ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

NGUYỄN THANH TÙNG – 23521744
ĐỖ ĐỨC MINH TRIẾT - 23521650

BÁO CÁO ĐỒ ÁN CUỐI KỲ
EVALUATING NETWORK CONGESTION CONTROL WITH IPERF3 AND LINUX TRAFFIC CONTROL (TC/NETEM)

GIẢNG VIÊN HƯỚNG DẪN
LÊ TRUNG QUÂN
NGUYỄN VĂN BẢO

TP. HỒ CHÍ MINH, 2025

Mục lục

Bảng

Hình ảnh

I. GIỚI THIỆU ĐỀ TÀI

Trong bối cảnh các hệ thống mạng ngày càng đa dạng và phức tạp, việc đánh giá và tối ưu hiệu suất của các cơ chế điều khiển tắc nghẽn (Congestion Control) đóng vai trò quan trọng nhằm đảm bảo chất lượng truyền tải dữ liệu. Khi mạng phải đối mặt với các điều kiện bất lợi như độ trễ cao, giới hạn băng thông, hoặc mất gói, hiệu quả của thuật toán điều khiển tắc nghẽn ảnh hưởng trực tiếp đến tốc độ và độ ổn định của kết nối.

Đề tài "Evaluating Network Congestion Control with iperf3 and Linux Traffic Control (TC/NetEm)" nhằm mục tiêu xây dựng môi trường mô phỏng và đánh giá các cơ chế điều khiển tắc nghẽn TCP trong nhiều kịch bản mạng khác nhau. Bằng cách kết hợp công cụ iperf3 để tạo luồng lưu lượng và Linux Traffic Control (TC/NetEm) để cấu hình độ trễ, giới hạn băng thông và các lỗi mạng, nhóm tiến hành thực nghiệm nhằm đo lường các chỉ số hiệu năng như throughput, RTT, độ trễ trung bình, kích thước cửa sổ nghẽn (cwnd) và hành vi ACK trong các điều kiện khác nhau.

Mục tiêu của nghiên cứu này là so sánh hiệu quả giữa các thuật toán điều khiển tắc nghẽn (CUBIC, BBR) và các hàng đợi mạng (qdisc) khác nhau dưới tác động của các yếu tố gây suy giảm hiệu năng (impairments). Qua đó, đề tài giúp hiểu rõ hơn về cách thức các thuật toán phản ứng với tắc nghẽn, hỗ trợ việc lựa chọn cấu hình phù hợp cho từng môi trường mạng thực tế.

II. CƠ SỞ LÝ THUYẾT

2.1. ARRIVAL PROCESSES (QUÁ TRÌNH ĐẾN)

Trong lý thuyết hàng đợi, **arrival process** mô tả cách các gói tin (hoặc khách hàng) đến hệ thống theo thời gian.

Các đặc trưng chính của arrival process gồm:

- **Tốc độ đến trung bình (λ - arrival rate)**: số gói đến trung bình trong một đơn vị thời gian.

- **Phân phối thời gian giữa các lần đến (inter-arrival time)**: thời gian giữa hai gói tin liên tiếp.

Các mô hình arrival phổ biến:

- **Deterministic Arrival (D)**: gói đến theo chu kỳ cố định.
- **Poisson Process (M)**: thời gian giữa các gói đến tuân theo phân phối mũ — đây là mô hình phổ biến nhất do tính chất "không nhớ" (*memoryless*).
- **General Arrival (G)**: thời gian đến có phân phối bất kỳ.

Trong mô hình hàng đợi, ký hiệu M/D/1, M/M/1, v.v… biểu diễn:

- Chữ đầu: loại arrival process (M – Markov/Poisson, D – deterministic).
- Chữ thứ hai: loại service time distribution.
- Số cuối: số lượng server (thường là 1).

2.2. MÔ HÌNH HÀNG ĐỢI CƠ BẢN

Hệ thống hàng đợi được mô tả bởi các thành phần:

Thành phần	**Ý nghĩa**
Arrival Process	Cách các gói tin đến hàng đợi
Service Process	Cách hệ thống xử lý (truyền) từng gói
Queue Discipline	Quy tắc chọn gói tiếp theo để phục vụ (ví dụ: FIFO, Priority, Round Robin)
Number of Servers	Số tiến trình phục vụ đồng thời
System Capacity	Số lượng gói tối đa hệ thống có thể chứa
Population Source	Tổng số nguồn có thể sinh gói (hữu hạn hoặc vô hạn)

Bảng 1. Hệ thống hàng đợi

1. Một số mô hình hàng đợi chuẩn:

- **M/M/1**: Arrival Poisson, service time exponential, 1 server.

$$E[N] = \frac{\rho}{1-\rho}, E[T] = \frac{1}{\mu - \lambda}$$

với $\rho = \lambda/\mu$.

- **M/M/1/K**: Có giới hạn hàng đợi K (mất gói khi hàng đầy).
- **M/D/1**: Service time cố định, giúp giảm độ dao động trễ.

Các mô hình này giúp mô phỏng và đánh giá **hiệu năng của bộ định tuyến hoặc link mạng**, như độ trễ trung bình, xác suất mất gói và thông lượng.

2.3. RED (RANDOM EARLY DETECTION)

RED là một cơ chế quản lý hàng đợi chủ động (**AQM – Active Queue Management**) nhằm tránh hiện tượng **tắc nghẽn toàn cục (global synchronization)** trong mạng TCP.

1. Nguyên lý hoạt động:

- RED không chờ hàng đợi đầy mới loại bỏ gói, mà **bắt đầu loại ngẫu nhiên sớm hơn**, khi độ dài trung bình của hàng vượt ngưỡng.
- Tính **độ dài trung bình (avg_len)** bằng bộ lọc trung bình trượt theo hàm mũ:

$$avg_{len} = (1 - w_q) \times avg_{len} + w_q \times q_{inst}$$

- Khi avg_{len} vượt qua **min_th** (ngưỡng dưới), gói sẽ bị **drop** hoặc **mark (ECN)** với xác suất tăng dần cho đến **max_th** (ngưỡng trên).

2. Ưu điểm:

- Giảm hiện tượng đồng bộ hóa mất gói của TCP.
- Duy trì độ trễ ổn định.
- Tăng thông lượng và hiệu quả đường truyền.

3. Nhược điểm:

- Khó cấu hình các tham số (min_th, max_th, w_q).

- Không tối ưu trong các mạng có tải thay đổi nhanh.

2.4. CUBIC (TCP CONGESTION CONTROL)

Cubic là thuật toán điều khiển tắc nghẽn TCP mặc định trong hầu hết các nhân Linux hiện nay (thay thế cho Reno và BIC).

1. Ý tưởng chính:

- Thay vì tăng tuyến tính như **Reno**, Cubic tăng theo **hàm bậc ba (cubic function)** của thời gian kể từ lần mất gói gần nhất.

$$cwnd(t) = C \times (t - K)^3 + W_{max}$$

  - $cwnd(t)$: cửa sổ nghẽn tại thời điểm t
  - W_{max}: cửa sổ tại thời điểm mất gói gần nhất
  - C: hằng số tăng
  - K: khoảng thời gian để cwnd đạt lại W_{max}

2. Ưu điểm:

- Phục hồi nhanh sau mất gói.
- Tận dụng tốt băng thông trong mạng tốc độ cao – độ trễ lớn (high-BDP).
- Giữ ổn định khi tải cao.

2.5. TCP BBR (Bottleneck Bandwidth and Round-trip Propagation Time)

1. Giới thiệu chung

BBR (Bottleneck Bandwidth and RTT) là một thuật toán TCP congestion control do Google phát triển và công bố năm 2016, hiện được tích hợp trong Linux kernel từ phiên bản 4.9 trở lên.

Không giống các thuật toán truyền thống như Reno hay Cubic, vốn dựa vào tín hiệu mất gói (packet loss) để xác định tắc nghẽn, BBR dựa trên mô hình băng thông – độ trễ thực tế của đường truyền để duy trì tốc độ truyền tối ưu.

Mục tiêu của BBR là tối đa hóa thông lượng mà vẫn giữ độ trễ thấp, giúp tránh hiện tượng *bufferbloat* (độ trễ tăng cao do hàng đợi bị đầy).

2. Nguyên lý hoạt động

BBR xây dựng một **mô hình ảo của đường truyền** dựa trên hai đại lượng cơ bản:

Tham số	**Ý nghĩa**
BtlBw (Bottleneck Bandwidth)	Băng thông tối đa mà liên kết có thể truyền (tính bằng bytes/s).
RTprop (Round-trip propagation time)	Thời gian truyền tối thiểu của một gói đi và về mà không bị hàng đợi ảnh hưởng.

Bảng 2. hai đại lượng cơ bản

=> Hai giá trị này được ước lượng liên tục dựa trên kết quả đo thực tế từ các gói ACK nhận được.

- Công suất gửi tối ưu được BBR tính là: **Pacing Rate=BtlBw**
- Kích thước cửa sổ nghẽn (congestion window): **cwnd=BtlBw×Rtprop**

=> Điều này giúp BBR gửi lượng dữ liệu vừa đủ để "lấp đầy đường ống" (*pipe*), không gây tắc nghẽn hàng đợi như Cubic.

3. Các giai đoạn (phases) của BBR

BBR hoạt động theo chu kỳ, gồm 4 pha chính:

Pha	**Mục tiêu**	**Mô tả**
Startup	Tìm băng thông cực đại	Tăng tốc độ gửi theo cấp số nhân cho đến khi không còn thấy băng thông tăng.
Drain	Xả hàng đợi	Giảm tốc độ gửi để làm rỗng hàng đợi tạm thời.
ProbeBW	Duy trì và kiểm tra băng thông	Tăng/giảm nhẹ tốc độ gửi để xác minh liệu có thêm băng

		thông khả dụng không.
ProbeRTT	Đo lại RTT thực	Giảm tốc độ gửi trong thời gian ngắn để đo lại RTprop chính xác.

Bảng 3. 4 pha chính BBR hoạt động

- Chu kỳ ProbeBW – ProbeRTT lặp liên tục để duy trì mô hình cập nhật của mạng.

4. So sánh BBR với Cubic

Đặc tính	**Cubic**	**BBR**
Nguyên lý	Dựa trên *packet loss*	Dựa trên *băng thông và RTT*
Điều chỉnh cwnd	Hàm bậc ba theo thời gian	Theo công thức cwnd = BtlBw × RTprop
Độ trễ trung bình	Có thể cao (bufferbloat)	Thấp và ổn định
Tốc độ phục hồi sau mất gói	Giảm cwnd mạnh	Ít bị ảnh hưởng bởi mất gói
Hiệu suất ở mạng tốc độ cao	Giảm hiệu quả do phụ thuộc vào RTT	Hiệu quả cao trong mạng có RTT lớn
Phù hợp	Liên kết truyền thống	Liên kết tốc độ cao, mạng 5G, WAN, Data Center

Bảng 4. So sánh BBR với Cubic

=> Nhờ đó, **BBR thường đạt thông lượng cao hơn và độ trễ thấp hơn** trong hầu hết các môi trường mạng hiện đại.

5. Ưu điểm và hạn chế

Ưu điểm:

- Duy trì **throughput cao** mà **độ trễ thấp**.
- **Không phụ thuộc vào packet loss**, tránh giảm tốc độ không cần thiết.
- **Ổn định** trong môi trường có RTT và băng thông thay đổi.
- Giảm đáng kể hiện tượng **bufferbloat**.

Hạn chế:

- **Không công bằng với Cubic** trong một số trường hợp: BBR có thể chiếm nhiều băng thông hơn.
- Phiên bản đầu tiên (BBR v1) có thể gây *burst* trong liên kết nhỏ; Google đã khắc phục trong **BBR v2** (ra mắt 2021).
- Cần **ACK chính xác và ổn định**, nếu bị trễ hoặc mất ACK, hiệu quả giảm.

6. Ứng dụng và triển khai

Được **Google triển khai trên YouTube, Google Cloud, và gRPC**, giúp giảm độ trễ video streaming hàng chục phần trăm.

Có thể kích hoạt trên Linux:

```
sudo sysctl -w net.ipv4.tcp_congestion_control=bbr
```

Bảng 5. Lệnh kích hoạt BBR trên Linux

Kiểm tra trạng thái:

```
sysctl net.ipv4.tcp_congestion_control
```

Bảng 6. Lệnh kiểm tra trạng thái trên Linux

2.6. Các hàng đợi (qdisc) phổ biến trong Linux

Qdisc (Queueing Discipline) là cơ chế trong Linux kernel để quản lý cách gói tin được xếp hàng và gửi đi. Một số qdisc thường dùng:

1. pfifo_fast / pfifo

- **pfifo**: đơn giản nhất, xếp hàng theo **FIFO (First In First Out)**, bỏ gói khi đầy.
- Thích hợp cho thử nghiệm cơ bản hoặc môi trường không cần kiểm soát trễ.

2. RED (Random Early Detection)

- Như mô tả ở trên, là qdisc AQM chủ động để giảm tắc nghẽn.
- Cấu hình trong Linux:

```
tc qdisc add dev eth0 root red limit 1000 min 200 max 600 avpkt 1000 burst 20 probability 0.02
```

Bảng 7. Ví dụ về cấu hình trong Linux của RED

3. fq_codel (Fair Queue Controlled Delay)

- Là qdisc hiện đại, **kết hợp Codel (Controlled Delay)** và **Fair Queuing**.
- Tự động phát hiện và loại bỏ gói trễ quá lâu, giảm bufferbloat.
- Không cần tinh chỉnh tham số, hoạt động hiệu quả trong mạng thực tế.

Qdisc	Đặc điểm chính	Loại
pfifo	Đơn giản, FIFO	Passive
RED	Loại ngẫu nhiên sớm, chống tắc nghẽn	AQM
fq_codel	Fair Queue + kiểm soát độ trễ tự động	AQM nâng cao

Bảng 8. So sánh qdisc

2.7. iperf - Công Cụ Đo Băng Thông và Hiệu Năng

Mục đích: iperf3 là công cụ dòng lệnh được sử dụng rộng rãi để đo lường thông lượng tối đa (maximum throughput) trên mạng IP. Nó có thể kiểm tra hiệu năng của cả TCP và UDP.

Nguyên lý hoạt động:

- Thiết lập mô hình Client-Server. Máy chủ (server) lắng nghe kết nối, máy khách (client) khởi tạo và gửi dữ liệu.
- iperf3 tạo ra các luồng dữ liệu (streams) và gửi đi trong một khoảng thời gian xác định, sau đó báo cáo lại lượng dữ liệu đã truyền tải, từ đó tính toán ra băng thông đạt được.

Các thông số quan trọng có thể đo được:

- Bandwidth (Băng thông): Tốc độ truyền tải dữ liệu (Mbits/sec, MBytes/sec).
- Jitter (Độ biến động trễ): Sự thay đổi về độ trễ giữa các gói tin (chỉ cho UDP).
- Packet Loss (Tỷ lệ mất gói): Phần trăm gói tin bị mất trên đường truyền (chỉ cho UDP).

Ứng dụng trong đồ án: Dùng để kiểm chứng băng thông thực tế của liên kết mạng, so sánh hiệu năng giữa các cấu hình khác nhau (ví dụ: thay đổi kích thước cửa sổ TCP - window size), hoặc đánh giá hiệu quả của một giải pháp mạng mới.

2.8. tc (Traffic Control) và netem - Công Cụ Điều Khiển và Mô Phỏng Mạng

Mục đích: tc là công cụ mạnh mẽ trong Linux để điều khiển lưu lượng mạng. Kết hợp với module netem (Network Emulator), nó cho phép mô phỏng các điều kiện mạng không lý tưởng ngay trên một máy chủ.

Nguyên lý hoạt động:

- tc hoạt động bằng cách áp dụng các "qdisc" (queueing disciplines - kỷ luật xếp hàng) lên các giao diện mạng. Các qdisc này sẽ kiểm soát cách các gói tin được gửi đi, nhận về hoặc xử lý.
- netem là một loại qdisc đặc biệt, cung cấp khả năng thêm độ trễ, tỷ lệ mất gói, jitter, và nhiều hiện tượng mạng phức tạp khác.

Các tình huống mạng có thể mô phỏng:

- Độ trễ (Delay): netem delay 100ms 10ms
- Mất gói (Packet Loss): netem loss 5%
- Jitter: netem delay 100ms 20ms (độ trễ trung bình 100ms, dao động ±20ms).
- Gói tin trùng lặp (Duplication), Đảo thứ tự (Reordering), v.v.

Ứng dụng trong đồ án: Tạo ra một môi trường thử nghiệm có điều kiện mạng giống thực tế (ví dụ: mạng di động, đường truyền vệ tinh, hoặc mạng WAN có độ trễ cao) để đánh giá xem ứng dụng của bạn hoạt động như thế nào trong những điều kiện đó.

2.9. tcpdump - Công Cụ Phân Tích Gói Tin Mạng

Mục đích: tcpdump là công cụ phân tích giao thức mạng dựa trên dòng lệnh. Nó cho phép "bắt" (capture) và hiển thị các gói tin đang được truyền đi hoặc nhận về trên một giao diện mạng.

Nguyên lý hoạt động:

- tcpdump sử dụng thư viện libpcap để lấy các gói tin từ lớp driver của card mạng.
- Nó có thể lọc và hiển thị nội dung của các gói tin dựa trên nhiều tiêu chí (địa chỉ IP, cổng, giao thức, cờ TCP, v.v.).

