùng một máy chủ. TCP hỗ trợ nhiều giao thức ứng dụng phổ biến nhất trên Internet và các ứng dụng kết quả, trong đó có WWW, thư điện tử và Secure Shell. Trong bộ giao thức TCP/IP, TCP là tầng trung gian giữa giao thức IP bên dưới và một ứng dụng bên trên. Các ứng dụng thường cần các kết nối đáng tin cậy kiểu đường ống để liên lạc với nhau, trong khi đó, giao thức IP không cung cấp những dòng kiểu đó, mà chỉ cung cấp dịch vụ chuyển gói tin không đáng tin cậy.

Trong mô hình OSI đơn giản TCP làm nhiệm vụ của tầng giao vận. Các ứng dụng gửi các dòng gồm các byte 8-bit tới TCP để chuyển qua mạng. TCP phân chia dòng byte này thành các đoạn (segment) có kích thước thích hợp (thường được quyết định dựa theo kích thước của đơn vị truyền dẫn tối đa (MTU) của tầng liên kết dữ liệu của mạng mà máy tính đang nằm trong đó). Sau đó, TCP chuyển các gói tin thu được tới giao thức IP để gửi nó qua một liên mạng tới mô đun TCP tại máy tính đích. TCP kiểm tra để đảm bảo không có gói tin nào bị thất lạc bằng cách gán cho mỗi gói tin một "số thứ tự" (sequence number). Số thứ tự này còn được sử dụng để đảm bảo dữ liệu được trao cho ứng dụng đích theo đúng thứ tự. Mô đun TCP tại đầu kia gửi lại "tin báo nhận" (acknowledgement) cho các gói tin đã nhận được thành công; một "đồng hồ" (timer) tại nơi gửi sẽ báo time-out nếu không nhận được tin báo nhận trong khoảng thời gian bằng một round-trip time (RTT), và dữ liệu (được coi là bị thất lạc) sẽ được gửi lại. TCP sử dụng checksum (giá trị kiểm tra) để xem có byte nào bị hỏng trong quá trình truyền hay không; giá trị này được tính toán cho mỗi khối dữ liệu tại nơi gửi trước khi nó được gửi, và được kiểm tra tại nơi nhận.

Khác với giao thức UDP – giao thức có thể gửi gói tin mà không cần thiết lập kết nối. Giao thức TCP đòi hỏi phải thiết lập kết nối trước khi truyền dữ liệu. Các kết nối sử dụng TCP có 3 giai đoạn:

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

Để thiết lập kết nối TCP sử dụng quá trình bắt tay ba 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:

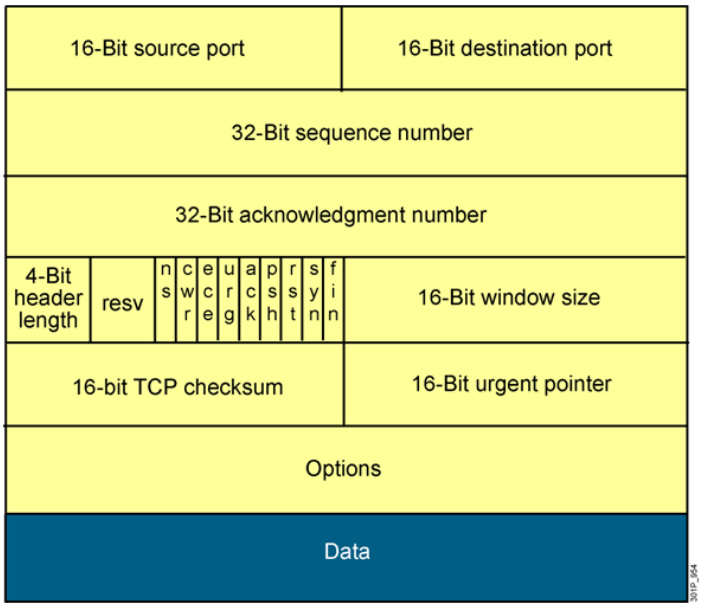
- Client yêu cầu mở cổng dịch vụ bằng cách gửi gói tin SYN (gói tin TCP) tới server, trong gói tin này, tham số sequence number được gán cho một giá trị ngẫu nhiên X, đó là số thứ tự khởi đầu - ISN (Initial sequence number), của bên gửi.
- Server hồi đáp bằng cách gửi lại phía client bản tin SYN-ACK, 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, đó là số thứ tự khởi đầu - ISN (Initial sequence number), của bên nhận.
- 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.

Một vài đặc điểm của TCP để có thể 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.

TCP Header: do TCP là giao thức đáng tin cậy nên header của TCP rất phức tạp



Hình 1.1: TCP Header

- Source Por (16 bits): Số hiệu cổng TCP của trạm nguồn.
- Destination Port (16 bit): Số hiệu cổng TCP của trạm đích.
- 32 bit sequence number: dùng để đánh số thứ tự gói tin (từ số sequence nó sẽ tính ra được số byte đã được truyền).
- 32 bit acknowledgement number: dùng để báo nó đã nhận được gói tin nào và nó mong nhận được byte mang số thứ tự nào tiếp theo.
- 4 bit header length: cho biết toàn bộ header dài bao nhiêu Word (1 Word = 4 byte).
- Reserved (6 bit): dành để dùng trong tương lai
- Control bit (các bit điều khiển):
    - URG: Vùng con trỏ khẩn (Ucgent Poiter) có hiệu lực.
    - ACK: Vùng báo nhận (ACK number) có hiệu lực.
    - PSH: Chức năng PUSH.
    - RST: Khởi động lại (reset) liên kết.
    - SYN: Đồng bộ hóa số hiệu tuần tự (sequence number).
    - FIN: Không còn dữ liệu từ trạm nguồn.
- Phần kí tự (trước 16 bit Window Size): là các bit dùng để điều khiển cờ (flag) ACK, cờ Sequence v.v.
- Checksum (16 bit): mã kiểm soát lỗi cho toàn bộ segment (header + data)
- 16 bit urgent pointer: được sử dụng trong trường hợp cần ưu tiên dữ liệu (kết hợp với bit điều khiển URG ở trên).

- Options (độ dài thay đổi): khai báo các option của TCP, trong đó có độ dài tối đa của vùng TCP data trong một segment.
- Data (độ dài thay đổi): chứa dữ liệu của tầng trên, có độ dài tối đa ngầm định là 536 byte. Giá trị này có thể điều chỉnh bằng cách khai báo trong vùng options.

UDP (User Datagram Protocol):

Khác với TCP, UDP không cung cấp sự tin cậy và thứ tự truyền nhận mà TCP làm, các gói dữ liệu có thể đến không đúng thứ tự hoặc bị mất mà không có thông báo. Do đó UDP cung cấp các dịch vụ vận chuyển không tin cậy. Tuy nhiên UDP nhanh và hiệu quả hơn đối với các mục tiêu như kích thước nhỏ và yêu cầu khắt khe về thời gian. Do bản chất không trạng thái của nó nên nó hữu dụng đối với việc trả lời các truy vấn yêu cầu truyền lượng dữ liệu nhỏ với số lượng lớn người yêu cầu.

UDP được dùng khi tốc độ là mong muốn và việc sửa lỗi là không cần thiết. Đây cũng là lý do mà UDP được dùng cho truyền thông đa phương tiện. Ví dụ như: các chương trình phát sóng trực tiếp, các luồng dữ liệu có độ ưu tiên cao. Giả sử bạn đang xem hình ảnh video trực tiếp, lúc này giao thức thường sử dụng là UDP thay vì TCP. Các máy chủ chỉ cần gửi một dòng của các gói tin UDP để máy tính xem. Nếu bạn bị mất kết nối của bạn trong vài giây, video sẽ đóng băng cho một thời điểm và sau đó chuyển đến các bit hiện tại của truyền hình, bỏ qua các bit bạn đã bị bỏ qua. Video hoặc âm thanh có thể bị bóp méo một lúc và video tiếp tục được chạy mà không có dữ liệu bị mất.

UDP Header:

Source Port	Destination Port
Message length	Checksum
DATA	

Hình 1.2: UDP Header

- Gồm 16 bit source port
- 16 bit destination port. Vậy port là gì? Có rất nhiều session sử dụng kết nối UDP vậy làm thế nào để định danh chúng? Để giải quyết điều này tầng Transport dùng 1 cặp source port và destination port để định danh 1 session đang truy nhập vào đường truyền của kết nối UDP. Ta có thể coi port là địa chỉ tầng Transport (giao thức DNS chạy UDP port 53, TFTP port 69…).
- 16 bit UDP Length: cho biết toàn bộ gói tin UDP dài tổng cộng bao nhiêu byte.

- 16 bit UDP checksum: sử dụng thuật toán mã vòng CRC để kiểm lỗi. Và chỉ kiểm tra một cách hạn chế.

Những ứng dụng phổ biến sử dụng UDP như DNS (Domain Name System), ứng dụng streaming media, Voice over IP, Trivial File Transfer Protocol (TFTP), và game trực tuyến.

UDP thích hợp với rất nhiều ứng dụng dựa vào một số đặc điểm được mô tả chi tiết hơn như sau:

- ✓ Không cần thiết lập kết nối (No connection establishment): UDP không yêu cầu quá trình thiết lập kết nối như TCP, do đó nó không làm chậm quá trình truyền dữ liệu. Đó là lý do tại sao DNS lại chạy nhanh hơn khi sử dụng UDP (DNS có thể chạy cả trên TCP lẫn UDP).
- ✓ Không cần lưu giữ trạng thái kết nối (No connection state): UDP không cần lưu giữ các thông tin về trạng thái hoạt động của kết nối như TCP (thí dụ: thông số gửi và nhận gói tin, ACK, sequence number, …), do đó tiêu tốn ít tài nguyên hệ thống hơn so với TCP, giúp các server có thể phục vụ nhiều client hơn.
- ✓ Tổng phí (overhead) cho phần tiêu đề các gói tin nhỏ hơn: trong khi header của TCP có kích thước 20 bytes thì header của UDP chỉ có 8 bytes, làm cho gói tin UDP nhỏ hơn và có thể truyền đi nhanh hơn.
- ✓ Tốc độ gửi không được điều hòa (Unragulated send rate): TCP có cơ chế điều tiết tốc tộ truyền khi gặp những đường truyền hỏng hay khi mạng bắt đầu bị tắc nghẽn, cơ chế này không thích hợp cho những ứng dụng thời gian thực (có thể chấp nhận một tỉ lệ mất gói tin nhất định, không cần phát lại để đảm bảo tính kịp thời). Trong khi đó, tốc độ phát của thực thể giao thức UDP chỉ phụ thuộc vào tốc độ gửi của ứng dụng sử dụng UDP để truyền chứ không phụ thuộc vào mạng có bị tắc nghẽn (congestion) hay không. Do đó ứng dụng có thể áp dụng các cơ chế khác nhau theo yêu cầu của nó.

1.2 Khái niệm hệ thống thời gian thực, multimedia, QoS và đảm bảo QoS

1.2.1 Hệ thống thời gian thực

[4] Là hệ thống mà trong đó sự đúng đắn của việc thực hiện các thao tác không chỉ phục thuộc vào việc thu được kết quả đúng mà còn phải đưa ra kết quả đúng thời điểm. Điều đó có nghĩa là tính đúng đắn của hệ thống thời gian thực không chỉ phục thuộc vào kết quả logic của thao tác mà còn phụ thuộc vào thời điểm tạo ra các kết quả. Giới hạn về thời gian mà các kết quả được đưa ra gọi là thời hạn kết thúc (deadline), đó là thời điểm muộn nhất có thể chấp nhận được để kết thúc một thao tác.

Đặc điểm của hệ thống thời gian thực:

- Các sự kiện bên trong và bên ngoài có thể xảy ra một cách định kỳ hoặc tự phát.
- Sự đúng đắn của hệ thống còn phụ thuộc cả vào việc đáp ứng các ràng buộc thời gian

1.2.2 Truyền thông đa phương tiện (multimedia)

Hệ thống truyền thông đa phương tiện là hệ thống cung cấp tích hợp các chức năng lưu trữ, truyền dẫn và trình diễn các kiểu phương tiện mang tin rời rạc (văn bản, đồ hoạ, ảnh…) và liên tục (audio, video) trên máy tính.

Thuật ngữ media: phương tiện truyền đạt thông tin (thí dụ: văn bản, âm thanh, hình ảnh); còn multimedia có nghĩa là truyền thông tin bằng nhiều phương tiện truyền đạt một cách đồng thời.

Các kiểu media trong hệ thống đa phương tiện gồm:

- Media độc lập với thời gian: thông tin không liên quan gì đến việc định thời luồng dữ liệu, ví dụ như văn bản, đồ hoạ, ảnh.
- Media phụ thuộc thời gian: thông tin có quan hệ chặt chẽ với thời gian, phải được trình diễn trước người sử dụng vào những thời điểm xác định. Ví dụ: animation (phim hoạt hoạ), audio (âm thanh), video, game online (trò chơi trực tuyến).

Hệ thống đa phương tiện cũng là hệ thống thời gian thực.

Các ứng dụng đa phương tiện (multimedia) là loại ứng dụng nhạy cảm với độ trễ, tuy nhiên chúng lại cho phép sự mất mát gói tin ở một mức độ nào đó. Như vậy so với các ứng dụng truyền thống thì tính chất của nó hoàn toàn ngược lại: chấp nhận mất mát nhưng yêu cầu độ trễ nhỏ, trong khi các ứng dụng truyền thống thì cho phép độ trễ lớn và không chấp nhận mất mát dữ liệu.

Các ứng dụng đa phương tiện có thể chia làm 3 lớp lớn như sau [4]:

1.2.2.1 Truyền audio và video đã được lưu trữ:

Đối với các ứng dụng loại này, người dùng tại các máy trạm truy cập đến các files audio, video… được lưu trữ sẵn trên các máy server. Audio có thể là các bài hát… Video có thể là những bộ phim…

Trong hầu hết các ứng dụng loại này, sau một thời gian trễ vài giây, các máy trạm có thể chạy được các file trong khi chúng vẫn tiếp tục nhận phần còn lại từ server. Gần như tất cả các website phim đều lưu ý người dùng, đợi load khoảng 10s thì bắt đầu xem. Nhiều ứng dụng còn cho phép tính năng tương tác với người dùng: cho phép người dùng pause, play, next, previous. Yêu cầu đối với độ trễ và sự biến thiên độ trễ là không chặt chẽ bằng ở trong ứng dụng thời gian thực như điện thoại Internet, video conference thời

gian thực, live stream... Các chương trình dùng để chạy các file audio/video được lưu trữ trên mạng hiện nay như: Windows Media Player, MPC…

1.2.2.2 Truyền audio và video thời gian thực:

Các ứng dụng cho phép người dùng nghe/xem được các chương trình phát thanh/truyền hình từ bất kỳ nơi nào trên thế giới. Chẳng hạn người dùng có thể nghe đài phát từ Anh, các kênh truyền hình trên khắp thế giới từ bất kỳ máy nào kết nối Internet. Đặc trưng của lớp ứng dụng này là nhiều người có thể đồng thời nhận được cùng một chương trình audio/video. Các ứng dụng này không cho phép tương tác người dùng. Cũng như lớp ứng dụng truyền audio/video được lưu trữ, độ trễ các ứng dụng loại này cho phép tối đa là 10s [12].

1.2.2.3 Ứng dụng tương tác audio và video thời gian thực:

Lớp ứng dụng này cho phép nhiều người dùng sử dụng audio/video để tương tác với nhau trong thời gian thực. Các ứng dụng cho audio, video thời gian thực ngày nay rất đa dạng như: voice chat, video call trong Facebook, Skype, các trang live stream... Trong các ứng dụng tương tác audio/video thời gian thực thì yêu cầu độ trễ nhỏ hơn vài trăm miligiây. Với âm thanh, độ trễ tốt nhất là nên nhỏ hơn 150 ms, với độ trễ từ 150-400ms thì có thể chấp nhận được, còn lớn hơn 400 ms thì không thể chấp nhận được [12].

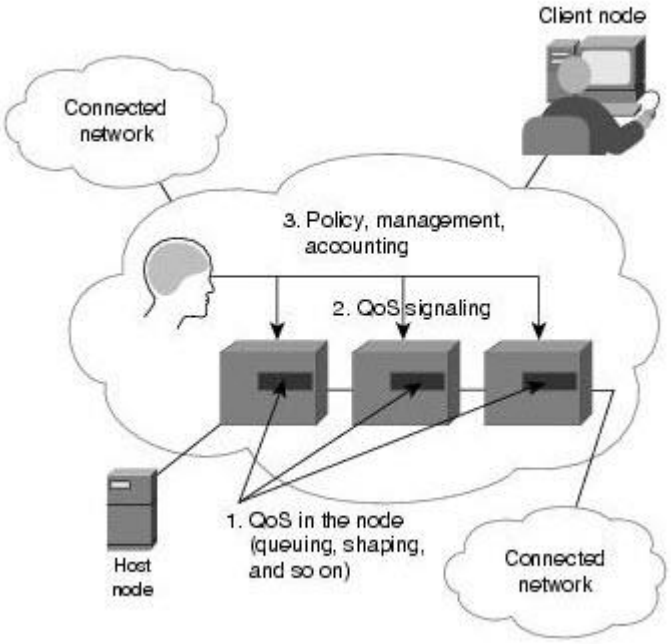
1.2.3 Khái niệm QoS và đảm bảo QoS

[4, 10] Các hệ thống đa phương tiện thường là phân tán, hoạt động trên mạng máy tính, các yêu cầu tài nguyên cho sự hoạt động của hệ thống thường là động, thí dụ trong ứng dụng Video conference thì yêu cầu về tài nguyên phụ thuộc số người tham gia. Do đó, cần có các giải pháp để đảm bảo chất lượng các dịch vụ (đảm bảo QoS) của ứng dụng thoả mãn yêu cầu của người dùng. Chất lượng dịch vụ là khái niệm khó định nghĩa chính xác, khái niệm của nó phụ thuộc vào góc nhìn, nhu cầu, yêu cầu của từng người sử dụng dịch vụ đó. Người dùng muốn chất lượng dịch vụ mình nhận được là như thế nào, ví dụ như: nghe nhạc, xem video, video call, live stream… thì người dùng chấp nhận hình nhòe ở mức nhất định, nhưng đảm bảo được độ trễ về truyền thông... Còn với những người dùng mạng với nhu cầu truyền dữ liệu không yêu cầu cao về thời gian, thì lúc này yêu cầu về việc gói tin được đảm bảo lại được đặt lên hàng đầu. Theo [10] định nghĩa thì **"QoS (Quality of Service) đề cập đến khả năng cung cấp dịch vụ của một mạng cho một lưu lượng mạng được lựa chọn nào đó qua các công nghệ khác nhau"**. Mục đích chính của QoS là điều khiển độ trễ và biến thiên độ trễ, giảm tỷ lệ mất mát gói tin cho các luồng lưu lượng của các ứng dụng thời gian thực và tương tác. Một điều quan trọng nữa là nó cung cấp quyền ưu tiên cho một hoặc một vài luồng trong khi

vẫn đảm bảo các luồng khác (có quyền ưu tiên thấp hơn) không mất quyền được phục vụ.

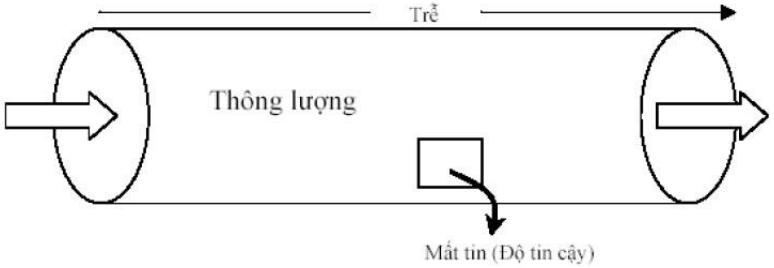
Đảm bảo chất lượng dựa trên cơ sở là quản lý tài nguyên vì QoS phụ thuộc vào tài nguyên khả dụng của hệ thống, việc quản lý tài nguyên ở đây là:

- Tính toán, ước lượng được hiệu năng sử dụng tài nguyên
- Dành tài nguyên cho dịch vụ
- Lập lịch truy cập tài nguyên



Hình 1.3: Kiến trúc cơ bản của QoS [10]

1.2.3.1 Các tham số QoS chính [4]



Hình 1.4: Các tham số QoS chính

Hình 1.4 minh hoạ các tham số QoS chính, bao gồm: độ trễ, thông lượng, tỷ lệ mất tin, jitter (biến thiên độ trễ).

- **Độ trễ:** là thời gian để truyền một gói tin từ nguồn đến đích, bao gồm thời gian phát một gói tin lên đường truyền, thời gian xử lý tại hàng đợi, thời gian xếp hàng

chờ tại hàng đợi và thời gian truyền tín hiệu trên đường truyền; nó phụ thuộc vào tốc độ truyền tin, tốc độ truyền tin càng lớn, độ trễ càng nhỏ và ngược lại. Trong các thành phần độ trễ nói trên, thời gian chờ thường biến động trọng một miền rất rộng, nó quyết định mức độ biến động trễ (jitter).

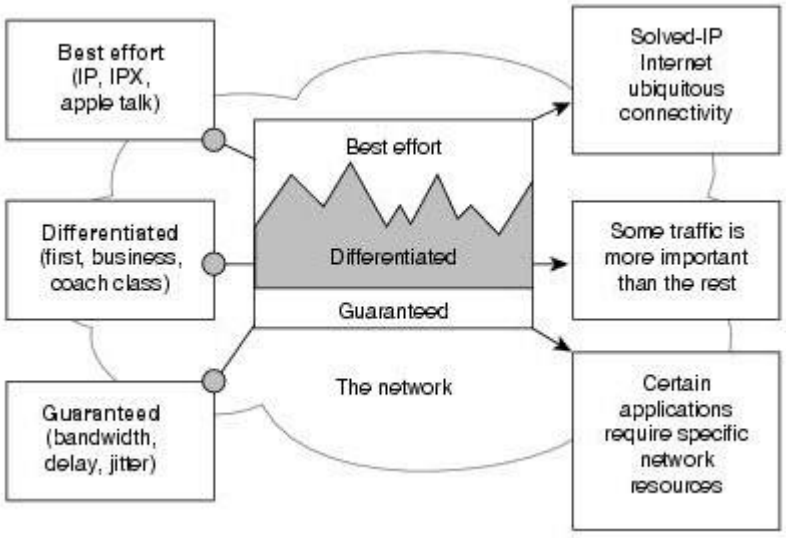
- **Thông lượng:** thông lượng quyết định khả năng truyền tin, thông lượng được tính bằng tổng số đơn vị dữ liệu được truyền trong 1 đơn vị thời gian, chẳng hạn số gói tin hoặc số bytes dữ liệu truyền đi trong 1s.
- **Tỷ số mất tin:** là số đơn vị dữ liệu bị mất cực đại trong một đơn vị thời gian.
- **Jitter:** là sự biến thiên của độ trễ.
- Ngoài ra còn có khái niệm **kích thước mất tin**: đó là số gói tin bị mất liên tiếp cực đại. Bên cạnh tỷ số mất tin ta có thể dùng khái niệm **độ tin cậy**: tỷ số mất tin tỷ lệ nghịch với độ tin cậy

Một số tham số khác:

- **Băng thông (bandwidth):** Băng thông biểu thị tốc độ truyền dữ liệu cực đại có thể đạt được giữa hai điểm kết nối hay là số lượng bit trên giây mà mạng sẵn sàng cung cấp cho các ứng dụng. Nếu có băng thông đủ lớn thì các vấn đề như nghẽn mạch, kỹ thuật lập lịch, phân loại, trễ… nói chung chúng ta sẽ không phải quan tâm nhiều, nhưng điều này khó xảy ra vì băng thông của mạng là có hạn. Khi được sử dụng như một tham số của QoS, băng thông là yếu tố tối thiểu mà một ứng dụng cần có để hoạt động được.

1.2.3.2 Các mức chất lượng dịch vụ - QoS [4]

Mức chất lượng dịch vụ (mức QoS) chỉ khả năng thực sự của QoS, đó là khả năng của mạng trong việc phân phát dịch vụ cho một lưu lượng mạng cụ thể. Các mức dịch vụ khác nhau theo mức độ nghiêm ngặt khác nhau của QoS, được đặc tả bởi sự giới hạn về dải thông, độ trễ, jitter và khả năng mất tin.



Hình 1.5: Ba mức QoS trong mạng không đồng nhất [10]

Có ba mức QoS trong mạng không đồng nhất [10]:

- Dịch vụ cố gắng tối đa (**Best-effort service)**: đây là dịch vụ truyền không đảm bảo, được đặc trưng bởi việc sử dụng hàng đợi FIFO, không có sự phân loại mức ưu tiên giữa các luồng.
- Dịch vụ có phân loại (**Differentiated services) - QoS mềm.** Một luồng nào đó được phục vụ tốt hơn những luồng khác (xử lý nhanh hơn, được cấp nhiều băng thông hơn, tỷ lệ mất tin giảm xuống).
- Dịch vụ đảm bảo (**Guaranteed service) - QoS cứng.** Đó là sự đặt trước tài nguyên mạng cho một luồng lưu lượng cụ thể.

Đới với từng trường hợp cụ thể ta sẽ triển khai các dịch vụ khác nhau. Các yếu tố được dùng để lựa chọn dịch vụ triển khai trong mạng:

- **Ứng dụng hoặc bài toán cần giải quyết:** Mỗi loại dịch vụ nêu trên thích hợp cho một loại ứng dụng cụ thể. Người dùng không nhất thiết phải sử dụng dịch vụ đảm bảo khi ứng dụng của họ không yêu cầu đến mức đó. Một dịch vụ phân loại – hoặc thậm chí là cố gắng tối đa – có thể phù hợp, tuỳ theo yêu cầu của ứng dụng người dùng.
- **Chi phí cho cài đặt và triển khai dịch vụ:** dịch vụ đi kèm với chi phí. Do đó phải cân đối được chất lượng và giá thành.

1.3 Dịch vụ cố gắng tối đa (Best Effort) và truyền thông đa phương tiện

[4] Dịch vụ cố gắng tối đa là dịch vụ phổ biến trên mạng Internet và mạng IP nói chung. Các gói tin được phục vụ theo chiến lược: đến trước sẽ được phục vụ trước, không cần quan tâm gói tin đó đến từ đâu hay dịch vụ của nó là gì. Đặc điểm của dịch vụ cố gắng tối đa là nó không từ chối việc đưa vào mạng bất kỳ một lưu lượng nào, tất cả mọi lưu lượng đến đều được nó đối xử như nhau, mạng chỉ đảm bảo một điều duy nhất là lưu lượng sẽ được truyền đi bằng một cách tốt nhất mà nó có thể nếu có đủ tài nguyên, nghĩa là không làm sinh thêm độ trễ nhân tạo và không làm mất gói tin một cách không cần thiết. Điều này dẫn đến việc nó rất khó đáp ứng được nhu cầu của những dịch vụ đòi hỏi độ trễ thấp như các dịch vụ trong truyền thông đa phương tiện. Là một vấn đề gây cản trở rất lớn cho sự phát triển các ứng dụng đa phương tiện trên Internet. Ở thời điểm hiện tại, đa phần các dịch vụ được nhà mạng cung cấp vẫn sử dụng nguyên tắc "cố gắng tối đa" này.

Với mạng ngày nay, các ứng dụng truyền thông đa phương tiện đã được cải thiện đáng kể như băng thông của mạng ngày càng nhiều. Tuy nhiên nó vẫn có nhiều hạn chế,

đặc biệt là khi trong mạng xảy ra tắc nghẽn, độ trễ bị tăng đến mức không thể chấp nhận được.

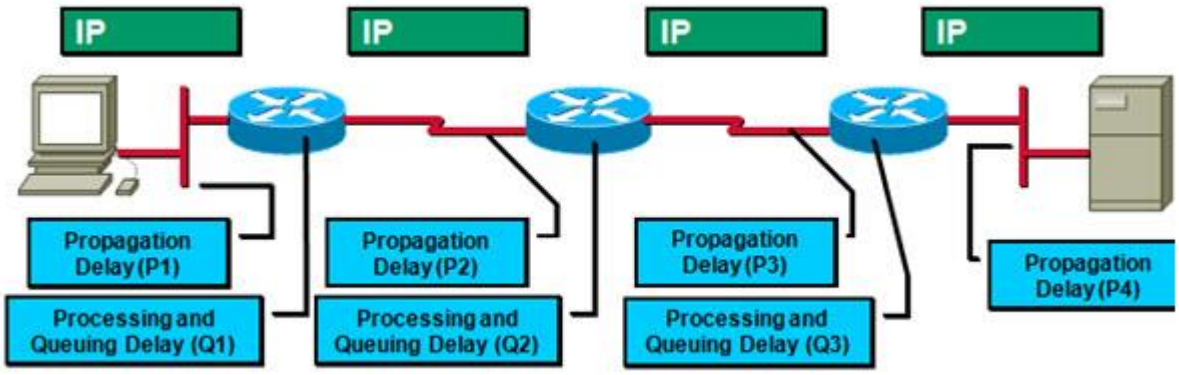
1.3.1 Hạn chế của dịch vụ cố gắng tối đa

1.3.1.1 Tỉ lệ mất mát gói tin có thể là rất lớn khi xảy ra tắc nghẽn

Ví dụ một gói tin UDP được tạo ra bởi ứng dụng thoại Internet. Nó được đóng gói trong một gói tin IP và gói tin IP chuyển tới bên nhận. Gói tin được truyền trên mạng qua các bộ đệm trong các router. Nếu một trong các bộ đệm của router đã đầy thì gói tin sẽ không được vào hàng đợi. Trong trường hợp này, gói tin bị loại bỏ và nó sẽ không tới được phía nhận.

Nếu thay bằng TCP, TCP có cơ chế biên nhận nên sẽ truyền lại các gói tin bị mất. Vấn đề là với các ứng dụng thời gian thực như thoại Internet, hay các ứng dụng video call, việc truyền lại là không chấp nhận. Chưa nói đến vấn đề với cơ chế điều khiển tắc nghẽn trong TCP, sau khi gói tin bị mất, tốc độ phát tại phía gửi có thể giảm tới mức thấp nhất, nhằm tránh tắc nghẽn, điều này ảnh hưởng nghiêm trọng tới chất lượng âm thanh tại phía nhận. Vì thế, hầu hết các ứng dụng thoại Internet đều chạy trên UDP và không thực hiện truyền lại các gói tin bị mất, do đó để đảm bảo được chất lượng dịch vụ, việc mất tin phải ở mức thấp và chấp nhận được, độ trễ cũng phải chấp nhận được. Cơ chế sửa lỗi FEC (Forward Error Correction) có thể được dùng để che đậy sự mất gói tin. Tuy nhiên, nếu đường truyền giữa bên gửi và bên nhận bị tắc nghẽn trầm trọng, tỉ lệ mất gói tin vượt quá 10-20%, khi đó sẽ không có cách nào đạt được chất lượng âm thanh mong muốn. Đây là hạn chế của dịch vụ cố gắng tối đa [12].