Thông tin chi tiết có thể thu thập:

- Toàn bộ header của gói tin (Ethernet, IP, TCP/UDP, v.v.).

- Dữ liệu payload (nếu không mã hóa).
- Thời gian chính xác khi gói tin được gửi/ nhận.
- Các sự kiện trong quá trình bắt tay TCP (SYN, SYN-ACK, ACK) và kết thúc kết nối (FIN, RST).

Ứng dụng trong đồ án: Phân tích sâu các vấn đề về hiệu năng (ví dụ: xác định nguyên nhân độ trễ bằng cách phân tích các khoảng thời gian giữa các gói tin, phân tích hiện tượng tắc nghẽn thông qua cờ TCP Window, hoặc gỡ lỗi các kết nối không thành công).

2.10. ss - Công Cụ Thống Kê Socket

Mục đích: ss (socket statistics) là công cụ hiện đại thay thế cho netstat. Nó dùng để điều tra thông tin về socket, kết nối mạng, và các bảng routing. [6]

Nguyên lý hoạt động:

- ss lấy thông tin trực tiếp từ nhân Linux thông qua giao diện netlink, làm cho nó nhanh hơn và hiệu quả hơn netstat trên các hệ thống có nhiều kết nối.

Các thông tin quan trọng:

- Danh sách tất cả các kết nối TCP, UDP, và UNIX socket.
- Trạng thái của kết nối (ESTABLISHED, TIME-WAIT, LISTEN, v.v.).
- Thông tin về bộ nhớ sử dụng (send/receive buffer), số gói tin đang chờ,...
- Thông tin chi tiết về cài đặt TCP (như rtt, cwnd - congestion window).

Ứng dụng trong đồ án: Giám sát trạng thái kết nối thời gian thực, phân tích vấn đề về kết nối (ví dụ: quá nhiều kết nối ở trạng thái TIME-WAIT), và thu thập các chỉ số hiệu năng cấp độ socket như kích thước cửa sổ tắc nghẽn.

2.11. ifstat - Công Cụ Thống Kê Lưu Lượng Giao Diện Mạng

Mục đích: ifstat là công cụ đơn giản để giám sát lưu lượng mạng trên các giao diện (interface) theo thời gian thực.

Nguyên lý hoạt động:

- ifstat định kỳ đọc các bộ đếm trong file /proc/net/dev (chứa thống kê về từng giao diện mạng) và tính toán tốc độ truyền tải dựa trên sự chênh lệch giữa các lần đọc.

Các thông tin hiển thị:

- Băng thông inbound (KB/s, MB/s) và outbound (KB/s, MB/s) của từng giao diện.
- Tổng số byte/packet đã truyền đi và nhận về.

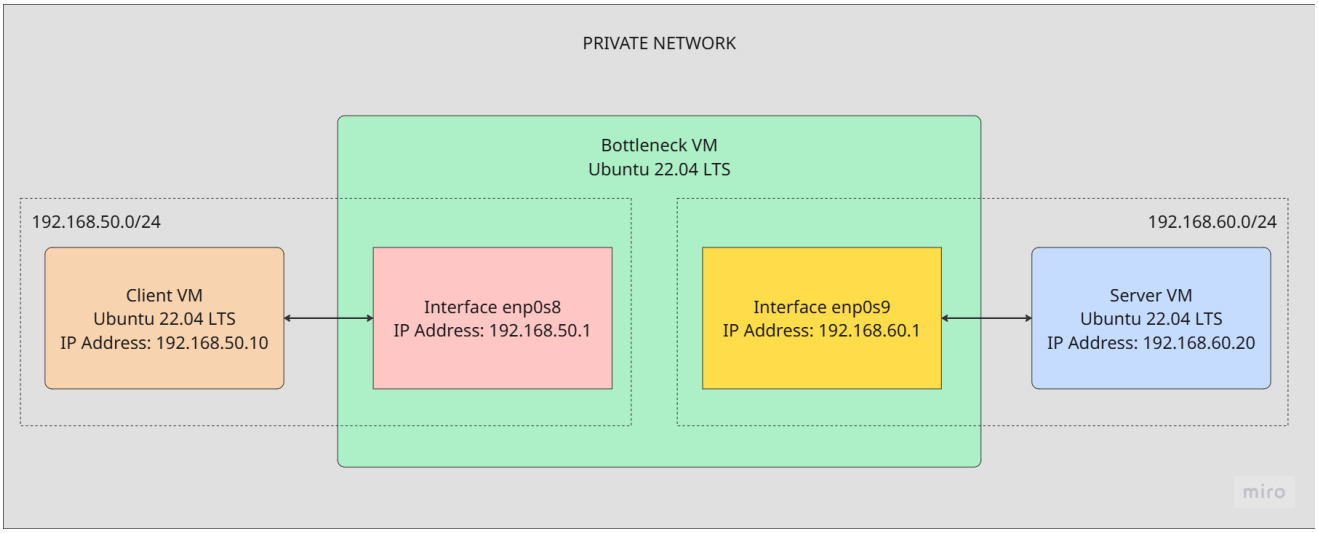
Ứng dụng trong đồ án: Cung cấp một cái nhìn tổng quan, trực quan về việc sử dụng băng thông trên các giao diện mạng. Hữu ích để xác nhận xem liệu lưu lượng thử nghiệm từ iperf3 có thực sự được tạo ra hay không, hoặc để giám sát mức độ tải tổng thể của hệ thống.

III. MÔ HÌNH TRIỂN KHAI

3.1. Mô hình tổng quan

- 3 máy ảo Ubuntu 22.04 LTS: Client, Bottleneck, Server.

3.2. Sơ đồ hệ thống



Hình 1. Sơ đồ hệ thống

3.3. Kịch bản thử nghiệm

- Trường hợp bandwidth bình thường và bandwidth bị giới hạn xuống 3 Mbps, 1 flow, CUBIC cho fq_codel/pfifo/RED.

- Trường hợp bandwidth bình thường và bandwidth bị giới hạn xuống 3 Mbps, 5/10 flows với impairment (delay, jitter, loss) cho fq_codel/pfifo/RED cùng với CUBIC/RED.

3.4. Cài đặt các công cụ và các thiết bị cần thiết:

- Cài 3 máy ảo ubuntu 22.04 trên VirtualBox, cấu hình interface phù hợp cho các máy ảo.

- Sau khi cấu hình 3 máy ảo, thực hiện update và cài các công cụ cần thiết trên 3 máy:

- Update: Đảm bảo cài công cụ không bị lỗi

```
sudo apt -y update
```

Bảng 9. Lệnh update hệ thống

- iperf3 [1]:

```
sudo apt -y install iperf3
```

Bảng 10. Lệnh cài iperf3

- iproute2 (tc, netem, ss) [12]:

```
sudo apt install iproute2
```

Bảng 11. Lệnh cài iproute2 (tc, netem, ss):

- tcpdump [5]:

```
sudo apt-get install tcpdump
```

Bảng 12. Lệnh cài tcpdump

- tshark [2]:

```
sudo add-apt-repository -y ppa:wireshark-dev/stable
sudo apt install -y tshark
sudo usermod -a -G wireshark $USER
```

Bảng 13. Lệnh cài tshark

- ifstat [13]:

```
sudo apt-get install ifstat
```

Bảng 14. Lệnh cài ifstat

- Cài thêm công cụ và cấu hình trên client:

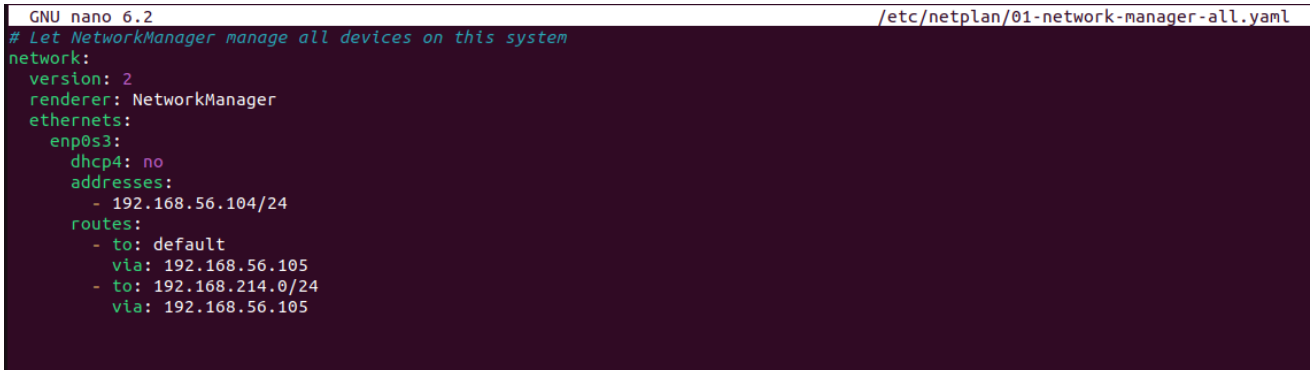
- python3, pandas, matplotlib [3][4]:

```
sudo apt install python3-pip -y
python3 -m pip install pandas
```

pip3 install matplotlib

Bảng 15. Lệnh cài python3,pandas,matplotlib

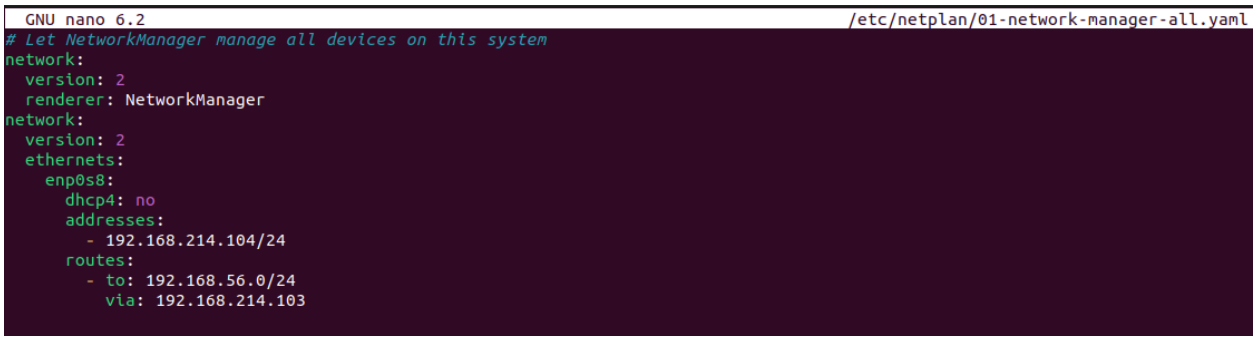
- Cấu hình file network-manager-all.yml để khi reboot tự động có route:



Hình 2. File network-manager-all.yml trên client

- Cấu hình trên server:

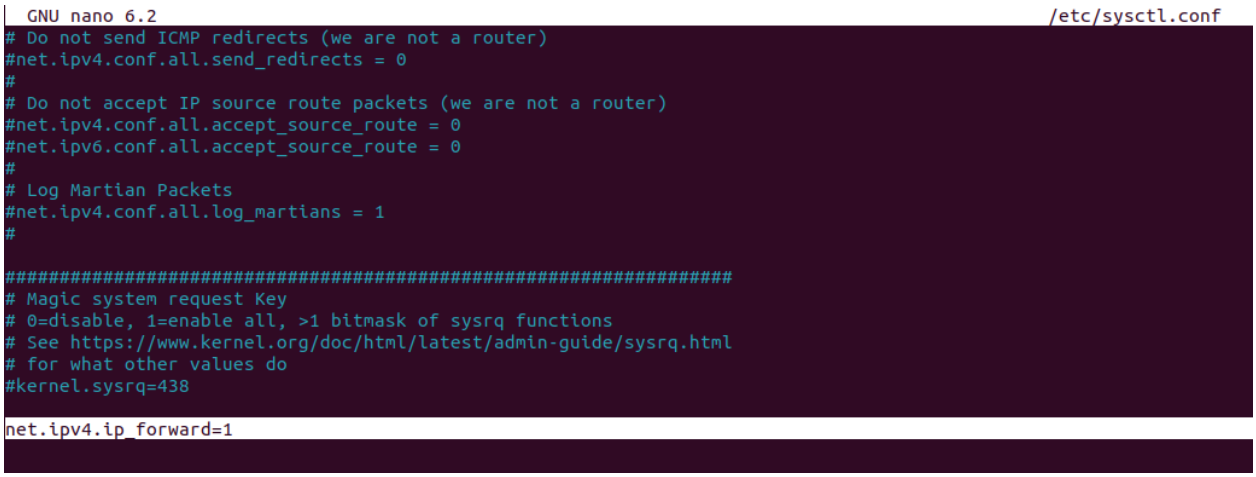
- Cấu hình file network-manager-all.yml để khi reboot tự động có route:



Hình 3. File network-manager-all.yml trên server

- Cấu hình trên bottleneck:

- Cấu hình file /etc/sysctl.conf trên bottleneck để khi reboot tự động bật ip forwarding:



Hình 4. File /etc/sysctl.conf trên bottleneck

3.5. Cấu hình script mô phỏng và code thu thập dữ liệu cho thực nghiệm:

1. run_experiments.sh:

Đây là một script chạy trong bash, dùng để chạy mô phỏng các kịch bản cần thực hiện, giúp việc tiến thành thực nghiệm trở nên nhanh chóng hơn.

a. Phần khai báo và cấu hình ban đầu:

```
#!/bin/bash


# ---- CONFIG ----
USER="root"
SERVER_IP="192.168.214.104"
BOTTLENECK_IP="192.168.56.105"
RUNS=3
SCENARIO="bwNORMAL+impairments_5_cubic_fq"
OUT_BASE="experiments"
TCP_TIME=15
UDP_TIME=15
UDP_BW="0"
TCP_FLOWS=5
UDP_FLOWS=5
```

```
CLIENT_IF="enp0s3"
SERVER_IF="enp0s8"
BOTTLENECK_IF="enp0s8"
SSH_OPTS="-o BatchMode=yes -o ConnectTimeout=8"
PORT=5201
# ---- END CONFIG ----

set -u
mkdir -p "${OUT_BASE}/${SCENARIO}"

timestamp() { date +"%F %T"; }
```

Bảng 16. Phần khai báo và cấu hình ban đầu

- #!/bin/bash: Chỉ định shell Bash để chạy script.
- Khai báo các biến cấu hình:

- USER="root": Người dùng SSH để kết nối tới các node từ xa.
- SERVER_IP, BOTTLENECK_IP: Địa chỉ IP của máy chủ và node bottleneck.
- RUNS=3: Số lần lặp lại phép đo.
- SCENARIO="bwNORMAL+impairments_5_cubic_fq": Tên kịch bản đo, dùng để đặt tên thư mục lưu kết quả.
- OUT_BASE="experiments": Thư mục gốc lưu toàn bộ kết quả đo.
- TCP_TIME=15, UDP_TIME=15: Thời gian chạy iperf3 cho TCP và UDP (giây).
- UDP_BW="0": Băng thông UDP (0 nghĩa là không giới hạn).
- TCP_FLOWS=5, UDP_FLOWS=5: Số luồng TCP và UDP.
- CLIENT_IF, SERVER_IF, BOTTLENECK_IF: Tên giao diện mạng trên từng node.
- SSH_OPTS="-o BatchMode=yes -o ConnectTimeout=8": Tùy chọn SSH để không cần nhập mật khẩu và có timeout 8 giây.
- PORT=5201: Cổng mặc định của iperf3.

- set -u: Script sẽ dừng nếu sử dụng biến chưa được khai báo.

- mkdir -p "${OUT_BASE}/${SCENARIO}": Tạo thư mục chứa kết quả cho toàn bộ kịch bản.