1.3.1.2 Độ trễ end-to-end có thể vượt quá giới hạn chấp nhận được



Hình 1.5: Độ trễ end-to-end

Độ trễ end-to-end là tổng của thời gian xử lý và chờ trong hàng đợi của các router dọc theo đường truyền từ người gửi đến người nhận, thời gian truyền và thời gian xử lý của phía nhận.

Delay = P1 + Q1 + P2 + Q2 + P3 + Q3 + P4

- Propagation (serialization): Sự chậm trễ truyền tải dữ liệu trên phương tiện truyền dẫn, hầu hết xảy ra ở các phần, chỉ phụ thuộc vào băng thông.
- Processing, queuing delay: Xảy ra trong Router.

Với các ứng dụng tương tác thời gian thực như điện thoại Internet, độ trễ end-to-end nhỏ hơn 150ms được coi là không có vấn đề gì (giác quan con người không cảm nhận được sự khác biệt), độ trễ từ 150-400ms là có thể được chấp nhận được nhưng không nên lớn đến mức như vậy, độ trễ lớn hơn 400ms là quá lớn, không thể chấp nhận được.

1.3.1.3 Jitter là không thể tránh khỏi [12]

Một trong những thành phần tạo nên độ trễ end-to-end là thời gian chờ ngẫu nhiên ở hàng đợi của router. Do thời gian chờ ngẫu nhiên này, độ trễ end-to-end có thể thay đổi đối với từng gói tin, sự biến đổi này được gọi là ***jitter*** (biến thiên độ trễ). Có thể loại bỏ jitter bằng các cách sau: đánh số số tuần tự các gói tin, gán nhãn thời gian cho các gói tin, tạm dừng chạy.

Sau đây chúng ta sẽ xem xét kỹ hơn các phương pháp này:

1.3.2 Tổng quan các phương pháp đảm bảo QoS cho truyền thông multimedia trên nền các dịch vụ Best Effort

1.3.2.1 Loại bỏ jitter ở phía nhận [4, 12]

Với một ứng dụng tiếng nói như điện thoại Internet…, phía nhận cố gắng cung cấp khả năng chạy đồng bộ các đoạn khi có jitter. Điều này có thể được thực hiện bằng cách kết hợp ba cơ chế sau:

- Chèn một số tuần tự vào mỗi đoạn, phía gửi sẽ tăng số tuần tự lên 1 khi có một gói tin được tạo ra.
- Gắn thêm một nhãn thời gian (timestamp) cho mỗi đoạn; phía gửi sẽ gán nhãn cho mỗi đoạn chính là thời điểm mà đoạn đó được tạo ra.
- Làm trễ việc chạy ở phía nhận. Thời gian làm trễ phải đủ lâu để các gói tin phải được nhận trước khi tới lượt (thứ tự) chúng được lập lịch chạy. Việc làm trễ có thể theo khoảng thời gian cố định hoặc thay đổi. Các gói tin không tới được phía nhận trước thứ tự chạy của chúng được xem như là bị mất.

Có 2 phương pháp làm trễ việc chạy ở phía nhận: **tạm dừng cứng** và **tạm dừng thích nghi.**

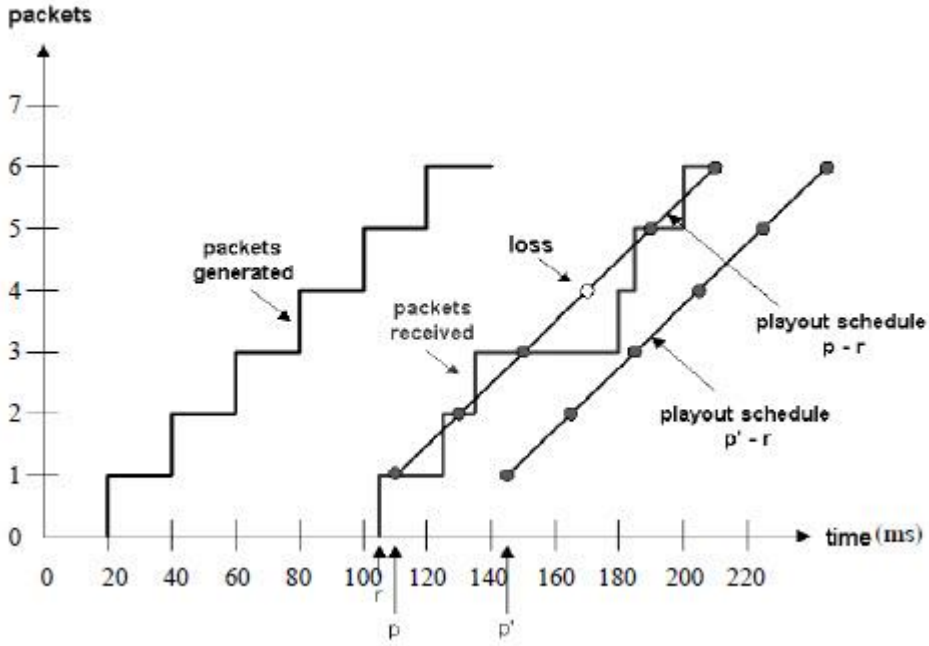
- **Tạm dừng cứng (fixed playout delay)**

Phía nhận thực hiện chạy mỗi đoạn đúng q ms sau khi đoạn đó được tạo ra. Nếu timestamp của một đoạn là t, phía nhận sẽ chạy đoạn đó tại thời điểm t + q (giả sử đoạn

tới phía nhận trước thời điểm theo thứ tự nó được chạy), nếu đoạn có timestamp t đến sau thời điểm t+q, nó bị coi như đã mất.

Trong chiến lược này, số tuần tự là không cần thiết. Việc quan trọng là lựa chọn được giá trị q tốt nhất. Như phần 1.3.1.2 đã trình bày, với ứng dụng điện thoại internet độ trễ cho phép lên tới 400 ms, còn đối với các ứng dụng audio tương tác thì yêu cầu q nhỏ hơn.

- Nếu tăng q thì tỉ lệ mất gói tin giảm (tốt), nhưng QoS giảm điều này chúng ta cần phải cân nhắc
- Nếu độ trễ E2E lớn thì dùng q lớn, nếu độ trễ nhỏ và jitter nhỏ thì dùng q nhỏ
- Nếu q << 400 ms, rất nhiều gói tin có thể bị coi như là bị mất, dù vẫn đến đích.



Hình 1.6: Mối quan hệ giữa thời gian tạm dừng và sự mất mát gói tin

Hình 1.6 minh họa mối quan hệ giữa thời gian tạm dừng và sự mất mát gói tin (gói tin bị coi là mất khi nó tới phía nhận sau thời điểm được chạy). Hình vẽ thể hiện thời gian mà tại đó gói tin được tạo và được chạy. Hai lựa chọn khác nhau về thời gian chạy được xét.

- Đường bậc thang bên trái: thời gian các gói tin được tạo ra và gửi đi, cách nhau 20ms.
- Đường bậc thang ở giữa: Trường hợp làm trễ với thời gian ngắn (= p-r), gói tin thứ 4 tới phía nhận sau thời gian được lập lịch nên bị coi là mất.
- Đường bậc thang bên phải: Trường hợp làm trễ với thời gian dài hơn (= p'-r) (*p' > p*): toàn bộ các gói tin đều tới phía nhận trước thời gian được lập lịch và được chạy.

- **Làm trễ thích nghi (adaptive playout delay)**

Với chiến lược tạm dừng cứng nêu ở trên, khi ta thiết lập một thời gian tạm ngừng lớn thì gần như tất cả các gói tin đều được gửi tới người nhận mà không xảy ra sự mất mát gói tin nào, tùy nhiên điều này sẽ làm cho độ trễ là lớn. Với các ứng dụng thoại internet thì độ trễ cao là điều không thể chấp nhận được, vì vậy cần có những biện pháp để để giảm tối đa thời gian trễ và vẫn đảm bảo sự mất mát gói tin là chấp nhận được.

Để giải quyết vấn đề trên, ta phải ước lượng được độ trễ và jitter, từ đó điều chỉnh thời gian làm trễ tại lúc bắt đầu mỗi cuộc hội thoại. Sau đây là thuật toán mà phía nhận dùng để hiệu chỉnh thời gian làm trễ (playout delay) theo sự thay đổi của độ trễ và jitter.

Gọi t_i là timestamp (thời điểm gói tin được sinh ra) của gói tin thứ i, r_i là thời điểm gói tin thứ i tới phía nhận, p_i là thời điểm gói tin thứ i được chạy ở phía nhận. Khi đó, độ trễ end-to-end của gói tin thứ i là $r_i – t_i$ (giá trị này có thể được thay đổi với tùy gói tin). Gọi d_i là độ trễ trung bình cho tới khi nhận được gói tin thứ i, d_i được tính theo công thức:

$d_i = (1-u)\, d_{i-1} + u\,(r_i - t_i)$, với u là một hằng số (ví dụ u = 0.01).

Như vậy d_i chính là giá trị trung bình đã được làm trơn của các độ trễ $r_1 – t_1$, $r_2 – t_2$,.... $r_i - t_i$. Gọi v_i là độ lệch trung bình giữa độ trễ và độ trễ trung bình, v_i được tính theo công thức:

$v_i = (1-u)\, v_{i-1} + u\,|\, r_i - t_i - d_i \,|$

d_i, v_i được tính cho mỗi gói tin nhận được, mặc dù chúng chỉ được dùng để tính thời điểm chạy cho gói tin đầu tiên của mỗi đoạn hội thoại.

Sau khi tính được các ước lượng này, phía nhận sẽ dùng thuật toán sau để tính thời điểm chạy cho các gói tin: Nếu gói tin i là gói tin đầu tiên của một đoạn hội thoại, thời điểm chạy của nó được tính theo công thức: $p_i = t_i + d_i + K*v_i$ trong đó K là hằng số dương (ví dụ K = 4). $K*v_i$ được thêm vào để thiết lập thời gian chạy đủ lâu để chỉ một phần nhỏ các gói tin bị coi là mất do đến muộn. Thời điểm chạy của các gói tin trong đoạn được tính bằng timestamp của nó cộng với khoảng thời gian từ khi gói đầu tiên được sinh ra tại phía gửi đến lúc nó được chạy tại phía nhận. Cụ thể như sau: Đặt $q_i = p_i - t_i$ là thời gian từ khi gói tin đầu tiên được tạo ra (bên gửi) cho tới khi nó được chạy (bên nhận). Khi đó gói tin thứ j trong đoạn sẽ được chạy tại thời điểm $pj = tj + q_j$.

Trong thuật toán trên ta đã giả sử rằng phía nhận biết được gói tin nào là gói tin đầu tiên được tạo ra. Nếu không có sự mất mát gói tin, phía nhận có thể xác định được điều này bằng cách so sánh timestamp của gói tin thứ i với timestamp của gói tin thứ i-1. Nếu $t_i - t_{i-1} > 20$ms, thì phía nhận sẽ biết rằng gói tin thứ i là gói tin đầu tiên của đoạn hội

thoại. Nhưng việc mất gói tin là hoàn toàn có thể xảy ra. Trong trường hợp này, 2 gói tin nhận được trong cùng một đoạn cũng có hiệu 2 timestamp lớn hơn 20ms. Khi đó, số tuần tự trở nên hữu ích. Phía nhận có thể dùng số tuần tự để xác định khi hiệu hai timestamp > 20 ms, đâu là trường hợp do gói tin đầu tiên được phát ra, đâu là trường hợp do đã có sự mất mát gói tin. Cụ thể là: nếu $t_i - t_{i-1} > 20$ms, và hai gói tin có số tuần tự liên tiếp nhau thì gói tin thứ i là gói tin đầu tiên của đoạn hội thoại, ngược lại nếu hai gói tin không liên tiếp thì chắc chắn là có sự mất mát gói tin.

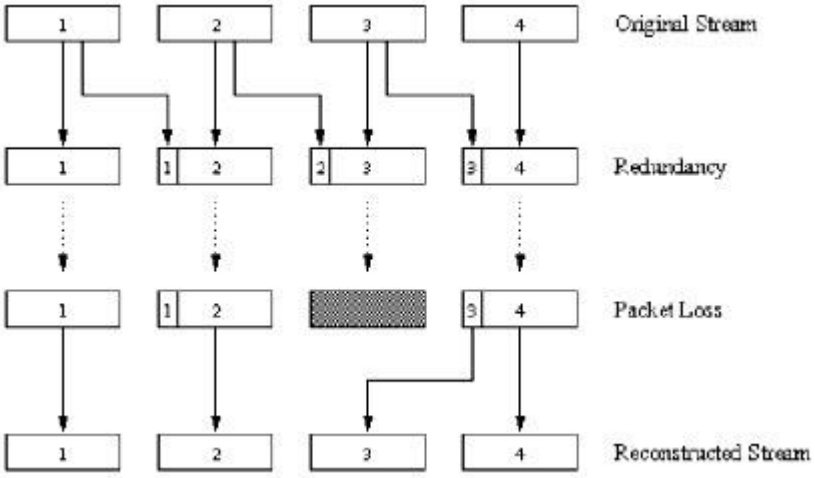
1.3.2.2 Khôi phục các gói tin bị mất

Như chúng ta đã nói ở phần 1.3 việc truyền lại gói tin bị mất là không phù hợp với ứng dụng thời gian thực như điện thoại Internet, vì có thực hiện truyền lại gói tin thì khi đến đích cũng đã quá hạn (deadline), gói tin đến sau thời điểm này là không có ý nghĩa. Có 2 kiểu lược đồ lường trước mất mát là: **sửa lỗi ở phía trước - FEC** và **xen kẽ (interleaving)** [4, 12].

- **Sửa lỗi ở phía trước - FEC**

Ý tưởng của FEC là thêm các thông tin bổ sung vào gói tin ban đầu, các thông tin này có thể được dùng để khôi phục các gói tin bị mất. Hiện nay người ta dùng 2 cơ chế FEC. Cơ chế thứ nhất: gửi một đoạn chứa thông tin bổ sung được mã hoá sau *n* đoạn. Đoạn chứa thông tin bổ sung đó thu được bằng cách thực hiện phép XOR *n* đoạn ban đầu. Theo cơ chế này, nếu một gói tin nào đó trong *n* + 1 gói tin bị mất, phía nhận có thể khôi phục hoàn toàn lại gói tin đó. Nhưng nếu có hai hay nhiều gói tin bị mất, phía nhận sẽ không thể tái tạo lại được. Nếu để cỡ của khối đó (tức là *n* + 1) nhỏ, một phần lớn các gói tin bị mất có thể được khôi phục. Tuy nhiên, cỡ khối càng nhỏ, băng thông đường truyền càng phải được tăng lên. Cụ thể, băng thông đường truyền tăng theo hệ số 1/n. Ví dụ nếu *n* = 4, băng thông phải được tăng 25%. Lược đồ này cũng làm tăng độ trễ bởi vì phía nhận phải nhận toàn bộ khối gói tin xong mới có thể thực hiện chạy chúng được. Cơ chế FEC thứ hai là gửi kèm một luồng âm thanh chất lượng thấp hơn luồng ban đầu như là luồng thông tin bổ sung (xem hình 1.3). Chẳng hạn, phía gửi tạo một luồng bình thường (được mã hóa PCM với tốc độ 64Kbps) và một luồng tốc độ bit thấp (được mã hóa GSM, có tốc độ 13 Kbps). Phía gửi sẽ tạo ra gói tin thứ *n* bằng cách lấy đoạn thứ *n* của luồng bình thường và đoạn thứ *n*-1 của luồng bổ sung. Khi có sự mất gói tin không liên tục, phía nhận có thể chạy đoạn mã hóa tốc độ bit thấp trong gói tin kế tiếp. Hiển nhiên là đoạn tốc độ bit thấp cho chất lượng âm thanh thấp hơn đoạn bình thường, nhưng trên tổng thể gồm nhiều đoạn tốc độ cao, một vài đoạn tốc độ thấp và không có sự mất mát các đoạn thì vẫn cho chất lượng âm thanh tốt. Trong lược đồ này, phía nhận chỉ phải nhận 2 gói tin trước khi thực hiện chạy chúng, vì thế phần độ trễ tăng thêm là nhỏ. Hơn

nữa, nếu mã hóa tốc độ bit thấp chiếm ít hơn nhiều so với mã hóa bình thường, tốc độ truyền tăng không đáng kể.

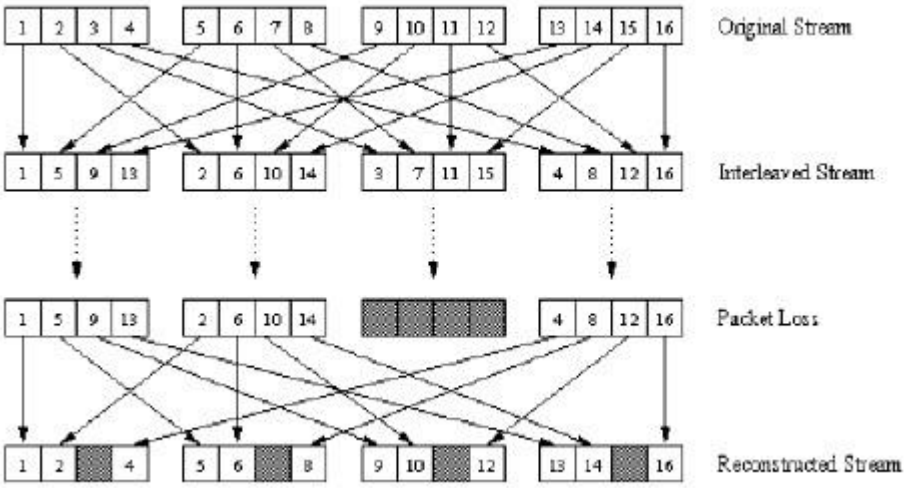


Hình 1.3: Cơ chế thứ 2 của FEC

Để xử lý sự mất gói tin liên tiếp, một biến thể của cơ chế trên có thể được sử dụng. Thay vì chèn đoạn tốc độ bit thấp thứ *n*-1 vào đoạn bình thường thứ *n*, phía gửi sẽ chèn đoạn tốc độ bit thấp thứ *n*-1 và *n*-2 , hoặc chèn đoạn tốc độ bít thấp thứ *n*-1 và thứ *n*-3. Việc chèn càng nhiều đoạn bổ sung vào giúp cho các ứng dụng có thể đáp ứng được trong nhiều môi trường best-effort, tuy nhiên nó cũng làm tang băng thông cần sử dụng và làm tăng độ trễ.

- **Cơ chế xen kẽ (Interleaving)**

Ứng dụng điện thoại Internet có thể gửi âm thanh theo kiểu xen kẽ, phía gửi sẽ xếp lại thứ tự các phần tử dữ liệu âm thanh trước khi truyền, các phần tử sát nhau sẽ được xếp cách một khoảng cố định trong luồng truyền đi. Cơ chế này có thể làm giảm ảnh hưởng của việc mất gói tin. Việc mất gói tin sẽ chỉ làm cho mỗi đoạn nhận được sau khi được khôi phục lại giảm chất lượng đi chút ít. Hình 1.4 minh hoạ cho cơ chế này. Mỗi đoạn chứa 4 phần tử, đoạn đầu tiên chứa các phần tử thứ: 1, 5, 9, 13; đoạn thứ 2 chứa phần tử thứ: 2, 6, 10, 14,.. Giả sử đoạn thứ 3 bị mất, các đoạn sau đó được tạo lại và mỗi đoạn bị mất đi một phần tử (phần tử thứ 3); chất lượng mỗi đoạn thu được chỉ bị giảm đi chút ít.



Hinh 1.4: Cơ chế xen kẻ

Cơ chế trên có thể cải thiện đáng kể chất lượng luồng nhận được. Bất lợi dễ thấy là tăng độ trễ bởi vì tất cả các đoạn cần phải được tạo lại trước khi chạy. Đây là hạn chế của nó khi áp cho các ứng dụng tương tác thời gian thực như điện thoại Internet; tuy nhiên nó thực hiện tốt cho các ứng dụng truyền audio/video được lưu trữ trên server. Thuận lợi là nó không yêu cầu tăng băng thông.

1.4 Hiệu năng và Đánh giá hiệu năng mạng

Hiệu năng là một vấn đề vô cùng quan trọng trong hệ thống nói chung và trong mạng máy tính nói riêng. Chúng ta có thể kiểm soát và quản lý hiệu năng trong mạng tốt và đơn giản với một mạng nhỏ, nhưng với một hệ thống mạng lớn thì việc đánh giá hiệu năng và đảm bảo được hiệu năng cho mạng lại vô cùng khó khăn và phức tạp. Việc đánh giá hiệu năng phải được tiến hành từ khi xây dựng giải pháp hệ thống, trong quá trình lắp đặt, cho đến khi triển khai hệ thống cho khách hàng [2, 3].

1.4.1 Hiệu năng

Khái niệm của hiệu năng có thể được diễn đạt theo nhiều cách khác nhau tuỳ theo góc độ nghiên cứu. Ví dụ như hiệu năng là một độ đo mức độ thực hiện công việc của một hệ thống. Đối với một hệ thống tính toán, đánh giá hiệu năng là xác định về mặt định tính và định lượng chất lượng phục vụ của hệ thống đó đối với một loại bài toán nhất định. Hiệu năng chủ yếu được xác định bởi sự kết hợp của các nhân tố:

- Tính sẵn sàng (availability).
- Thông lượng (throughput)
- Thời gian đáp ứng (response time).

Đối với mạng máy tính, hiệu năng còn được xác định dựa trên các nhân tố khác nữa như độ trễ (delay), độ tin cậy (reliability), tỉ lệ lỗi (error rate)... [3]

1.4.2 Các phương pháp đánh giá hiệu năng mạng

- **Mô hình giải tích**

Dựa trên các tính toán toán học để đánh giá hiệu năng của một mạng. Ưu điểm cua nó là chi phí thực hiện thấp nhất, bởi vì khi muốn tính hiệu năng với các tham số khác ta chỉ cần thay đổi các tham số hệ thống và cấu hình mạng với một chi phí rất thấp (chỉ cần thực hiện lại các thao tác tính toán đã xây dựng trước). Nhược điểm của phương pháp này là các bài toán khi đánh giá ta phải đơn giản hóa nó đi bằng các giả thiết thích hợp, hoặc tách bài toán thành các nhiều cấp nhỏ hơn. Các kết quả thu được thường không phản ánh chính xác thực tế, cho nên phương pháp này thường chỉ được sử dụng ngay trong giai đoạn đầu của việc thiết kế mạng, giúp cho người thiết kế dự đoán được các giá trị giới hạn của hiệu năng. Ngoài ra, các kết quả của phương pháp này bắt buộc phải được kiểm nghiệm bằng các phương pháp khác, như mô phỏng hoặc đo (được trình bày dưới đây).

- **Mô phỏng**

Trong việc đánh giá hiệu năng mạng, mô phỏng là một kỹ thuật sử dụng máy tính để thực hiện các thí nghiệm về mạng. Với cách này chúng ta có thể theo dõi được sự hoạt động của mạng theo tuần tự thời gian thời gian. Đối với phương pháp đánh giá hiệu nay này, việc quan trọng nhất là xây dựng được bộ mô phỏng. Việc này đòi hỏi nhiều công sức và thời gian. Độ lớn của bộ mô phỏng tuỳ thuộc vào độ phức tạp của thí nghiệm mô phỏng. Nó thường được xây dựng có cấu trúc, cho phép mô-đun hoá chương trình mô phỏng thành tập các chương trình con, sao cho việc sửa đổi, bổ sung các chương trình con được dễ dàng. Ngoài ra, các chương trình mô phỏng cũng phải được xây dựng sao cho tối ưu hoá về mặt tốc độ nhằm làm giảm thời gian chạy mô phỏng. Phương pháp này có ưu điểm là người đo có thể tùy ý chọn các thiết bị ảo, từ đó đánh giá mức độ phù hợp của mạng trong hệ thống.

- **Đo**

Đo là phương pháp xác định hiệu năng bằng cách đo các tham số trên mạng thực. Hai vấn đề quan trọng mà việc đo phải thực hiện đó là:

- ✓ Thu thập số liệu để lập mô hình dữ liệu vào cho các phương pháp đánh giá hiệu năng khác.
- ✓ Kiểm chứng các mô hình khác dựa trên các số liệu đo được.

Việc đo phải được thực hiện liên tục từ lúc bắt đầu xây dựng giải pháp cho hệ thống, đến lúc lắp đặt và sau khi đưa hệ thống vào hoạt động. Trong giai đoạn xây dựng giải

pháp, đo sẽ giúp quyết định những thiết bị nào phù hợp nhất với yêu cầu của khách hàng. Trong giai đoạn lắp đặt, việc đo hiệu năng giúp người ta điều chỉnh lại cấu hình cho phù hợp. Việc đo hiệu năng mạng sau khi hệ thống đi vào hoạt động giúp quản trị viên theo dõi tình trạng của mạng để có các phương án khắc phục các sự cố nếu có. Hiện nay, hầu như tất cả các hệ thống mạng đều tích hợp bên trong nó các công cụ đo và đánh giá hiệu năng, nhờ đó có thể đo hiệu năng bất cứ lúc nào trong suốt thời gian vận hành của hệ thống.

Nhược điểm của Đo là nó phải được thực hiện trên mạng thực, nên chi phí tùy thuộc vào công cụ mà ta thực hiện. Vì đo là phương pháp thực tế, nên nó phải được thực hiện nhiều lần, lặp đi lặp lại, trên các khoảng thời gian liên tục và khác nhau, để có thể thu thập được dữ liệu một cách tốt nhất và từ đó mới có thể đưa ra các kết luận chính xác dựa trên các kết quả đo này.

Trong khuôn khổ luận văn này, phương pháp mô phỏng sẽ được chúng tôi lựa chọn để đo hiệu năng và đánh giá các kết quả dựa trên kết quả mô phỏng. Bộ mô phỏng được sử dụng là bộ mô phỏng NS 2.35. Cài đặt thông qua source từ link: https://sourceforge.net/projects/nsnam/files/allinone/ns-allinone-2.35/ trên hệ điều hành Ubuntu 16.04. Một vài hỗ trợ của NS 2.35 dành cho mạng truyền thống và mạng diffserv sẽ được chúng tôi trình bày ở dưới các chương có mô phỏng tương ứng.

CHƯƠNG 2. CÁC KẾ HOẠCH QUẢN LÝ HÀNG ĐỢI ĐỘNG CHO TRUYỀN THÔNG ĐA PHƯƠNG TIỆN TRÊN KIẾN TRÚC MẠNG TRUYỀN THỐNG

Trong lý thuyết hàng đợi, người ta đã chứng minh rằng thời gian trung bình mà các gói tin đi qua hàng đợi, bao gồm thời gian các gói tin phải chờ trong hàng đợi cộng với thời gian chúng được phục vụ, tỉ lệ thuận với chiều dài trung bình của hàng đợi và tỉ lệ nghịch với tốc độ đến hàng đợi trung bình [3]. Điều này được Little phát biểu bằng công thức: $L = \lambda.W$ ($\Rightarrow W = L/\lambda$), trong đó: L (packets) là chiều dài trung bình của hàng đợi, λ (packets/s) là tốc độ đến trung bình của các gói tin, và W là thời gian đi qua hàng đợi (s).

Mục tiêu chính của các chiến lược quản lý hàng đợi là giữ cho chiều dài hàng đợi đủ nhỏ và ổn định, đảm bảo độ trễ trung bình của các gói tin không vượt quá mức cho phép, nhưng vẫn có hệ số sử dụng đường truyền cao [1, 15]. Hai yêu cầu trên là trái ngược nhau, chính vì vậy cần có sự thoả hiệp. Để biểu diễn sự thoả hiệp, người ta đưa ra một đại lượng gọi là "công suất", đó là tỷ lệ giữa thông lượng và độ trễ. Điểm làm việc tối ưu là điểm có "công suất" cực [2]. Nội dung chính của chương 2 là trình bày về 2 cơ chế lập lịch phổ biến và 2 chiến lược quản lý hàng đợi động AQM tiêu biểu trong mạng truyền thống nhằm đảm bảo chất lượng dịch vụ QoS cho truyền thông đa phương tiện. Qua đó chúng ta sẽ so sánh 2 chiến lược quản lý hàng đợi động AQM qua mô phỏng cụ thể.

2.1 Cách tiếp cận truyền thống và hệ quả

Để quản lý kích thước hàng đợi trong kỹ thuật truyền thống, ta thiết lập độ dài tối đa cho mỗi hàng đợi. Các gói tin đến sẽ được cho vào hàng đợi, khi hàng đợi đầy, các gói tin đến sau đó sẽ bị loại bỏ, đến khi hàng đợi có chỗ trống do có gói tin được chuyển đi thì càng gói tin tiếp theo mới được nhận vào hàng đợi (đây là chính sách FIFO, trong NS được biết đến với tên là DropTail). Phương pháp này được sử dụng trong Internet trong nhiều năm, song nó có 2 nhược điểm chính sau đây [8, 1]:

2.1.1 Hiện tượng Lock-Out và Global Synchronization

Trong một vài tính huống đặc biệt, DropTail cho phép một hoặc một vài dòng lưu lượng độc quyền chiếm hàng đợi, làm cho các gói tin của các kết nối khác không thể được nhận vào, ta nói chúng bị "lock-out". Khi hiện tượng "lock-out" xảy ra, nếu thực thể gửi sử dụng giao thức TCP thì chúng sẽ bị timeout khi đi qua hàng đợi, lúc này theo thuật toán tránh tắc nghẽn chúng sẽ đồng thời giảm kích thước cửa sổ phát và thực hiện rút lui theo hàm mũ theo thuật toán tránh tắc nghẽn, làm cho lưu lượng trên mạng giảm

mạnh. Đó là hiện tượng đồng thời giảm lưu lượng trên toàn mạng, được gọi là - "global synchronization", gây lãng phí dải thông của mạng [20].

2.1.2 Hiện tượng Full Queues

Với "Tail-Drop", hàng đợi có thể thường xuyên ở trạng thái đầy hoặc gần đầy trong khoảng thời gian dài, vì việc loại bỏ gói tin đến chỉ thực hiện khi hàng đợi đã đầy. Ngoài ra, vì lưu lượng trên Internet thường xuyên có sự bùng nổ, các gói tin đến hàng loạt một lúc gây nên bùng nổ trong mạng, để đáp ứng được nhu cầu phục vụ ta phải tăng kích thước hàng đợi đủ để có thể hấp thu được sự bùng nổ dữ liệu. Nhưng việc tăng kích thước hàng đợi sẽ dẫn đến độ trễ và Jitter tăng cao. Nên việc sử dụng "Tail-Drop" sẽ dẫn đến việc độ trễ và thăng giáng độ trễ của các gói tin đi qua mạng có giá trị cao. Ngoài "Tail-Drop", hai phương pháp khác có thể được áp dụng khi hàng đợi đầy là loại bỏ ngẫu nhiên hoặc loại bỏ ở đầu khi đầy. Cả hai trường hợp này có thể giải quyết được vấn đề "lock-out" như đã nói ở trên nhưng không giải quyết được vấn đề thứ 2 là "full-queue".

2.2 Chiến lược AQM

Hàng đợi đầy chính là dấu hiệu của tắc nghẽn. Các chiến lược truyền thống chỉ thực hiện loại bỏ gói tin khi hàng đợi đầy, lúc này thì mạng đã ở trong tình trạng tắc nghẽn khó kiểm. Để giải quyết vấn đề này, chúng ta cho phép gateway loại bỏ các gói tin trước khi hàng đợi đầy, khi đó các thực thể đầu cuối có thể phản ứng lại tắc nghẽn khi hiện tượng này mới chớm có dấu hiệu xuất hiện. Cách tiếp cận này được gọi là quản trị hàng đợi động – **Active Queues Manegement AQM**. Với chiến lược này, gateway sẽ quyết định cách thức loại bỏ gói tin trong hàng đợi của nó. Điều cần phải chú ý là, AQM phải được gắn với các giao thức vận chuyển có áp dụng các cơ chế điều khiển lưu lượng và tránh tắc nghẽn kiểu end-to-end (như TCP), và nó không có tác dụng đối với các giao thức vận chuyển không áp dụng cơ chế điều khiển lưu lượng (như UDP).

Nhìn chung, các chiến lược AQM đem lại các lợi ích được trình bày dưới đây:

2.2.1 Giảm số gói tin bị loại bỏ tại router

Sự bùng nổ các gói tin là không thể tránh được trong mạng chuyển mạch gói. Bằng việc giữ kích thước trung bình của hàng đợi nhỏ, AQM sẽ cung cấp dung lượng lớn hơn để hấp thu các bùng nổ xảy ra một cách ngẫu nhiên mà không cần loại bỏ hàng loạt gói tin đến khi hàng đợi bị đầy.

2.2.2 Giảm độ trễ

AQM giữ kích thước hàng đợi trung bình nhỏ, dẫn đến độ trễ trung bình của các gói tin cũng nhỏ. Điều này đặc biệt tốt với các ứng dụng tương tác thời gian thực trong truyền thông đa phương tiện như các ứng dụng thoại, video call… ứng dụng mà độ trễ thấp đồng nghĩa với hiệu quả tốt.

2.2.3 Tránh hiện tượng Lock-Out

AQM đảm bảo rằng hầu như luôn luôn có vị trí trống trong bộ đệm khi một gói tin đến, do đó tránh được hiện tượng Lock-Out. Cũng với lý do đó, AQM có thể làm cho router không chống lại các dòng có thông lượng thấp nhưng có độ đột biến cao.

2.3 Chiến lược RED

2.3.1 Giới thiệu

RED (Random Early Detection of congestion; Random Early Drop) là một trong những thuật toán AQM đầu tiên được đề xuất vào năm 1993 bởi Sally Floyd và Van Jacobson, hai nhà khoa học của Phòng thí nghiệm Lawrence Berkeley, thuộc Đại học Califonia, Mỹ [16]. Do những ưu điểm nổi bật của nó so với các thuật toán quản lý hàng đợi trước đó, RED đã được cài đặt và triển khai trên mạng Internet. RED là một chiến lược AQM đặc biệt, áp dụng cho mạng chuyển mạch gói, được thiết kế để đi cùng với giao thức điều khiển tắc nghẽn TCP. Ưu điểm chính của RED là tính đơn giản, không yêu cầu tất cả các gateway và máy tính trên Internet cùng phải sử dụng kỹ thuật này, mà có thể triển khai dần.

Điểm cơ bản nhất trong công trình của họ là cho rằng **nơi hiệu quả nhất để phát hiện tắc nghẽn và phản ứng lại hiện tượng này chính là tại các gateway**. Các thực thể nguồn cũng có thể làm điều này thông qua thời gian phục vụ ước lượng tại gateway, thông qua độ trễ end-to-end, qua sự thay đổi thông lượng, hoặc từ các gói tin bị loại bỏ. Tuy nhiên, khung nhìn của một kết nối cụ thể bị giới hạn bởi thời gian phát tin (mỗi kết nối có một thời gian phát gói tin nhất định) và số lượng dữ liệu phát, ngoài ra nó cũng không biết những gateway nào đang bị tắc nghẽn, không phân biệt được độ trễ lan truyền và độ trễ hàng đợi (độ trễ mà gói tin phải chờ trong hàng đợi). Chỉ có gateway mới có cái nhìn đúng đắn về trạng thái của hàng đợi, sự chia sẻ đường truyền của các kết nối đi qua nó tại mọi thời điểm, cũng như yêu cầu chất lượng dịch vụ của các dòng lưu lượng.

2.3.2 Nguyên tắc hoạt động

- Phát hiện sớm tắc nghẽn, bằng cách thường xuyên giám sát kích thước hàng đợi trung bình. Tránh tắc nghẽn bằng cách loại bỏ các gói tin trong hàng đợi theo một số quy tắc nhất định, để giữ cho kích thước hàng đợi trung bình đủ nhỏ, làm cho mạng hoạt động ở vùng có độ trễ thấp và thông lượng cao, trong khi vẫn cho phép kích thước hàng đợi dao động trong một miền nhất định để hấp thụ các thăng giáng ngắn hạn.
- Báo hiệu sớm tắc nghẽn. Khi có dấu hiệu của tắc nghẽn, ngoài biện pháp loại bỏ các gói tin nêu trên, cần áp dụng biện pháp đánh dấu vào trường ECN (Explicit Congestion Notification) của gói tin với một xác suất nhất định, để báo hiệu cho nguồn giảm lưu lượng đưa vào mạng (thông tin ECN được bên đích gửi cho bên nguồn trong gói tin ACK). Việc đánh dấu được thực hiện ngẫu nhiên để tránh

hiện tượng global synchronization và không chống lại các dòng lưu lượng có giá trị trung bình thấp nhưng độ thăng giáng cao [17, 21].

2.3.3 Mục tiêu

Mục tiêu chính của RED là phát hiện sớm tắc nghẽn bằng cách theo dõi kích thước hàng đợi trung bình, tránh tắc nghẽn bằng cách điều khiển kích thước hàng đợi trung bình nằm trong một vùng đủ nhỏ và ổn định. RED còn hướng đến các mục tiêu như tránh hiện tượng đồng bộ toàn cục, không chống lại các dòng lưu lượng có đặc tính đột biến và duy trì cận trên của kích thước hàng đợi trung bình ngay cả khi không có được sự hợp tác từ các giao thức tầng giao vận.

Để đạt được các mục tiêu đó, các gateways tránh tắc nghẽn phải làm được những nhiệm vụ dưới đây:

- **Việc đầu tiên là phát hiện sớm tắc nghẽn, giữ cho kích thước hàng đợi trung bình đủ nhỏ, làm cho mạng hoạt động ở vùng có độ trễ thấp và thông lượng cao, trong khi vẫn cho phép kích thước hàng đợi dao động trong một miền nhất định để hấp thụ các thăng giáng ngắn hạn.** Như kết quả từ nghiên cứu của Sally Floyd và Van Jacobson [16], gateway là nơi thích hợp nhất để phát hiện tắc nghẽn và cũng là nơi thích hợp nhất để quyết định chọn kết nối cụ thể nào để thông báo tắc nghẽn.
- **Việc thứ hai là thông báo tắc nghẽn tới nguồn phát.** Việc này được thực hiện bằng cách đánh dấu và thông báo cho nguồn phát giảm lưu lượng xuống. Thông thường gateway sẽ loại bỏ gói tin. Tuy nhiên, nếu tắc nghẽn được phát hiện trước khi hàng đợi đầy thì nên đánh dấu gói tin thay vì loại bỏ nó để báo hiệu tắc nghẽn. (xem phần 2.4.2)
- **Một mục tiêu quan trọng mà các gateway cần đạt được là tránh hiện tượng đồng bộ toàn cầu và không chống lại các dòng lưu lượng có đặc tính đột biến.** RED gateway chọn ngẫu nhiên các gói tin đến để đánh dấu; với phương pháp này xác suất đánh dấu một gói tin từ một kết nối cụ thể tỉ lệ với phần băng thông được chia sẻ của kết nối đó tại gateway.
- **Một mục tiêu nữa là điều khiển được kích thước hàng đợi trung bình ngay cả khi không có sự hợp tác từ các thực thể nguồn phát**. Mục tiêu này được RED thể hiện bằng cách loại bỏ các gói tin đến khi kích thước hàng đợi trung mình vượt quá ngưỡng cho phép.

2.3.4 Giải thuật

```
for each packet arrival
   calculate the average queue size avg
   if min_th ≤ avg < max_th
      calculate probability p_a
      with probability p_a:
         mark the arriving packet
   else if max_th ≤ avg
      mark the arriving packet
```

Hình 2.5: Giải thuật tổng quát cho RED gateways [16]

Giải thuật tại RED gateway gồm hai giải thuật tách biệt: Giải thuật tính kích thước hàng đợi trung bình quyết định mức độ bùng nổ cho phép trong hàng đợi tại gateway và giải thuật tính xác suất đánh dấu quyết định mức độ thường xuyên của việc đánh dấu gói tin của gateway. Giải thuật tính xác suất đánh dấu phải đảm bảo sao cho các gói tin được đánh dấu tại những khoảng không gian đều nhau, để tránh hiện tượng đồng bộ toàn cầu, trong khi vẫn giữ kích thước hàng đợi trung bình ở một giới hạn nhất định.

2.3.4.1 Tính kích thước hàng đợi trung bình [16, 1]

Mỗi khi có một gói tin đến, sẽ tính kích thước hàng đợi trung bình – avg bằng bộ lọc thông thấp (Low Pass Filter):

$$avg = (1 - w_q) * avg + w_q * q$$

- q: kích thước hàng đợi hiện thời.
- w_q: trọng số hàng đợi, nhận giá trị trong miền 0..1. w_q còn được gọi là hệ số làm trơn, w_q càng nhỏ thì mức độ làm trơn càng cao, w_q càng lớn thì avg càng bám sát giá trị tức thời của q. Như vậy w_q quyết định độ lớn và độ kéo dài cho phép của sự bùng nổ lưu lượng.

Sau đó avg được sử dụng để tính xác suất loại bỏ gói tin, như trình bày dưới đây.

2.3.4.2 Tính xác xuất loại bỏ gói tin [16, 1]

Khi có một gói tin đến, nó sẽ bị loại bỏ với xác suất pa, được tính như dưới đây:

$$p_a = pb/(1\text{-}count*p_b)$$

$$p_b = max_p*(avg - min_{th})/(max_{th} - min_{th})$$

- max_p: xác suất loại bỏ tối đa.
- min_{th}: ngưỡng dưới của hàng đợi.
- max_{th}: ngưỡng trên của hàng đợi.
- count: số gói không bị loại bỏ kể từ gói bị loại bỏ cuối cùng trước đó.

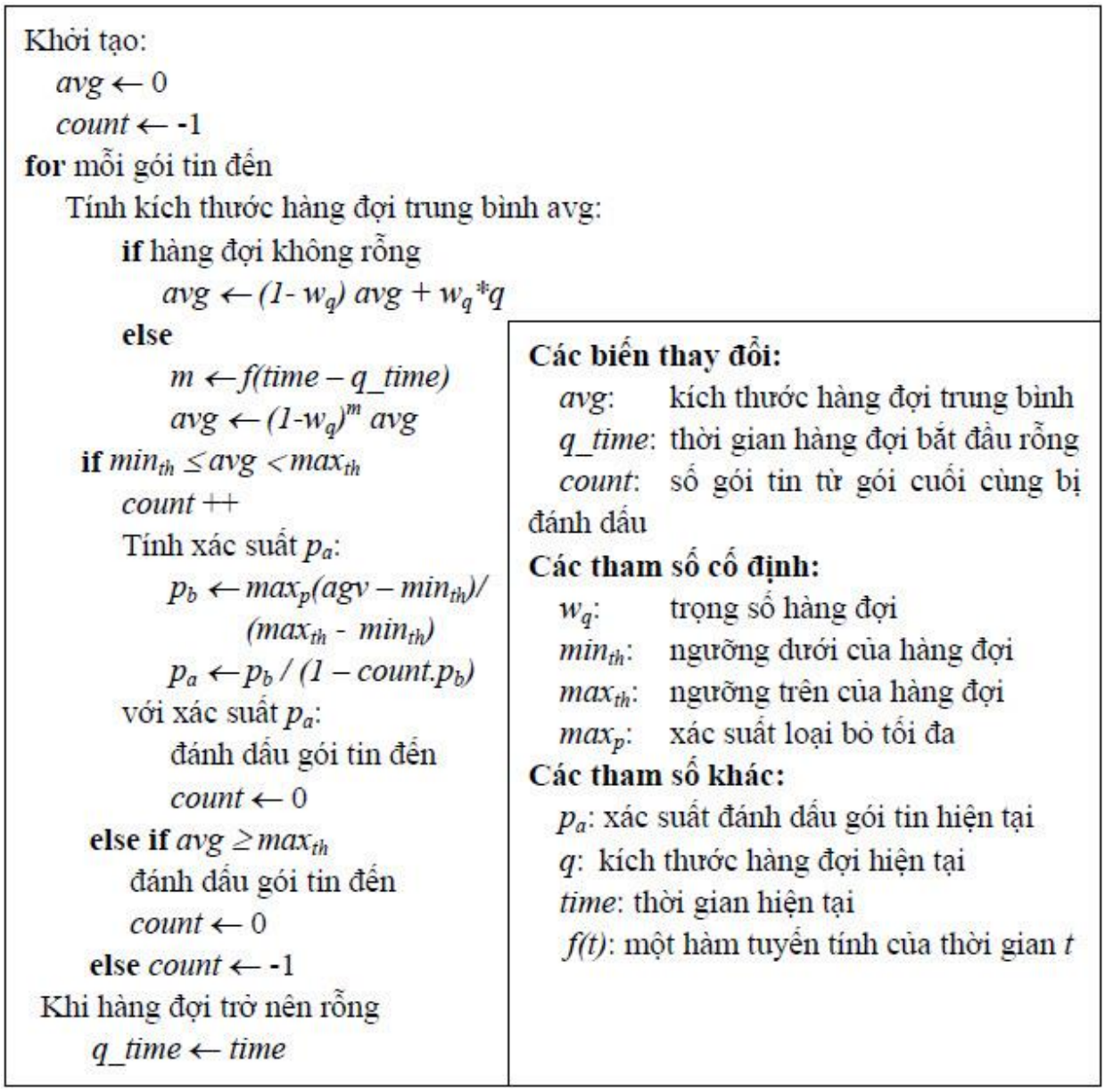
Chúng ta thấy:

- Khi avg ≤ min_{th}, p_b = 0, nên không có gói tin nào bị loại bỏ.
- Khi avg tăng từ min_{th} đến max_{th}, xác suất loại bỏ gói gói tin p_b tăng từ 0 đến max_p.
- Khi avg ≥ max_{th}, p_b =1, nên tất cả các gói tin đều bị loại bỏ.
- p_a tăng chậm khi count tăng.
- Giá trị maxp sẽ quyết định tần số loại bỏ gói là lớn hay nhỏ, nó quyết định avg sẽ nằm ở mức nào trong khoảng từ min_{th} đến max_{th}. Vì vậy tùy từng yêu cầu mà có thể thiết lập max_p cho phù hợp. Thí dụ, để avg nằm trong khoảng giữa min_{th} và max_{th}, giá trị max_p phù hợp là 0.02. Tuy nhiên nếu tắc nghẽn thường xuyên xảy ra, cần chọn max_p lớn hơn, thí dụ bằng 0.1.

RED có một tuỳ chọn: để đảm bảo xác suất một gói bị loại bỏ tỉ lệ với thông lượng tính bằng bit/s chứ không phải packet/sec, p_b được tính như sau [22]:

$$pb = \frac{PacketSize}{MaxPacketSize} * maxp * \frac{(avg - minth)}{(maxth - minth)}$$

Trong trường hợp này, một gói tin lớn dễ bị loại bỏ hơn là một gói tin nhỏ.



Hình 2.6: Giải thuật RED chi tiết

2.3.5 Thiết lập tham số cho RED

Từ thuật toán nêu ở phần 2.4.4 ta thấy, RED có 4 tham số cố định cần phải đặt trước, đó là các tham số: Trọng số hàng đợi w_q, ngưỡng dưới của hàng đợi min_{th}, ngưỡng trên của hàng đợi max_{th}, xác suất loại bỏ tối đa max_p. Các tham số này của RED rất quan trọng, nó ảnh hưởng trực tiếp đến chất lượng của thuật toán. Ở khuôn khổ luận văn này, tôi xin phép được dùng lại các khuyến nghị về việc chọn các tham số cho RED từ nghiên cứu và đánh giá của tác giả [1]:

- Nên chọn $max_{th/}$ min_{th} = 3
- min_{th} không nên nhỏ hơn 5

- Nên chọn $max_p = 0.1$. Tuy nhiên tùy theo từng yêu cầu cụ thể mà ta thiết lập max_p một cách phù hợp. max_p cao thì tỷ lệ loại bỏ gói tin cao, lúc này kích thước hàng đợi trung bình sẽ nằm gần min_{th}. max_p thấp thì tỷ lệ loại bỏ gói tin thấp, lúc này kích thước hàng đợi trung bình sẽ nằm gần max_{th}
- Nên chọn $w_p = 0.002$

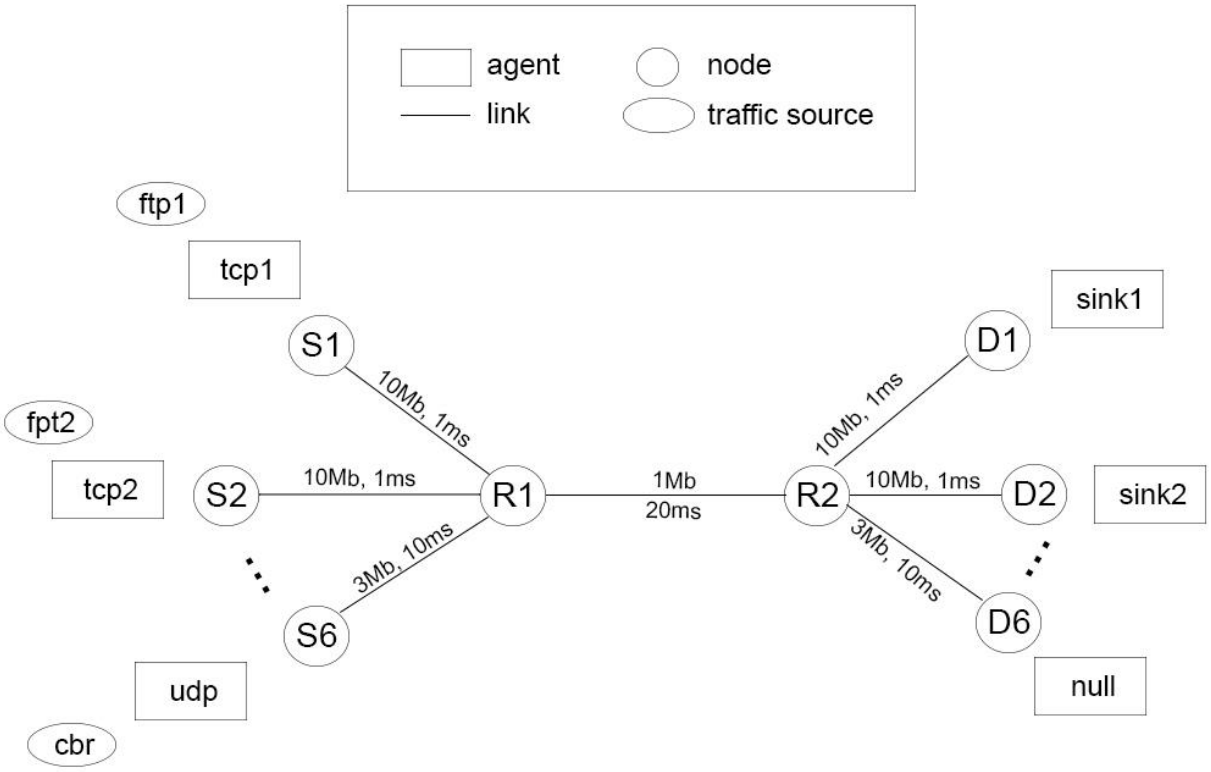
2.3.6 Mô phỏng RED và so sánh với DropTail

Tiến hành mô phỏng RED trên bộ mô phỏng NS2 – phiên bản 2.35. Áp dụng kết quả từ [1] làm cơ sở để chúng tôi chọn các tham số RED phục vụ cho các mô phỏng sau này của chúng tôi. Sau đây là phần mô phỏng của chúng tôi nhằm so sánh và đánh giá hiệu năng giữa chiến lược RED và DropTail.

Chú ý: Có một điều lưu ý là với gói tin kích thước ví dụ 1000 bytes thì sẽ có thêm 40 bytes ở header được thêm vào [11, tr.77].

2.3.6.1 Cấu hình mạng mô phỏng

File code TCL mô phỏng và file tính toán các thông số như kích thước hàng đợi, thông lượng đường truyền, hệ số sử dụng đường truyền, độ trễ… tương ứng với mô phỏng ở hình 2.7 được tôi trình bày ở phụ lục với tên là: **red.tcl** và **redPerl.pl**



Hình 2.7: Cấu hình mạng mô phỏng so sánh giữa RED và DropTail

Mạng mô phỏng gồm 14 node được đánh số và thứ tự như hình 2.7; 5 luồng tcp gắn với 5 node 1, 2, 3, 4, 5 và một luồng udp gắn với node 6. Tất cả các luồng cùng chia sẻ đường truyền chung từ node r1 đến node r2. Các nguồn hướng đến các đích tương ứng như: s1 đi đến d1, s2 đi đến d2.... Đường truyền giữa các node đều là full-duplex, không lỗi:

- s1-r1, s2-r1, s3-r1, s4-r1, s5-r1: 10Mbps, 1ms
- d1-r2, d2-r2, d3-r2, d4-r2, d5-r2: 10Mbps, 1ms
- s6-r1: 3Mbps, 10ms.
- r1-r2: 1Mbps, 20ms.

Từ s1 đến s5 là các luồng TCP với nguồn FTP (fpt1, fpt2…). S6 là UDP nguồn CBR (nguồn sinh lưu lượng với tốc độ không đổi). Các thực thể nhận TCP tương ứng là sink1, sink2… Thực thể nhận UDP là null. Kích thước gói in TCP bằng 1000 bytes, hàng đợi Q đặt giữa node r1 và r2 có kích thước là 100 gói tin. Các thực thể gửi FPT được đưa vào mạng trong các khoảng thời gian bắt đầu từ cách nhau 0.1s bắt đầu từ ftp1 ở 0.1s và kết thúc ở. Thực thể CBR gửi vào mạng trong các khoảng: 6s – 7s, 15s – 17s,

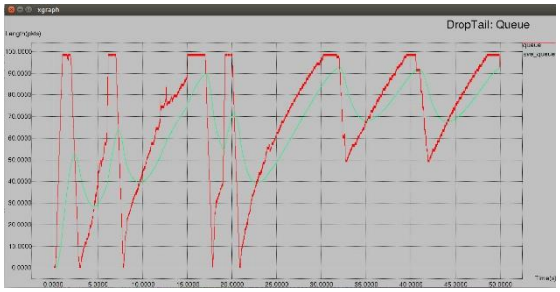
19.0s - 20.0s, tốc độ truyền 2Mbps; mục đích đưa nguồn CBR trong các khoảng thời gian như vậy là để gây lưu lượng đột biến cho mạng. Tổng thời gian mô phỏng là 50s.

Tôi sẽ thay đổi chính sách quản lý tại hàng đợi Q lần lượt là DropTail và RED và so sánh các kết quả. Với mỗi mô phỏng tôi đưa ra 3 đồ thị: kích thước hàng đợi trung bình, thông lượng của mỗi kết nối và kích thước cửa sổ để đánh giá các đại lượng liên quan như độ trễ trung bình, thông lượng của từng kết nối, và nghiên cứu hiện tượng đồng bộ toàn cục. Sau đây là chi tiết về kết quả thu được từ các mô phỏng.

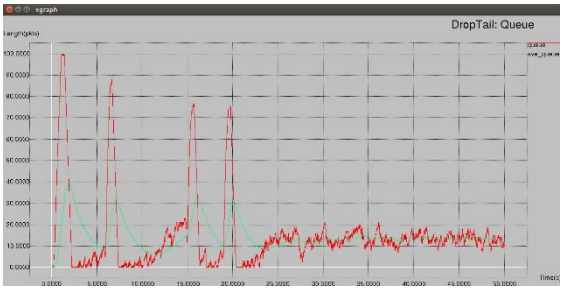
Bảng 2.1: Bảng kết quả thống kê của mô phỏng 1 so sánh giữa RED và DropTail

Chiến lược	Kích thước hàng đợi trung bình (gói tin)	Độ trễ hàng đợi trung bình (ms)	Hệ số sử dụng đường truyền (%)
DropTail	67.00	557.00	94.33
RED	15.00	123.00	93.41

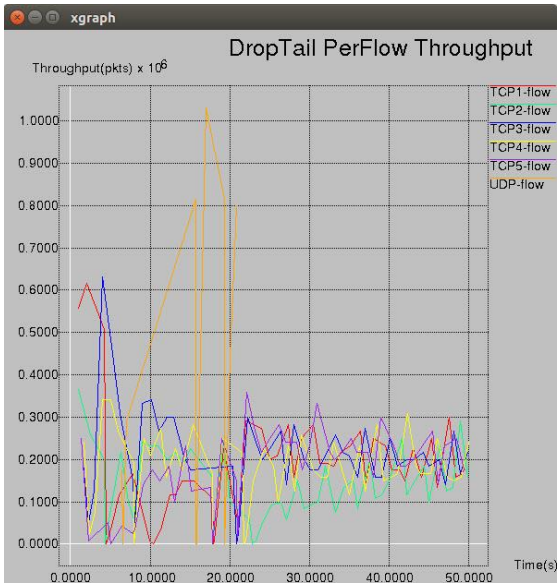
Đồ thị 2.1: Đồ thị kết quả thu được từ mô phỏng so sánh giữa RED và DropTail



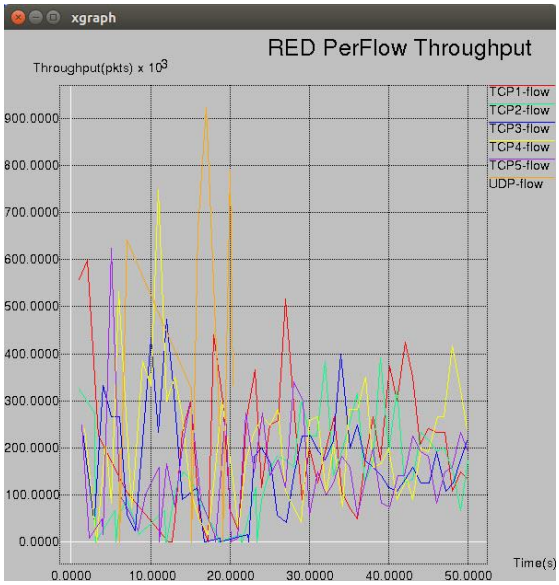
a. Kích thước hàng đợi trung bình



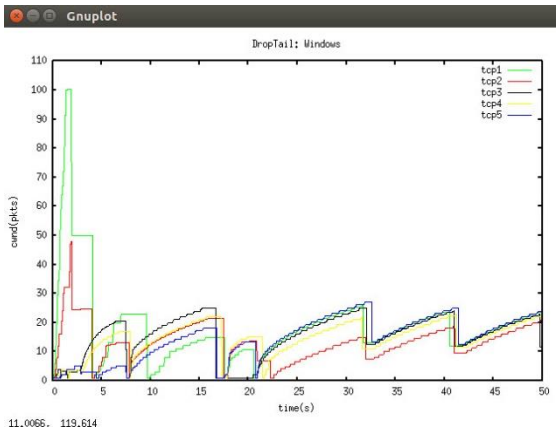
a. Kích thước hàng đợi trung bình



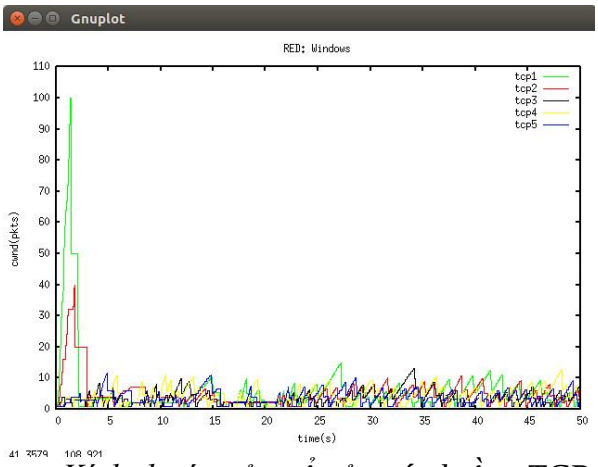
b. Thông lượng của mỗi luồng



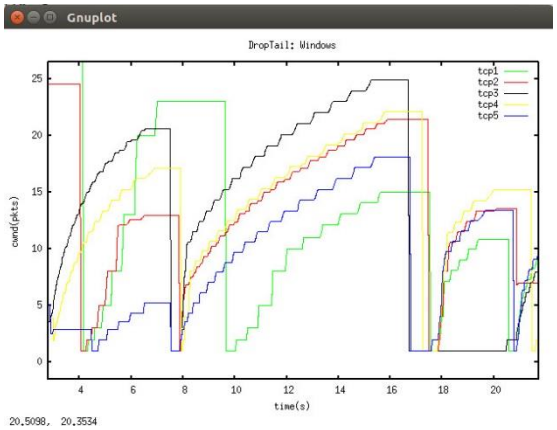
b. Thông lượng của mỗi luồng



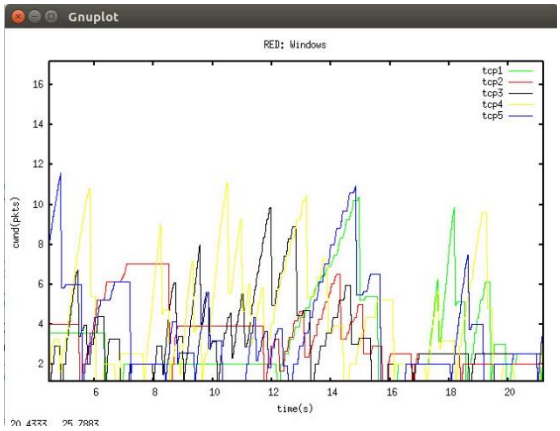
c. Kích thước cửa sổ của các luồng TCP



c. Kích thước cửa sổ của các luồng TCP



d. Kích thước cửa sổ của các luồng TCP phóng to giai đoạn đột biến



d. Kích thước cửa sổ của các luồng TCP phóng to giai đoạn đột biến

Hình 2.8: Kết quả mô phỏng với DropTail

Hình 2.9: Kết quả mô phỏng với RED

- **Mô hình mạng như trên với chính sách DropTail**

Từ kết quả thu được ở mô phỏng hình 2.7, nhìn vào hình 2.8 ta thấy:

✓ Trong các khoảng thời gian: 5s – 7s, 15s – 17s, 19-20s khi ta đưa luồng cbr vào để tạo lưu lượng đột biến trong mạng, nguồn cbr phát với tốc độ 2Mb vượt quá dung lượng đường truyền chung dẫn đến ở hàng đợi Q xảy ra hiện tượng lock-out. Hiện tượng Lock-out xảy ra dẫn đến hiện tượng global synchronization xảy ra - các kết nối tcp đồng loạt giảm kích thước cửa sổ phát và kích thước cửa sổ về đến 0 (hình 2.8c – 8s, 18s, 21s), đi cùng với kích thước cửa sổ giảm đồng loạt là thông lượng các kết nối tcp giảm xuống đột ngột (hình 2.8b). Hiện tượng sụt giảm thông

lượng ngay cả khi luồng cbr ngừng hoạt động là do cơ chế rút lui theo hàm mũ của TCP. Mặt khác trong thời thời gian này kích thước hàng đợi hầu như đầy (hình 2.8a), dẫn tới độ trễ hàng đợi cao.