- timestamp() { date +"%F %T"; }: Hàm in ra thời gian hiện tại để ghi log.

b. Phần đo đạc chính:

```
for run in $(seq 1 "$RUNS"); do
  OUTDIR="${OUT_BASE}/${SCENARIO}/${SCENARIO}_run_${run}"
  mkdir -p "$OUTDIR"
  REMOTE_TMP="/tmp/exp_${SCENARIO}_run${run}"

  echo "$(timestamp) [MAIN] Starting run ${run}, output -> ${OUTDIR}"

  # ===== BOTTLENECK CAPTURE =====
     echo "$(timestamp) [BOTTLENECK] Start tcpdump & ifstat on ${BOTTLENECK_IP} (${BOTTLENECK_IF})"
  ssh $SSH_OPTS ${USER}@${BOTTLENECK_IP} "
    set -u
    rm -rf ${REMOTE_TMP} 2>/dev/null || true
    mkdir -p ${REMOTE_TMP}
    sudo pkill tcpdump || true
     sudo nohup tcpdump -i ${BOTTLENECK_IF} port ${PORT} and \(tcp or udp\) -s 128 -w ${REMOTE_TMP}/bottleneck.pcap >/dev/null 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/bottleneck_tcpdump.pid
                nohup ifstat -i ${BOTTLENECK_IF} -t 1 > ${REMOTE_TMP}/ifstat_bottleneck_${BOTTLENECK_IF}.log 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/ifstat_bottleneck.pid
  " || echo "$(timestamp) [WARN] SSH to bottleneck failed"
```

```
  # ===== SERVER CAPTURE =====
   echo "$(timestamp) [SERVER] Start tcpdump, ifstat, iperf3 server and ss on ${SERVER_IP} (${SERVER_IF})"
  ssh $SSH_OPTS ${USER}@${SERVER_IP} "
    set -u
    rm -rf ${REMOTE_TMP} 2>/dev/null || true
    mkdir -p ${REMOTE_TMP}
    sudo pkill tcpdump || true
    sudo nohup tcpdump -i ${SERVER_IF} port ${PORT} and \(tcp or udp\) -s 128 -w ${REMOTE_TMP}/server.pcap >/dev/null 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/server_tcpdump.pid

    pkill iperf3 || true
     nohup iperf3 -s > ${REMOTE_TMP}/iperf3_server.log 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/iperf3_server.pid

                nohup ifstat -i ${SERVER_IF} -t 1 > ${REMOTE_TMP}/ifstat_server_${SERVER_IF}.log 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/ifstat_server.pid

    SS_BIN=\$(command -v ss || true)
    if [ -z "\$SS_BIN" ]; then
      echo '[ERROR] ss not found on server' >&2
    else
              nohup bash -lc 'while true; do "\$SS_BIN" -tinm >> ${REMOTE_TMP}/ss_server.txt; sleep 1; done' > ${REMOTE_TMP}/ss_server.nohup 2>&1 < /dev/null & echo \$! > ${REMOTE_TMP}/ss_server.pid
    fi
```

```
  " || echo "$(timestamp) [WARN] SSH to server failed"

  # ===== CLIENT CAPTURE =====
   echo "$(timestamp) [CLIENT] Start local collectors (ss, ifstat) and lightweight tcpdump"
   nohup bash -c "while true; do ss -tinm >> \"${OUTDIR}/ss_client.txt\"; sleep 1; done" & CLIENT_SS_PID=$!
   nohup ifstat -i ${CLIENT_IF} -t 1 > "${OUTDIR}/ifstat_client_${CLIENT_IF}.log" 2>&1 < /dev/null & CLIENT_IFSTAT_PID=$!

   sudo pkill tcpdump || true
   nohup sudo tcpdump -i ${CLIENT_IF} port ${PORT} and \(tcp or udp\) -s 128 -w "${OUTDIR}/client_iperf3.pcap" >/dev/null 2>&1 < /dev/null & CLIENT_TCPDUMP_PID=$!
   sleep 1

   # ===== TCP TEST =====
   echo "$(timestamp) [TEST] Running TCP iperf3 (client -> ${SERVER_IP}) for ${TCP_TIME}s"
     iperf3 -c ${SERVER_IP} -P ${TCP_FLOWS} -t ${TCP_TIME} -J > "${OUTDIR}/tcp.json" || echo "$(timestamp) [WARN] iperf3 TCP returned non-zero"

   # ===== UDP TEST =====
   echo "$(timestamp) [TEST] Waiting 3s before UDP test to ensure server ready"
   sleep 3

   echo "$(timestamp) [TEST] Running UDP iperf3 (client -> ${SERVER_IP}) for ${UDP_TIME}s bw=${UDP_BW}"
   if [ "$UDP_BW" = "0" ]; then
```

```
    iperf3 -c ${SERVER_IP} -u -b 0 -P ${UDP_FLOWS} -t ${UDP_TIME} -J > "${OUTDIR}/udp.json" || echo "$(timestamp) [WARN] iperf3 UDP returned non-zero"
  else
      iperf3 -c ${SERVER_IP} -u -b ${UDP_BW} -P ${UDP_FLOWS} -t ${UDP_TIME} -J > "${OUTDIR}/udp.json" || echo "$(timestamp) [WARN] iperf3 UDP returned non-zero"
  fi
```

Bảng 17. Phần đo đạc chính

- for run in $(seq 1 "$RUNS"); do: Lặp lại quá trình đo đạc RUNS lần (theo cấu hình ban đầu).

- OUTDIR và REMOTE_TMP: Tạo thư mục lưu kết quả cục bộ và thư mục tạm trên các node từ xa.

- echo "$(timestamp) [MAIN] Starting run ...": In thông báo bắt đầu mỗi lần chạy đo.

- Giai đoạn BOTTLENECK CAPTURE:

- Kết nối SSH đến node bottleneck.
- Xóa dữ liệu tạm cũ và tạo thư mục mới.
- Dừng bất kỳ tiến trình tcpdump đang chạy.
- Khởi chạy tcpdump để bắt gói tin (TCP và UDP) tại giao diện bottleneck, lưu vào file .pcap.
- Chạy ifstat trên bottleneck để ghi lại lưu lượng mạng theo thời gian thực.

- Giai đoạn SERVER CAPTURE:

- Kết nối SSH đến node server.
- Xóa dữ liệu cũ và tạo thư mục tạm.
- Dừng tcpdump cũ nếu có.

- Khởi chạy tcpdump tại interface server để bắt gói tin và lưu vào file .pcap.
- Dừng tiến trình iperf3 cũ và khởi chạy iperf3 -s (chế độ server) để lắng nghe kết nối từ client.
- Chạy ifstat để ghi log tốc độ mạng.
- Kiểm tra sự tồn tại của lệnh ss, sau đó chạy vòng lặp thu thập thông tin socket TCP mỗi giây, ghi vào ss_server.txt.

- Giai đoạn CLIENT CAPTURE:

- Chạy ss trong vòng lặp để ghi thông tin kết nối vào ss_client.txt.
- Chạy ifstat để ghi log băng thông mạng tại interface client.
- Dừng tcpdump cũ (nếu có), rồi khởi chạy tcpdump để bắt gói tin của iperf3 tại client.
- sleep 1: Chờ 1 giây để đảm bảo mọi tiến trình đã khởi động xong.

- Giai đoạn chạy thử nghiệm TCP trên iperf3 ở chế độ client:

- echo "[TEST] Running TCP iperf3...": In thông báo bắt đầu đo TCP.
- -c ${SERVER_IP}: Kết nối đến server.
- -P ${TCP_FLOWS}: Sử dụng số luồng TCP song song.
- -t ${TCP_TIME}: Chạy trong thời gian quy định (15 giây).
- -J: Xuất kết quả dưới dạng JSON.
- Lưu kết quả vào tcp.json.

- Giai đoạn chạy thử nghiệm TCP trên iperf3 ở chế độ client:

- sleep 3: Chờ 3 giây để server ổn định sau thử nghiệm TCP.
- if ["$UDP_BW" = "0"]: Kiểm tra giá trị băng thông UDP. Nếu bằng 0 thì chạy iperf3 UDP không giới hạn tốc độ. Nếu khác 0 thì chạy UDP với băng thông giới hạn theo cấu hình.
- Lưu kết quả đầu ra dưới dạng JSON vào udp.json.

c. Phần kết thúc đo đạc và thu thập dữ liệu từ các node:

```
# ===== STOP CLIENT =====
 echo "$(timestamp) [CLIENT] Stopping local collectors..."
 kill ${CLIENT_SS_PID} ${CLIENT_IFSTAT_PID} 2>/dev/null || true
 echo "$(timestamp) [CLIENT] Stopping client tcpdump"
 sudo kill ${CLIENT_TCPDUMP_PID} 2>/dev/null || true
 sleep 1
 # ===== STOP SERVER =====
 echo "$(timestamp) [SERVER] Stopping remote collectors and changing ownership..."
 ssh $SSH_OPTS ${USER}@${SERVER_IP} "
  set -u
        [ -f ${REMOTE_TMP}/iperf3_server.pid ] && kill \$(cat
${REMOTE_TMP}/iperf3_server.pid) 2>/dev/null || true
      [ -f ${REMOTE_TMP}/server_tcpdump.pid ] && sudo kill \$(cat
${REMOTE_TMP}/server_tcpdump.pid) 2>/dev/null || true
        [ -f ${REMOTE_TMP}/ifstat_server.pid ] && kill \$(cat
${REMOTE_TMP}/ifstat_server.pid) 2>/dev/null || true
   [ -f ${REMOTE_TMP}/ss_server.pid ] && kill \$(cat ${REMOTE_TMP}/ss_server.pid)
2>/dev/null || true
  sudo chown ${USER}:${USER} ${REMOTE_TMP}/server.pcap 2>/dev/null || true
 " || echo "$(timestamp) [WARN] SSH to server stop failed"
 # ===== STOP BOTTLENECK =====
  echo "$(timestamp) [BOTTLENECK] Stopping remote collectors and changing
ownership..."
 ssh $SSH_OPTS ${USER}@${BOTTLENECK_IP} "
  set -u
      [ -f ${REMOTE_TMP}/bottleneck_tcpdump.pid ] && sudo kill \$(cat
${REMOTE_TMP}/bottleneck_tcpdump.pid) 2>/dev/null || true
        [ -f ${REMOTE_TMP}/ifstat_bottleneck.pid ] && kill \$(cat
```

```
${REMOTE_TMP}/ifstat_bottleneck.pid) 2>/dev/null || true
   sudo chown ${USER}:${USER} ${REMOTE_TMP}/bottleneck.pcap 2>/dev/null || true
  " || echo "$(timestamp) [WARN] SSH to bottleneck stop failed"
  # ===== COPY FILES =====
  echo "$(timestamp) [COPY] Copying pcaps and logs to ${OUTDIR}..."
                                          scp                                ${SSH_OPTS}
${USER}@${BOTTLENECK_IP}:"${REMOTE_TMP}/bottleneck.pcap"  "${OUTDIR}/"  ||
echo "$(timestamp) [WARN] scp bottleneck failed"
      scp     ${SSH_OPTS}     ${USER}@${SERVER_IP}:"${REMOTE_TMP}/server.pcap"
"${OUTDIR}/" || echo "$(timestamp) [WARN] scp server pcap failed"
   scp  ${SSH_OPTS}  ${USER}@${SERVER_IP}:"${REMOTE_TMP}/iperf3_server.log"
"${OUTDIR}/" || true
     scp     ${SSH_OPTS}     ${USER}@${SERVER_IP}:"${REMOTE_TMP}/ss_server.txt"
"${OUTDIR}/" || true
                                          scp                                ${SSH_OPTS}
${USER}@${SERVER_IP}:"${REMOTE_TMP}/ifstat_server_${SERVER_IF}.log"
"${OUTDIR}/" || true
                                          scp                                ${SSH_OPTS}
${USER}@${BOTTLENECK_IP}:"${REMOTE_TMP}/ifstat_bottleneck_${BOTTLENEC
K_IF}.log" "${OUTDIR}/" || true
  # cleanup
  ssh $SSH_OPTS ${USER}@${SERVER_IP} "rm -rf ${REMOTE_TMP}" 2>/dev/null ||
true
    ssh   $SSH_OPTS   ${USER}@${BOTTLENECK_IP}   "rm   -rf   ${REMOTE_TMP}"
2>/dev/null || true


  echo "$(timestamp) [DONE] Run ${run} complete. Results in ${OUTDIR}"
  echo "------------------------------------------------------------"
  sleep 5
```

```
done
echo "$(timestamp) All runs for ${SCENARIO} finished."
```

Bảng 18. Phần kết thúc đo đạc và thu thập dữ liệu từ các node

- Dừng các tiến trình tại client:

- In thông báo “[CLIENT] Stopping local collectors...”.
- Dừng các tiến trình ss, ifstat tại client bằng kill.
- Dừng tiến trình tcpdump tại client bằng sudo kill.
- sleep 1: Chờ 1 giây để đảm bảo tiến trình dừng hoàn toàn.

- Dừng các tiến trình tại server:

- Kết nối SSH đến server để dừng các tiến trình đo.
- iperf3_server.pid: dừng iperf3 server.
- server_tcpdump.pid: dừng tcpdump.
- ifstat_server.pid: dừng ifstat.
- ss_server.pid: dừng vòng lặp ghi ss.
- sudo chown: Đổi quyền sở hữu file .pcap để có thể sao chép về client.

- Dừng các tiến trình tại bottleneck:

- Kết nối SSH đến node bottleneck.
- Dừng tiến trình tcpdump và ifstat.
- Đổi quyền sở hữu file .pcap để có thể sao chép về client.

- Dùng scp để sao chép file từ bottleneck và server về thư mục OUTDIR:

- bottleneck.pcap, server.pcap: File chứa gói tin bắt được.
- iperf3_server.log: Log hoạt động iperf3 server.
- ss_server.txt: Thống kê socket TCP trên server.
- ifstat_server_*.log, ifstat_bottleneck_*.log: Log throughput mạng.
- Sau khi sao chép xong, xóa thư mục tạm REMOTE_TMP trên các node để giải phóng dung lượng.

- Kết thúc một lần đo:

- echo "[DONE] Run ${run} complete...": Thông báo hoàn thành một lần đo.
- sleep 5: Nghỉ 5 giây trước khi bắt đầu vòng lặp tiếp theo.

- Sau khi hoàn tất các lần đo, in thông báo: "All runs for ${SCENARIO} finished." để xác nhận toàn bộ kịch bản đo đã hoàn tất.ss

d. Kết quả:

- File script sẽ chạy các kịch bản và thu các dữ liệu sau đối với từng kịch bản:

- Các file pcap là các gói tin bắt được trên client, bottleneck, server.
- Các file log ifstat đo băng thông của interface theo thời gian trên client, bottleneck, server.
- Các file ss_client.txt, ss_server.txt đo trang thái socket của client và server.
- Các file tcp.json, udp.json thu dữ liệu truyền TCP, UDP từ iperf3 trên client.
- Các file log iperf3_server thu dữ liệu nhận từ iperf3 trên server.

2. summary.py:

Đây là file code python, file này có nhiệm vụ thu thập dữ liệu từ iperf3 và ifstat của các kịch bản dưới dạng file CSV, thuận tiện cho việc phân tích dữ liệu.

a. Phần khai báo và thiết lập ban đầu:

```
#!/usr/bin/env python3
import os
import re
import json
import pandas as pd


ROOT_DIR = "experiments"
```

Bảng 19. Phần khai báo và thiết lập ban đầu

- #!/usr/bin/env python3: Cho phép chạy script bằng Python 3 trên các hệ điều hành khác nhau.

- import os, re, json, pandas as pd: Import các thư viện cần thiết:

  - os: làm việc với file và thư mục.
  - re: xử lý biểu thức chính quy (regex).
  - json: đọc/ghi file JSON.
  - pandas: tạo bảng dữ liệu và xuất ra file CSV.

- ROOT_DIR = "experiments": Xác định thư mục gốc chứa toàn bộ dữ liệu các kịch bản.

b. Hàm get_flow_count(scenario_name):

```
def get_flow_count(scenario_name: str):
    m = re.search(r"_(\d+)_", scenario_name)
    if m:
        return int(m.group(1))
    return 1
```

Bảng 20. Hàm get_flow_count(scenario_name)

- Dùng regex để tìm số lượng flow trong tên scenario (ví dụ bw3_5 → 5).
- Nếu không tìm được, mặc định trả về 1.

c. Hàm parse_iperf_json(path):

```
def parse_iperf_json(path):
    """Parse iperf3 JSON (TCP or UDP)."""
    try:
        with open(path) as f:
            data = json.load(f)
    except Exception as e:
        print(f"[!] Error reading {path}: {e}")
        return {}

    if "error" in data:
```

```
        return {}

    proto = data.get("start", {}).get("test_start", {}).get("protocol", "").upper()
    end = data.get("end", {})

    if proto == "TCP":
        per_flow_bw, per_flow_retrans = [], []
        if "streams" in end:
            for s in end["streams"]:
                recv = s.get("receiver", {})
                send = s.get("sender", {})
                if "bits_per_second" in recv:
                    per_flow_bw.append(recv["bits_per_second"] / 1e6)
                    per_flow_retrans.append(send.get("retransmits", 0))
        elif "sum_received" in end:
            recv = end["sum_received"]
            send = end.get("sum_sent", {})
            per_flow_bw = [recv.get("bits_per_second", 0) / 1e6]
            per_flow_retrans = [send.get("retransmits", 0)]
        return {
            "protocol": proto,
            "per_flow_Mbps": per_flow_bw,
              "retrans": sum(per_flow_retrans) / len(per_flow_retrans) if per_flow_retrans
else 0,
        }

    if proto == "UDP":
        per_flow_bw, per_flow_loss, per_flow_jitter = [], [], []
        if "streams" in end:
```

```
        for s in end["streams"]:
            udp = s.get("udp", {})
            if "bits_per_second" in udp:
                per_flow_bw.append(udp["bits_per_second"] / 1e6)
                per_flow_loss.append(udp.get("lost_percent", 0))
                per_flow_jitter.append(udp.get("jitter_ms", 0))
    elif "sum" in end:
        s = end["sum"]
        per_flow_bw = [s.get("bits_per_second", 0) / 1e6]
        per_flow_loss = [s.get("lost_percent", 0)]
        per_flow_jitter = [s.get("jitter_ms", 0)]
    return {
        "protocol": proto,
        "per_flow_Mbps": per_flow_bw,
        "lost_pct": sum(per_flow_loss) / len(per_flow_loss) if per_flow_loss else 0,
        "jitter_ms": sum(per_flow_jitter) / len(per_flow_jitter) if per_flow_jitter else 0,
    }

return {}
```

Bảng 21. Hàm parse_iperf_json(path)

- Đọc file kết quả iperf3 (dạng JSON) và trích xuất thông tin về băng thông, retransmit (TCP) hoặc loss/jitter (UDP).