- ✓ Ngay cả khi nguồn đột biến cbr ngừng hoạt động, hiện tượng global synchronization vẫn xuất hiện (ở các thời điểm 32s, 41s), các kết nối cùng tăng kích thước cửa sổ cho đến khi đạt đến ngưỡng thì đồng thời giảm kích thước cửa số phát xuống, kích thước hàng đợi (hình 2.8a) cũng vì thế mà dao động trong một miền rất rộng so với giá trị trung bình và chiều dài hàng đợi trung bình thường xuyên ở mức cao (cỡ 80 ± 12). Kích thước hàng đợi trung bình bám rất sát với kích thước hàng đợi thực tế.

- **Mô hình mạng như trên với chính sách RED**

RED được mô phỏng với các tham số như sau: min_{th} = 5, max_{th} = 15, max_p = 0.1 và w_q = 0.002 [1]. Kết quả tôi thu được từ mô phỏng được thể hiện qua đồ thị ở hình 2.9.

- ✓ Trong khoảng thời gian mà chúng ta đưa lưu lượng đột biến cbr vào (5s – 7s, 15s -17s, 19s – 20s), cũng xảy ra hiện tượng kích thước cửa sổ các kết nối tcp giảm xuống (hình 2.9d), kéo theo nó là thông lượng của chúng cũng giảm theo, kích thước hàng đợi trong giai đoạn có đột biết cũng tăng cao nhưng không bị tràn, đặc biệt là sau khi ra khỏi giai đoạn này (9s – 15s, 18s – 19s, 21s – 50s) thì các kết nối này nhanh chóng tăng kích thước cửa sổ lên, thông lượng vì thế nhanh chóng được hồi phục; mặt khác kích thước hàng đợi tăng lên nhưng nhanh chóng được kéo xuống (hình 2.9a).
- ✓ Trong giai đoạn không có đột biến thì RED luôn duy trì được kích thước hàng đợi trung bình (ở khoảng ~ 12 ± 2 gói tin)

Ngoài 2 đồ thị thu được từ mô phòng, các thông số khác tôi thu được ở bảng 2.1 phía trên cho ta thấy: hàng đợi trung bình khi ta dung chiến lược RED thấp hơn nhiều lần khi ta dung chiến lược DropTail (15 so với 67). Kéo theo độ trễ hàng đợi trung bình của RED thấp hơn nhiều so với DropTail (123 so với 557) tuy nhiên vẫn đảm bảo được hệ số sử dụng đường truyền là gần bằng nhau (94.33 và 93.41).

2.3.6.2 Kết luận so sánh giữa RED và DropTail

Từ mô phỏng (hình 2.7) trên ta có thể thấy:

Hàng đợi sử dụng DropTail không thể tránh khỏi hiện tượng Lock-out và global synchronization cho dù có lưu lượng đột biết đưa vào mạng hay không, DropTail không

hỗ trợ sự chia sẻ giải thông công bằng giữa các kết nối; đặc biệt là khi có lưu lượng bùng nổ xảy ra trong mạng thì gần như toàn bộ đường truyền lúc này chỉ phục vụ riêng cho lưu lượng bùng nổ (hình 2.8b, 5s – 7s, 15s – 17s, 19s – 20s) đưa vào mạng, điều này dẫn đến việc không bảo vệ được các kết nối đang hoạt động.

Trong khi đó RED tránh được hiện tượng đồng bộ toàn cầu (global synchronization) khi giữ cho kích thước hàng đợi trung bình nhỏ, ngay cả khi có lưu lượng đột biến được đưa vào mạng. Vì RED giữ kích thước hàng đợi trung bình nhỏ, nên có bộ đệm lớn để hấp thục các luồng lưu lượng đột biến xảy ra trong mạng, tránh được các hiện tượng như lock-out và global synchronization. Ta thấy đột biến trong khoảng thời gian ngắn hạn đã được ngăn cản, đặc biệt là sau khoảng thời gian xảy ra tắc nghẽn, thông lượng được hồi phục rất nhanh. Chia sẻ giải thông tương đối công bằng giữa các kết nối có tính chất giống nhau (như các luồng tcp). Vì RED duy trì kích thước hàng đợi nhỏ nên điều này giúp nó đạt được độ trễ thấp hơn rất nhiều so với DropTail, trong khi vẫn đảm bảo hệ số sử dụng đường truyền (bảng 2.1).

2.4 Adaptive-RED (A-RED)

Từ mô phỏng phía trên (hình 2.7) và nhận xét từ mô phỏng ở phần 2.4.6.2 ta thấy RED có nhiều ưu điểm hơn so với DropTail. Tuy nhiên từ đồ thị thu được từ mô phỏng và công thức loại bỏ gói tin của RED (mục 2.4.4.2), ta thấy RED có hai điểm yếu cơ bản cần phải được khắc phục:

Điểm yếu thứ nhất là kích thước hàng đợi trung bình rất nhạy cảm với các mức độ tắc nghẽn và việc thiết lập các tham số cho RED router. Khi đường truyền tắc nghẽn nhẹ (hình 2.9a – 12s, 24s), và/hoặc max_p cao thì kích thước hàng đợi trung bình gần min_{th}; khi đường truyền tắc nghẽn nặng hơn, và/hoặc max_p thấp, kích thước hàng đợi trung bình gần, hoặc thậm chí cao hơn max_{th} (hình 2.9a – 7s, 16s, 20s). Kết quả là, độ trễ hàng đợi trung bình rất nhạy cảm với tải (lưu lượng đưa vào mạng) và các tham số, dẫn tới không thể đoán trước được. Trong truyền thông đa phương tiện, độ trễ là một yếu tố rất quan trọng đảm bảo đến chất lượng dịch vụ, mạng phải có một sự ước lượng tốt để có thể ước lượng chính xác độ trễ trung bình trong các router khi xảy ra tắc nghẽn. Để đạt được độ trễ trung bình có thể đoán trước được, cần điều chỉnh liên tục các tham số RED để đáp ứng sự thay đổi điều kiện của lưu lượng mạng hiện tại.

Điểm yếu thứ hai của RED là thông lượng cũng nhạy cảm với tải và các tham số RED. Nhìn vào hình 2.9a và 2.9b ta thấy, khi kích thước hàng đợi trung bình trở nên lớn hơn max_{th} (giai đoạn 7s, 17s, 20s) thì thông lượng giảm đáng kể và tỉ lệ loại bỏ gói tin tăng. **Để tránh nhược điểm này cũng cần đến sự điều chỉnh liên tục các tham số**

RED. Đây cũng chính là nguyên tắc cốt lõi của thuật toán A-RED cải tiến từ RED gốc.

2.4.1 Thuật toán A-RED

Các mục tiêu của A-RED [18, tr.5-6]:

- max_p được thay đổi không chỉ để giữ cho kích thước hàng đợi trung bình nằm trong khoảng min_{th} và max_{th} mà còn giữ cho kích thước hàng đợi trung bình nằm trong khoảng nửa min_{th} và max_{th}, - là khoảng (q_{low}, q_{high}), với $q_{low} = min_{th} + 0.4(max_{th} – min_{th})$, $q_{high} = min_{th} + 0.6(max_{th} – min_{th})$.
- max_p được thay đổi chậm, sau những khoảng thời gian lớn hơn một thời gian khứ hồi (round-trip time), và được thực hiện với chi phí thấp.
- max_p phải được khống chế để duy trì trong miền [0.01, 0.5] (tương ứng với [1%, 50%]).
- Thay vì phải nhân lên nhiều lần khi tăng và giảm max_p, thuật toán sử dụng chính sách tăng theo cấp số cộng giảm theo cấp số nhân, tức là khi tăng thì cộng thêm một lượng đủ nhỏ (α), khi giảm thì nhân với một giá trị nhỏ hơn 1 (β). Các giá trị α, β được chọn sao cho kích thước hàng đợi trung bình quay trở lại miền mục tiêu (khoảng (q_{low}, q_{high})) không quá 25s.

A-RED kế thừa lại thuật toán RED gốc, bổ sung thêm thuật toán hiệu chỉnh max_p, sau đó max_p được dùng trong thuật toán RED. **Ưu điểm chính của A-RED là thay đổi các giá trị max_p không thường xuyên và thay đổi chậm.** max_p chỉ được thay đổi khi cần thiết sau những khoảng thời gian dài. Hầu như việc thay đổi này chỉ ở những thời điểm ngay sau khi có đột biến xảy ra tắc nghẽn (lưu lượng tăng hoặc giảm đột ngột). Để đảm bảo cho A-RED vẫn hoạt động tốt sau những thời điểm đột biến này, max_p luôn được giữ trong khoảng [0.01, 0.5] [18]. Điều này đảm bảo trong suốt thời gian thay đổi trạng thái của mạng hiệu năng tổng thể của RED, độ trễ trung bình, thông lượng chịu ảnh hưởng với một mức độ chấp nhận được.

```
Every interval seconds:
    if (avg > target and max_p ≤ 0.5)
        increase max_p:
        max_p ← max_p + α;
    elseif (avg < target and max_p ≥ 0.01)
        decrease max_p:
        max_p ← max_p * β;

Variables:
avg:  average queue size

Fixed parameters:
interval:  time; 0.5 seconds
target:  target for avg;
    [min_th + 0.4 * (max_th − min_th),
        min_th + 0.6 * (max_th − min_th)].
α:  increment; min(0.01, max_p/4)
β:  decrease factor; 0.9
```

Hình 2.10: Thuật toán hiệu chỉnh max_p trong A-RED

2.4.2 Thiết lập các tham số

2.4.2.1 Phạm vi của max_p [18, tr.7]

Cận trên của max_p được thiết lập là 0.5 vì:

- Tỷ lệ loại bỏ gói tin vượt quá 50% là không thể chấp nhận được, do đó ta không cần thiết tối ưu RED với tỷ lệ như vậy.
- Thứ hai, bởi vì khi sử dụng RED trong chế độ *gentle* (*gentle mode*) [19], tỉ lệ loại bỏ gói tin thay đổi từ 0 đến max_p khi kích thước hàng đợi trung bình thay đổi từ min_{th} đến max_{th}, và tỉ lệ loại bỏ gói tin thay đổi từ max_p đến 1 khi kích thước hàng đợi trung bình thay đổi từ max_{th} đến 2*max_{th}. Do đó, với việc thiết lập max_p bằng 0.5, tỉ lệ loại bỏ gói tin thay đổi từ 0 đến 1 khi kích thước hàng đợi trung bình chạy từ min_{th} đến 2*max_{th}. Điều này giúp tăng cường hiệu năng cho RED ngay cả khi tỉ lệ loại bỏ gói tin lớn hơn 50%.

Cận dưới của max_p được thiết lập 0.01 với mong muốn hạn chế miền của max_p. Tùy nhiên như mục 2.4.5 tôi đã trình bày, giá trị max_p được lựa chọn tùy vào mục tiêu hướng đến của bài toán chúng ta đặt ra. Với các mô phỏng trong luận văn về RED, tôi xin lấy kết quả khảo sát các tham số của tác giả [1] để xét giá trị khi thực hiện mô phỏng: max_p = 0.1

2.4.2.2 Các tham số α và β [18, tr.7, 5]

Hai tham số α và β phải được thiết lập sao cho với điều kiện bình thường, một thay đổi max_p không làm cho kích thước hàng đợi trung bình giảm đột ngột từ trên miền mục tiêu xuống dưới miền đó, hoặc ngược lại. Để cho đơn giản, giả sử khi max_p được thay đổi, xác suất loại bỏ gói tin *p* được giữ không đổi, và kích thước hàng đợi trung bình *avg* thay đổi theo sự thay đổi của max_p.

Giả sử $p < max_p$, khi max_p tăng lên một lượng α, *avg* mục tiêu sẽ giảm đi từ

$$\min_{th} + \frac{p}{\max_p}(\max_{th} - \min_{th}) \quad \text{xuống} \quad \min_{th} + \frac{p}{\max_p + \alpha}(\max_{th} - \min_{th})$$

Tức là giảm đi

$$\frac{\alpha}{\max_p + \alpha}\frac{p}{\max_p}(\max_{th} - \min_{th})$$

Khi giá trị này < 0.2 ($max_{th} - min_{th}$) thì kích thước hàng đợi trung bình sẽ không thay đổi đột ngột từ trên miền mục tiêu xuống dưới miền mục tiêu trong một khoảng thời gian *interval*. Vì $p/max_p < 1$, nên ta phải chọn α sao cho $\alpha/(max_p+\alpha) < 0.2$ hay $\alpha < 0.25*max_p$.

Tương tự, để việc *giảm theo cấp số nhân* của max_p không làm cho *avg* tăng đột ngột từ dưới miền mục tiêu lên trên miền đó sau mỗi lần thay đổi max_p, theo cách phân tích tương tự như với α, ta phải có:

$$\frac{p(1-\beta)}{\max_p \beta}(\max_{th} - \min_{th}) < 0.2(\max_{th} - \min_{th}) \Rightarrow \frac{1-\beta}{\beta} < 0.2 \Rightarrow \beta > 0.83$$

Các tham số mặc định cho α (= 0.01) và β (= 0.9) trong thuật toán A-RED được tôi dùng cho mô phỏng (hình 3.11) dưới đây là thoả mãn các cận vừa xem xét.

2.4.2.3 Thiết lập các tham số RED: maxth và wq [18, tr.7, 5]

Theo nhiều kiến nghị thì max_{th} nên gấp ba lần min_{th}. Khi đó, kích thước hàng đợi trung bình mong muốn sẽ nằm trong khoảng quanh $2*min_{th}$. Do đó cần phải xác định được min_{th}.

Mặt khác, người ta đã chứng minh được rằng, w_q phụ thuộc vào tốc độ đường truyền, đường truyền tốc độ cao yêu cầu giá trị w_q nhỏ, vì vậy trong [18] các tác giả đã thiết lập w_q là một hàm của băng thông đường truyền:

$$w_q = 1 - \exp(-1 / C) \qquad (*)$$

trong đó C là băng thông đường truyền, tính theo packet/s.

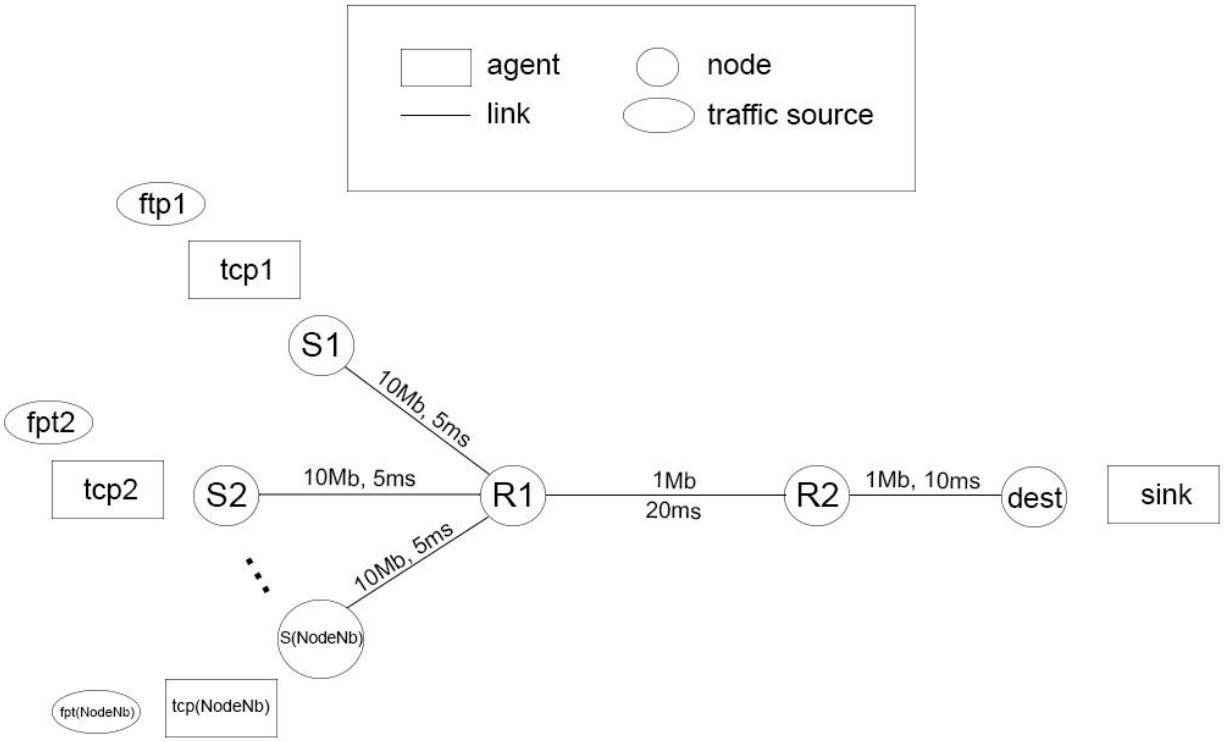
Trong bộ mô phỏng NS2, để mô phỏng A-RED, ta chỉ cần thay đổi tham số adaptive_ từ 0 chuyển thành 1 và set thêm 2 giá trị cho α và β và giữ nguyên các cài đặt của RED.

2.4.3 Mô phỏng A-RED

Tôi sẽ thực hiện mô phỏng để đánh giá chiến lược RED và A-RED thông qua việc hấp thụ các lưu lượng đột biến được đưa vào mạng.

2.4.3.1 Cấu hình mạng mô phỏng

File code TCL mô phỏng và file tính toán các thông số như kích thước hàng đợi, thông lượng đường truyền, hệ số sử dụng đường truyền, độ trễ… tương ứng với mô phỏng ở hình 2.11 được tôi trình bày ở phụ lục với tên là: **ared.tcl** và **aredPerl.pl**



Hình 2.11: Cấu hình mạng mô phỏng so sánh giữa RED và A-RED

Topo mạng được tôi minh họa trên như hình 2.11. Trong mô phỏng lần này, mạng gồm NodeNb node (Ở đây NodeNb được tôi để trong mô phỏng là 55 node), đích đến của 55 node này là node Dest, chúng đi qua kênh truyền chung giữa R1 và R2. Các thực thể gửi đều là TCP, kích thước cửa sổ gửi tối đa là 50 gói tin và đều xuất phát từ các nguồn sinh lưu lượng FTP. Các đường truyền đều là duplex-link, không lỗi. Đường truyền từ các nút nguồn (S1, S2…) đến R1 có băng thông 10Mbps, độ trễ 5ms; đường

truyền từ R2 đến các nút nhận Dest có băng thông là 1Mbps, độ trễ là 10ms. Các kết nối cùng chia sẻ đường truyền chung R1- R2 có băng thông là 1Mb và độ trễ là 20ms. Hàng đợi Q được đặt giữa R1-R2 có kích thước tối đa 100 gói tin, các gói tin có kích thước 1000 bytes. Thời gian mô phỏng là 80s.

- Các tham số RED được thiết lập là: min_{th} = 5, max_{th} = 15, max_p = 0.1 và w_q = 0.002 [1]
- Với A-RED, w_q được thiết lập tự động theo công thức (*), α = 0.02 và β = 0.9 [18]

Để thấy rõ ưu điểm của A-RED so với RED, cũng như việc A-RED hoạt động tốt hơn so với RED khi mạng phải chịu tải lớn (mục 2.5, 2 khuyết điểm của RED) chúng tôi đã thực hiện 2 mô phỏng:

- Lần một tôi sẽ cho mạng tăng đột biết lưu lượng
- Lần hai tôi sẽ cho mạng giảm lưu lượng đột ngột

Sau đây là kết quả mô phỏng và đánh giá từ mô phỏng:

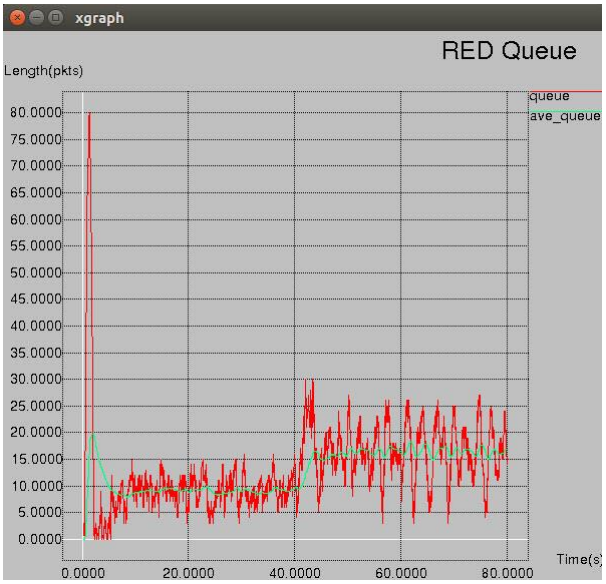
➢ Mô phỏng tăng cường độ tắc nghẽn:

Với mô phỏng này chúng tôi sẽ cho 5 luồng từ tcp1 đến tcp5 đi vào mạng từ giây thứ 0.1s và mỗi luồng cách nhau 0.1s (tương ứng là: 0.1s, 0.2s, 0.3s, 0.4s, 0.5s). Đến giây thứ 40, 50 luồng từ luồng tcp6 đến tcp50 được đưa vào mạng (mỗi luồng đưa cách nhau 0.5s) nhằm gây đột biết cho mạng (lưu lượng mạng tăng đột ngột). Sau đây là kết quả của mô phỏng so sánh giữa RED và A-RED

Bảng 2.2: Bảng kết quả thống kê của mô phỏng trường hợp 1 so sánh giữa RED và A-RED

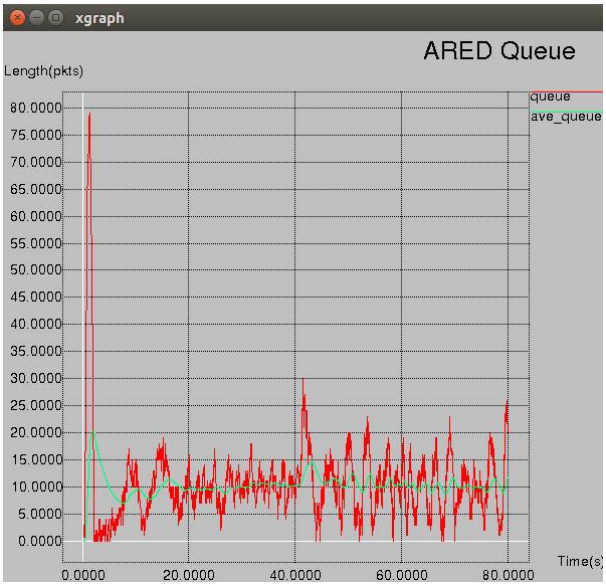
Chiến lược	Kích thước hàng đợi trung bình (gói tin)	Độ trễ hàng đợi trung bình (ms)	Hệ số sử dụng đường truyền (%)	Thông lượng tổng (%)	Tỷ lệ loại bỏ gói tin (%)
RED	13.63	113.40	99.93	0.99	16.33
A-RED	10.99	91.40	99.95	1.00	17.31

Đồ thị 2.2: Đồ thị kết quả thu được từ mô phỏng trường hợp 1 so sánh giữa RED và A-RED



a. Kích thước hàng đợi trung bình

Hình 2.12: Kết quả hàng đợi trung bình của RED trong mô phỏng trường hợp 1 so sánh RED và A-RED



a. Kích thước hàng đợi trung bình

Hình 2.13: Kết quả hàng đợi trung bình của A-RED trong mô phỏng trường hợp 1 so sánh RED và A-RED

Từ đồ thị 2.2 và bảng thống kê 2.2 thu được từ kết quả mô phỏng với cấu hình mạng mô phỏng như hình 2.11 ta thấy:

- Khi lưu lượng đưa vào mạng được tăng lên (40s), kích thước hàng đợi trung bình sẽ bị tăng lên, dẫn đến độ trễ sẽ tăng lên. Tuy nhiên nhìn vào đồ thị hình 2.12 và 2.13, ta thấy với hàng đợi cài đặt là A-RED, thì chỉ mất khoảng 4s, A-RED đã kéo hàng đợi trung bình quay trở lại mức thấp và ổn định trở lại (hình 3.7), còn RED sau khi có lưu lượng đưa vào mạng tăng lên, kích thước hàng đợi tăng và giữ nguyên cho đến hết thời gian mô phỏng.
- Nhìn vào bảng thống kê 2.2 ta có thể thấy rõ hơn những ưu điểm của A-RED so với RED như : kích thước hàng đợi trung bình thấp hơn RED (10.99 so với 13.63) dẫn đến độ trễ cũng thấp hơn, thông lượng tổng thể cao hơn RED, hệ số sử dụng đường truyền tốt hơn và tỷ lệ loại bỏ gói tin cũng cao hơn một chút.

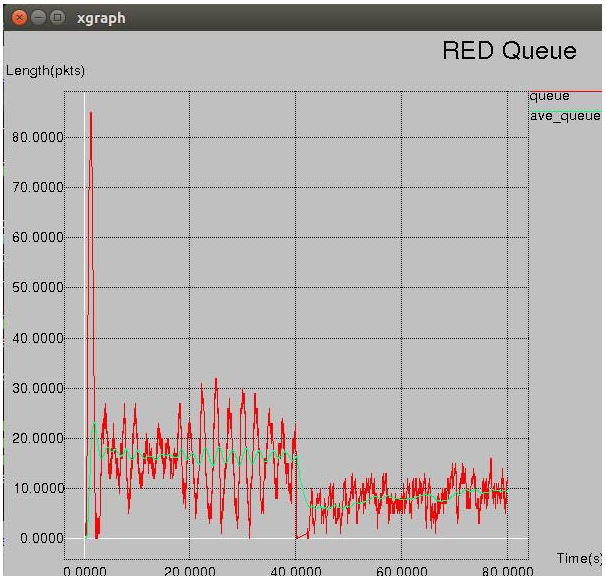
➢ Mô phỏng giảm cường độ tắc nghẽn:

Với mô phỏng này chúng tôi sẽ cho 55 luồng từ tcp1 đến tcp55 đi vào mạng từ giây thứ 0.1s và mỗi luồng cách nhau 0. Đến giây thứ 40, 50 luồng từ luồng tcp6 đến tcp50 ngừng hoạt động nhằm gây đột biết cho mạng (lưu lượng mạng giảm đột ngột). Sau đây là kết quả của mô phỏng so sánh giữa RED và A-RED

Bảng 2.3: Bảng kết quả thống kê của mô phỏng trường hợp 2 so sánh giữa RED và A-RED

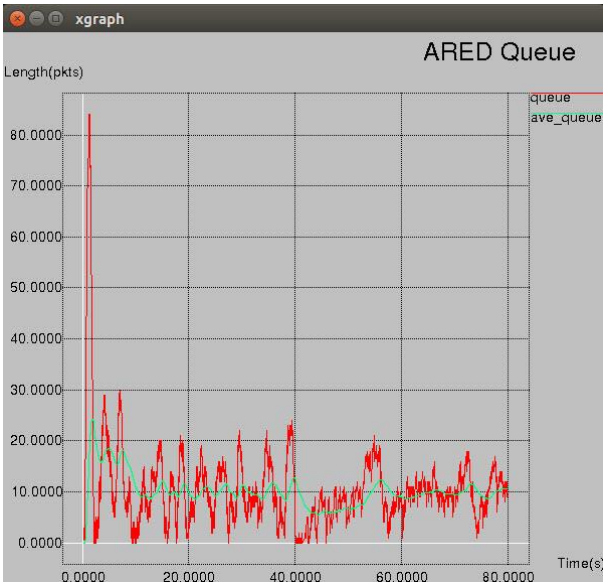
Chiến lược	Kích thước hàng đợi trung bình (gói tin)	Độ trễ hàng đợi trung bình (ms)	Hệ số sử dụng đường truyền (%)	Thông lượng tổng (%)	Tỷ lệ loại bỏ gói tin (%)
RED	12.89	107.27	98.08	0.98	17.07
A-RED	10.85	90.29	99.29	0.99	17.29

Đồ thị 2.3: Đồ thị kết quả thu được từ mô phỏng trường hợp 2 so sánh giữa RED và A-RED



a. Kích thước hàng đợi trung bình

Hình 2.14: Kết quả hàng đợi trung bình của RED trong mô phỏng trường hợp 2 so sánh RED và A-RED



a. Kích thước hàng đợi trung bình

Hình 2.15: Kết quả hàng đợi trung bình của A-RED trong mô phỏng trường hợp 2 so sánh RED và A-RED

Từ đồ thị 2.3 và bảng thống kê 2.3 thu được từ kết quả mô phỏng với cấu hình mạng mô phỏng như hình 3.7 ở trường hợp 2 này ta thấy:

- Khi lưu lượng đưa vào mạng giảm đột ngột (40s), với RED kích thước hàng đợi trung bình sẽ bị giảm xuống nhanh chóng và ổn định ở mức thấp. Với A-RED kích thước hàng đợi trung bình cũng giảm xuống nhưng không đột ngột như RED (42s) và nó nhanh chóng quay trở lại ổn định ở mức mục tiêu ban đầu (10 ± 2)

- Nhìn vào bảng thống kê 2.3 ta càng thấy rõ hơn hiệu năng của A-RED tốt hơn so với RED

CHƯƠNG 3. CÁC KẾ HOẠCH QUẢN LÝ HÀNG ĐỢI ĐỘNG CHO TRUYỀN THÔNG ĐA PHƯƠNG TIỆN TRONG KIẾN TRÚC CÁC DỊCH VỤ PHÂN LOẠI

Ở chương 2, tôi đã trình bày các nghiên cứu và đánh giá của mình về 2 thuật toán quản lý hàng đợi động AQM tiêu biểu trong kiến trúc mạng truyền thống là RED và A-RED. Tuy nhiên như đã trình bày ở chương 2, cả 2 thuật toán này chỉ được thực hiện trên kiến trúc mạng truyền thống, các luồng lưu lượng đến đều được xử lý như nhau và lưu lượng được xử lý, vận chuyển theo kiểu "cố gắng tối đa". Trong chương này tôi sẽ trình bày việc áp dụng AQM trong kiến trúc các dịch vụ phân loại DiffServ. DiffServ được định nghĩa bởi Tiểu ban đặc nhiệm kỹ thuật Internet - IETF, nhằm cung cấp cho các mạng IP sự đảm bảo chất lượng dịch vụ QoS cho các lưu lượng khác nhau cùng chia sẻ kênh truyền chung. DiffServ căn cứ vào giá trị của một trường đặc biệt trong header của gói tin IP (cụ thể là trường Diffierentiated Service (DS) trong header các gói tin được thiết lập một giá trị nào đó, gọi là Code Point (CP)) để báo cho router biết cách xử lý nào (ở đây là cách xử lý trên từng chặng - PHB nào) cần được áp dụng cho mỗi gói tin.

Các nội dung chính của chương 3 được trình bày như sau:

- Đầu tiên tôi sẽ trình bày về mô hình DiffServ. Các tham số, cấu hình… của DiffServ trong bộ mô phỏng NS2. Chương 4 cũng là phần quan trọng nhất của Luận Văn, tôi sẽ trình bày kết quả mô phỏng DiffServ, đánh giá mức độ ảnh hưởng của luồng đột biến so với các luồng ưu tiên với bộ mô phỏng NS2.
- Tiếp theo tôi sẽ trình bày về thuật toán quản lý hàng đợi động RIO. Một thuật toán mở rộng của RED áp dụng cho kiến trúc mạng DiffServ khi các gói tin đến sẽ được xử lý khác nhau tùy thuộc vào mức độ ưu tiên của nó. Phần mô phỏng so sánh và đánh giá giữa RED và RIO sẽ được tôi thực hiện ở chương 4.

3.1 Mô hình DiffServ

Như chúng ta đã biết, Internet truyền thống chạy trên nền dịch vụ BestEffort, với kiến trúc truyền thống, tất cả các gói tin khi đến hàng đợi, đều được xử lý như nhau, không cần biết gói tin đó có mức độ ưu tiên cao hay thấp. Để khắc phục vấn đề này, một kiến trúc khác đã được phát triển đó là kiến trúc IntServ, IntServ (Integrated Services) là kiến trúc các dịch vụ tích hợp, được phát triển bởi IETF, nhằm cung cấp khả năng đảm bảo dịch vụ cho các lưu lượng bằng cách dùng giao thức đặt trước tài nguyên mạng RSVP [12]. Theo kiến trúc của mô hình IntServ, nếu từ nguồn đến đích có đủ tài nguyên thì các luồng lưu lượng đặt trước tài nguyên từ nguồn đó sẽ được đảm bảo dịch vụ trên đường truyền. Tuy nhiên hạn chế của IntServ là không có khả năng mở rộng theo các

luồng lưu lượng, cài đặt router phức tạp, khó quản lý, vì nó phải đảm nhiệm rất nhiều việc; các router phải thường xuyên cập nhật lại trạng thái các lưu lượng qua nó, dẫn đến làm chậm tốc độ truyền. Đã có nhiều cố gắng thay đổi các khuyến điểm của IntServ nhằm đảm bảo chất lượng dịch vụ tốt hơn cho mạng IP, DiffServ ra đời từ những vấn đề đó. IETF đã phát triển một kiến trúc mới, đó là kiến trúc dịch vụ phân loại DiffServ (Differentiated Services). Hai khả năng quan trọng nhất của DiffServ là khả năng mở rộng và khả năng linh hoạt. Khả năng mở rộng của DiffServ để phục vụ cho một số lượng lớn người dùng, cho phép tất cả kết nối cùng chia sẻ một đường truyền chung trên Internet tại một thời điểm. Khả năng linh hoạt thể hiện trong việc cung cấp các dịch vụ khác nhau cho các lưu lượng khác nhau, phân phối mềm dẻo băng thông cho mỗi kết nối khi có sự thay đổi trạng thái lưu lượng qua router [5].

Sau đây tôi xin trình bày các vấn đề sau:

- Cấu trúc DiffServ
- Đánh dấu gói DiffServ
- Đối xử theo từng chặng PHB
- DiffServ trong bộ mô phỏng NS2

3.1.1 Cấu trúc DiffServ

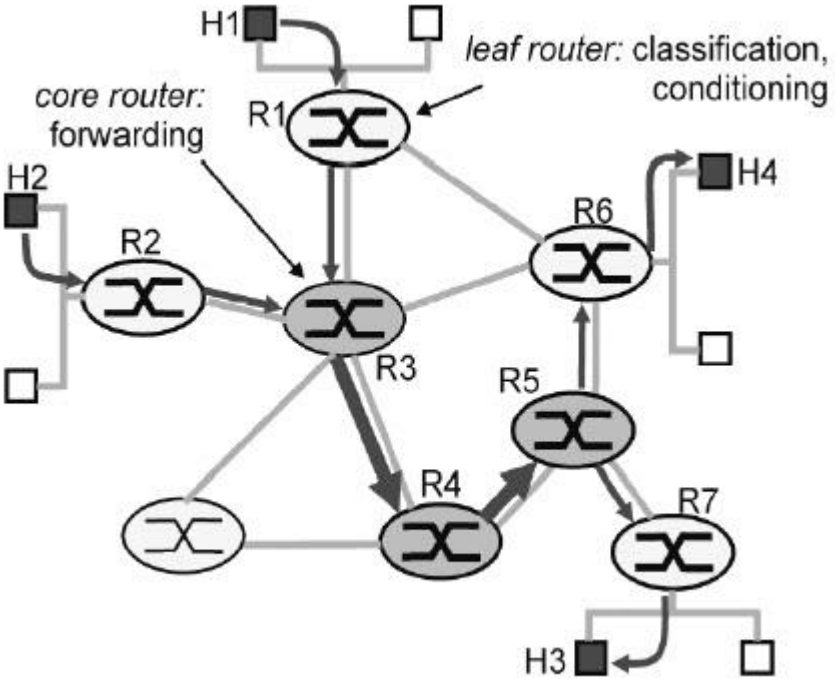
DiffServ không xử lý theo từng luồng lưu lượng riêng biệt mà ghép chúng vào một số lượng hạn chế các lớp lưu lượng. DiffServ hướng tới xử lý trong từng dịch vụ phân biệt thay vì xử lý từ đầu cuối tới đầu cuối như mô hình IntServ [6].

- Thay vì thực hiện QoS xuyên suốt và thống nhất trên cả đường truyền như mô hình IntServ, mô hình DiffServ thực hiện QoS riêng lẻ trên từng router.
- Mô hình DiffServ thực hiện quản lý tài nguyên hiệu quả do không dành riêng tài nguyên cho bất kỳ một dịch vụ nào.
- Mô hình DiffServ không thực hiện báo hiệu, bắt tay khi thiết lập luồng nên không bị mất băng thông cho phần báo hiệu nên tiết kiệm băng thông và có khả năng mở rộng lớn rất phù hợp trong mô hình hệ thống mạng lớn.

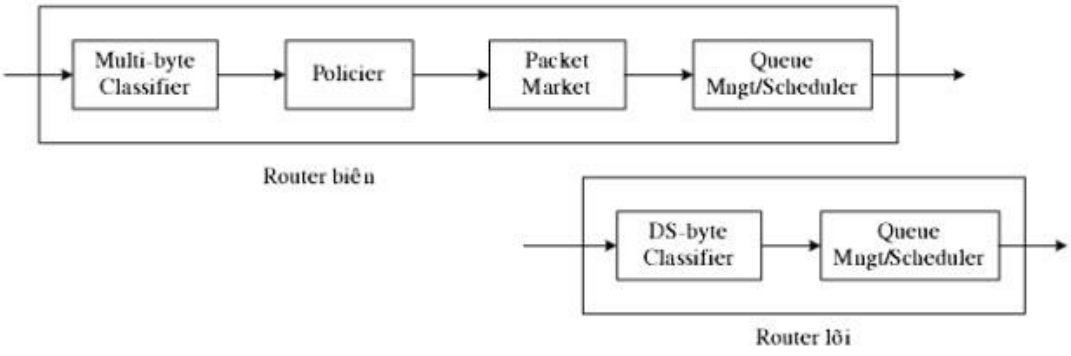
Nguyên lý hoạt động của mô hình DiffServ như sau: Các gói tin được phân loại ra thành nhiều nhóm ưu tiên từ thấp đến cao tùy theo đặc điểm của từng dịch vụ, thiết bị sẽ tiến hành cung cấp tài nguyên theo từng nhóm, nhóm nào có thứ tự cao hơn thì sẽ được cung cấp quyền được sử dụng tài nguyên ưu tiên hơn, tài nguyên sẽ được các nhóm thấp hơn dùng nếu nhóm trên không sử dụng nữa. Tất cả các quá trình này sẽ được thực hiện riêng lẻ trên từng thiết bị.

Cấu trúc của mô hình DiffServ bao gồm nhiều class lưu lượng cho từng dịch vụ cụ thể và mỗi class được cung cấp một lượng tài nguyên xác định. Để phân biệt các class, DiffServ sử dụng một thông tin gọi là điểm mã phân biệt dịch vụ DSCP (Differentiated Service Code Point). DSCP có tiền thân là vùng ToS (Type of Service) trong IP header.

DiffServ gồm 2 thành phần chính đó là mạng biên và mạng lõi.



Hình 3.1: Kiến trúc DiffServ đơn giản [5]



Hình 3.2: Mô hình DiffServ tại mạng biên và mạng lõi [6]

Mạng biên của DiffServ có chức năng phân loại các gói tin đến và đánh dấu chúng. Đây có thể là nguồn sinh lưu lượng có cài đặt các chính sách phân loại gói tin hoặc là một router có hỗ trợ DiffServ đầu tiên (router nằm ở biên), được gọi là mạng biên (edge network). Tại đây các gói tin đi đến sẽ được phân loại và đánh dấu, cụ thể là

trường Diffierentiated Service (DS) trong header các gói tin được thiết lập một giá trị nào đó, gọi là Code Point (CP). Trên hình 3.1, R1, R2 là các router biên. Các gói tin đi từ H1 đến H3 được đánh dấu tại R1, trong khi các gói tin gửi từ H2 đến H4 được đánh dấu tại R2. Dấu mà một gói tin nhận được đánh xác định lớp lưu lượng mà chúng thuộc về. Các lớp lưu lượng khác nhau sẽ nhận được dịch vụ khác nhau trong phần mạng lõi. Sau khi đã được đánh dấu, một packet có thể được chuyển trực tiếp vào trong [5, 6].

Mạng lõi của DiffServ có chức năng chính là chuyển tiếp gói tin. Khi một gói tin đã được đánh dấu đi đến, gói tin được chuyển tiếp tới chặng tiếp theo thông qua chính sách từng chặng (PHB Per Hop Behavior) được định nghĩa trước. PHB ảnh hưởng đến việc chia sẻ băng thông và bộ đệm giữa các lớp lưu lượng cạnh tranh tại router. Một nguyên tắc chính của cấu trúc DiffServ là PHB của một router sẽ chỉ dựa trên duy nhất dấu của gói tin, mà không cần biết gói tin đó đến từ đâu. Ví dụ, nếu các gói tin được gửi từ H1 đến H3, hoặc từ H2 đến H4 nhận được cùng một dấu, thì tại các router thuộc phần mạng lõi chúng sẽ được đối xử như nhau, mà không cần biết chúng xuất phát từ H1 hay H2. Do đó cấu trúc DiffServ không đòi hỏi các router phải lưu giữ các thông tin trạng thái cho từng cặp nguồn đích, giảm bớt gánh nặng xử lý và tài nguyên đặt lên router

3.1.2 Đánh dấu gói DiffServ

Như đã trình bày ở phần 3.1.1, một gói tin sẽ được đánh dấu tại mạng biên, trước khi vào mạng lõi. Ở đây, chúng ta sẽ giả thiết rằng việc đánh dấu xảy ra ở một router được nối trực tiếp với bên gửi, gọi là router biên. Hinh 3.2 minh họa chức năng phân loại và đánh dấu gói tin ở mạng biên và chính sách chuyển tiếp PHB ở mạng lõi.

Việc phân loại và đánh dấu sẽ giúp thực hiện các cơ chế QoS ở những bước sau:

- **Quản lý tắc nghẽn:** Cơ chế quản lý tắc nghẽn được thực hiện trên các interface của thiết bị mạng. Khi gói tin đến các interface này, các gói tin sẽ được phân vào từng hàng đợi có mức độ ưu tiên khác nhau.
- **Tránh tắc nghẽn:** Cơ chế loại bỏ gói tin khỏi hàng đợi trước khi hàng đợi đây (nếu hàng đợi đầy có thể gây ra hiện tượng tắc nghẽn).
- **Đặt ngưỡng:** Cơ chế đặt ngưỡng trên, ngưỡng dưới cho bandwidth. Bandwidth sẽ được đảm bảo một ngưỡng dưới tối thiểu và khi lớn hơn ngưỡng trên thì gói tin có thể bị đánh rớt hay được đưa vào hàng đợi.
- **Nén Header:** Header chiếm phần lớn trong 1 gói tin nhưng không mang thông tin thật sự, cơ chế nén header sẽ giúp tiết kiệm được băng thông (nhờ làm giảm số lượng bits truyền đi).

- **Fragmentation (phân mảnh):** Các gói tin có độ dài lớn có thể gây ra delay và tắc nghẽn. Cơ chế phân mảnh sẽ phân các gói tin này thành các gói tin nhỏ hơn để tránh tắc nghẽn.

Bộ phân loại (Classifier) sẽ đánh dấu các gói tin dựa vào nhiều yếu tố như tiêu đề của gói tin, ví dụ, địa chỉ nguồn, địa chỉ đích, cổng nguồn, cổng đích,… hoặc dựa trên tốc độ đến của gói tin so với các ngưỡng nào đó. Giá trị trường DS sau đó sẽ nhận được một giá trị tại bộ đánh dấu (Marker). Trong phần mô phỏng DiffServ kết hợp RIO dưới đây, tôi sử dụng một thuật toán đánh dấu gọi là srTCM (Single Rate Three ColorMaker), nó sử dụng các tham số CIR, CBS, EBS để đánh dấu các gói tin và phân loại các gói tin vào 3 hàng đợi ảo trong NS2, phần này sẽ được tôi trình bày bằng mô phỏng ở phần dưới. Ngay khi các gói tin được đánh dấu, chúng được chuyển tiếp cho router tiếp theo để đi tới đích. Tại mỗi router hỗ trợ DiffServ sau đó, những gói tin đã được đánh dấu này nhận được dịch vụ theo các dấu của chúng.

3.1.3 Đối xử theo từng chặng PHB

Khi router nhận gói tin với giá trị DSCP nào đó thì chính giá trị DSCP cho biết yêu cầu QoS cho gói tin đó. DSCP sẽ xác định một hành vi Perhop Behavior (PHB). Hành vi PHB dùng để kích hoạt và hỗ trợ QoS cho các gói tin được đánh dấu bằng giá trị DSCP Chính sách đối xử từng chặng PHB được cài đặt bên trong các router ở phần mạng lõi. PHB có những đặc điểm sau:

- Các lớp lưu lượng khác nhau, tức là các lớp được đánh dấu khac nhau sẽ nhận được các phục vụ khác nhau (cách chuyển tiếp khác nhau).
- PHB không chỉ định cụ thể cơ chế nào phải được áp dụng để đạt được mục đích, vì vậy bất kỳ một cơ chế quản lý cấp phát tài nguyên (băng thông/bộ đệm) hay cơ chế chuyển tiếp nào cũng có thể được sử dụng miễn là đạt được tiêu chí hiệu năng mong muốn.

Như việc, PHB không yêu cầu một chính sách quản lý hàng đợi cụ thể nào, vì vậy tất cả các chính sách như FIFO, PQ hay RED, RIO… đều có thể được sử dụng. Trong bộ mô phỏng NS2 các chính sách quản lý hàng đợi được hỗ trợ mặc định như: WRED, RIO-C, RIO-D, DROP… (xem chi tiết trong file: ns-2.35\diffserv\ dsred.cc)

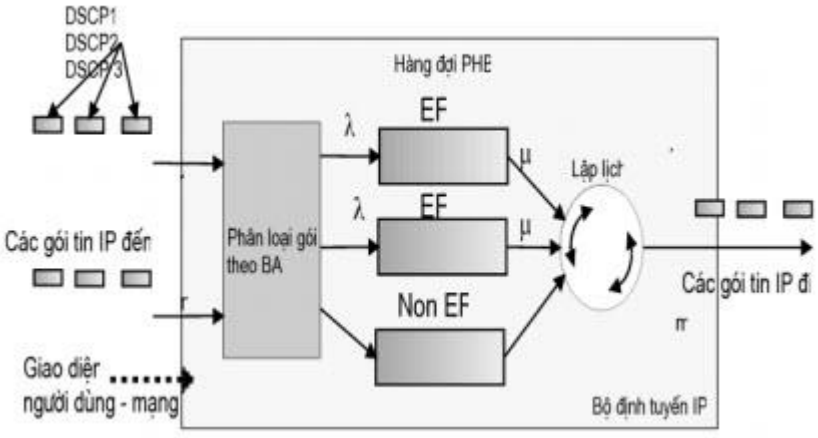
Sau đây là một số giá trị PHB chuẩn:

- **Giá trị mặc định (Default):** Tương đương với yêu cầu Best-Effort.
- **Expedited Forwarding (EF) PHB** - Chuyển tiếp nhanh: Gói tin được gán giá trị này sẽ có độ delay nhỏ nhất và packet loss thấp nhất.

- **Assured Forwading (AF) PHB** - Chuyển tiếp bảo đảm: Những gói tin có thể được đảm bảo chọn cho một PHB với yêu cầu bandwidth, delay, jitter, packet loss hoặc cả độ ưu tiên cho truy cập đến dịch vụ mạng.

Hiện tại có hai chuẩn PHP được quan tâm nhiều nhất, đó là: **Expedited Forwarding** (EF) và **Assured Forwarding** (AF).

- **EF** PHB chỉ định rằng tốc độ ra khỏi router của một lớp lưu lượng nào đó phải bằng hoặc vượt quá một tốc độ cho trước. Nghĩa là trong mọi khoảng thời gian, lớp lưu lượng đó được đảm bảo nhận được đủ băng thông sao cho tốc độ ra của nó bằng hoặc vượt quá tốc độ được đặt trước này. EF PHB đảm bảo ngay cả khi có nhiều lớp lưu lượng khác lấn át router và tài nguyên của đường truyền, thì một lượng đủ tài nguyên vẫn phải được tạo sẵn cho lớp để đảm bảo rằng nó nhận được một tốc độ tối thiểu [5]. Chuyển tiếp nhanh được yêu cầu đưa ra các dịch vụ với khả năng tổn hao thấp, trễ thấp, thay đổi trễ thấp và đảm bảo băng thông. Một bộ định tuyến EF phải đảm bảo lưu lượng EF được đưa đến những bộ nhớ đệm nhỏ vì rung pha và trễ gây nên bởi thời gian mà gói sử dụng trong bộ nhớ đệm và hàng đợi. Khi xảy ra hiện tượng quá tải, nút biên miền DS không cho phép lưu lượng dạng này đi vào trong miền vì nó là nguyên nhân gây tắc nghẽn tại các bộ định tuyến trong miền DS. Vấn đề này được điều chỉnh bởi thỏa thuận mức dịch vụ SLA (Service Level Argreement) và xác định lưu lượng truyền có điều kiện [6]



Hình 3.3: Xử lý chuyển tiếp nhanh EF PHB

- **AF** PHB phức tạp hơn. Chuyển tiếp đảm bảo với đặc điểm phân phối dữ liệu đảm bảo với khả năng mất gói thấp là điều kiện tốt nhất khi sử dụng các giao thức không thực hiện xử lý sửa lỗi hoặc không có giải pháp truyền lại gói. AF PHB bao gồm 4 lớp chuyển tiếp và mỗi lớp chuyển tiếp có 3 mức ưu tiên loại bỏ gói

tin, mỗi lớp được gán một băng thông và khoảng nhớ đệm xác định [6]. Khi tắc nghẽn xuất hiện trong một lớp AF, router có thể loại bỏ các gói tin dựa trên giá trị ưu tiên loại bỏ (drop preference) của chúng. Bằng cách thay đổi lượng tài nguyên cấp phát cho từng lớp, một nhà cung cấp dịch vụ Internet (ISP) có thể cung cấp các mức độ thực hiện khác nhau cho các lớp lưu lượng AF khác nhau.

Hình 3.4: Các phân lớp AF PHB [6]

AF PHB có thể được sử dụng để cung cấp các mức độ dịch vụ khác nhau cho các hệ thống cuối. Trong các hệ thống này việc quyết định số lượng tài nguyên cấp phát cho từng lớp dịch vụ phải gắn liền với việc giới hạn các tham số liên quan. Chẳng hạn có 3 lớp dịch vụ khác nhau là A, B, C, giả sử A được cung cấp x% băng thông tổng cộng, B được cấp x/2% và không còn ràng buộc gì thêm. Như vậy bề ngoài có thể nói lớp A được cấp phát dịch vụ tốt hơn lớp B, tuy nhiên nếu lưu lượng của các gói tin thuộc lớp A cao gấp nhiều lần so với lớp B, thì thực ra các gói tin lớp A được phục vụ ít hơn lớp B. Như vậy trong trường hợp này, kích thước tài nguyên là chưa đủ để đánh giá chất lượng dịch vụ của một lớp, còn phải có thêm ràng buộc về tốc độ dữ liệu tối đa cho các gói tin của lớp đó nữa [5].

Trong bộ mô phỏng NS-2, các lớp AF được cài đặt với các hàng đợi vật lý khác nhau, mỗi hàng đợi vật lý đó lại có tương ứng các hàng đợi ảo. Mỗi gói tin với các mức ưu tiên khác nhau sẽ được đưa vào một hàng đợi ảo tượng ứng.

3.1.4 DiffServ trong bộ mô phỏng NS2

Trong bộ mô phỏng NS2, DiffServ được hỗ trợ 4 lớp traffic, mỗi lớp có 3 loại quyền ưu tiên loại bỏ cho phép ta xử lý lớp traffic theo nhiều cách khác nhau. Các gói tin trong một lớp đơn sẽ được đưa vào trong 1 hàng đợi vật lý, DiffServ trong NS2 được hỗ trợ với 4 hàng đợi vật lý, mỗi hàng đợi vật lý được khuyên dùng tối đa là 3 hàng đợi ảo. Các tham số này được thể hiện trong code thông qua phần cài đặt:

```
set  numQueues_ 4 ;#hàng đợi vật lý
```

setNumPrec 3 ;#hàng đợi ảo

Chức năng, hàm, các phương thức… của DiffServ trong NS2 nằm trong bộ Source cài đặt NS2 ở thư mục: \ns-2.35\diffserv

Tại các hàng đợi ảo chúng ta có thể cấu hình để ưu tiên các gói tin theo ý muốn. Mỗi gói tin đến sẽ được gắn với 1 giá trị Code Point và các tham số tương ứng với hàng đợi mà chúng ta sử dụng.

DiffServ trong NS2 gồm 3 thành phần chính đó là:

- **Policy:** bộ luật (chính sách), nó được thiết lập bởi người quản trị mạng, nó cung cấp các mức dịch vụ mà luồng đến được nhận.
- **Edge router:** router biên có nhiệm vụ đánh dấu CP vào gói tin. Giá trị CP này tương ứng với giá trị ở bộ luật Policy trên.
- **Core router:** các router trong lõi có nhiệm vụ thực hiện các bộ luật dựa trên CP của các gói tin.

Hàng đợi mặc định được sử dụng trong module DiffServ là RED (mục 2.4). Tuy nhiên RED trong DiffServ có những điểm khác với RED mặc định được dùng trong NS2. RED trong DiffServ được định nghĩa trong lớp dsREDQueue, thừa kế từ lớp Queue. Trong mô hình DiffServ được sử dụng trong bộ mô phỏng NS2, có 4 hàng đợi động được support mặc định ngay khi ta cài đặt NS2 (ở đây tôi dùng phiên bản NS-2.35) là: DROP, WRED, RIO-C, RIO-D. Trong phần mô phỏng, tôi sẽ mô phỏng DiffServ áp dụng hàng đợi RIO-C và bộ đánh dấu srTCM.

Với NS2, Hàng đợi trong DiffServ được cài đặt thông qua 2 khai báo cơ bản: 1/ setMREDMode “hàng đợi muốn dùng”. Ví dụ: setMREDMode RIO-C. 2/ set limit_ “kích thước hàng đợi”. Ví dụ: set limit_ 100

Trong NS2 bảng PHB được định nghĩa là một mảng gồm 3 trường tượng ứng là: codePt_ queue_ prec_ [11, tr.88]

```
struct phbParam {
    int codePt_; // corresponding code point
     int queue_; // physical queue
    int prec_; // virtual queue (drop precedence)
};
```

Ví dụ PHB trong NS 2 được code là: addPHBEntry 10 0 0 ;#Code Point là 10, hàng đợi vật lý 0 và hàng đợi ảo 0.

Các khai báo mặc định của NS2 chúng ta có thể xem chi tiết ở: **ns-2.35/tcl/libs/ns-default.tcl** [13].

- **Bộ luật (chính sách) policy trong DiffServ với NS2:**

Lớp Policy và những lớp con của nó định nghĩa các phương án xử lý gói tin được đánh dấu ngoài mạng biên. Một policy được xác định bởi nút nguồn và nút đích, tất cả những luồng có chung nút nguồn và nút đích đều được quy về 1 traffic đơn. Policy khi khai báo được dùng với 3 tham số chính: policier type, meter type và initial code point. Khi một gói tin tới mạng biên, nó sẽ được phân tích để biết nó thuộc luồng nào, sau đó policier sẽ đánh dấu gói tin đó dựa trên việc phân tích luồng, tiếp theo gói tin sẽ được cho vào hàng đợi.

Trong NS2, Policy table (bảng chính sách) được dùng để lưu trữ các bộ luật của mỗi luồng đến. Bảng này là tập hợp của nhiều mảng, mỗi mảng gồm có nhiều trường, và tùy thuộc vào kiểu luật (policer type) mà có trường được sử dụng và trường không cần sử dụng. Sau đây là 16 trường trong Policy Table, sẽ dựa vào ví dụ để chỉ ra một số trường tương ứng trong khai báo:

Ví dụ về 1 khai báo luật trong code: addPolicyEntry [$S id] [$Dest id] srTCM 10 $CIR $CBS $EBS

1: Source node ID: Định danh nốt nguồn. Ở ví dụ là [$S id] (trả về một số duy nhất)

2: Destination node ID: Định danh node đích. Ở ví dụ là [$Dest id]

3: Policer type: Kiểu policer. Ở ví dụ là srTCM. Có 5 kiểu được định nghĩa như sau:

- TSW2CM (TSW2CMPolicer): Time Sliding Window with 2 Color Marking – Cửa sổ trượt theo thời gian với việc đánh dấu 2 màu, sử dụng biến CIR (trường thứ 6) và 2 hàng đợi ảo. Những hàng đợi ảo có số thấp hơn sẽ loại bỏ gói tin khi ngưỡng CIR bị vi phạm.
- TSW3CM (TSW3CMPolicer): Time Sliding Window with 2 Color Marking - Cửa sổ trượt theo thời gian với việc đánh dấu 3 màu, sử dụng biến CIR (trường thứ 6), PIR (trường thứ 11) và 3 hàng đợi ảo. Khi PIR bị vi phạm thì sẽ loại bỏ gói tin trong hàng đợi ảo ưu tiên thấp nhất. Khi CIR bị vi phạm sẽ loại bỏ gói tin ở hàng đợi ảo thấp thứ 2.
- Token Bucket (TokenBucketPolicer): Sử dụng CIR, CBS và 2 hàng đợi ảo. Mỗi gói tin sẽ bị ưu tiên loại bỏ bằng cách đặt vào hàng đợi ảo thấp hơn khi kích cỡ của nó lớn hơn CBS và tần suất gửi vi phạm CIR.
- Single Rate Three Color Marker (srTCMPolicer): Sử dụng CIR, CBS, EBS để ưu tiên gói tin từ 3 hàng đợi ảo.

- Two Rate Three Color Marker (trTCMPolicer): Sử dụng biến CIR, CBS, PIR và PBS để ưu tiên gói tin từ 3 hàng đợi ảo.

4: Meter type: kiểu đo

5: Initial Code Point: Code Point ban đầu.

6: CIR - Committed Information Rate (bps): tốc độ cam kết đảm bảo

7: CBS - Committed Burst Size (bytes): kích thước khối dữ liệu bùng nổ được cam kết.

8: C bucket (Current size of the committed bucket): tốc độ truyền hiện tại được cam kết.

9: EBS - Excess Burst Size (bytes): kích thước khối dữ liệu bùng nổ vượt mức được cam kêt.

10: E bucket - Current size of the excess bucket: kích thước hiện tại của khối dữ liệu vượt mức được cam kết.

11: PIR - Peak Information Rate: tốc độ truyền dữ liệu cực đại.

12: PBS - Peak burst size: kích thước khối dữ liệu bùng nổ cực đại.

13: P bucket (Current size of the peak bucket)

14: Arrival time of the last packet: thời gian tới của gói tin cuối cùng.

15: Average sending rate: tốc độ gửi trung bình.

16: TSW window length: độ dài cửa sổ TSW

3.2 Thuật toán RIO

3.2.1 Ý tưởng của RIO

RIO là một mở rộng của RED để bước đầu hỗ trợ cho đảm bảo chất lượng dịch vụ QoS. Chúng ta biết rằng chính sách của RED hay A-RED đối xử một cách công bằng với tất cả các gói tin đến mà không cần biết mức độ ưu tiên của gói tin đó. Trên thực tế, người dùng hoàn toàn có quyền yêu cầu các mức chất lượng dịch vụ khác nhau tuỳ theo mức giá cả thoả thuận với nhà cung cấp dịch vụ mạng và những nhà cung cấp dịch vụ phải có trách nhiệm đáp ứng được điều đó. Vì thế nếu dùng RED hay A-RED thì sẽ dẫn đến việc không công bằng khi tất cả các nguồn đều được xử lý như nhau.

Từ nhu cầu đó, David D. Clark và Wenjia Fang đã đề xuất thuật toán RIO [9] cải tiến từ RED bằng cách phân loại các gói tin đến theo hai mức ưu tiên. Việc phân loại các gói tin đến theo mức độ ưu tiên được RIO thực hiện bằng cách gắn thẻ "*In*" hoặc "*Out*" cho mỗi gói tin đến dựa trên thoả thuận chung giữa khách hàng và nhà cung cấp dịch vụ. Theo đó, gói tin được gắn thẻ "*In*" (in-profile) nếu nó là gói tin ưu tiên đã được thỏa

thuận; ngược lại gói tin được gắn thẻ "*Out*" (out-of-profile) khi nó nằm ngoài thỏa thuận, hay được hiểu là gói tin không ưu tiên. Khi trong mạng có tắc nghẽn, thì việc loại bỏ gói tin sẽ theo cơ chế là gói tin có mức độ ưu tiên thấp hơn sẽ bị loại bỏ trước, ở đây là gói tin có gắn thẻ "Out" sẽ bị loại trước. Một đặc điểm nữa của RIO là tất cả các gói tin đến đều được gộp chung vào trong một hàng đợi duy nhất, điều này, theo [9] là phù hợp đối với mạng ngày nay.

RIO viết tắt của **RED with In/Out bit**. RIO kế thừa tất cả các ưu điểm của RED như đạt được thông lượng cao, độ trễ thấp, kích thước hàng đợi trung bình nhỏ, hấp thu đột biến tốt, ngoài ra khác với RED, RIO còn phân loại các gói tin đến, nó phân biệt đối xử với các gói tin *Out* trong thời gian tắc nghẽn. Ở những *giờ cao điểm* (thời điểm có mức tắc nghẽn cao), RIO sử dụng bộ đôi thuật toán RED cho việc loại bỏ gói tin, một cho các gói *In* và một cho các gói *Out*. Bằng cách thiết lập các bộ tham số cho hai thuật toán khác nhau, RIO có khả năng ưu tiên các gói *In* và loại bỏ tích cực các gói *Out* hơn.