- Dùng json.load() để đọc dữ liệu JSON từ file.
- Nếu có lỗi hoặc không tồn tại, trả về {}.
- Lấy giao thức từ data["start"]["test_start"]["protocol"] → xác định là **TCP** hay **UDP**.

- Trường hợp TCP:

- Duyệt qua end["streams"] để lấy thông tin từng luồng: bits_per_second (băng thông), retransmits (số gói truyền lại).
- Nếu không có streams, lấy tổng từ sum_received và sum_sent.
- Trả về dict chứa: protocol, per_flow_Mbps, retransmits.

- Trường hợp UDP:

- Duyệt qua end["streams"] để lấy thông tin: bits_per_second (băng thông), lost_percent (tỷ lệ mất gói), jitter_ms (độ dao động thời gian truyền).
- Nếu không có streams, lấy dữ liệu từ data["end"]["sum"].
- Trả về dict chứa: protocol, per_flow_Mbps, lost_pct, jitter_ms.

- Nếu không phải TCP/UDP hợp lệ → trả về {}.

d. Hàm parse_ifstat(path):

```
def parse_ifstat_kB(path):
    """Parse ifstat average KB/s -> return (rx, tx)."""
    if not os.path.exists(path):
        return 0, 0
    rx, tx = [], []
    with open(path) as f:
        for line in f:
            if re.match(r"^\s*\d", line):
                parts = line.split()
                if len(parts) >= 3:
                    try:
                        rx.append(float(parts[1]))
                        tx.append(float(parts[2]))
                    except ValueError:
                        pass
```

```
    if not rx:
        return 0, 0
    return sum(rx) / len(rx), sum(tx) / len(tx)
```

Bảng 22. Hàm parse_ifstat(path):

- Đọc file ifstat.txt để tính tốc độ trung bình nhận (RX) và gửi (TX) dữ liệu theo KB/s.
- Nếu file không tồn tại → trả (0, 0).

- Đọc từng dòng trong file:

  - Chỉ xử lý các dòng bắt đầu bằng chữ số (chứa dữ liệu tốc độ).Nếu có "MB/s" → scale = 8.0.
  - Tách dòng theo khoảng trắng và lấy cột rx và tx.
  - Thêm vào danh sách rx[] và tx[] sau khi ép kiểu float.

- Nếu danh sách rỗng → trả về (0, 0).

- Trả về (trung bình rx, trung bình tx).

e. Hàm jain_fairness(values):

```
def jain_fairness(values):
    if not values or sum(values) == 0:
        return 0
    s = sum(values)
    return (s ** 2) / (len(values) * sum(v ** 2 for v in values))
```

Bảng 23. Hàm jain_fairness(values)

- Tính chỉ số công bằng Jain để đo độ chia sẻ băng thông giữa các flow theo công thức: fairness = $\frac{(\sum x_i)^2}{n*\sum(x_i^2)}$

- Nếu danh sách rỗng hoặc tổng bằng 0 thì trả về 0.

f. Hàm summarize_run(run_dir, flow_count):

```
def summarize_run(run_dir, flow_count):
    out = {"run_id": os.path.basename(run_dir)}
    tcp_path = os.path.join(run_dir, "tcp.json")
    udp_path = os.path.join(run_dir, "udp.json")

    tcp_flows, udp_flows = [], []
    tcp_retrans, udp_loss, udp_jitter = [], [], []

    # --- TCP ---
    if os.path.exists(tcp_path):
        data = parse_iperf_json(tcp_path)
        if data and data["protocol"] == "TCP":
            tcp_flows.extend(data["per_flow_Mbps"])
            tcp_retrans.append(data.get("retrans", 0))
    else:
        print(f"[!] Missing tcp.json in {run_dir}")

    # --- UDP ---
    if os.path.exists(udp_path):
        data = parse_iperf_json(udp_path)
        if data and data["protocol"] == "UDP":
            udp_flows.extend(data["per_flow_Mbps"])
            udp_loss.append(data.get("lost_pct", 0))
            udp_jitter.append(data.get("jitter_ms", 0))
    else:
        print(f"[!] Missing udp.json in {run_dir}")

    # --- Aggregates ---
```

```
    if tcp_flows:
        out["tcp_avg_bw_Mbps"] = sum(tcp_flows) / len(tcp_flows)
        out["tcp_fairness"] = jain_fairness(tcp_flows)
        out["tcp_avg_retrans"] = sum(tcp_retrans) / len(tcp_retrans)
    if udp_flows:
        out["udp_avg_bw_Mbps"] = sum(udp_flows) / len(udp_flows)
        out["udp_avg_lost_pct"] = sum(udp_loss) / len(udp_loss)
        out["udp_avg_jitter_ms"] = sum(udp_jitter) / len(udp_jitter)

    # --- ifstat ---
    roles = ["client", "bottleneck", "server"]
    for role in roles:
        pattern = f"ifstat_{role}_"
            file_match = [f for f in os.listdir(run_dir) if f.startswith(pattern) and
f.endswith(".log")]
        if not file_match:
            continue
        path = os.path.join(run_dir, file_match[0])
        _, tx_kBps = parse_ifstat_kB(path)
        mbps = tx_kBps * 8 / 1000  # KB/s -> Mbps
        out[f"{role}_bw"] = mbps

    return out
```

Bảng 24. Hàm summarize_run(run_dir, flow_count)

- Tổng hợp kết quả của một lần chạy thử nghiệm (run) trong thư mục run_dir, bao gồm dữ liệu TCP, UDP và ifstat.

- out = {"run_id": os.path.basename(run_dir)}: tạo dictionary chứa kết quả đầu ra, đặt tên run dựa trên thư mục.
- tcp_path, udp_path: xác định đường dẫn đến file tcp.json và udp.json trong thư mục.
- tcp_flows, udp_flows, tcp_retrans, udp_loss, udp_jitter: khởi tạo danh sách để lưu dữ liệu băng thông, retransmission, mất gói, và jitter.
- Kiểm tra tồn tại file tcp.json:

- Nếu có, gọi parse_iperf_json() để đọc dữ liệu.
- Nếu dữ liệu thuộc TCP, lấy throughput của từng flow và số lần retransmission.
- Nếu không có file, in cảnh báo.
- Kiểm tra tồn tại file udp.json:
- Nếu có, đọc dữ liệu bằng parse_iperf_json().
- Nếu dữ liệu thuộc UDP, lấy throughput, tỷ lệ mất gói, và jitter.
- Nếu không có file, in cảnh báo.

- Tính trung bình cho dữ liệu TCP:

- tcp_avg_bw_Mbps: trung bình băng thông TCP trên các flow.
- tcp_fairness: tính chỉ số công bằng Jain từ các flow.
- tcp_avg_retrans: trung bình số lần retransmission.

- Tính trung bình cho dữ liệu UDP:

- udp_avg_bw_Mbps: trung bình băng thông UDP.
- udp_avg_lost_pct: trung bình tỷ lệ mất gói.
- udp_avg_jitter_ms: trung bình jitter.

- Đọc dữ liệu ifstat của các vai trò client, bottleneck, server:

- Tìm file ifstat_<role>_*.log trong thư mục.
- Nếu có, gọi parse_ifstat_kB() để lấy tốc độ truyền (KB/s).
- Chuyển đổi sang Mbps (KB/s × 8 / 1000) và lưu vào out.

- Trả về out chứa toàn bộ kết quả tổng hợp của run.

g. Hàm summarize_run(run_dir, flow_count):

```
def summarize_scenario(path, flow_count):
    runs = sorted([
        os.path.join(path, d) for d in os.listdir(path)
        if os.path.isdir(os.path.join(path, d)) and "_run_" in d
    ])

    rows = []
    for r in runs:
        print(f"[*] Processing {r}")
        rows.append(summarize_run(r, flow_count))

    if not rows:
        print(f"[!] No runs found in {path}")
        return None

    df = pd.DataFrame(rows)
    avg = df.select_dtypes("number").mean()
    avg_row = {"run_id": "avg"} | avg.to_dict()
    df = pd.concat([df, pd.DataFrame([avg_row])], ignore_index=True)

    out_path = os.path.join(path, "summary.csv")
    df.to_csv(out_path, index=False)
    print(f"[*] Saved {out_path}")

    return os.path.basename(path), avg
```

Bảng 25. Hàm summarize_run(run_dir, flow_count)

- Tổng hợp kết quả của toàn bộ các run trong một scenario.

- Tìm tất cả thư mục con bắt đầu bằng "run".

- Với mỗi run: In tên run đang xử lý. Gọi summarize_run() và thêm kết quả vào rows.

- Nếu không có run nào thì in cảnh báo và dừng.

- Chuyển rows thành DataFrame df.

- Tính trung bình toàn bộ các cột số học (df.mean()), tạo hàng avg_row.

- Gộp avg_row vào cuối bảng.

- Ghi file summary.csv trong thư mục scenario.

- Trả về (tên_scenario, giá_trị_trung_bình).

h. Hàm main() và thực thi chính:

```
def main():
    scenario_summaries = []
    for scen in sorted(os.listdir(ROOT_DIR)):
        scen_path = os.path.join(ROOT_DIR, scen)
        if not os.path.isdir(scen_path):
            continue
        flow_count = get_flow_count(scen)
        print(f"\n=== Scenario: {scen} ===")
        result = summarize_scenario(scen_path, flow_count)
        if result:
            scen_name, avg_metrics = result
            avg_metrics["scenario"] = scen_name
            scenario_summaries.append(avg_metrics)

    if scenario_summaries:
        df = pd.DataFrame(scenario_summaries)
        df = df.set_index("scenario")
```

```
        out_path = os.path.join(ROOT_DIR, "all_scenarios_summary.csv")
        df.to_csv(out_path)
        print(f"\n[*] Global summary saved to {out_path}")
    else:
        print("[!] No scenarios summarized.")

if __name__ == "__main__":
    main()
```

Bảng 26. Hàm main() và thực thi chính

- Điều khiển toàn bộ quá trình phân tích.
- Khởi tạo danh sách scenario_summaries = [].
- Duyệt tất cả thư mục con trong ROOT_DIR (experiments). Nếu là thư mục hợp lệ:

  - Gọi get_flow_count() để xác định số luồng.
  - In tiêu đề scenario.
  - Gọi summarize_scenario() để tạo file summary.csv cho từng scenario.
  - Lưu kết quả trung bình vào scenario_summaries.

- Tạo DataFrame chứa các dòng tóm tắt của từng scenario.
- Đặt cột scenario làm chỉ mục.
- Xuất ra file tổng all_scenarios_summary.csv trong thư mục gốc.
- Nếu không có scenario nào được xử lý thì in cảnh báo.
- Khi chạy file trực tiếp (python3 script.py), chương trình sẽ bắt đầu từ hàm main().
- Nếu được import dưới dạng module, code trong main() sẽ **không tự động chạy.**

i. Kết quả:

- File code này sẽ xuất ra các file summary.csv thông kê dữ liệu từ iperf3 và ifstat của từng kịch bản, file all_scenarios_summary.csv tổng hợp dữ liệu từ tất cả kịch bản để phục vụ cho việc so sánh các kịch bản.

3. pcap_summary.py:

File code python này dùng để chuyển dữ liệu từ các file .pcap của các kịch bản dưới dạng file CSV, thuận tiện cho việc phân tích dữ liệu.

a. Phần khởi tạo và cấu hình:

```
#!/usr/bin/env python3
import os
import re
import pandas as pd
import subprocess

CLIENT_IP = "192.168.56.104"
SERVER_IP = "192.168.214.104"
```

Bảng 27. Phần khởi tạo và cấu hình

- #!/usr/bin/env python3: Cho phép chạy file bằng Python 3 trên nhiều hệ điều hành.
- import os, pandas as pd, subprocess: Import các thư viện cần thiết:
  - os: làm việc với file và thư mục.
  - pandas: xử lý dữ liệu dạng bảng và xuất CSV.
  - subprocess: chạy lệnh hệ thống (như tshark).
- CLIENT_IP và SERVER_IP: Địa chỉ IP của client và server dùng để lọc gói tin trong file PCAP.

b. Hàm tshark_fields(pcap, fields, display_filter):

```
def tshark_fields(pcap, fields, display_filter):
    cmd = ["tshark", "-r", pcap, "-Y", display_filter, "-Tfields"]
    for f in fields:
        cmd += ["-e", f]
    cmd += ["-E", "separator=\t"]
    try:
```

```
        out = subprocess.run(cmd, capture_output=True, text=True, check=True)
        lines = [l.split("\t") for l in out.stdout.strip().split("\n") if l.strip()]
        if not lines:
            return pd.DataFrame()
        df = pd.DataFrame(lines, columns=fields)
        for f in fields:
            df[f] = pd.to_numeric(df[f], errors="coerce")
        return df.dropna()
    except subprocess.CalledProcessError:
        return pd.DataFrame()
```

Bảng 28. Hàm tshark_fields(pcap, fields, display_filter)

- Chạy lệnh tshark để trích xuất các trường dữ liệu cụ thể từ file .pcap và trả về dưới dạng DataFrame.
- cmd: tạo danh sách tham số cho lệnh tshark, gồm đường dẫn file, bộ lọc hiển thị, và các trường cần trích xuất.
- subprocess.run(...): thực thi lệnh tshark và thu đầu ra.
- lines = [l.split("\t") ...]: tách từng dòng kết quả thành danh sách giá trị bằng dấu tab.
- df = pd.DataFrame(...): chuyển dữ liệu thành DataFrame với tên cột là các trường được chỉ định.
- df[f] = pd.to_numeric(...): chuyển đổi các giá trị thành dạng số để tiện xử lý.
- return df.dropna(): trả về DataFrame đã loại bỏ giá trị rỗng.
- except subprocess.CalledProcessError: nếu lệnh tshark thất bại, trả về DataFrame trống.

c. Hàm summarize_pcap_metrics(pcap_path):

```
def summarize_pcap_metrics(pcap_path):
    fname = os.path.basename(pcap_path)
    summary = {"pcap": fname}

    if not os.path.exists(pcap_path) or os.path.getsize(pcap_path) == 0:
```

```
    return None

# Determine role and tshark filter
if "client" in fname:
    tcp_filter = f"tcp and (ip.src=={CLIENT_IP} or ip.dst=={SERVER_IP})"
    role = "client"
elif "server" in fname:
    tcp_filter = f"tcp and (ip.src=={SERVER_IP} or ip.dst=={CLIENT_IP})"
    role = "server"
else:
    tcp_filter = "tcp"
    role = "bottleneck"

# ==============================
# Metrics by node type
# ==============================

# ---- CLIENT: sender pacing + ACK timing ----
if role == "client":
    df_time = tshark_fields(pcap_path, ["frame.time_relative"], tcp_filter)
    if not df_time.empty and len(df_time) > 1:
        diffs = df_time["frame.time_relative"].diff().dropna() * 1000
        summary["gap_avg_ms"] = diffs.mean()
    else:
        summary["gap_avg_ms"] = 0

    ack_filter = tcp_filter + " and tcp.flags.ack==1"
    df_ack = tshark_fields(pcap_path, ["frame.time_relative"], ack_filter)
    if not df_ack.empty and len(df_ack) > 1:
```

```
        adiffs = df_ack["frame.time_relative"].diff().dropna() * 1000
        summary["ack_interval_avg_ms"] = adiffs.mean()
    else:
        summary["ack_interval_avg_ms"] = 0

    summary["rtt_avg_ms"] = 0
    summary["rtt_std_ms"] = 0
    summary["cwnd_avg_kB"] = 0

# ---- BOTTLENECK: RTT + queue delay + cwnd proxy ----
elif role == "bottleneck":
    df_tcp = tshark_fields(pcap_path,
                           ["tcp.analysis.ack_rtt", "tcp.analysis.bytes_in_flight"],
                           tcp_filter)
    if not df_tcp.empty:
        summary["rtt_avg_ms"] = df_tcp["tcp.analysis.ack_rtt"].mean() * 1000
        summary["rtt_std_ms"] = df_tcp["tcp.analysis.ack_rtt"].std() * 1000
          summary["cwnd_avg_kB"] = df_tcp["tcp.analysis.bytes_in_flight"].mean() /
1024
    else:
        summary["rtt_avg_ms"] = 0
        summary["rtt_std_ms"] = 0
        summary["cwnd_avg_kB"] = 0

    df_time = tshark_fields(pcap_path, ["frame.time_relative"], tcp_filter)
    if not df_time.empty and len(df_time) > 1:
        diffs = df_time["frame.time_relative"].diff().dropna() * 1000
        summary["gap_avg_ms"] = diffs.mean()
    else:
```

```
        summary["gap_avg_ms"] = 0

    ack_filter = tcp_filter + " and tcp.flags.ack==1"
    df_ack = tshark_fields(pcap_path, ["frame.time_relative"], ack_filter)
    if not df_ack.empty and len(df_ack) > 1:
        adiffs = df_ack["frame.time_relative"].diff().dropna() * 1000
        summary["ack_interval_avg_ms"] = adiffs.mean()
    else:
        summary["ack_interval_avg_ms"] = 0

# ---- SERVER: ACK response behavior ----
elif role == "server":
    ack_filter = tcp_filter + " and tcp.flags.ack==1"
    df_ack = tshark_fields(pcap_path, ["frame.time_relative"], ack_filter)
    if not df_ack.empty and len(df_ack) > 1:
        adiffs = df_ack["frame.time_relative"].diff().dropna() * 1000
        summary["ack_interval_avg_ms"] = adiffs.mean()
    else:
        summary["ack_interval_avg_ms"] = 0

    summary["rtt_avg_ms"] = 0
    summary["rtt_std_ms"] = 0
    summary["cwnd_avg_kB"] = 0
    summary["gap_avg_ms"] = 0

return summary
```

Bảng 29. Hàm summarize_pcap_metrics(pcap_path)

- Trích xuất các trường dữ liệu số (như RTT, bytes_in_flight, thời gian khung hình) từ file PCAP theo theo vai trò của node (client, server, bottleneck).
- fname = os.path.basename(pcap_path): lấy tên file pcap.
- Kiểm tra file tồn tại và có dung lượng > 0, nếu không trả về None.
- Xác định role (vai trò) dựa vào tên file — client, server, hoặc bottleneck — và tạo bộ lọc tshark phù hợp.
- Với client:

- Tính khoảng thời gian giữa các gói tin (gap_avg_ms) để đo pacing.
- Tính khoảng thời gian giữa các ACK (ack_interval_avg_ms) để đo tốc độ phản hồi ACK.
- Đặt RTT và cwnd bằng 0 vì client không đo trực tiếp.

- Với bottleneck:

- Trích xuất RTT và bytes_in_flight để tính RTT trung bình, độ lệch chuẩn và cwnd trung bình.
- Tính thêm gap_avg_ms và ack_interval_avg_ms tương tự client.

- Với server:

- Chỉ tính khoảng thời gian ACK (ack_interval_avg_ms).
- Đặt RTT, cwnd, gap bằng 0 vì server chủ yếu phản hồi.

- Trả về dictionary chứa tất cả các chỉ số được đo.

d. Hàm parse_ss_file(ss_path):

```
def parse_ss_file(ss_path):
    if not os.path.exists(ss_path):
        return 0, 0
    rtts, cwnds = [], []
    with open(ss_path) as f:
        for line in f:
```

```
        if "rtt" in line and "cwnd" in line:
            m_rtt = re.search(r"rtt[:=](\d+\.\d+)", line)
            m_cwnd = re.search(r"cwnd[:=](\d+)", line)
            if m_rtt:
                rtts.append(float(m_rtt.group(1)))
            if m_cwnd:
                cwnds.append(int(m_cwnd.group(1)))
    avg_rtt = sum(rtts) / len(rtts) if rtts else 0
    avg_cwnd = sum(cwnds) / len(cwnds) / 1024 if cwnds else 0  # KB
    return avg_rtt, avg_cwnd
```

Bảng 30. Hàm parse_ss_file(ss_path)

- Đọc file ss_client.txt và trích xuất giá trị trung bình của RTT và CWND từ lệnh ss.
- Kiểm tra file tồn tại, nếu không có trả về 0, 0.
- Đọc từng dòng và dùng regex để tìm giá trị rtt và cwnd.
- Lưu các giá trị tìm thấy vào danh sách rtts và cwnds.
- Tính trung bình RTT (ms) và CWND (KB) từ danh sách.
- Trả về hai giá trị trung bình (avg_rtt, avg_cwnd).

e. Hàm normalize_run_name(run_folder):

```
def normalize_run_name(run_folder):
    match = re.search(r"run[_-]?(\d+)", run_folder)
    return f"run_{match.group(1)}" if match else run_folder
```

Bảng 31. Hàm normalize_run_name(run_folder)

- Chuẩn hóa tên thư mục "run".
- Dùng regex tìm số trong tên thư mục (ví dụ: run_* → run1).
- Nếu khớp thì trả về định dạng run_*. Nếu không khớp thì trả lại tên gốc.

f. Hàm process_all_runs(root="experiments") và phần thực thi:

```
def process_all_runs(root="experiments"):
    all_summaries = []

    for scenario in sorted(os.listdir(root)):
        scenario_path = os.path.join(root, scenario)
        if not os.path.isdir(scenario_path):
            continue

        scenario_summaries = []
        for run in sorted(os.listdir(scenario_path)):
            run_path = os.path.join(scenario_path, run)
            if not os.path.isdir(run_path):
                continue
            run_id = normalize_run_name(run)

            ss_path = os.path.join(run_path, "ss_client.txt")
            ss_rtt, ss_cwnd = parse_ss_file(ss_path)

            for pcap_name in ["client_iperf3.pcap", "server.pcap", "bottleneck.pcap"]:
                pcap_path = os.path.join(run_path, pcap_name)
                summary = summarize_pcap_metrics(pcap_path)
                if summary:
                    summary["run"] = run_id
                    summary["scenario"] = scenario
                    summary["ss_avg_rtt_ms"] = ss_rtt
                    summary["ss_avg_cwnd"] = ss_cwnd
                    scenario_summaries.append(summary)
                    all_summaries.append(summary)
```

```
    # Write per-scenario summary
    if scenario_summaries:
        df_summary = pd.DataFrame(scenario_summaries).fillna(0)
        df_summary = df_summary[[
            "pcap", "run", "rtt_avg_ms", "rtt_std_ms", "cwnd_avg_kB",
            "gap_avg_ms", "ack_interval_avg_ms", "ss_avg_rtt_ms", "ss_avg_cwnd"
        ]]
        out_csv = os.path.join(scenario_path, "pcap_summary.csv")
        df_summary.to_csv(out_csv, index=False)
        print(f"[+] Wrote {out_csv}")


#
# ============================================================
# Global summary (averaged, cleaned version)
#
# ============================================================
if all_summaries:
    df_all = pd.DataFrame(all_summaries).fillna(0)
    df_avg = df_all.groupby(["scenario"]).agg({
        "gap_avg_ms": "mean",            # pacing behavior
        "ack_interval_avg_ms": "mean",   # ACK behavior
        "ss_avg_rtt_ms": "mean",         # real RTT seen by TCP
        "ss_avg_cwnd": "mean"            # real congestion window
    }).reset_index()

    out_csv = os.path.join(root, "all_scenarios_pcap.csv")
    df_avg.to_csv(out_csv, index=False)
    print(f"[+] Wrote aggregated averages to {out_csv}")
```

```
if __name__ == "__main__":
    process_all_runs()
```

Bảng 32. Hàm process_all_runs(root="experiments") và phần thực thi

- Quét toàn bộ các scenario và run để xử lý tất cả file .pcap và tạo bảng tổng hợp.
- all_summaries = []: danh sách để lưu toàn bộ kết quả từ mọi scenario.
- Lặp qua từng scenario trong thư mục gốc:

  - Bỏ qua nếu không phải thư mục.
  - Với mỗi run trong scenario, xác định đường dẫn và chuẩn hóa tên run.
  - Đọc file ss_client.txt để lấy giá trị RTT và CWND trung bình.
  - Với mỗi file pcap (client_iperf3.pcap, server.pcap, bottleneck.pcap), gọi summarize_pcap_metrics() để trích xuất số liệu.
  - Thêm thông tin run, scenario, ss_rtt, ss_cwnd vào kết quả và lưu lại.

- Sau khi xử lý xong mỗi scenario: Tạo DataFrame từ danh sách summary và lưu thành file pcap_summary.csv trong thư mục scenario.
- Sau khi hoàn thành toàn bộ:

  - Tạo DataFrame tổng hợp df_all từ tất cả summary.
  - Tính trung bình theo từng scenario cho các chỉ số chính (gap_avg_ms, ack_interval_avg_ms, ss_avg_rtt_ms, ss_avg_cwnd).
  - Ghi kết quả ra file all_scenarios_pcap.csv trong thư mục gốc.

- if name == "main": kiểm tra nếu script được chạy trực tiếp (không phải import).
- Gọi hàm process_all_runs() để thực hiện toàn bộ quy trình phân tích và tổng hợp dữ liệu.

g. Kết quả:

- File code python này sẽ xuất ra file pcap_summary.csv cho từng kịch bản, 1 file all_scenarios_pcap.csv tổng hợp dữ liệu từ các kịch bản để phục vụ cho việc so sánh.

3. plot_result.py:

File code python này dùng để vẽ biểu đồ từ file all_scenarios_summary.csv và all_scenarios_pcap.csv để phục vụ việc so sánh các kịch bản với nhau trực quan hơn.

a. Phần khởi tạo và thiết lập biểu đồ:

```
#!/usr/bin/env python3
import os
import pandas as pd
import matplotlib.pyplot as plt
import re


plt.style.use("seaborn-v0_8-whitegrid")
```

Bảng 33. Phần khởi tạo và thiết lập biểu đồ

- #!/usr/bin/env python3: Cho phép script chạy bằng Python 3 trên các hệ thống khác nhau.
- import os, pandas as pd, matplotlib.pyplot as plt, re:

- os: thao tác với file và thư mục.
- pandas: đọc và xử lý dữ liệu CSV.
- matplotlib.pyplot: vẽ biểu đồ.
- re: xử lý biểu thức chính quy (regex).

- plt.style.use("seaborn-v0_8-whitegrid"): áp dụng style “seaborn” cho biểu đồ — giúp đồ thị có nền sáng và dễ nhìn.

b. Hàm safe_read_csv(path):

```
def safe_read_csv(path):
    if os.path.exists(path):
        print(f"[+] Loaded {path}")
        return pd.read_csv(path)
    else:
        print(f"[!] Missing {path}")
```

```
    return pd.DataFrame()
```

Bảng 34. Hàm safe_read_csv(path)

- Đọc file CSV an toàn, tránh lỗi khi file không tồn tại. Kiểm tra file có tồn tại không bằng os.path.exists(path).
- Nếu có thì in thông báo [+] Loaded <path> và đọc file CSV bằng pd.read_csv(path).
- Nếu không có thì in cảnh báo [!] Missing <path> và trả về DataFrame rỗng (pd.DataFrame()).
- Giúp chương trình không bị dừng nếu thiếu dữ liệu.

c. Hàm shorten_name(name):

```
def shorten_name(name: str):
    """
    Convert scenario names into readable short names.
    Examples:
      bw3_1_cubic_fq          -> bw3 1 cubic
      bw3_10_impairments_bbr_fq -> bw3 10 imp bbr
      bwNormal_2_bbr_fq       -> norm 2 bbr
    """
    name = name.replace("bwNormal_", "norm ").replace("bw3_", "bw3 ")
    name = name.replace("_impairments", " imp").replace("_fq", "")
    name = name.replace("_", " ")
    name = re.sub(r"\s+", " ", name).strip()
    return name
```

Bảng 35. Hàm shorten_name(name)

- Rút gọn và làm đẹp tên scenario để hiển thị trong biểu đồ.
- Thay thế các phần trong tên bằng phiên bản ngắn hơn, ví dụ:

- "bwNormal_" → "norm "

- "bw3_" → "bw3 "
- "_impairments" → " imp"
- "_fq" → xóa bỏ.

- Thay các dấu _ bằng khoảng trắng (_ → " ").

- Dùng re.sub(r"\s+", " ", name) để gộp nhiều khoảng trắng liên tiếp thành một.

- Trả về tên ngắn gọn, dễ đọc (ví dụ: bw3_10_impairments_bbr_fq → bw3 10 imp bbr).

d. Hàm plot_pcap_summary(pcap_csv):

```
def plot_pcap_summary(pcap_csv):
    df = safe_read_csv(pcap_csv)
    if df.empty:
        print("[!] No data for pcap metrics.")
        return

    if "protocol" in df.columns:
        df["short_name"] = df["scenario"].apply(shorten_name) + " (" + df["protocol"] +
")"
    else:
        df["short_name"] = df["scenario"].apply(shorten_name)

            metrics = ["gap_avg_ms", "ack_interval_avg_ms", "ss_avg_rtt_ms",
"ss_avg_cwnd"]
    labels = [
        "Gap avg (ms)",
        "ACK interval avg (ms)",
        "ss Avg RTT (ms)",
        "ss Avg CWND"
```

```
    ]


    fig, axes = plt.subplots(len(metrics), 1, figsize=(12, 8))
    for i, (metric, label) in enumerate(zip(metrics, labels)):
        ax = axes[i]
        if metric not in df.columns:
            print(f"[!] Missing column: {metric}")
            continue
        df_sorted = df.sort_values(metric, ascending=False)
        ax.barh(df_sorted["short_name"], df_sorted[metric], color="steelblue")
        ax.set_xlabel(label, fontsize=11)
        ax.set_ylabel("Scenario", fontsize=11)
        ax.set_title(f"{label} by Scenario", fontsize=13, weight="bold")
        ax.grid(True, linestyle="--", alpha=0.5)
    plt.tight_layout()
    plt.savefig("pcap_plot.png", dpi=250, bbox_inches="tight")
    print("[+] Saved pcap_plot.png")
```

Bảng 36. Hàm plot_pcap_summary(pcap_csv)

- Vẽ biểu đồ tóm tắt các chỉ số từ file CSV chứa dữ liệu PCAP (RTT, CWND, GAP, ACK interval).
- df = safe_read_csv(pcap_csv): Đọc file CSV chứa dữ liệu PCAP vào DataFrame.
- if df.empty: Kiểm tra nếu file trống thì bỏ qua.
- if "protocol" in df.columns: Kiểm tra xem có cột “protocol” trong dữ liệu hay không để tạo tên rút gọn cho từng scenario.
- metrics: Danh sách các cột cần vẽ gồm gap_avg_ms, ack_interval_avg_ms, ss_avg_rtt_ms, ss_avg_cwnd.
- labels: Nhãn hiển thị cho từng chỉ số tương ứng với metrics.
- fig, axes = plt.subplots(...): Tạo nhiều biểu đồ con (mỗi metric một biểu đồ ngang).

- Vòng lặp for i, (metric, label) in enumerate(...):

- Sắp xếp dữ liệu giảm dần theo từng metric.
- Vẽ biểu đồ thanh ngang (barh) thể hiện giá trị của từng scenario.
- Đặt tiêu đề, nhãn trục, và lưới (grid) cho dễ đọc.

- plt.tight_layout() và plt.savefig(...): Căn chỉnh bố cục và lưu hình thành file pcap_plot.png.

e. Hàm plot_bw_summary(bw_csv):

```
def plot_bw_summary(bw_csv):
    df = safe_read_csv(bw_csv)
    if df.empty:
        print("[!] No data for bandwidth metrics.")
        return

    df["short_name"] = df["scenario"].apply(shorten_name)

    # Updated metrics (using avg BW instead of total BW)
    metrics = [
        ("tcp_avg_bw_Mbps", "TCP Average Bandwidth (Mbps)", "seagreen"),
        ("tcp_fairness", "TCP Fairness", "teal"),
        ("tcp_avg_retrans", "TCP Retransmissions", "royalblue"),
        ("udp_avg_bw_Mbps", "UDP Average Bandwidth (Mbps)", "darkorange"),
        ("udp_avg_jitter_ms", "UDP Avg Jitter (ms)", "darkred"),
        ("udp_avg_lost_pct", "UDP Avg Packet Loss (%)", "firebrick"),
    ]

    n = len(metrics)
    fig, axes = plt.subplots(nrows=n, ncols=1, figsize=(12, 2.6 * n))
    if n == 1:
```

```
    axes = [axes]

for i, (metric, label, color) in enumerate(metrics):
    ax = axes[i]
    if metric not in df.columns:
        print(f"[!] Missing column: {metric}")
        continue

    df_sorted = df.sort_values(metric, ascending=False)
    bars = ax.barh(df_sorted["short_name"], df_sorted[metric], color=color)

    # Log scale only for Bandwidth
    if "Bandwidth" in label:
        ax.set_xscale("log")
        ax.set_xlim(left=max(0.1, df_sorted[metric].min() * 0.5))
        ax.set_xlabel(f"{label} (log scale)", fontsize=11)
    else:
        ax.set_xlabel(label, fontsize=11)

    ax.set_ylabel("Scenario", fontsize=11)
    ax.set_title(label, fontsize=13, weight="bold")
    ax.grid(True, linestyle="--", alpha=0.5)

    # Add numeric labels
    for bar in bars:
        width = bar.get_width()
        ax.text(
            width * 1.02,
            bar.get_y() + bar.get_height() / 2,
```

```
            f"{width:.2f}",
            va="center",
            fontsize=8,
        )

plt.tight_layout()
plt.savefig("summary_plot.png", dpi=250, bbox_inches="tight")
print("[+] Saved summary_plot.png")
```

Bảng 37. Hàm plot_bw_summary(bw_csv)

- Vẽ biểu đồ tóm tắt các chỉ số hiệu năng TCP và UDP (bandwidth, fairness, retransmission, jitter, loss).
- df = safe_read_csv(bw_csv): Đọc dữ liệu từ file CSV chứa các thông số hiệu năng.
- if df.empty: Kiểm tra file trống.
- df["short_name"]: Rút gọn tên scenario để hiển thị ngắn gọn.
- metrics: Danh sách các cột cần vẽ (6 chỉ số: 3 cho TCP, 3 cho UDP) kèm tiêu đề và màu sắc.
- fig, axes = plt.subplots(...): Tạo biểu đồ con cho từng metric.
- Vòng lặp for i, (metric, label, color) in enumerate(metrics):

  - Kiểm tra cột có tồn tại trong dữ liệu không.
  - Sắp xếp dữ liệu giảm dần theo giá trị metric.
  - Vẽ biểu đồ thanh ngang (barh) với màu tương ứng.
  - Với các chỉ số chứa "Bandwidth", sử dụng thang log (ax.set_xscale("log")) để dễ so sánh.
  - Thêm tiêu đề, nhãn, lưới, và giá trị số bên cạnh mỗi thanh.