RIO là một kỹ thuật AQM cơ bản phù hợp với việc thiết lập xử lý từng chặng theo chuẩn AF. Như vừa trình bày, nguyên lý của RIO là phân loại và đánh dấu các gói tin đến thành các gói In hoặc Out trước khi chuyển tiếp nó. RIO đã được mở rộng để xử lý n mức độ ưu tiên (nhiều hơn 2 mức độ ưu tiên) theo một nguyên lý tương tự. Khi đó xác suất loại bỏ các gói tin có mức ưu tiên j ($1 \leq j < n$) phụ thuộc vào kích thước trung bình của *hàng đợi ảo mức j* (là hàng đợi tạo bởi chỉ các gói tin có mức ưu tiên từ 1 đến j). Đối với các gói tin có mức ưu tiên n thì xác suất loại bỏ là một hàm của kích thước hàng đợi "vật lý" (hàng đợi tổng cộng-*total queue*). Phương pháp gốc này có tên là RIO-C (*RIO-Coupled*) [14]

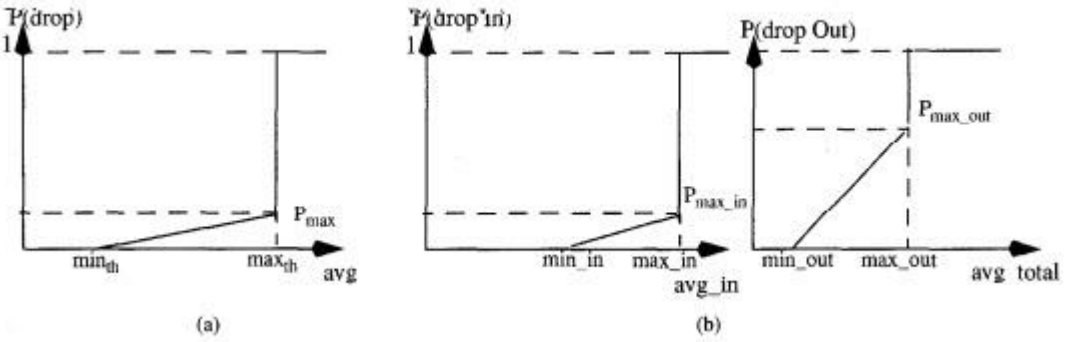
3.2.2 Thuật toán RIO

Vì thuật toán RIO kế thừa lại của RED, chỉ khác có phân loại và đánh dấu các gói tin đến là In hay là Out dựa vào các cơ chế đánh dấu khác nhau. Ở đây tôi ví dụ trường hợp đơn giản là các gói tin đến chúng ta chỉ có 2 mức ưu tiên là In và Out. Lúc này với mỗi mức ưu tiên chúng ta sẽ có 3 bộ tham số tương ứng (mục 2.4.4). Sau khi các gói tin đi đến được đánh dấu, tương tự như thuật toán của RED (mục 2.4.4), RIO sẽ tiến hành so sánh kích thước hàng đợi trung bình với 2 ngưỡng minth và maxth. Tuy nhiên vì nó 2 mức ưu tiên nên thuật toán của RIO sẽ khác đôi chút. Nếu gói tin đến là In, gateway sẽ tính avg_in (kích thước hàng đợi trung bình của gói tin In), nếu gói đến là out, gateway sẽ tính avg_total (kích thước trung bình của cả gói In và Out – kích thước trung bình của toàn bộ gói tin đến). Do đó xác suất loại bỏ gói tin In sẽ phụ thuộc vào avg_in, xác suất loại bỏ gói tin Out sẽ phụ thuộc và avg_total. Thuật toán của RIO được viết lại như hình 3.6 [9]

```
For mỗi gói tin đến
    if nó là gói tin In
        tính kích thước trung bình hàng đợi In: avg_in;
    tính kích thước trung bình hàng đợi tổng avg_total;
if nó là gói tin In
    if min_in < avg_in < max_in
        tính xác suất P_in;
        loại bỏ gói tin này với xác suất P_in;
    else if max_in < avg_in
        loại bỏ gói tin này.
if nó là gói tin Out
    if min_out <avg_total < max_out
        tính xác suất P_out;
        loại bỏ gói tin này với xác suất P_out;
    else if max_out <avg_total
        loại bỏ gói tin này.
```

Hình 3.5: Thuật toán RIO

Như đã trình bày ở trên, với mỗi mức ưu tiên chúng ta sẽ có 3 bộ tham số tương ứng như của RED (mục 2.4.4). Ví dụ với gói tin In, ba tham số lần lượt là **min_in**, **max_in** và **Pmax_in** xác định 3 miền tương tự RED: miền bình thường **[0, min_in)**, miền trách tắc nghẽn **[min_in, max_in)**, và miền điều khiển tắc nghẽn **[max_in,)**. Tương tự ba tham số **min_out, max_out**, và **Pmax_out** xác định miền ứng với thuật toán cho các gói tin Out.



Hình 3.6: Thuật toán RED (a) và RIO (b) [9]

Việc phân biệt đối xử với các gói tin Out được thực hiện bằng cách lựa chọn cẩn thận các tham số **(min_in, max_in, Pmax_in)**, và **(min_out, max_out, Pmax_out)**, mục tiêu hướng đến là RIO sẽ loại bỏ các gói tin Out nhanh hơn so với các gói In, bằng cách chọn **min_out** bé hơn **min_in**. Trong pha tránh tắc nghẽn, nó loại bỏ gói tin Out

với xác suất lớn hơn so, bằng cách thiết lập **pmax_out** lớn hơn **pmax_in**. Cuối nó với pha điều khiển tắc nghẽn đối với các gói tin Out sớm hơn so với các gói tin In, bằng cách cho **max_out** nhỏ hơn **max_in**. Thực tế, các gói tin Out sẽ bị loại bỏ khi có tắc nghẽn xảy ra, và sẽ loại bỏ tất cả khi tắc nghẽn đó kéo dài, gói tin được đánh dấu In chỉ bị loại bỏ trong trường hợp bất khả kháng, và điều này là không bảo giờ xảy ra với mạng được điều khiển tốt, khi gói tin In bị loại bỏ trong pha điều khiển tắc nghẽn là một việc không được chấp nhận trong thực tế.

Việc lựa chọn **avg_total** để quyết định xác suất loại bỏ các gói tin **Out** là một lựa chọn không hề đơn giản. Khác với các gói In – các gói được phục vụ với quyền ưu tiên cao vì đã đăng ký sử dụng, các gói tin **Out** chính là các lưu lượng của những luồng không đăng ký dịch vụ nhưng vẫn sử dụng mạng, và không có một dấu hiệu hợp lệ nào cho việc quyết định bao nhiêu gói tin **Out** là thích hợp. Nếu ta dùng kích thước hàng đợi trung bình của các gói **Out** cho việc loại bỏ các gói tin **Out**, thì việc chọn các tham số tương ứng rất khó, và không có sự tương quan rõ ràng nào so với các tham số đối với các gói In. Bằng cách sử dụng **avg_total** gateway có thể giữ được kích thước hàng đợi nhỏ và thông lượng cao bất kể lưu lượng nào trộn lẫn vào.

Ở trong chương tiếp theo (chương 4), tôi sẽ đánh giá và nhận xét từ mô phỏng so sánh giữa RED và RIO, sau đó là phần mô phỏng chính quan trọng của Luận Văn để đánh giá mức ảnh hưởng của các luồng lưu lượng đột biết lên các luồng ưu tiên trong mạng kiến trúc DiffServ.

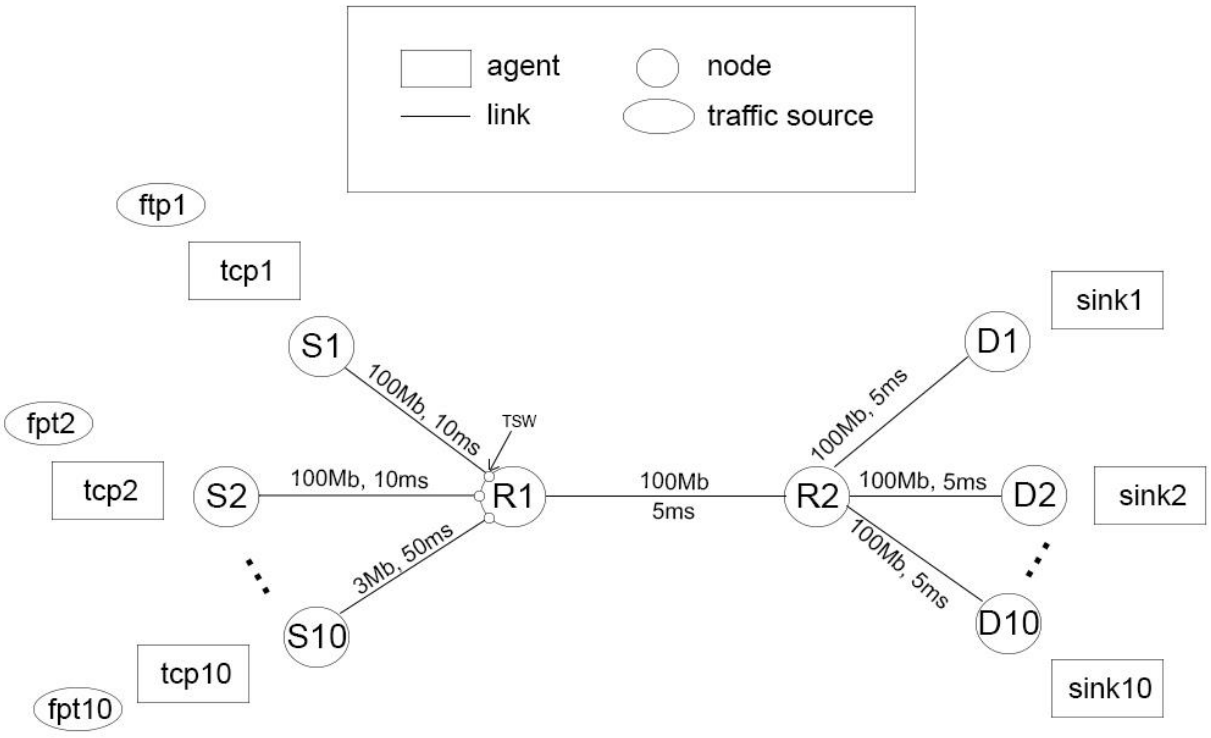
CHƯƠNG 4. ĐÁNH GIÁ RED, RIO VÀ SỰ ẢNH HƯỞNG CỦA LUỒNG ĐỘT BIẾN GÂY RA CHO CÁC LUỒNG ƯU TIÊN TRONG KIẾN TRÚC MẠNG DIFFSERV, SỬ DỤNG AQM RIO BẰNG MÔ PHỎNG

Ở chương này tôi sẽ tiến hành mô phỏng, đánh giá, so sánh giữa RED và RIO. Mô phỏng RIO trong kiến trúc mạng DiffServ, đánh giá sự ảnh hưởng của lưu lượng đột biến tác động thế nào lên các lưu lượng đang tồn tại trong mạng theo mức độ ưu tiên khác nhau. Đầu tiên chúng ta sẽ đến với phần mô phỏng đánh giá, so sánh giữa RED và RIO.

4.1 Đánh giá RIO và so sánh với RED

Tham khảo tài liệu [9].

4.1.1 Cấu hình mạng mô phỏng



Hình 4.1: Cấu hình mạng mô phỏng so sánh RIO và RED

Mạng mô phỏng gồm 22 node, 10 node nguồn, 10 node đích, các luồng đi từ nguồn đến đích đi qua kênh truyền chung R1 – R2. Hàng đợi Q đặt ở R1-R2 có kích thước 100 gói tin. Ở mô phỏng này để đánh giá RED và RIO, sự ảnh hưởng của độ trễ RTT đến thông lượng đường truyền và sự phân loại các luồng ưu tiên, đánh dấu In, Out cho các luồng đi đến. Có 10 đường truyền chia thành 5 cặp, mỗi cặp sẽ có chung độ trễ như nhau và tăng lần lượt từ 10ms đến 50ms. Với RIO, tôi sẽ sử dụng kỹ thuật đánh dấu TSW [7, mục 3.1.4], chia thành 2 cặp tương ứng: nguồn 1,3,5,7,9 sẽ có RT (Mbps) là 1Mbps, các nguồn còn lại có RT (Mbps) là 5 Mbps, ở đây ta coi như có 2 luồng với 2 mức độ ưu tiên khác nhau, một luồng được ưu tiên gấp 5 lần luồng kia. Kỹ thuật đánh dấu TSW sẽ đánh dấu các gói tin thành In hoặc Out [9], tùy thuộc vào tốc độ đến của chúng so với tốc độ trung bình RT nào đó. Nếu gói tin đến có tốc độ đến lớn hơn RT nó sẽ bị đánh dấu là Out và ngược lại sẽ có dấu là In. Các luồng khi đi đến R1 sẽ được đánh dấu tại lối vào R1 (coi như là một router biên)

- R1 – R2 : 30Mbps, 5ms

Tất cả các luồng sau khi được đánh dấu sẽ được chuyển tiếp trong một lớp AF Các luồng TCP được đưa vào mạng cùng 1 lúc ở 0.1s và kết thúc ở 20s. Thời gian mô phỏng là 20s, kích thước các gói tin là 1000 bytes. Các tham số sử dụng tương ứng với RED và RIO là:

- **RED:** min_{th} = 10, max_{th} = 30, max_p = 0.1, wq = 0.002.
- **RIO:** min_in = 40, max_in = 80, $pmax_{in}$ = 0.02, w_{qin} = 0.002; min_out = 10, max_in = 30, $pmax_{in}$ = 0.1, w_{qout} = 0.01; [9]

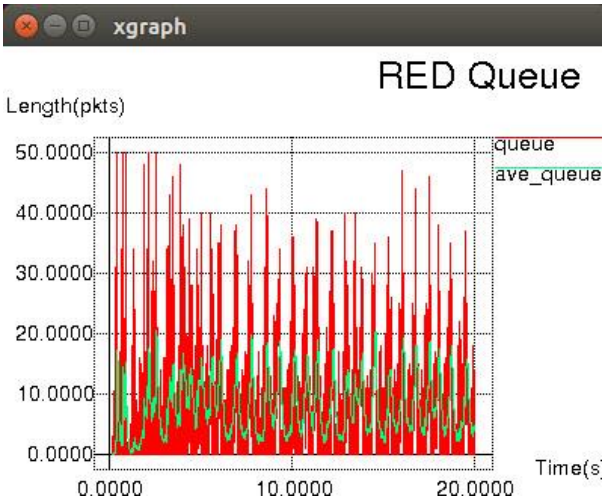
Chú ý: Thông lượng của từng luồng bằng tổng số gói tin của luồng đấy chia cho tổng thời gian mô phỏng. Chi tiết tính được tôi để ở phần phụ lục. Có một điều lưu ý là với gói tin kích thước 1000 bytes thì sẽ có thêm 40 bytes ở header được thêm [11, tr.77]

4.1.2 Kết quả mô phỏng

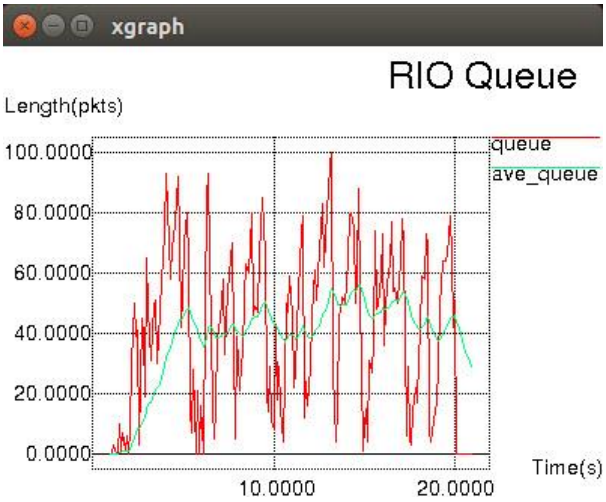
Bảng 4.1: So sánh giữa RED và RIO (RIO sử dụng bộ đánh dấu TSW)

Luồng	RTT (ms)	Thông lượng trung bình với RED (Mbps)	R_T (Mbps)	Thông lượng trung bình với RIO (Mbps)
1	10	4.90	1	3.27
2	10	5.02	5	5.95
3	20	2.83	1	3.09
4	20	2.62	5	3.99
5	30	2.01	1	1.81

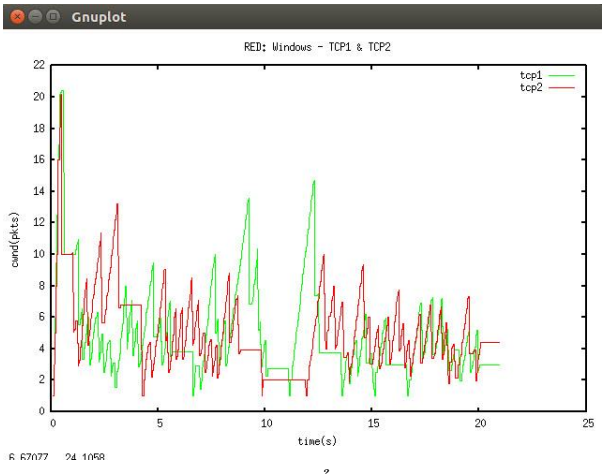
6	30	2.09	5	2.96
7	40	2.38	1	1.64
8	40	1.57	5	2.70
9	50	1.34	1	1.38
10	50	1.18	5	1.55
Total		25.93		28.34



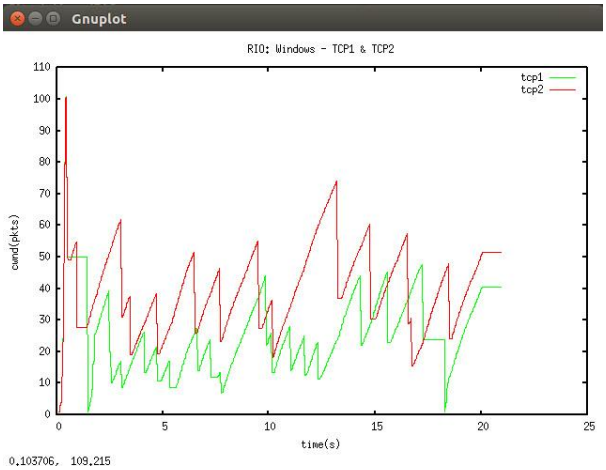
a. Kích thước hàng đợi trung bình



a. Kích thước hàng đợi trung bình



b. Kích thước cửa sổ cặp tcp1 và tcp2



b. Kích thước cửa sổ cặp tcp1 và tcp2

Hình 4.2: Kết quả mô phỏng với RED

Hình 4.3: Kết quả mô phỏng với RIO-TSW

4.1.3 Nhận xét cá nhân

Từ hình 4.1, 4.2 và bảng 4.1 ta thấy:

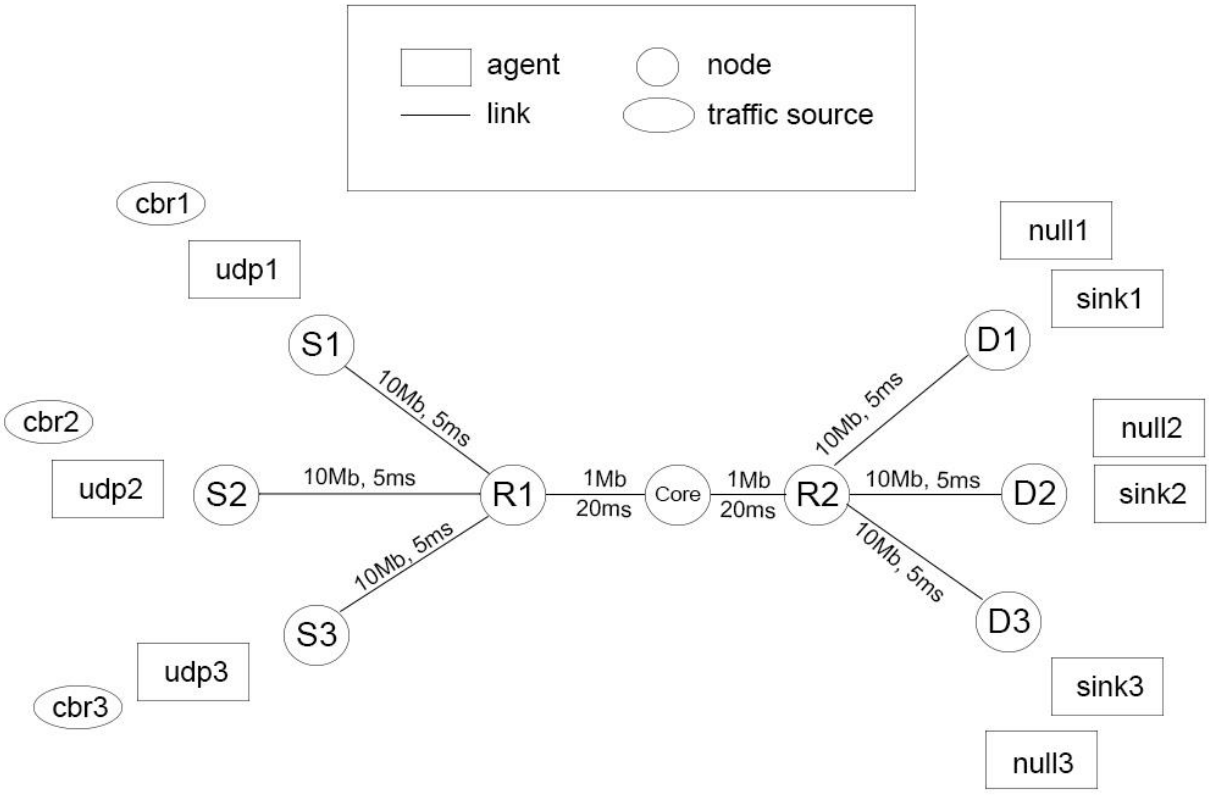
- Với RED các luồng được đối xử như nhau (Hình 4.2b), với RIO 2 luồng đã được đối xử theo cấp độ ưu tiên khác nhau (Hình 4.3b), luồng tcp2 có mức độ ưu tiên cao hơn tp1 (R_T của tcp2 cao hơn tcp 1) nên cửa sổ phát của nó luôn được phát nhiều hơn tcp1, dẫn đến băng thông của nó cũng cao hơn băng thông của luồng tcp1 (5.95 so với 3.27).
- Từ bảng 4.1 ta thấy với RED, thông lượng của các kết nối sẽ giảm dần khi độ trễ tăng cao. Kể cả khi độ trễ là như nhau, thì thông lượng của các luồng cũng chênh lệch lớn (bảng 4.1 luồng 7, 8). Với RIO ta thấy thông lượng của các luồng tỷ lệ với mức độ ưu tiên (ở đây là 1Mbps và 5Mbps), các luồng cũng có thông lượng giảm dần khi RTT tăng, nhưng các luồng ưu tiên vẫn luôn được đảm bảo đạt được thông lượng tốt hơn các luồng ưu tiên thấp hơn. Tổng thông lượng của RIO là 28.34Mbps tốt hơn so với RED là 25.93Mbps (94.48% so với 86.44%)

4.2 Mô phỏng DiffServ sử dụng AQM RIO-C, mục tiêu đánh giá sự đảm bảo chất lượng dịch vụ trong truyền thông đa phương tiện.

Mục đích của mô phỏng DiffServ với các luồng ưu tiên khác nhau, kết hợp thuật toán quản lý hàng đợi động RIO ở phần này của tôi là để nghiên cứu và đánh giá mức độ ảnh hưởng của các luồng lưu lượng ưu tiên khi có luồng đột biến gây tác động vào mạng. Từ đó nhận xét về các vấn đề có thể xảy ra trong đảm bảo chất lượng dịch vụ QoS cũng như vai trò đảm bảo chất lượng dịch vụ của mô hình DiffServ kết hợp thuật toán quản lý hàng đợi động RIO trong truyền thông đa phương tiện.

Mục tiêu của chiến lược quản lý hàng đợi động AQM trong các mạng DiffServ có thêm sự khác biệt về mục tiêu so với trong các mạng Best-effort. Trong khi mục tiêu của AQM trong các mạng Best-effort là để tránh tắc nghẽn (chương 2) thì trong các mạng DiffServ nó còn thêm mục tiêu nữa là loại bỏ có ưu tiên (chương 3).

4.2.1 Cấu hình mạng mô phỏng



Hình 4.4 Cấu hình mạng mô phỏng DiffServ

Mạng mô phỏng gồm 8 nốt. Node S1 hướng đến D1, node S2 hướng đến D2, node S3 hướng đến S3. Cái luồng từ S1 đến S3 cùng chia sẽ kênh truyền chung từ R1 đến R2. Node R1 đến Core và node Core đi R2. Ở đây ta giả sử S1 là các luồng void với độ ưu tiên cao nhất, tiếp theo S2 là các luồng xử lý video với độ ưu tiên thứ 2 và cuối cùng là các luồng dữ liệu khác đi từ S3 với độ ưu tiên thấp nhất. Thông tin đường truyền của các node trong mạng như sau:

- R1 – Core, Core – R2: 1Mbps, 20ms
- S1 – R1, S2 – R1, S3 – R1, R2 – D1, R2 – D2, R2 – D3: 10Mbps, 5ms

Các luồng khi đi đến R1 sẽ được đánh dấu thông qua bộ đánh dấu srTCM (mục 3.1.4) được cài đặt ở router R1, hàng đợi vật lý đồng tời được đặt tại lối ra của router R1, với kích thước tối đa là 1000 gói tin. Khi các gói tin đi qua R1 sẽ nhận được một dấu, tương ứng với 3 mức ưu tiên trong hàng đợi vật lý. Tất cả các gói tin sau khi được đánh dấu sẽ được chuyển tiếp vào Core, đi qua R2 và đến nút đích tương ứng (D1, D2, D3). Hàng đợi được đặt ở R1 đến Core là RIO-C (RIO Coupled). Tất cả 3 node gửi đều là cbr (constant bit rate - nguồn sinh lưu lượng với tốc độ không đổi), thực thể nhận lần lượt là

sink0, null0, sink1, null1, sink2, null2. Tốc độ truyền của cả 3 đều bằng 0.6Mbps. Thời gian mô phỏng là 50s.

Các tham số khác sử dụng cho bộ đánh dấu srTCM:

Bảng 4.1: Các tham số sử dụng srTCM

	S1 – D1	**S2 – D2**	**S3 – D3**
	srTCM	**srTCM**	**srTCM**
CIR	4000	4000	4000
CBS	10000	10000	20000
EBS	20000	15000	30000

Các thông số của RIO tương ứng với 3 mức ưu tiên: đỏ, vàng và xanh trong đó mức xanh là ưu tiên cao nhất và đỏ là thấp nhất:

Bảng 4.2: Các tham số của RIO-C

	Minth	**Maxth**	**Pmax**	**Wq**
Green	20	40	0.01	0.002
Yellow	10	50	0.1	0.002
Red	10	50	0.3	0.002

Độ ưu tiên có thứ tự lần lượt là: luồng S1 đến D1, luồng S2 đến D2, luồng S3 đến D3.

Để đánh giá độ ảnh hưởng của luồng đột biến gây ra cho các luồng ưu tiên khác trong mạng, tôi sẽ thực hiện 3 mô phỏng:

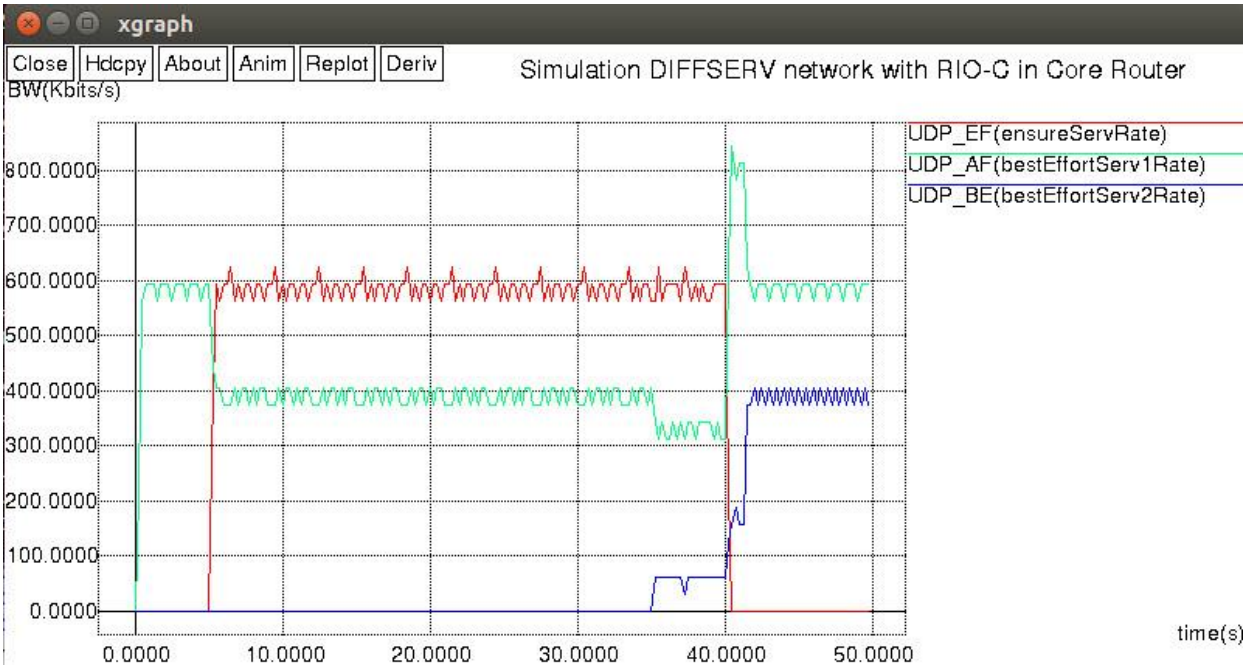
- **Trường hợp 1:** Luồng S2 phát từ 0.1s đến kết thúc mô phỏng. Luồng S1 phát từ khoảng 5s đến đến 40s. Luồng S3 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến hết.
- **Trường hợp 2:** Luồng S3 phát từ 0.1s đến kết thúc mô phỏng. Luồng S1 phát từ khoảng 5s đến đến 40s. Luồng S2 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến 12s.
- **Trường hợp 3:** Luồng S3 phát từ 0.1s đến kết thúc mô phỏng. Luồng S2 phát từ khoảng 5s đến đến 40s. Luồng S1 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến 12s.