- plt.tight_layout() và plt.savefig(...): Căn chỉnh bố cục và lưu hình thành file summary_plot.png.

f. Phần thực thi chính:

```
if __name__ == "__main__":
```

```
    os.makedirs("experiments", exist_ok=True)
    plot_pcap_summary("experiments/all_scenarios_pcap.csv")
    plot_bw_summary("experiments/all_scenarios_summary.csv")ss
```

Bảng 38. Phần thực thi chính

- if __name__ == "__main__": đảm bảo đoạn code chỉ chạy khi file được thực thi trực tiếp.
- os.makedirs("experiments", exist_ok=True): tạo thư mục experiments nếu chưa có.
- Gọi plot_pcap_summary("experiments/all_scenarios_pcap.csv"): vẽ biểu đồ PCAP từ file tổng hợp.
- Gọi plot_bw_summary("experiments/all_scenarios_summary.csv"): vẽ biểu đồ hiệu năng TCP/UDP.

g. Kết quả:

- File code python này sẽ xuất ra file summary_plot.png và file pcap_plot.png là 2 biểu đồ trực quan tương ứng với dữ liệu của file all_scenarios_summary.csv và all_scenarios_pcap.csv.

3.6. Trình tự thực hiện các kịch bản:

Trước khi chạy mỗi kịch bản, cần thực hiện cấu hình phù hợp cho kịch bản đó:

- Kiểm tra và cấu hình congestion control trên client và server:

```
sysctl net.ipv4.tcp_congestion_control
sudo sysctl -w net.ipv4.tcp_congestion_control=cubic/bbr
```

Bảng 39. Lệnh kiểm tra và cấu hình congestion control

- Kiểm tra và cấu hình qdisc:

- Giới hạn băng thông 3 Mbit

```
sudo tc qdisc del dev <interface> root 2>/dev/null; #Xóa sạch sẽ qdisc trước đó
sudo tc qdisc add dev <interface> root handle 1: tbf rate 3mbit burst 32kbit latency 50ms;
sudo tc qdisc add dev <interface> parent 1: handle 10: fq_codel/pfifo/RED
tc qdisc show dev <interface> #Kiểm tra cấu hình qdisc hiện tại
```

Bảng 40. Lệnh giới hạn băng thông 3 Mbit

- Mô phỏng trễ + mất gói

```
sudo tc qdisc del dev <interface>  root 2>/dev/null;
sudo tc qdisc add dev <interface>  root handle 1: netem delay 100ms 20ms distribution normal loss 0.5%;
sudo tc qdisc add dev <interface>  parent 1: handle 2: fq_codel/pfifo/RED
tc qdisc show dev <interface>
```

Bảng 41. Lệnh mô phỏng trễ + mất gói

- Kết hợp đầy đủ: giới hạn băng thông + trễ + mất gói

```
sudo tc qdisc del dev <interface> root 2>/dev/null;
sudo tc qdisc add dev <interface> root handle 1: tbf rate 3mbit burst 32kbit latency 50ms;
sudo tc qdisc add dev <interface> parent 1: handle 2: netem delay 100ms 20ms distribution normal loss 0.5%;
sudo tc qdisc add dev <interface>s parent 2: handle 3: fq_codel/pfifo/RED
tc qdisc show dev <interface>
```

Bảng 42. Lệnh giới hạn băng thông + trễ + mất gói

- Thay đổi cấu hình trong file script tương ứng với kịch bản rồi chạy script mô phỏng kịch bản:

```
bash run_experiments.sh
```

Bảng 43. Lệnh chạy script mô phỏng

- Chạy summary.py:

```
python3 summary.py
```

Bảng 44. Lệnh chạy summary.py

- Chạy pcap_summary.py:

```
python3 pcap_summary.py
```

Bảng 45. Lệnh chạy pcap_summary.py

- Chạy plot_result.py:

```
python3 plot_results.py
```

Bảng 46. Lệnh chạy plot_result.py

3.7. Phân tích dữ liệu các kịch bản:

1. Tổng quan về các nguồn dữ liệu:

Nguồn dữ liệu	Mục tiêu đo	Mức độ	Loại dữ liệu chính	Vai trò
iperf3 (client + server)	Đo throughput, fairness, retransmission, jitter	Ứng dụng (Application Layer)	TCP/UDP bandwidth, retrans, jitter, lost	Xác định hiệu năng tổng thể (goodput, stability)
ifstat (client/server/bottleneck)	Giám sát traffic thực tế trên interface	Link Layer (Data Link)	Tx/Rx rate trên NIC	Kiểm chứng tốc độ gửi/thực đạt so với iperf3
pcap (client/server/bottleneck)	Đo RTT, cwnd, delay, ACK timing	Network Layer (Packet level)	RTT per packet, inter-arrival, ack interval	Phân tích chi tiết flow control, delay, queue behavior
ss (ss_client.txt)	Snapshot TCP socket stats (kernel view)	Transport Layer (Kernel TCP stack)	cwnd, RTT, retrans, rtt_var	Đo thông tin chính xác nhất về congestion window và RTT trung bình thực tế từ TCP kernel

Bảng 47. Tổng quan về các nguồn dữ liệu

2. Tổng quan về metric:

Metric	Nguồn	Giao thức	Ý nghĩa	Dùng để phân tích
tcp_avg_bw_Mbps	iperf3	TCP	Băng thông trung bình đạt được (Goodput)	Hiệu quả truyền TCP với qdisc/CC hiện tại
tcp_fairness	iperf3	TCP	Mức độ chia sẻ công bằng giữa các flow (Jain Index)	Đánh giá khả năng fair share của qdisc/CC
tcp_avg_retrans	iperf3	TCP	Tổng số packet retransmitted	Mức độ tắc nghẽn và ổn định CC
udp_avg_bw_Mbps	iperf3	UDP	Throughput UDP trung bình	Đánh giá mức giới hạn băng thông và fairness với TCP
udp_avg_lost_pct	iperf3	UDP	% packet bị mất	Độ ổn định mạng / buffer overflows
udp_avg_jitter_ms	iperf3	UDP	Dao động delay	Độ ổn định đường truyền (quan trọng với VoIP/stream)
rtt_avg_ms	pcap	TCP	RTT trung bình giữa gói TCP	Quan sát độ trễ mạng thực tế (chỉ nên xem

				tương đối)
rtt_std_ms	pcap	TCP	Độ lệch chuẩn RTT	Mức biến động delay (jitter TCP)
cwnd_avg_kB	pcap	TCP	Congestion window trung bình (ước lượng từ trace)	Chỉ để tham khảo (ít chính xác)
gap_avg_ms	pcap	TCP	Khoảng cách giữa các gói gửi (inter-packet gap)	Đánh giá pacing behavior của CC (BBR có gap ổn định hơn CUBIC)
ack_interval_avg_ms	pcap	TCP	Khoảng giữa các ACK	Quan sát feedback từ receiver
ss_avg_rtt_ms	ss	TCP	RTT trung bình kernel thấy	Giá trị *chính xác* về delay tại TCP layer
ss_avg_cwnd	ss	TCP	cwnd trung bình (kernel-level)	Dùng phân tích congestion control trực tiếp
Tx/Rx rate	ifstat	TCP/UDP	Throughput thực tế tại interface	Kiểm chứng tổng traffic thực tế

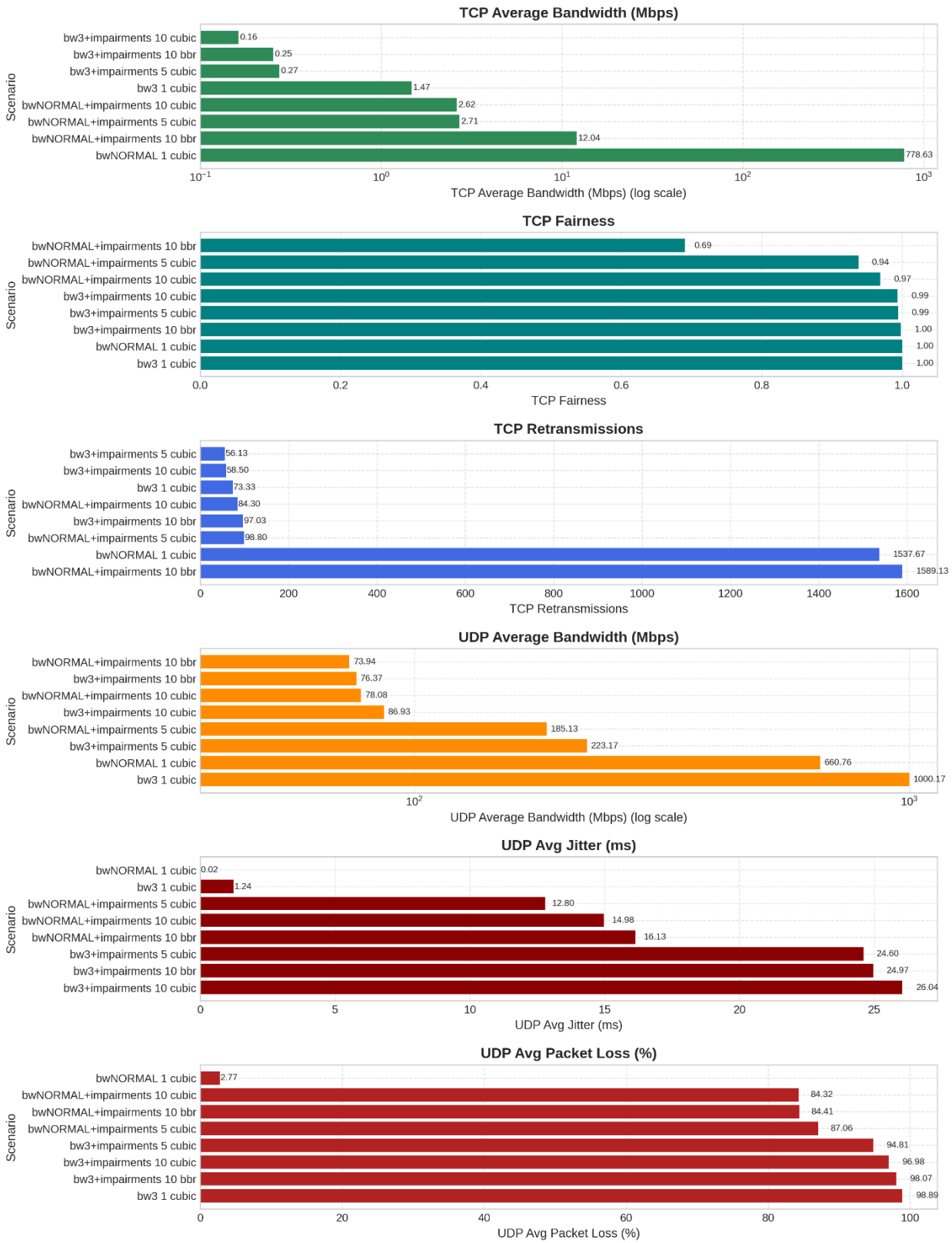
Bảng 48. Tổng quan về metric

3. So sánh tổng quan các kịch bản sử dụng fq_codel:

Các kịch bản chạy trên qdisc fq_codel:

scenario	tcp_avg_bw_Mbps	tcp_fairness	tcp_avg_retrans	udp_avg_bw_Mbps	udp_avg_lost_pct	udp_avg_jitter_ms	client_bw	bottleneck_bw	server_bw
bw3+impairments_10_bbr_fq	0.253349132	0.998054965	97.03333333	76.37269377	98.0686952	24.96541599	299.2028227	2.026951451	0.047033496
bw3+impairments_10_cubic_fq	0.162750919	0.993157833	58.5	86.93199731	96.98459242	26.03912118	313.8711041	1.831757135	0.024470894
bw3+impairments_5_cubic_fq	0.274126599	0.994021045	56.13333333	223.1721878	94.80635289	24.60249997	447.3988919	1.806971429	0.018448667
bw3_1_cubic_fq	1.469430497	1	73.33333333	1000.167019	98.88869249	1.240174132	479.7875948	2.224502626	0.018978333
bwNORMAL+impairments_10_bbr_fq	12.03688383	0.690466334	1589.133333	73.94294006	84.40871746	16.1336114	339.7720312	114.9997451	0.814135935
bwNORMAL+impairments_10_cubic_fq	2.6180427	0.968985379	84.3	78.07675179	84.32403153	14.97836531	314.2322329	53.75599107	0.262465691
bwNORMAL+impairments_5_cubic_fq	2.714619403	0.937968751	98.8	185.128836	87.05760841	12.7966077	377.5249412	47.44040317	0.134769138
bwNORMAL_1_cubic_fq	778.6256758	1	1537.666667	660.756817	2.770199341	0.019720088	702.064874	864.2728501	1.549850833

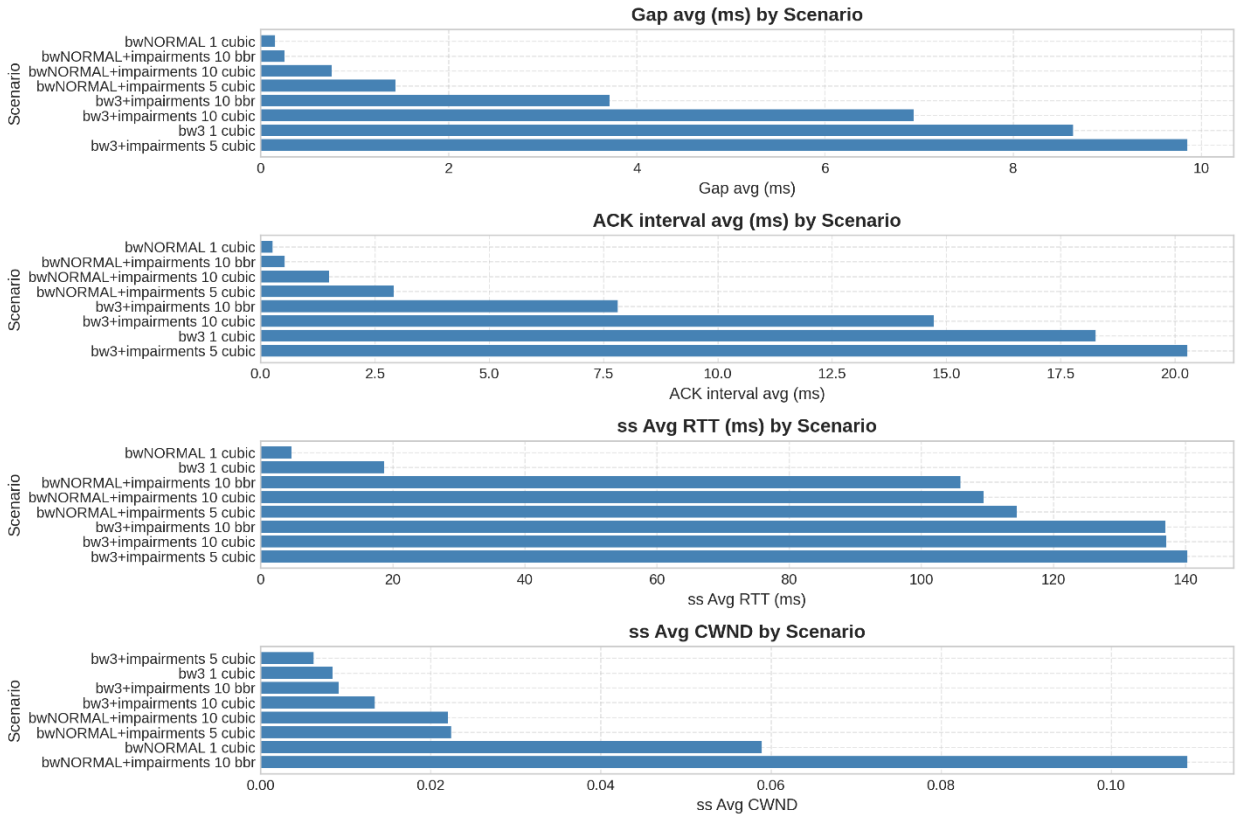
Hình 5. Tổng quan dữ liệu iperf3 và ifstat của các kịch bản chạy trên qdisc fq_codel



Hình 6. Biểu đồ tổng quan dữ liệu iperf3 và ifstat của các kịch bản chạy trên qdisc fq_codel

scenario	gap_avg_ms	ack_interval_avg_ms	ss_avg_rtt_ms	ss_avg_cwnd
bw3+impairments_10_bbr_fq	3.710368476	7.810756991	136.9691736	0.009199952
bw3+impairments_10_cubic_fq	6.940295613	14.72766	137.0544494	0.013439134
bw3+impairments_5_cubic_fq	9.852466614	20.27215638	140.2764732	0.006248265
bw3_1_cubic_fq	8.638786785	18.26846847	18.66754074	0.008456308
bwNORMAL+impairments_10_bbr_fq	0.253482932	0.531180957	105.909754	0.108953705
bwNORMAL+impairments_10_cubic_fq	0.755142676	1.503916911	109.3943313	0.022050952
bwNORMAL+impairments_5_cubic_fq	1.434507444	2.916344364	114.4309881	0.022401903
bwNORMAL_1_cubic_fq	0.156294388	0.261630894	4.685044444	0.058940972

Hình 7. Tổng quan dữ liệu pcap và ss của các kịch bản chạy trên qdisc fq_codel



Hình 8 Biểu đồ tổng quan dữ liệu pcap và ss của các kịch bản chạy trên qdisc fq_codel

Kết quả thực nghiệm cho thấy cả hai thuật toán điều khiển tắc nghẽn CUBIC và BBR khi kết hợp với fq_codel đều đạt được khả năng kiểm soát độ trễ tốt và duy trì truyền tải ổn định, tuy nhiên hiệu năng và hành vi của chúng khác biệt rõ rệt tùy theo điều kiện mạng.

Trong các kịch bản băng thông giới hạn (bw3), cả hai thuật toán đều bị giới hạn nghiêm trọng về throughput (chỉ khoảng 0.16–0.27 Mbps đối với CUBIC và BBR khi có 10 luồng), phản

ánh sự chi phối hoàn toàn của giới hạn đường truyền. Mặc dù vậy, BBR thường ghi nhận số lượng retransmission cao hơn CUBIC, cho thấy cơ chế dự đoán băng thông của BBR có thể dẫn đến việc gửi vượt quá năng lực đường truyền trong môi trường hẹp. CUBIC tỏ ra ổn định hơn trong điều kiện này, giữ được mức công bằng cao và độ trễ ổn định nhờ fq_codel, dù phải đánh đổi throughput thấp. Do đó, CUBIC phù hợp hơn với môi trường có băng thông thấp, nhiều luồng, và yêu cầu công bằng ổn định giữa các kết nối.

Ngược lại, trong các kịch bản băng thông bình thường có impairments (bwNORMAL), BBR thể hiện ưu thế rõ rệt về throughput, đạt trung bình khoảng 12 Mbps, vượt xa CUBIC (~2–3 Mbps). Tuy nhiên, kết quả cũng cho thấy fairness của BBR thấp hơn đáng kể (≈0.69) và số lượng retransmission rất lớn (~1589 gói/run), cho thấy BBR có xu hướng chiếm ưu thế tài nguyên khi cạnh tranh với các luồng khác. Trong cùng điều kiện, CUBIC đạt công bằng tốt hơn (≈0.95) nhưng không tận dụng hết băng thông khả dụng. fq_codel vẫn đảm bảo RTT thấp cho cả hai, song không thể loại bỏ hoàn toàn hiện tượng mất gói UDP khi đường truyền bị quá tải.

Tổng thể, có thể kết luận rằng:

- CUBIC + fq_codel thích hợp cho mạng có băng thông hạn chế hoặc nhiều luồng cạnh tranh, nơi tính công bằng và ổn định là ưu tiên hàng đầu.
- BBR + fq_codel phù hợp hơn với mạng có băng thông trung bình đến cao, độ trễ thấp, nơi throughput là mục tiêu chính, dù có thể đánh đổi công bằng và mức retransmission cao hơn.

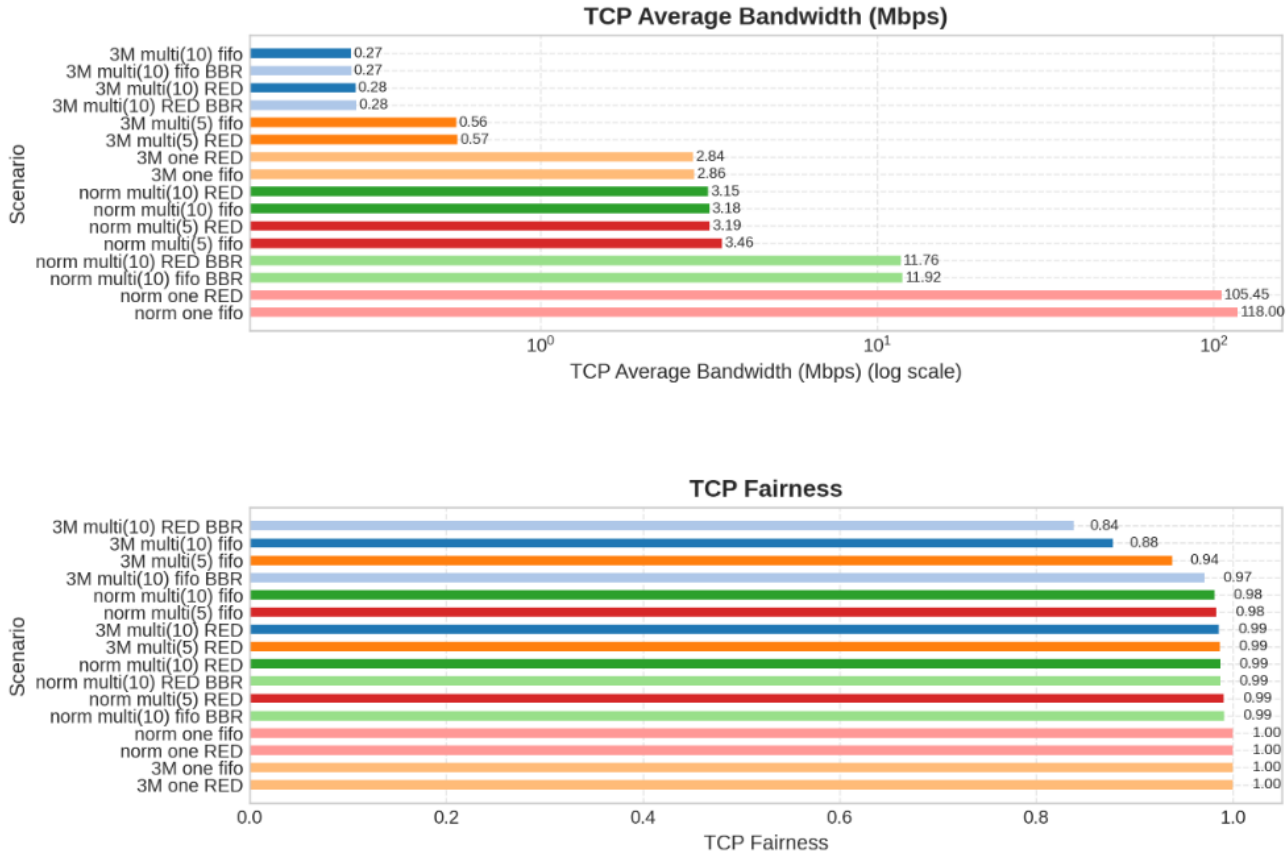
Như vậy, việc lựa chọn thuật toán điều khiển tắc nghẽn cần dựa trên đặc tính mạng và mục tiêu truyền tải: CUBIC cho môi trường chia sẻ công bằng và ổn định, còn BBR cho môi trường ưu tiên hiệu suất và tốc độ truyền tải.

4. So sánh tổng quan các kịch bản sử dụng pfifo/RED:

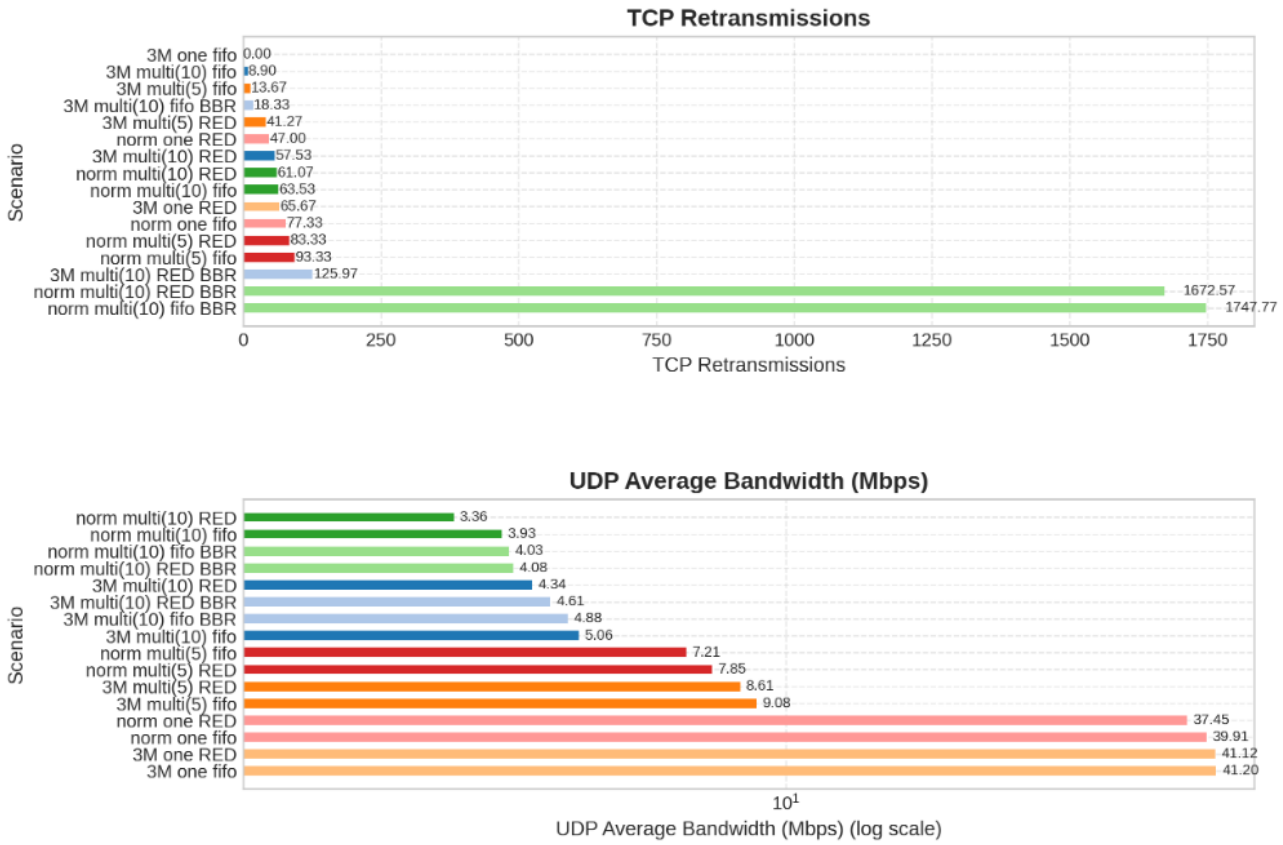
Các kịch bản chạy trên qdisc pfifo/RED:

scenario	tcp_avg_bw	tcp_fairness	tcp_avg_retra	udp_avg_bw	udp_avg_lost	udp_avg_ji	client_bw	bottleneck_b	server_bw
bw3Mbps_multiflow_10_RED	0.2825852	0.98578164	57.53333333	4.34086974	93.04838095	3.9193863	18.929087	1.932593654	0.0480841
bw3Mbps_multiflow_10_RED_BBR	0.28409607	0.83814863	125.9666667	4.60817109	93.59233842	9.9016722	20.933608	1.986838209	0.0562585
bw3Mbps_multiflow_10_pfifo	0.27377053	0.87780749	8.9	5.06203495	87.99915092	63.416444	20.325291	2.061864299	0.0585885
bw3Mbps_multiflow_10_pfifo_BBR	0.27416978	0.97095858	18.33333333	4.88387754	84.66084661	3.286152	20.425599	1.894227626	0.0388121
bw3Mbps_multiflow_RED	0.56769355	0.98680649	41.26666667	8.60550108	93.03504836	3.6926923	21.251694	2.318006709	0.0504184
bw3Mbps_multiflow_pfifo	0.56200218	0.93806273	13.66666667	9.08258397	91.68325024	3.3295806	19.201111	2.212779339	0.0438418
bw3Mbps_oneflow_RED	2.841842	1	65.66666667	41.1196503	92.98970295	3.5736418	19.173117	2.615035056	0.041725
bw3Mbps_oneflow_pfifo	2.86278588	1	0	41.2012582	30.16729485	0.088178	17.716203	1.765987689	0.0332751
bwNORMAL_multiflow_10_RED	3.15311803	0.98712099	61.06666667	3.35723068	0.973461917	3.7968489	18.867707	22.04851752	0.5040832
bwNORMAL_multiflow_10_RED_BBR	11.7623897	0.98730276	1672.566667	4.08164371	0.967834276	3.552947	75.871679	57.85728372	1.7334407
bwNORMAL_multiflow_10_pfifo	3.18049098	0.98130226	63.53333333	3.92857803	1.338314068	3.4523993	30.812708	24.15068016	0.5164664
bwNORMAL_multiflow_10_pfifo_BBR	11.9152559	0.990904	1747.766667	4.02575964	1.021768938	4.2937411	74.912174	56.96519272	1.7237621
bwNORMAL_multiflow_RED	3.1868836	0.99079458	83.33333333	7.84904841	1.060617853	3.398499	19.109409	17.36260437	0.2815808
bwNORMAL_multiflow_pfifo	3.46448891	0.98319701	93.33333333	7.21111687	1.025349662	3.6131941	18.679425	17.01913373	0.3085005
bwNORMAL_oneflow_RED	105.45265	1	47	37.454955	0	0.3931746	64.362635	52.03317919	0.1895748
bwNORMAL_oneflow_pfifo	117.997032	1	77.33333333	39.9119331	0	0.3679128	77.234157	72.56665167	0.2510242

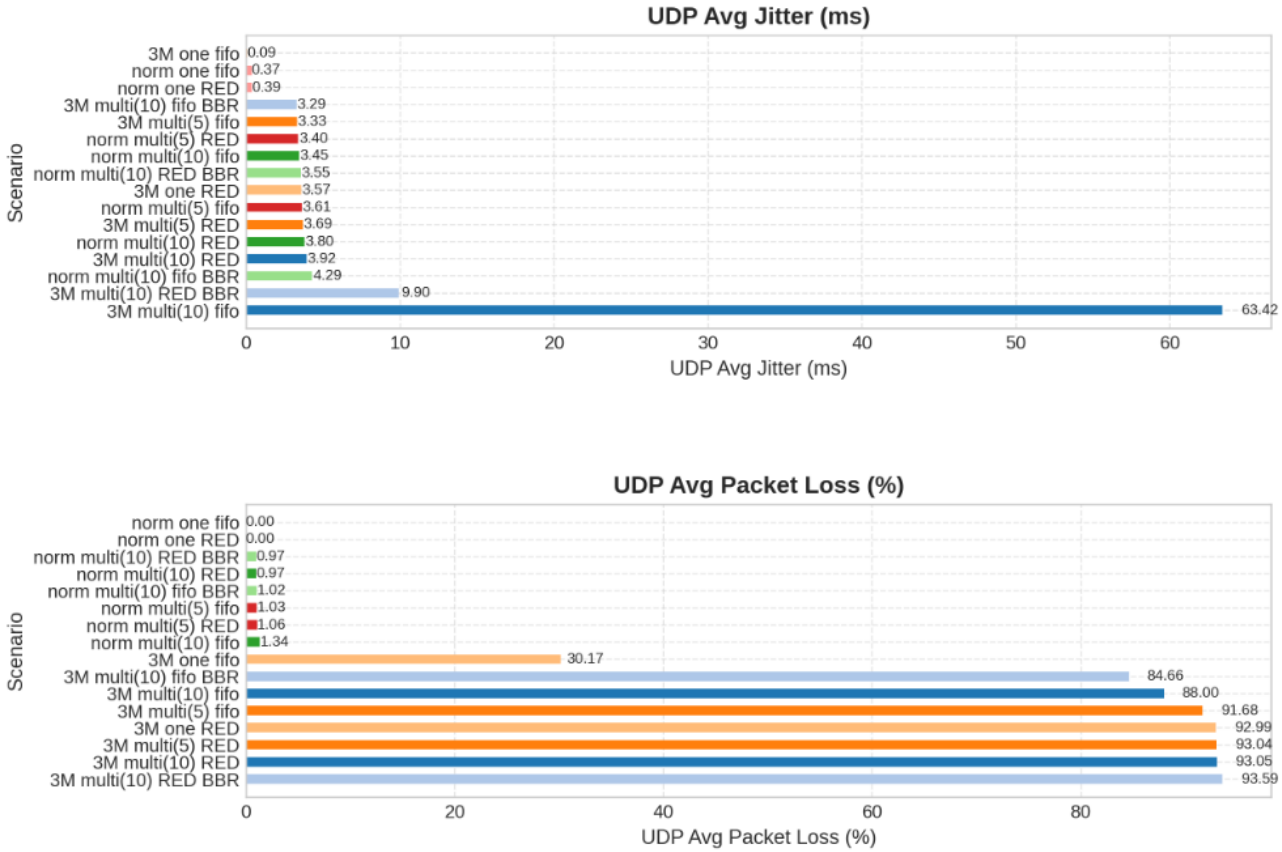
Hình 9. Tổng quan dữ liệu iperf3 và ifstat của các kịch bản chạy trên qdisc pfifo và RED