4.2.2 Kết quả mô phỏng và nhận xét với từng trường hợp

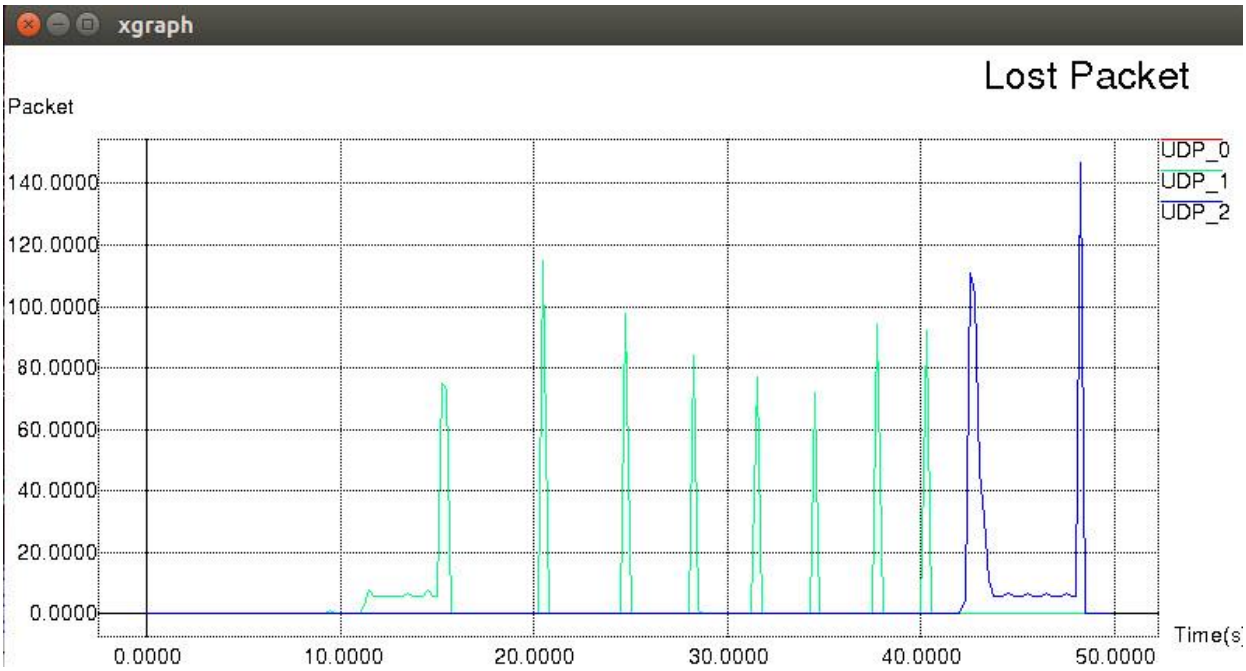
4.2.2.1 Trường hợp 1

Bài toán: Luồng S2 phát từ 0.1s đến kết thúc mô phỏng. Luồng S1 phát từ khoảng 5s đến đến 40s. Luồng S3 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến hết.

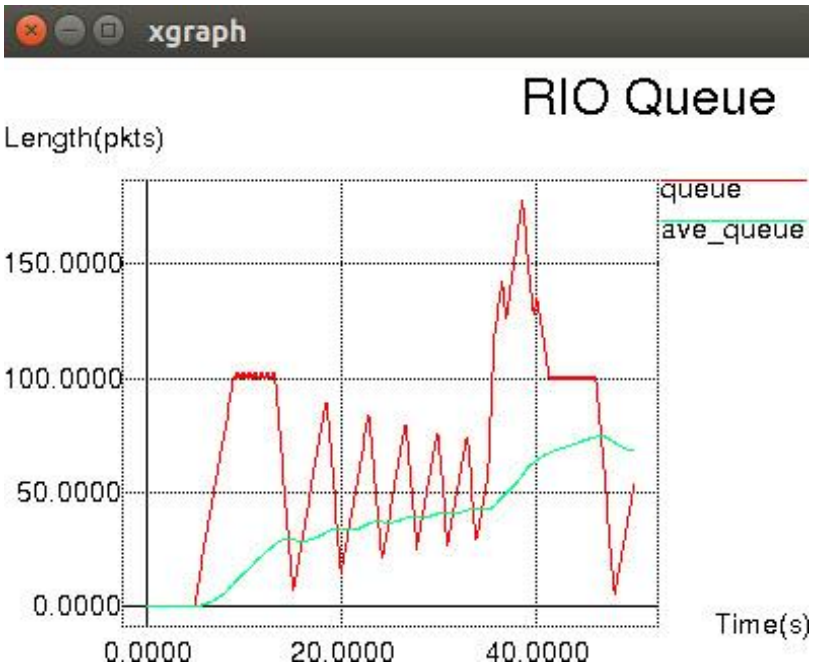
Kết quả mô phỏng:



Hình 4.5: Băng thông của đường truyền tương ứng với 3 luồng lưu lượng



Hình 4.6: Tỷ lệ mất gói tin tương ứng theo thời gian với 3 luồng



Hình 4.7: Kích thước hàng đợi RIO-C

Bảng 4.3: Một số kết quả thu được từ mô phỏng 1

	Số gói tin gửi	Số gói tin bị loại bỏ	Tỷ lệ loại bỏ (%)
Udp0	2626	0	0
Udp1	3738	880	23.5
Udp2	1068	567	53.0

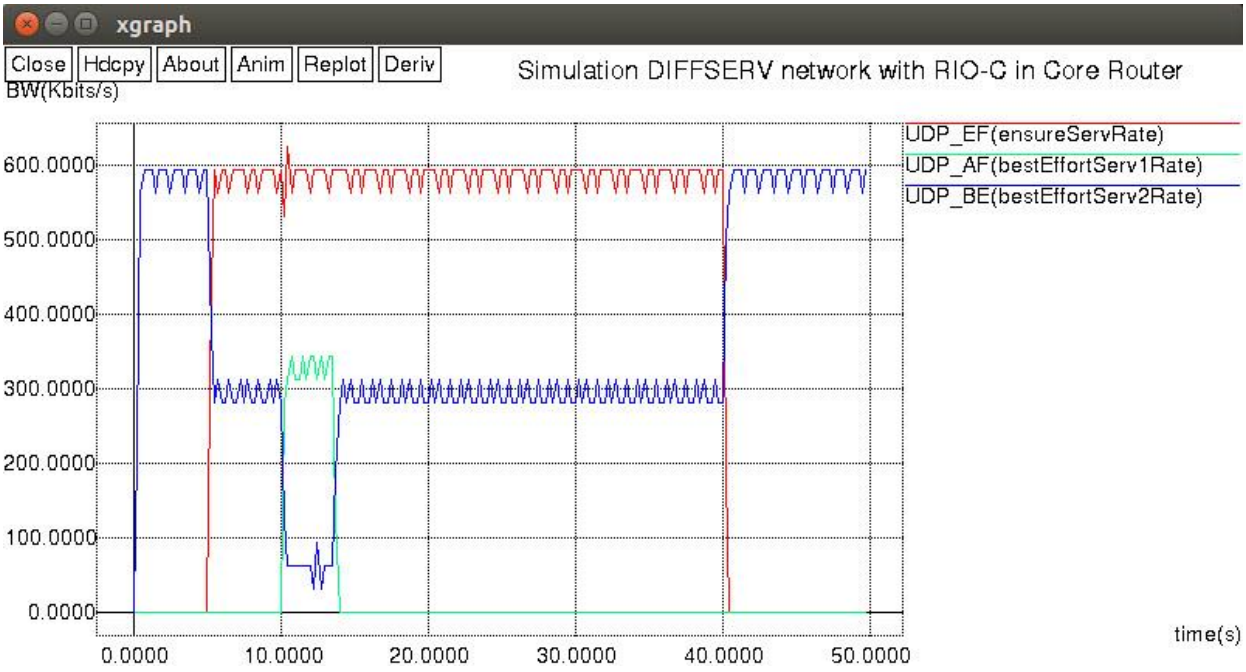
Ta thấy trong thời gian 5s đầu tiên, khi trong mạng chỉ có mình luồng S2 thì nó nó sẽ được chiếm trọn 0.6Mb băng thông trên đường truyền (hình 4.4), từ giây thứ 5 khi S1 bắt đầu được phát do S1 có độ ưu tiên cao hơn S2 nên S1 chiếm 0.6Mb băng thông, còn 0.4Mb còn lại S2 mới được sử dung. Khi S3 bắt đầu phát (giây 35), kích thước hàng đợi (hình 4.6) bắt đầu tăng (lên đến ~180 gói tin) nhưng sau đó khoảng ~3s nó bắt đầu được kéo về ở mức thấp (~ 130 gói tin). Đến giay thứ 40, khi S1 dừng phát, lúc này hàng đợi được kéo về thấp ngay lập tức. Khi S3 chưa được chạy, trong gia đoạn tắc nghẽn (10s, 20s, 25s, 28s, 32s…) các gói tin của S2 với độ ưu tiên thấp sẽ bị loại bỏ, S1 không có gói tin nào bị loại bỏ. Ở giai đoạn lưu lượng S3 bắt đầu được chạy và S1 dừng chạy (từ giây 40 đến hết mô phỏng), ta thấy băng thông của S2 được dùng đùng như nó mong muốn, phần còn lại được phục vụ cho S3, khi có tắc nghẽn xảy ra (42s…) các gói tin của S3 bị loại bỏ, S2 lúc này được phát với ưu tiên cao nhất, không có gói tin nào bị loại bỏ

(Hình 4.5). Ở giây thứ 35 khi luồng S3 được đưa vào mạng, ta thấy, S1 không bị ảnh hưởng gì về băng thông, nhưng băng thông của S2 bị tụt xuống một phần nhỏ (hình 4.4), để nhường 1 phần đó cho S3, vì S2 có độ ưu tiên cao hơn S3 và S3 chỉ được đảm bảo một lượng tốc độ cam kết giới hạn được cài đặt ban đầu. Từ 40s trở đi, lúc này S1 dừng phát, S2 và S3 chia sẽ băng thông đường truyền theo mức độ ưu tiên.

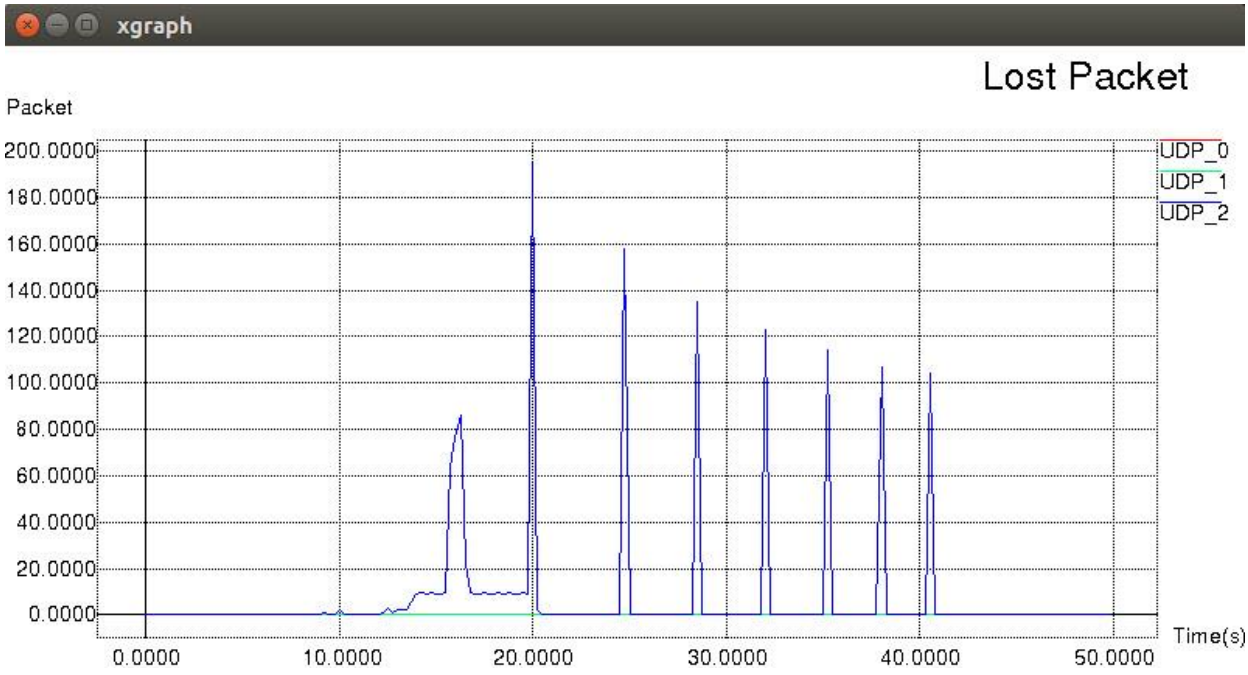
4.2.2.2 Trường hợp 2

Bài toán: Luồng S3 phát từ 0.1s đến kết thúc mô phỏng. Luồng S1 phát từ khoảng 5s đến đến 40s. Luồng S2 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến 12s.

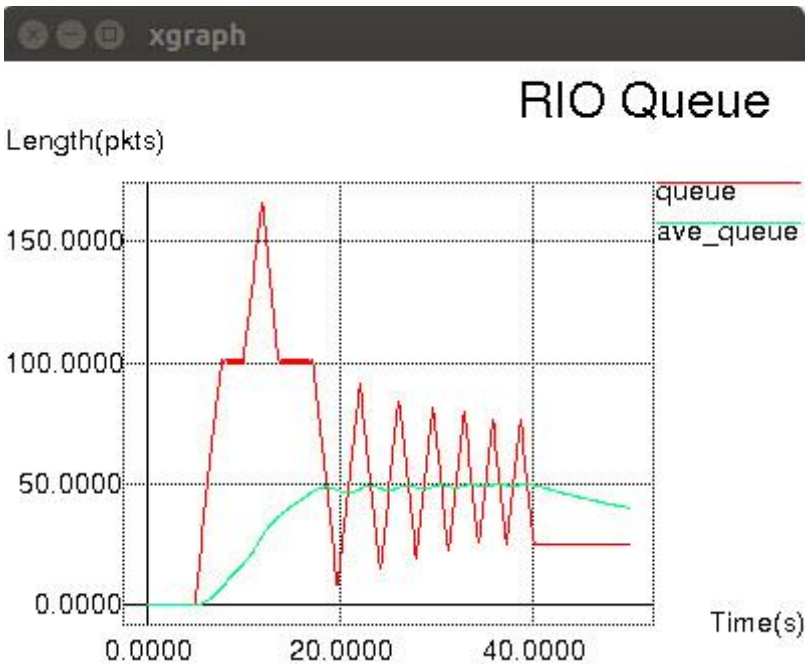
Kết quả mô phỏng:



Hình 4.8: Băng thông của đường truyền tương ứng với 3 luồng lưu lượng



Hình 4.9: Tỷ lệ mất gói tin tương ứng theo thời gian với 3 luồng



Hình 4.10: Kích thước hàng đợi RIO-C

Bảng 4.4: Một số kết quả thu được từ mô phỏng 2

	Số gói tin gửi	Số gói tin bị loại bỏ	Tỷ lệ loại bỏ (%)

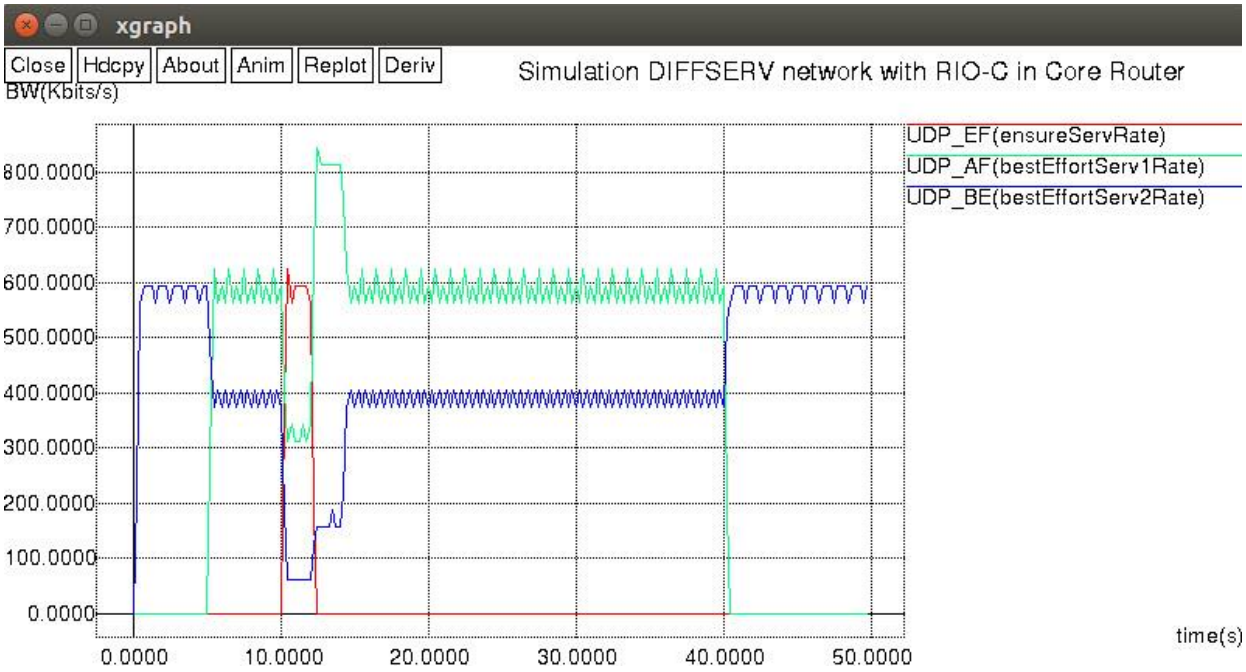
Udp0	2626	0	0
Udp1	150	0	0
Udp2	3713	1394	37.5

Ta thấy trong thời gian 5s đầu tiên, khi trong mạng chỉ có mình luồng S3 thì nó nó sẽ được chiếm trọn 0.6Mb băng thông trên đường truyền (hình 4.7), từ giây thứ 5 khi S1 bắt đầu được phát do S1 có độ ưu tiên cao hơn S3 nên S1 chiếm 0.6Mb băng thông theo mong muốn của nó (rate 0.6Mbps), còn 0.4Mb còn lại S3 mới được sử dung. Từ 0s đến 10s, kích thước hàng đơi ổn định, không có hiện tượng loại bỏ gói tin xay ra (hình 4.8). Đến giây thứ 10 khi S2 bắt đầu được đưa vào mạng, lúc này nhanh chóng kích thước hàng đợi bị tăng lên (hình 4.9). Đến khoảng 15s thì nó đạt cao nhất với khoảng 170 gói tin trong hàng đợi, lúc này ta thấy các gói tin bắt đầu được loại bỏ để tránh tình trạng tắc nghẽn và ưu tiên luồng có độ ưu tiên cao. Ở giây thứ 10 đến 14, S2 được phát, lúc này băng thông của S1 vẫn không bị ảnh hưởng, S1 vẫn được chiếm đầy đủ băng thông của nó, 0.4Mb còn lại, được S2 chiếm đến 0.34Mb băng thông còn S3 sau khi S2 phát, do quyền ưu tiên thấp hơn nên nó chỉ được phát đúng như cam kết CIR ban đầu (hình 4.7). Bước vào giai đoạn "ổn định" (hình 4.8, từ ~ 15s) tỷ lệ loại bỏ gói tin tập trung ở luồng S3. Tất cả các gói tin từ S1 và S2 ở trường hợp này đều được đảm bảo 100% không có gói tin nào bị loại bỏ. Việc loại bỏ gói tin chỉ xảy ra với S3. Từ giấy thứ 40 trở đi (hình 4.7), S1 ngừng phát, nên lúc này chỉ còn S3 chiếm băng thông của nó, không phải cạnh tranh với các luồng khác, vì thế không gói tin nào bị loại bỏ nữa (Hình 4.8). Từ bảng 4.4 ta có thể thấy rõ hơn, tỷ lệ loại bỏ gói tin ở luồng S3 có độ ưu tiên thấp nhất lên đến 37.5%.

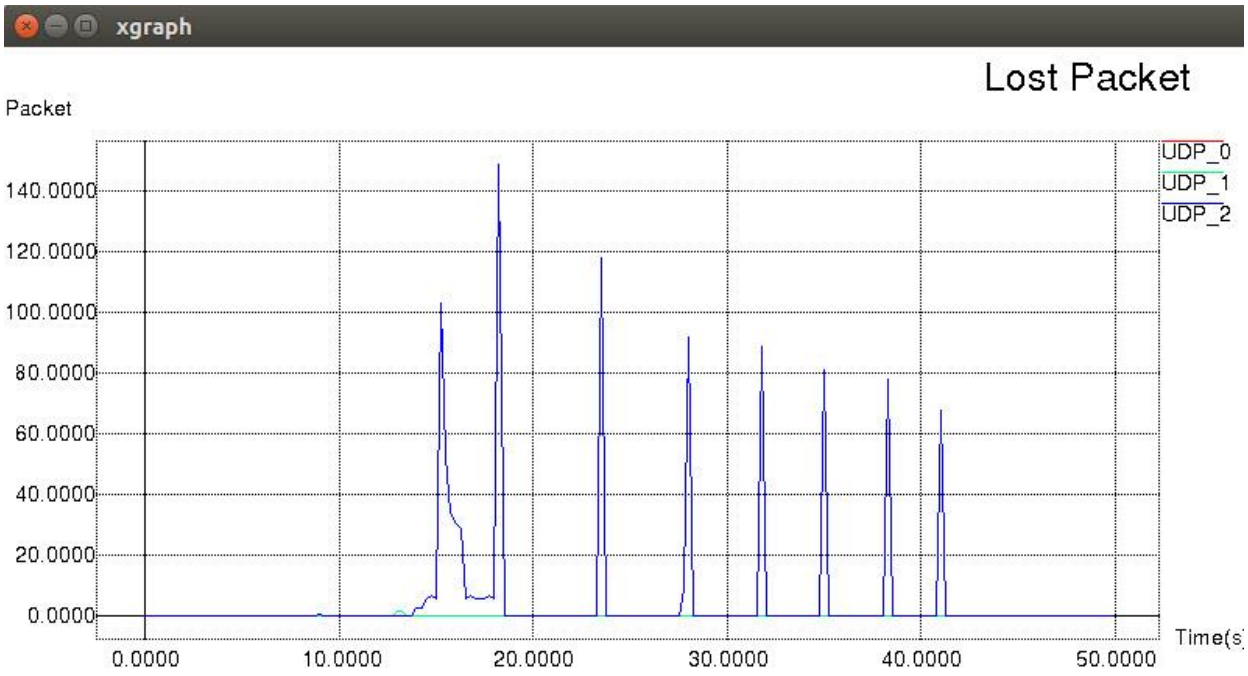
4.2.2.3 Trường hợp 3

Bài toán: Luồng S3 phát từ 0.1s đến kết thúc mô phỏng. Luồng S2 phát từ khoảng 5s đến đến 40s. Luồng S1 là luồng đột biến phát sinh trong mạng. Sẽ phát trong khoảng thời gian ngăn từ 10s đến 12s.

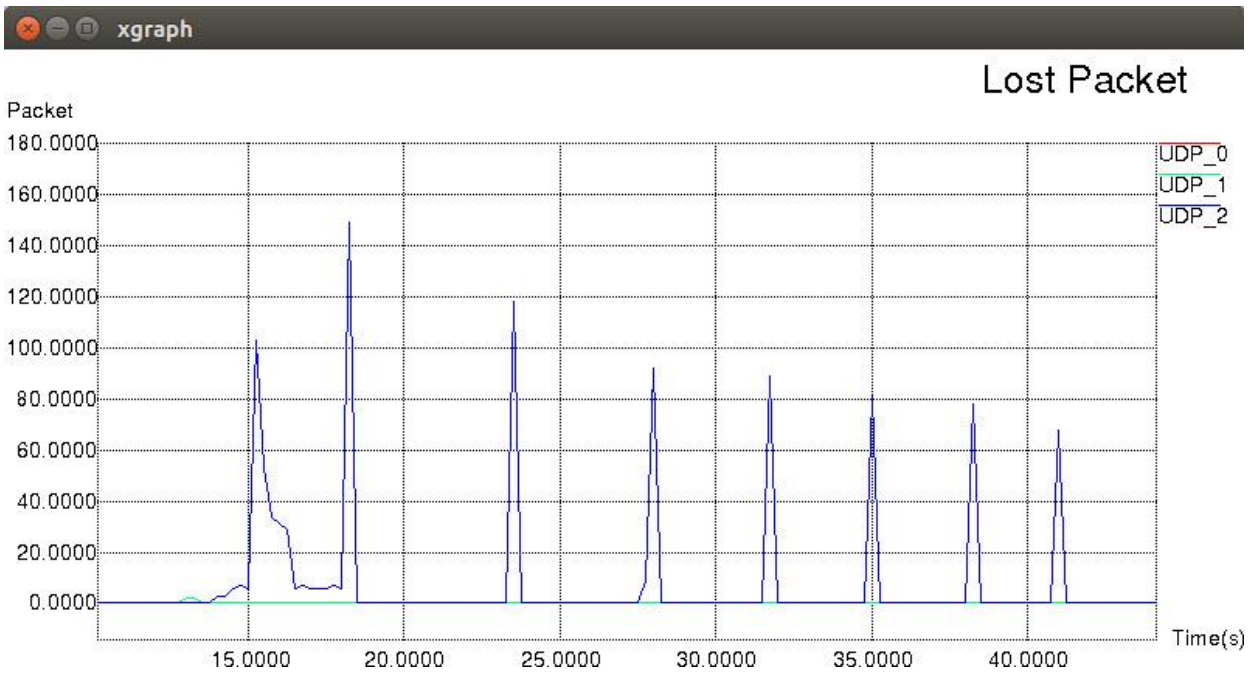
Kết quả mô phỏng:



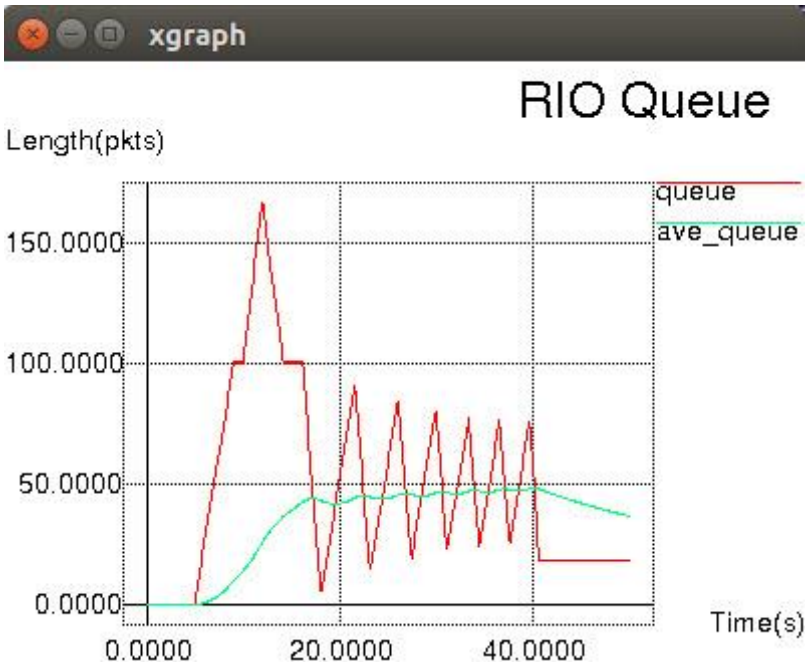
Hình 4.11: Băng thông của đường truyền tương ứng với 3 luồng lưu lượng



Hình 4.12: Tỷ lệ mất gói tin tương ứng theo thời gian với 3 luồng



Hình 4.13: Tỷ lệ mất gói tin tương ứng theo thời gian với 3 luồng (phóng to giai đoạn loại bỏ)



Hình 4.14: Kích thước hàng đợi RIO-C

Bảng 4.5: Một số kết quả thu được từ mô phỏng 3

	Số gói tin gửi	Số gói tin bị loại bỏ	Tỷ lệ loại bỏ (%)
Udp0	150	0	0

Udp1	2626	4	0.15
Udp2	3720	1004	26.9

Trong 10s đầu tiên 2 luồng S2 và S3 cùng nhau chia sẽ chung đường truyền, S2 chiếm đầy đủ 0.6Mb băng thông vì nó là luồng có quyền ưu tiên cao hơn S3 (hình 4.10), Từ 0s đến 10s, kích thước hàng đơi ổn định, không xảy ra hiện tượng tắc nghẽn nên không có hiện tượng loại bỏ gói tin xảy ra (hình 4.11). Đến giây thứ 10 khi S1 bắt đầu được đưa vào mạng, lúc này nhanh chóng kích thước hàng đợi bị tăng lên (hình 4.9). Đến khoảng 15s thì nó đạt cao nhất với khoảng 170 gói tin trong hàng đợi, đây cũng là lúc ta thấy các gói tin bắt đầu được loại bỏ tăng cao (hình 4.11) để tránh tình trạng tắc nghẽn và ưu tiên luồng có độ ưu tiên cao. Từ hình 4.12 ta thấy ở giây ~ 13 đã có một ít rất nhỏ các gói tin của S2 bị loại bỏ. Khi S1 được đưa vào mạng ở 10s, nó ngay lập tức chiếm đủ băng thông của nó (0.6Mb), khiến luồng S2 bị đẩy tụt xuống, và S3 chỉ còn chiếm đúng với lượng tốc độ mà nó được cam kết, lúc này kích thước hàng đợi được đẩy lên cao (4.13). Sau giai đoạn luồng S1 được chay (~15s), lúc này trong mạng chỉ còn 2 luồng S2 và S3. Trong giai đoạn "ổn định" này, tỷ lệ loại bỏ gói tin tập trung loại bỏ ở luồng S3, luồng S2 không bị loại bỏ thêm gói tin nào. Ta thấy với việc S1 là luồng ưu tiên cao nhất được thêm vào mạng, các luồng còn lại đều "phải nhường" cho nó chạy, và hiện tượng loại bỏ gói tin sẽ xảy ra với các luồng còn lại, việc loại bỏ sẽ ưu tiên loại bỏ ở luồng thấp trước, nhưng khi cần thiết, luồng cao hơn (S2) vẫn bị loại bỏ 1 vài các gói tin để nhường băng thông đảm bảo cho luồng ưu tiên nhất (S1).

KẾT LUẬN VÀ PHƯƠNG HƯỚNG NGHIÊN CỨU TIẾP THEO

A. KẾT LUẬN

Trong các chiến lược quản lý hàng đợi thì chiến lược quản lý hàng đợi động có nhiều những ưu điểm nổi bật. Trong đó phải kể đến 2 thuật toán nổi bật dành cho đảm bảo chất lượng dịch vụ trên kiến trúc mạng truyền thống là RED và A-RED. Đặc điểm nổi bật của nó là đem lại độ trễ nhỏ và thông lượng cao cho các luồng. RED với nhiều những ưu điểm nổi trội, tuy nhiên nó vẫn có những khuyến điểm có thể nhìn thấy ngay (mục 2.5) như RED nhạy cảm với tham số cài đặt ban đầu và cũng nhạy cảm với tải của mạng, vì thế A-RED ra đời với mục đích khắc phục các nhược điểm của RED. A-RED kế thừa hoàn toàn thuật toán RED gốc và thêm vào đó là khả năng hiệu chỉnh maxp giúp cho hàng đợi luôn nằm trong khoảng mong muốn, giúp cho mạng chạy ổn định hơn bất chấp hiện tượng về lưu lượng.

Tuy nhiên, RED và A-RED đối xử với các luồng đến là như nhau, tất cả các gói tin đều được đối xử một cách công bằng, không có ưu tiên bất kỳ gói tin nào, điều này trong thực thế là một khuyết điểm. Khi các gói tin đến hoàn toàn có độ ưu tiên khá nhau, và ví thế chúng phải được đối xử khác nhau trên kênh truyền chung. Từ lý do đó, RIO được hình thành. RIO là một mở rộng khác của RED, giữ nguyên lại thuật toán RED gốc, kế thừa hoàn toàn ưu điểm của RED nhưng có thêm khả năng cung cấp dịch vụ phân loại các gói tin đến. Các gói tin đến sẽ được phân loại theo 2 lớp ưu tiên là In và Out. Trong đó các gói tin thuộc lớp In có độ ưu tiên cao hơn các gói tin thuộc lớp Out. Khi có tắc nghẽn xảy ra, RIO sẽ ưu tiên loại bỏ các gói tin Out trước, sau đó mới đến các gói tin In. Trong thực tế, các gói tin ưu tiên Out nếu bị loại bỏ là điều khó chấp nhận được.

Trong kiến trúc mạng truyền thống, tất cả các gói tin trong mạng đều được đối xử như nhau, điều này trong mạng ngày này là không ổn và khổn đảm bảo được chất lượng dịch vụ trong truyền thông đa phương tiện. Với kiến trúc mạng DiffServ – kiến trúc mạng hỗ trợ các dịch vụ phân loại, nó thỏa mãn được nhu cầu chia sẽ nhiều kết nối trên một kênh truyền chung mà vẫn đảm bảo được chất lượng dịch vụ cho các luồng đã được đăng ký dịch vụ trước. Thuật toán quản lý hàng đợi động RIO-C trong DiffServ có khả năng cung cấp cho mỗi lớp ưu tiên trong DiffServ một dịch vụ tương ứng với độ ưu tiên của nó.

Sau khi kết hợp tìm hiểu và nghiên cứu lý thuyết đi kèm với mô phỏng các kế hoạch quản lý hàng đợi động trong truyền thông đa phương tiện cụ thể là về các chiến lược RED, A-RED, RIO, DiffServ tôi đã đạt được các kết quả sau:

RED có nhiều ưu điểm của nó so với DropTai như kích thước hàng đợi nhỏ, đỗ trễ thấp, thông lượng cao… những điều đã được chứng minh qua mô phỏng. Nhưng RED cũng còn những khuyết điểm (mục 2.5) cụ thể đã được nếu ra ở cuối chương 2.4. Đây cũng chính là lý do hình thành nên thuật toán A-RED. Đối với A-RED, qua mô phỏng đơn giản (2.5.3) với nhiều luồng kết nối, thay đôi về tải trong mạng, tôi thấy rằng, A-RED khắc phục rất tốt các khuyết điểm của RED và vẫn kế thừa hoàn toàn các ưu điểm của RED. Với việc hiệu chỉnh giá trị maxp không liên tục, nó giúp duy trì kích thước hàng đợi nằm trong vùng mong muốn và vẫn giữ cho thông lượng trong mang cao.

Với RIO, RIO là một thuật toán kế thừa RED gốc, và tiếp tục cải tiến để có thể đối xử khác nhau với các luồng đến có độ ưu tiên khác nhau. Với việc nghiên cứu RIO kết hợp trong DiffServ nhằm đảm bảo chất lượng dịch vụ trong truyền thông đa phương tiện, tôi thấy: trong mô hình DiffServ, RIO được dùng để cung cấp cho mỗi lớp ưu tiên trong DiffServ một chất lượng dịch vụ khác nhau.

Với các luồng ưu tiên đã được xét, việc có các luồng đột biến tác động vào mạng, không gây ảnh hưởng đến các luồng được ưu tiên cao. Các luồng ưu tiên cao nhất, luôn được sử dụng tối đa các dịch vụ mà nó được cung cấp, bất chấp việc trong mạng đang xảy ra vấn đề tắc nghẽn khi có các lưu lượng ngoài luồng (các lưu lượng có mức ưu tiên thấp hơn) tác động vào mạng. Đây là ưu điểm cốt lõi của mô hình DiffServ trong việc đảm bảo chất lượng dịch vụ cho truyền thông đa phương tiện**.** Tuy nhiên vì RIO kế thừa từ RED và áp dụng với mô hình mạng kiến trúc các dịch vụ phân loại, nên RIO cũng kế thừa các khuyết điểm của RED, như việc RIO phụ thuộc vào các tham số chúng ta khai báo, tải trong mạng… hay ban đầu với 2 mức ưu tiên là In và Out ta có tương ứng với mỗi mức là bộ 3 tham số đi kèm. Vậy nên khi mở rộng với n mức ưu tiên khác nhau ta có 3n + 1 tham số đi kèm. Điều này gây khó khăn cho người quan trị mạng khi thiết lập tham số, chưa tính đến việc ta còn không kiểm soát được kích thước hàng đợi trung bình trong mạng nằm ở vùng mục tiêu. Đây cũng là hướng nghiên cứu tiếp theo của tôi.

B. PHƯƠNG HƯỚNG NGHIÊN CỨU TIẾP THEO

Tôi sẽ tiếp tục nghiên cứu về A-RIO. Một thuật toán cải tiến của RIO, một sự kết hợp giữa A-RED và RIO. Cụ thể là:

1. Đánh giá sự tối ưu của A-RIO so với RIO trong kiến trúc mạng DiffServ.
2. Sự ảnh hưởng của các kiểu lưu lượng với độ ưu tiên khác nhau lên A-RIO trong mô hình DiffServ.
3. Chất lượng dịch vụ trong truyền thông đa phương tiện khi sử dụng A-RIO trong DiffServ để cung cấp các dịch vụ khác nhau cho các lớp ưu tiên.

TÀI LIỆU THAM KHẢO

A. TÀI LIỆU TIẾNG VIỆT

[1]. Vũ Duy Lợi, Nguyễn Đình Việt, Ngô Thị Duyên, Lê Thị Hợi (2004), "*Đánh giá hiệu năng chiến lược quản lý hàng đợi RED bằng bộ mô phỏng NS*", Kỷ yếu Hội thảo Khoa học Quốc gia lần thứ hai về Nghiên cứu, Phát triển và Ứng dụng Công nghệ Thông tin và Truyền thông (ICT.rda'04), (Hà nội, 24-25/9/2004). NXB Khoa học và Kỹ thuật, Hà Nội, 5/2005, trang 394-403.

[2]. PGS.TS. Nguyễn Đình Việt, Bài giảng đánh giá hiệu năng mạng máy tính, 2013.

[3]. PGS.TS. Nguyễn Đình Việt (2003), *Nghiên cứu phương pháp đánh giá và cải thiện hiệu năng giao thức TCP cho mạng máy tính*, Luận án tiến sĩ, Khoa Công nghệ, Đại học Quốc gia Hà nội.

[4]. PGS.TS. Nguyễn Đình Việt, *Bài giảng Mạng và Truyền số liệu nâng cao*, 2013.

[5]. Luận Văn Cao Học, Lê Đình Danh, 2007.

[6]. Nguyễn Thị Phương Nhung, Lê Đình Thanh, Hồ Sĩ Đàm (2010), *Đánh giá hiệu năng đảm bảo chất lượng dịch vụ trên nền mạng IP qua mô hình DiffServ*.

B. TÀI LIỆU TIẾNG ANH

[7]. A. S.Tanenbaum, Computer Network, Fourth Edition, Prentice Hall, March, 2003.

[8]. B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, et al: "*Recommendations on Queue Management and Congestion Avoidance in the Internet*", RFC 2309, April 1998.

[9]. David D.Clark, Wenjia Fang (1998), "*Explixit Allocation of Best Effort Packet Delivery Service*", Labratory for Computer Sciences Computer Science Department, Massachusetts Institute of Technology Princeton University.

[10]. http://docwiki.cisco.com/wiki/Quality_of_Service_Networking

[11]. Eitan Altman and Tania Jiménez, "*NS Simulator for beginners*", 2003

[12]. James F. Kurose and Keith W. Ross, "*Computer Networking: A Top-Down Approach Featuring the Internet*", Pearson, Sixth Edition, 2013.

[13]. http://www.isi.edu/nsnam/ns/ns-documentation.html

[14]. Julio Orozco, David Ros (2003), "*An Adaptive RIO (A-RIO) Queue Management Algorithm*", Reseach Report PI-1526, IRISA.

[15]. Mischa Schwartz: "*Telecommunication Networks: Protocols, Modeling and Analysis*", Addition-Wesley, 1987.

[16]. Sally Floy, Van Jacobson (1993), "*Random Early Detection Gateways for Congestion Avoidance*", IEEE/ACM Transactions on Networking, Vol. 1, no 4.

[17]. Sally Floyd, D. Black: "*The Addition of Explicit Congestion Notification (ECN) to IP*", RFC 3168, Sep. 2001.

[18]. Sally Floy, Ramakrishna, and Scott Shenker (2001), "*Adaptive RED: An Algorithm for Increasing the Roburstness of RED's Active Queue Management*", AT&T Center for Internet Research at ICSI.

[19]. S. Kanhere, H. Sethu and B. Parekh (2002), "*Fair and Efficient Packet Scheduling Using Elastic Round Robin*", IEEE Transactions on parallel and distributed systems, Vol. 13. No. 3, pp 324-336.

[20]. Van Jacobson: "*Congestion Avoidance and Control*", Proceeding of SIGCOMM '88, (Stanford, CA, Aug. 1988), ACM.

[21]. Van Jacobson: "*Notes on using RED for Queue Management and Congestion Avoidance*", Network Research Group, Berkeley National Laboratory, Berkeley, CA 94720, June 8, 1998.

[22]. Wesley M. Eddy, Mark Allman: "*A Comparison of RED's Byte and Packet Modes*", Computer Networks, Vol. 42, Issue 2, June 2003.

PHỤ LỤC

- **File red.tcl và redPerl.pl (mục 2.4.6.1)**

➢ **File Red.tcl: Tính kích thước hàng đợi, hàng đợi trung bình và vẽ đồ thị**

```
set ns [new Simulator]

set numSrcs 5
set Duration 50

# Setting
set tcp_packet_size_ 1000
set udp_packet_size_ 1000
set tcp_window_size_ 100
set udp_rate_ 2mb
set queue_size_ 100
set aqm_ "RED"
set windown_size_with_time_ "Rwin.tr"
set queue_out_ "queueR.out"
set bw 1

Queue/RED set bytes_ false
Queue/RED set queue_in_bytes_ false
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 15
Queue/RED set linterm_ 10
Queue/RED set queue_in_bytes_ 0
Queue/RED set q_weight_ 0.002

#Create 6 nodes
for {set j 0} { $j <= 6} {incr j} {
        set s($j) [$ns node]
        set d($j) [$ns node]
}
set r1 [$ns node]
```

```
set r2 [$ns node]

for {set i 1} {$i <= $numSrcs} { incr i} {
        $ns duplex-link $s($i) $r1 10Mb 1ms DropTail
        $ns queue-limit $s($i) $r1 20
        $ns duplex-link $r2 $d($i) 10Mb 1ms DropTail
        $ns queue-limit $r2 $d($i) 20
}
$ns duplex-link $s(6) $r1 3Mb 10ms DropTail
$ns duplex-link $r2 $d(6) 3Mb 10ms DropTail

$ns duplex-link $r1 $r2 [expr $bw]Mb 20ms $aqm_
$ns queue-limit $r1 $r2 $queue_size_

set tq [open $queue_out_ w]
$ns trace-queue $r1 $r2 $tq

for {set i 1} {$i <= $numSrcs} {incr i} {
        set tcp($i) [$ns create-connection TCP/Reno $s($i)  TCPSink $d($i) $i]
        $tcp($i) set packetSize_ $tcp_packet_size_
        $tcp($i) set window_ $tcp_window_size_
        set ftp($i) [$tcp($i) attach-source FTP]
}

set udp [new Agent/UDP]
$ns attach-agent $s(6) $udp
$udp set fid_ 6

set null [new Agent/Null]
$ns attach-agent $d(6) $null

set cbr [new Application/Traffic/CBR]
$cbr set packet_size_ 1000
$cbr set rate_ $udp_rate_
$cbr set random_ false
$cbr attach-agent $udp
$ns connect $udp $null
```

```
for {set i 1} {$i <= $numSrcs} {incr i} {
        set t [expr $i*0.1]
        $ns at $t "$ftp($i) start"
        $ns at $Duration "$ftp($i) stop"
}

$ns at 6.0  "$cbr start"
$ns at 7.0  "$cbr stop"
$ns at 15.0  "$cbr start"
$ns at 17.0  "$cbr stop"
$ns at 19.0  "$cbr start"
$ns at 20.0  "$cbr stop"

set wind [open $windown_size_with_time_ w]
proc plotWindow {tcpSource file k} {
        global ns numSrcs
        set time 0.03
        set now       [$ns now]
        set cwnd [$tcpSource set cwnd_]
        if {$k == 1} {
                puts -nonewline $file "$now $cwnd\t"
        } elseif {$k > 1 && $k < $numSrcs} {
                puts -nonewline $file "$cwnd\t"
        } elseif {$k == $numSrcs} {
                puts -nonewline $file "$cwnd\n"
        }
        $ns at [expr $now+$time] "plotWindow $tcpSource $file $k"
}
for {set i 1} {$i <= $numSrcs} {incr i} {
        $ns at 0.1 "plotWindow $tcp($i) $wind $i"
}

set qfile [$ns monitor-queue $r1 $r2 [open q.tr w] 0.01]
set trq [open cur.tr w]
set sum 0
set n 0
```

```
proc record {} {
        global ns qfile trq sum n
        set time 0.01
        set now       [$ns now]
        $qfile instvar pkts_ parrivals_ pdepartures_ pdrops_
        set n [expr $n + 1]
        set sum [expr $sum + $pkts_]
        puts $trq "$now $pkts_"
        $ns at [expr $now+$time] "record"
}
$ns at 0.1 "record"

proc finish {} {
        global ns trq sum n aqm_ bw
        $ns flush-trace
        close $trq
        exec awk { {avg = 0.995*avg + 0.005*$2; printf("%f\t%f\n", $1, avg) >
"ave.tr";}} cur.tr
        set f [open queue.tr w]
        puts $f \"queue
        exec cat cur.tr >@ $f
        puts $f \n\"ave_queue
        exec cat ave.tr >@ $f
        close $f

        set mean [expr $sum/$n]
        set delay [expr 1000*$mean*1040*8/($bw*1000*1000)]
        puts [format "%s: mean = %0.2f pkts    delay = %0.2f ms" $aqm_ $mean $delay]

        exec xgraph queue.tr -t "DropTail: Queue" -bb -nb -x "Time(s)" -y
"Length(pkts)" &
   exit 0
}

$ns at [expr $Duration] "finish"
$ns run
```

- **File redPerl.pl: Dùng để tính hệ số sử dụng đường truyền (%), và thông lượng các kết nối tcp.**

```
$file = $ARGV[0];
$time = $ARGV[1];

$st = `cat $file | grep ^- > thru.txt`;
chop($st);

$duration = 50;
$bw = 1;

$totPkts = `cat thru.txt | wc -l`;
$util = $totPkts*1040*8*100/($duration*$bw*1024*1024);
printf("Total packets = %ld \t util = %.2f\n",$totPkts, $util);

$sum = 0;
for($i = 1; $i <= 6; $i++){
        $p = $i * 2;
        $xa = 1 + $p;
        $str = `cat thru.txt | grep "$i $p.0 $xa.0" > tcp$i`;
        chop($str);
}

for($i = 1; $i <= 6; $i++){
        $clock = 0.0;
        open (DATA, "<tcp$i") || die "Can't open data in file tcp$i!";
```

```
        open (tcp, ">tcp$i.txt") || die "Can't open data in file tcp$i.txt!";
        if ($i == 6) { printf(tcp "\"UDP-flow\n");}
        else {printf(tcp "\"TCP$i-flow\n");}
        while(<DATA>){
                @x = split(' ');
                if($x[1] - $clock <= $time){
                        $sum ++;
                } else {
                        printf(tcp "%.2f\t%.2f\n", $x[1], $sum*1040*8/$time);
                        $clock += $time;
                        $sum = 0;
                }
        }
        printf(tcp "%.2f\t%.2f\n", $x[1], $sum*1040*8/$time);
        close DATA;
        close tcp;
}
exit 0;
```

- **File ared.tcl (mục 2.5.3)**

```
# Le Xuan Anh 10-11-2016
set ns [new Simulator]
# Setting
set tcp_packet_size_ 1000
set tcp_window_size_ 50
set queue_size_ 100
```

```
set aqm_ 1 ;# 0 is RED, 1 is A-RED
set title "ARED"
set numSrcs 55
set finish_time_ 80
set firstTimeStart 0.1
set secondTimeStart 40
set tf [open queue$title.out w]
set qmon [open stats$title.tr w]

# AQM config
Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 15
Queue/RED set linterm_ 10
Queue/RED set q_weight_ 0.002
Queue/RED set queue_in_bytes_ 0
Queue/RED set adaptive_ $aqm_
Queue/RED set alpha_ 0.02
Queue/RED set beta_ 0.9

set R1 [$ns node]
set R2 [$ns node]
set dest [$ns node]

$ns duplex-link $R1 $R2 1Mbps 20ms RED
$ns queue-limit $R1 $R2 $queue_size_
$ns queue-limit $R2 $R1 $queue_size_
```

```
$ns duplex-link $R2 $dest 1Mbps 10ms DropTail
for {set i 1} { $i <= $numSrcs} {incr i} {
        set s($i) [$ns node]
        $ns duplex-link $s($i) $R1 10Mbps 5ms DropTail
        set tcp($i) [$ns create-connection TCP/Newreno $s($i) TCPSink $dest $i]
        $tcp($i) set window_ $tcp_window_size_
        $tcp($i) set packetSize_ $tcp_packet_size_
        set ftp($i) [$tcp($i) attach-source FTP]
}

for {set i 1} { $i <= 5} {incr i} {
        $ns at [expr $firstTimeStart*$i] "$ftp($i) start"
        $ns at $finish_time_ "$ftp($i) stop"
}

#for {set i 6} { $i <= $numSrcs} {incr i} {
#       $ns at [expr $secondTimeStart + 0.5*($i-5)] "$ftp($i) start"
#       $ns at $finish_time_ "$ftp($i) stop"
#}
for {set i 6} { $i <= $numSrcs} {incr i} {
        $ns at [expr $firstTimeStart*$i] "$ftp($i) start"
        $ns at $secondTimeStart "$ftp($i) stop"
}
set redq [[$ns link $R1 $R2] queue]
set trq [open all.q w]
$redq trace curq_
```

```
$redq trace ave_
$redq attach $trq
set qfile [$ns monitor-queue $R1 $R2 [open qf.tr w] 0.01]
set n 0
set sum 0
proc record {} {
        global ns qfile qmon n sum
        set now         [$ns now]
        $qfile instvar pkts_ parrivals_ pdepartures_ pdrops_
        puts $qmon "$now $pkts_ $parrivals_ $pdepartures_ $pdrops_"
        $ns at [expr $now + 0.01] "record"

}

$ns at 0.1 "record"
$ns trace-queue $R1 $R2 $tf



# Define 'finish' procedure (include post-simulation processes)

proc finish {} {
        global ns trq qmon tf n sum title
        close $trq
        close $qmon
        close $tf
        exec grep "a" all.q > ave1.tr
```

```
        exec grep "Q" all.q > cur1.tr
        exec awk { {printf("%f\t%f\n", $2, $3) > "ave.tr";}} ave1.tr
        exec awk { {printf("%f\t%f\n", $2, $3) > "cur.tr";}} cur1.tr
        set f [open queue.tr w]
        puts $f \"queue
        exec cat cur.tr >@ $f
        puts $f \n\"ave_queue
        exec cat ave.tr >@ $f
        close $f
        exec xgraph queue.tr -t "$title Queue" -bb -nb -x "Time(s)" -y "Length(pkts)" &
        exit 0
}
$ns at $finish_time_ "finish"
$ns run
```

- **File mô phỏng RIO và DiffServ (chương 4)**

```
remove-all-packet-headers       ; # removes all except common
add-packet-header Flags IP RTP TCP  ; # hdrs reqd for validation test

# FOR UPDATING GLOBAL DEFAULTS:
Agent/TCP set minrto_ 1
# default changed on 10/14/2004.
Queue/RED set bytes_ false
# default changed on 10/11/2004.
Queue/RED set queue_in_bytes_ false
# default changed on 10/11/2004.
```

```
#Queue/RED set q_weight_ 0.002
#Queue/RED set thresh_ 5
#Queue/RED set maxthresh_ 15
# The RED parameter defaults are being changed for automatic configuration.

Class TestSuite

proc usage {} {
        global argv0
        puts stderr "usage: ns $argv0 <RIO> "
        exit 1
}
# Set up parameters for ds redqs for a link.
TestSuite instproc create_DSQueue {} {
   $self instvar packetSize numSrcs targetdelay min_in max_in min_out max_out pmax_in pmax_out pmax in_wq out_wq minth maxth
   $self instvar ns cir1 cir2 s C Ed d MaxQueueSize aqm qCEd

   # Set DS RED parameters from Edge to Core:
for {set i 1} { $i <= $numSrcs} {incr i} {
# Set DS RED parameters from si(=ei) to Core:
        set qEC($i) [[$ns link $s($i) $C] queue]
        $qEC($i) meanPktSize $packetSize
        $qEC($i) set limit_ $MaxQueueSize
        $qEC($i) set numQueues_ 1
        $qEC($i) setNumPrec 2
```

```
        if {$i % 2 == 1} {
                $qEC($i) addPolicyEntry [$s($i) id] [$d($i) id] TSW2CM 10 $cir1
        } else {
                $qEC($i) addPolicyEntry [$s($i) id] [$d($i) id] TSW2CM 10 $cir2
        }
        $qEC($i) addPolicerEntry TSW2CM 10 11
        $qEC($i) addPHBEntry 10 0 0
        $qEC($i) addPHBEntry 11 0 1

        $qEC($i) setMREDMode DROP
        $qEC($i) configQ 0 0 $MaxQueueSize

# Set DS RED parameters from Core to si:
        set qCE($i) [[$ns link $C $s($i)] queue]
        $qCE($i) set limit_ $MaxQueueSize
        $qCE($i) meanPktSize 40
        $qCE($i) set numQueues_ 1
        $qCE($i) setNumPrec 2
        $qCE($i) setMREDMode DROP
        $qCE($i) configQ 0 0 $MaxQueueSize
        $qCE($i) addPHBEntry 10 0 0
        $qCE($i) addPHBEntry 11 0 1
}

# Set DS RED parameters from Core to Ed:
        set qCEd [[$ns link $C $Ed] queue]
```

```
        $qCEd set limit_ $MaxQueueSize
        $qCEd meanPktSize $packetSize

        $qCEd set numQueues_ 1
        $qCEd setNumPrec 2

        $qCEd addPHBEntry 10 0 0
        $qCEd addPHBEntry 11 0 1

        if {$aqm == "RIO"} {
                $qCEd setMREDMode RIO-D
                $qCEd setGentleMode 0
                $qCEd configQ 0 0 $min_in $max_in $pmax_in $in_wq
                $qCEd configQ 0 1 $min_out $max_out $pmax_out $out_wq
        }
# Set DS RED parameters from Ed to Core:
        set qEdC [[$ns link $Ed $C] queue]
        $qEdC set limit_ $MaxQueueSize
        $qEdC meanPktSize 40
        $qEdC set numQueues_ 1
        $qEdC setNumPrec 2
        for {set i 1} {$i <= $numSrcs} {incr i} {
        $qEdC addPolicyEntry [$d($i) id] [$s($i) id] TSW2CM 10 $cir1
        }
        $qEdC addPolicerEntry TSW2CM 10 11
        $qEdC addPHBEntry 10 0 0
```

```
        $qEdC addPHBEntry 11 0 1
        $qEdC setMREDMode DROP
        $qEdC configQ 0 0 $MaxQueueSize
}
TestSuite instproc setTopo {} {
   $self instvar ns linkBW numSrcs C Ed s d trq
   # Set up the network topology shown above:
        set C [$ns node]
        set Ed [$ns node]
        for {set i 1} { $i <= $numSrcs} {incr i} {
                set s($i) [$ns node]
                set d($i) [$ns node]
                set t [expr ($i+1)/2]
                set dly [expr 10*$t]ms
                $ns simplex-link $s($i) $C 100Mb $dly dsRED/edge
                $ns simplex-link $C $s($i) 100Mb $dly dsRED/core
        }

        set BW [expr $linkBW]Mb
        $ns simplex-link $C $Ed $BW 5ms dsRED/core
        $ns simplex-link $Ed $C $BW 5ms dsRED/edge
        for {set i 1} { $i <= $numSrcs} {incr i} {
                $ns duplex-link $Ed $d($i) 100Mb 5ms DropTail
        }
        $self enable_tracequeue $C $Ed
        $ns at 1.0 "$self record"
```

```
}
TestSuite instproc setTraffic {} {
   $self instvar ns packetSize testTime aqm numSrcs s d
   #set up traffic
        $self create_DSQueue
# Set up TCP connections
        for {set i 1} {$i <= $numSrcs} {incr i} {
                set tcp($i) [$ns create-connection TCP/Newreno $s($i)  TCPSink $d($i) $i]
                $tcp($i) set packetSize_ $packetSize
                $tcp($i) set window_ 100
                set ftp($i) [$tcp($i) attach-source FTP]
                $ns at 0.1 "$ftp($i) start"
                $ns at $testTime "$ftp($i) stop"
        }
        set wind [open riowin.tr w]
        for {set i 1} {$i <= $numSrcs} {incr i} {
                $ns at 0.1 "$self plotWindow $tcp($i) $wind $i"
}
}
# Create the simulation scenario for the test suite.
TestSuite instproc create-scenario {} {
   $self instvar ns
        $self setTopo
        $self setTraffic
}
```

```
TestSuite instproc record {} {
        $self instvar qfile trq ns sum n
        set time 0.1
        set now [$ns now]
        $qfile instvar pkts_
        set sum [expr $sum + $pkts_]
        set n [expr $n + 1]
        puts $trq "$now $pkts_"
        $ns at [expr $now+$time] "$self record"
}
TestSuite instproc enable_tracequeue {c e} {
        $self instvar qfile ns tf
        set qfile [$ns monitor-queue $c $e [open qf.tr w] 0.1]
        [$ns link $c $e] queue-sample-timeout
        $ns trace-queue $c $e $tf
}
TestSuite instproc init {} {
   $self instvar ns tf trq testTime packetSize numSrcs MaxQueueSize linkBW
        $self instvar cir1 cir2
        $self instvar aqm
        $self instvar pmax_in pmax_out pmax min_in max_in min_out max_out minth
maxth in_wq out_wq
        $self instvar sum n
        global AQM
   set ns [new Simulator]
        set tf [open out.tr w]
```

```
        set trq [open traceq.tr w]
        set testTime 20.0
        set packetSize 1000
        set numSrcs 10
        set MaxQueueSize 100
        set linkBW 30
        set cir1 1000000
        set cir2 5000000
        $self set aqm $AQM
        if {$aqm == "RIO"}{
                        set min_in 40
                        set max_in 80
                        set pmax_in 0.02
                        set min_out 10
                        set max_out 30
                        set pmax_out 0.1
                        set in_wq 0.002
                        set out_wq 0.01
                }
        set sum 0
        set n 0
    $self next
    $ns at [expr $testTime + 1.0] "$self finish"
}

TestSuite instproc run {} {
```

```
    $self instvar ns
    $self create-scenario
    $ns run
}

TestSuite instproc plotWindow {tcpSource file k} {
    $self instvar ns numSrcs
        set time 0.03
        set now         [$ns now]
        set cwnd [$tcpSource set cwnd_]
        if {$k == 1} {
                puts -nonewline $file "$now $cwnd\t"
        } elseif {$k > 1 && $k < $numSrcs} {
                puts -nonewline $file "$cwnd\t"
        } elseif {$k == $numSrcs} {
                puts -nonewline $file "$cwnd\n"
        }
        $ns at [expr $now+$time] "$self plotWindow $tcpSource $file $k"
}


TestSuite instproc plotQueue file  {
        $self instvar trq
        #
        # Plot the queue size and average queue size, for RED gateways.
        #
```

```
        set awkCode {
                {
                        avg = 0.95*avg + 0.05*$2;
                        printf("%f\t%f\n", $1, avg) > "avg.tr";
                }
        }

        if { [info exists trq] } {
                close $trq
        }
        exec awk $awkCode traceq.tr

        set f [open queue.tr w]
        puts $f "TitleText: $file"
        puts $f "Device: Postscript"

        puts $f \"queue
        exec cat traceq.tr >@ $f
        puts $f \n\"ave_queue
        exec cat avg.tr >@ $f
        close $f
        exec xgraph queue.tr -geometry 400x300 -t "RIO Queue" -zg black -zw 1 -bb -nb -
bg white -x "Time(s)" -y "Length(pkts)" &
}
TestSuite instproc getStats {} {
        $self instvar qCEd testTime linkBW packetSize sum n aqm
```

```
        set TotPkts [$qCEd getStat pkts]
        set drops [$qCEd getStat drops]
        set edrops [$qCEd getStat edrops]
        set TxPkts [expr $TotPkts - $drops - $edrops]
        puts "Totpkts = $TotPkts \nTxPkts = $TxPkts \nldrops = $drops \nedrops = $edrops"
        for {set i 10} {$i<=12} {incr i} {
                set Totpkts_VQ($i) [$qCEd getStat pkts $i]
                set drops_VQ($i) [$qCEd getStat drops $i]
                set edrops_VQ($i) [$qCEd getStat edrops $i]
                set TxPkts_VQ($i) [expr $Totpkts_VQ($i) - $drops_VQ($i) - $edrops_VQ($i)]
#       puts "$i $pkts_VQ($i) $TxPkts_VQ($i) $drops_VQ($i) $edrops_VQ($i)"
        }

        set avg [expr 1.0*$sum/$n]
        set delay [expr 8*$avg/$linkBW]
        set totthru [expr ($TxPkts*$packetSize*8)/$testTime]
        set util [expr $totthru/($linkBW*10000)]
        set dgreen [expr 100*$edrops_VQ(10)/$TotPkts]
        puts "totthru = $totthru"
#       set str [format "%0.0f\t%0.2f\t%0.2f\t%0.2f\t%0.2f" $per $avg $delay $util $dgreen]
#       puts $str
}
TestSuite instproc finish {} {
   global tracefd
```

```
    $self instvar ns tf aqm qCEd nf
        $self plotQueue "$aqm"
#       $self getStats
        close $tf
    exit 0
}
set AQM ARIO
proc runtest {arg} {
        global AQM
        set b [llength $arg]
        if {$b == 0} {
                set AQM RIO
        } elseif {$b == 1} {
                set AQM $arg
        } else {
                usage
        }
        set t [new TestSuite]
        $t run
}
global argv arg0
runtest $argv
```