Hình 10. TCP Average Bandwidth và TCP Fairness trên 2 qdisc pfifo và RED



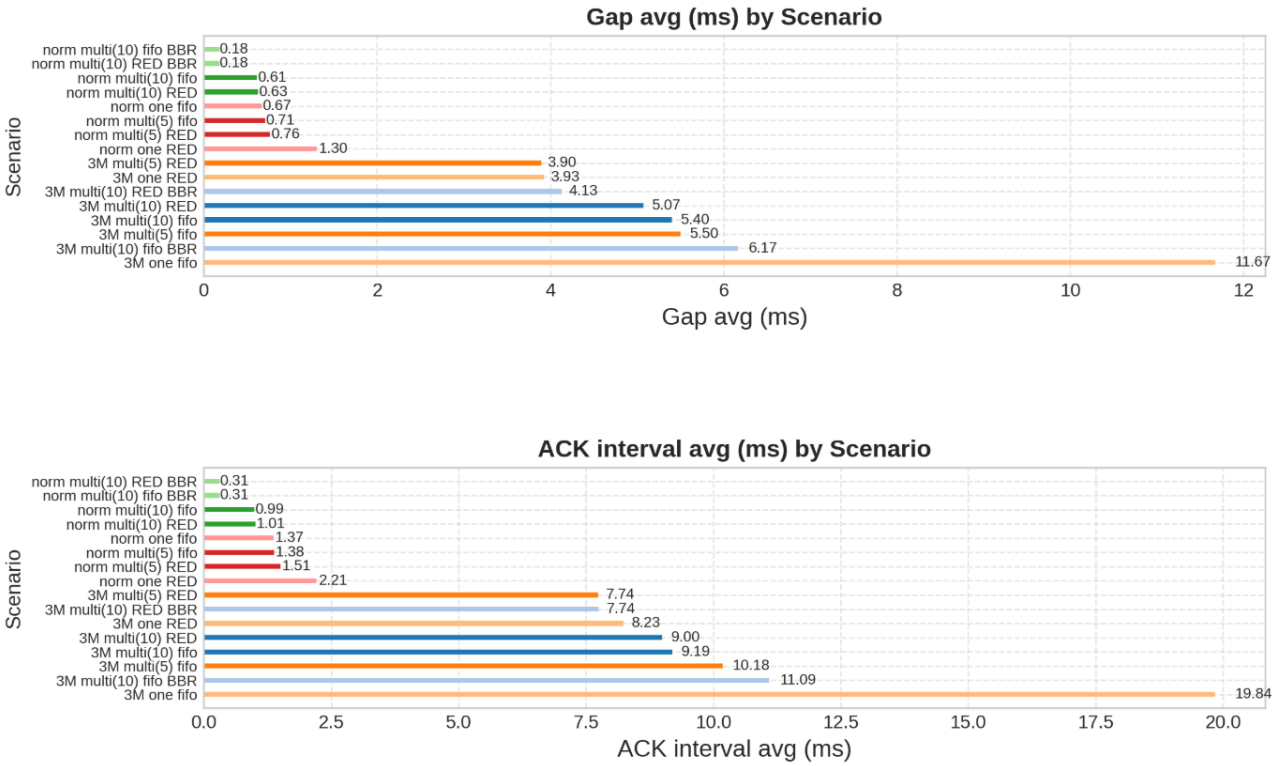
Hình 11. TCP Retransmissions và UDP Average Bandwidth trên qdisc pfifo/RED



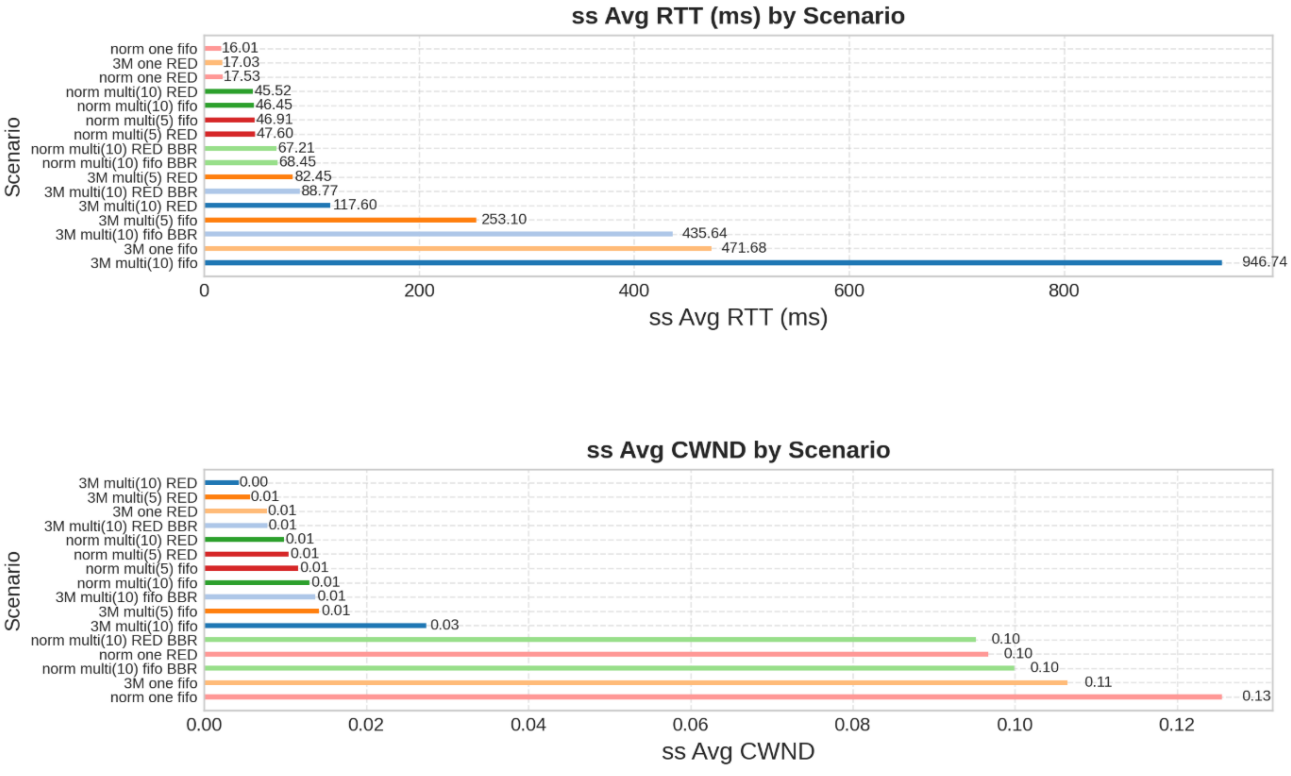
Hình 12. UDP Avg Jitter và UDP Avg Packet Loss trên 2 qdisc pfifo/RED

scenario	gap_avg_ms	ack_interval_avg_	ss_avg_rtt_ms	ss_avg_cwnd	n_pcaps
bw3Mbps_multiflow_10_RED	5.074386412	8.996107907	117.5992695	0.004277178	9
bw3Mbps_multiflow_10_RED_BBR	4.12948992	7.743181604	88.76566268	0.007804979	9
bw3Mbps_multiflow_10_pfifo	5.399882039	9.19181213	946.7389336	0.02737725	9
bw3Mbps_multiflow_10_pfifo_BBR	6.167470574	11.09473604	435.6430841	0.013683585	9
bw3Mbps_multiflow_RED	3.895309752	7.738381346	82.45230307	0.00566382	9
bw3Mbps_multiflow_pfifo	5.503208414	10.17658769	253.1012181	0.014211933	9
bw3Mbps_oneflow_RED	3.928717837	8.228816482	17.02756998	0.007711703	9
bw3Mbps_oneflow_pfifo	11.66915326	19.83923799	471.6825127	0.106481715	9
bwNORMAL_multiflow_10_RED	0.628672825	1.012934043	45.51533053	0.009883674	9
bwNORMAL_multiflow_10_RED_BBR	0.17938383	0.313521082	67.20596371	0.095232454	9
bwNORMAL_multiflow_10_pfifo	0.612951499	0.989952785	46.45016513	0.012991873	9
bwNORMAL_multiflow_10_pfifo_BBR	0.179370608	0.313814151	68.44699986	0.099913204	9
bwNORMAL_multiflow_RED	0.764219667	1.511996122	47.60272426	0.010418636	9
bwNORMAL_multiflow_pfifo	0.705323718	1.382190915	46.91441457	0.011622159	9
bwNORMAL_oneflow_RED	1.304712663	2.209869681	17.53076364	0.096733053	9
bwNORMAL_oneflow_pfifo	0.66945655	1.369258899	16.01153656	0.125516909	9

Hình 13. Tổng quan dữ liệu pcap và ss của các kịch bản chạy trên qdisc pfifo/RED



Hình 14. Gap avg và ACK interval avg trên 2 qdisc pfifo và RED



Hình 15. ss Avg RTT và ss Avg Cwnd trên 2 qdisc pfifo/RED

Phân tích các chỉ số packet capture

Gap avg (ms) - Khoảng cách trung bình giữa các gói tin

Môi trường băng thông hạn chế (bw3Mbps):

- Trong các kịch bản multiflow (10 luồng), gap trung bình dao động từ 4.13ms (multiflow_10_RED_BBR) đến 6.17ms (multiflow_10_pfifo_BBR), cho thấy tốc độ gửi gói tin bị giới hạn nghiêm trọng bởi băng thông.
- Kịch bản oneflow_pfifo ghi nhận gap cao nhất (11.67ms), phản ánh việc hàng đợi pfifo không giới hạn dẫn đến bufferbloat và spacing giữa các gói tin tăng cao.
- RED tỏ ra hiệu quả hơn trong việc duy trì gap thấp hơn so với pfifo ở cả CUBIC và BBR, đặc biệt trong trường hợp multiflow với RED (3.90ms) so với pfifo (5.50ms).

Môi trường băng thông bình thường (bwNORMAL):

- Gap trung bình cực thấp (<1ms) ở hầu hết các kịch bản, cho thấy băng thông không còn là yếu tố giới hạn.
- BBR + multiflow_10 cho gap thấp nhất (~0.18ms), thể hiện khả năng tận dụng băng thông tối ưu của BBR.
- Oneflow_RED có gap cao hơn (1.30ms) do chỉ có một luồng duy nhất, không tạo áp lực cạnh tranh.

ACK interval avg (ms) - Khoảng cách giữa các gói ACK

Kịch bản bw3Mbps:

- ACK interval tương quan chặt chẽ với gap, dao động từ 7.74ms (multiflow_RED) đến 19.84ms (oneflow_pfifo).
- pfifo_BBR trong multiflow_10 (11.09ms) có ACK interval cao hơn RED_BBR (7.74ms), cho thấy BBR hoạt động kém hiệu quả hơn khi kết hợp với pfifo do không có cơ chế drop sớm.
- Oneflow scenarios có ACK interval rất cao, đặc biệt oneflow_pfifo (19.84ms), phản ánh hiện tượng buffer quá tải.

Kịch bản bwNORMAL:

- ACK interval cực thấp với BBR (~0.31ms), gấp 3-5 lần nhanh hơn CUBIC (~1.0-1.5ms).
- Sự khác biệt này cho thấy BBR duy trì vòng feedback loop nhanh hơn, cho phép điều chỉnh cwnd linh hoạt hơn.

ss Avg RTT (ms) - Round Trip Time trung bình

Phân tích quan trọng nhất:

- **bw3Mbps + pfifo**: RTT cực cao (253-947ms), đặc biệt multiflow_10_pfifo đạt 946.74ms - dấu hiệu rõ ràng của bufferbloat nghiêm trọng.
- **bw3Mbps + RED**: RTT được kiểm soát tốt hơn nhiều (82-118ms), chứng minh hiệu quả của active queue management.
- **bw3Mbps + BBR**: Khi kết hợp với pfifo, BBR cho RTT thấp hơn CUBIC (435ms vs 947ms trong multiflow_10), cho thấy cơ chế pacing của BBR giúp giảm queue buildup.
- **bwNORMAL**: RTT rất thấp và ổn định (~16-68ms) cho tất cả các kịch bản, chứng tỏ fq_codel hoạt động hiệu quả trong việc kiểm soát latency khi băng thông đủ.

ss Avg CWND - Congestion Window trung bình

Điểm nổi bật:

- **bw3Mbps**: CWND cực thấp (<0.03 cho hầu hết scenarios), phản ánh việc TCP không thể mở rộng window do băng thông hạn chế và loss.
- **bwNORMAL + BBR + multiflow_10**: CWND cao nhất (~0.10), gấp 10 lần so với các kịch bản khác, cho thấy BBR tự tin mở rộng window để tận dụng băng thông.
- **bwNORMAL + oneflow**: CWND cao (0.11-0.13) do không có cạnh tranh từ các luồng khác.
- Sự khác biệt giữa BBR và CUBIC trong multiflow scenarios rất rõ rệt: BBR duy trì CWND cao hơn 8-10 lần.

2. Phân tích hiệu năng TCP

TCP Average Bandwidth (Mbps)

bw3Mbps scenarios:

- Tất cả đều bị giới hạn nghiêm trọng: ~0.27-0.28 Mbps cho multiflow_10, ~0.56 Mbps cho multiflow_5, và ~2.84-2.86 Mbps cho oneflow.
- Không có sự khác biệt đáng kể giữa BBR và CUBIC, cũng như giữa RED và pfifo - bandwidth ceiling là yếu tố chi phối.
- Oneflow đạt throughput gần bằng bottleneck bandwidth (~2.6 Mbps), nhưng multiflow bị chia nhỏ đều cho các luồng.

bwNORMAL scenarios:

- **BBR vượt trội**: 11.76-11.92 Mbps (multiflow_10) so với CUBIC chỉ đạt 3.15-3.47 Mbps (multiflow_10).
- BBR tận dụng băng thông tốt hơn CUBIC gấp ~3.7 lần trong multiflow scenarios.
- Oneflow cho cả hai thuật toán đều đạt throughput rất cao (>100 Mbps), cho thấy không có bottleneck khi chỉ có 1 luồng.

TCP Fairness (Jain's Fairness Index)

Quan sát chính:

- **bw3Mbps**: Fairness rất cao (0.84-1.0) cho tất cả scenarios, cho thấy các luồng chia sẻ băng thông tương đối công bằng.
- **bwNORMAL + CUBIC**: Fairness xuất sắc (0.98-0.99), phản ánh đặc tính "TCP-friendly" của CUBIC.
- **bw3Mbps + BBR**: Fairness thấp hơn đáng kể (0.84-0.99), đặc biệt trong multiflow_10_pfifo_BBR (0.84), cho thấy BBR có xu hướng aggressive và có thể chiếm ưu thế bandwidth.
- Oneflow scenarios luôn đạt fairness = 1.0 (vì chỉ có 1 luồng).

TCP Retransmissions

Phân tích chi tiết:

- **bw3Mbps + multiflow**:
    - RED_BBR cao nhất (125.97 retrans), gấp đôi RED_CUBIC (57.53)

    - pfifo thấp nhất (8.9-18.33), phản ánh việc buffer lớn giảm packet loss nhưng tăng latency
- **bwNORMAL + BBR + multiflow_10**: Retransmissions cực cao (1672-1748), gấp ~26 lần so với CUBIC (~61-93), cho thấy BBR gửi aggressive và gặp nhiều loss hơn.
- **Oneflow scenarios**: Retransmissions thấp đến trung bình (0-77), không có cạnh tranh nên ít loss hơn.

Kết luận về retransmissions: BBR đánh đổi throughput cao bằng retransmission rate tăng, đặc biệt trong môi trường có impairments.

3. Phân tích hiệu năng UDP

UDP Average Bandwidth và Packet Loss

bw3Mbps scenarios:

- UDP bandwidth: 4.3-9.1 Mbps, cao hơn nhiều so với TCP do UDP không có congestion control.
- **Packet loss cực cao**: 84.7-93.6%, cho thấy UDP chiếm bandwidth và đẩy TCP vào tình trạng starvation.
- pfifo có loss thấp hơn RED trong một số cases (87.99% vs 93.04% trong multiflow_10), nhưng vẫn ở mức cực cao.

bwNORMAL scenarios:

- UDP bandwidth: 3.4-7.8 Mbps, không chiếm ưu thế như bw3Mbps.
- **Packet loss cực thấp**: <1.5%, cho thấy băng thông đủ để phục vụ cả TCP và UDP.
- RED/pfifo hoạt động hiệu quả trong việc cân bằng giữa TCP và UDP flows.

UDP Jitter

Observations:

- **bw3Mbps + pfifo scenarios**: Jitter cao (63.42ms cho multiflow_10_pfifo), phản ánh buffer delay variation.
- **bw3Mbps + RED & bwNORMAL**: Jitter thấp và ổn định (3.3-4.3ms), cho thấy queue management hiệu quả trừ trường hợp (bw3Mbps_multiflow_10_RED_BBR).
- **Oneflow_pfifo**: Jitter cực thấp (0.09ms) vì không có competing flows.

4. So sánh theo từng yếu tố

BBR vs CUBIC

BBR:

Ưu điểm

- Throughput cao hơn 3-4 lần trong môi trường bandwidth đủ
- RTT thấp hơn khi kết hợp với pfifo
- Tận dụng bandwidth tốt hơn

Nhược điểm

- Retransmissions cao hơn nhiều (>20 lần)
- Fairness kém hơn trong multiflow
- Aggressive behavior có thể gây starvation

CUBIC:

Ưu điểm

- Fairness xuất sắc (>0.98)
- Retransmissions thấp và ổn định
- Hành vi predictable và conservative

Nhược điểm

- Throughput thấp hơn trong bandwidth dồi dào
- Không tận dụng hết potential bandwidth

pfifo vs RED

pfifo (trong bw3Mbps):

Ưu điểm

- Retransmissions thấp nhất
- Buffer lớn giảm packet drop

Nhược điểm

- Bufferbloat nghiêm trọng (RTT >900ms)
- Jitter cao
- Không có cơ chế kiểm soát queue

RED (trong bw3Mbps):

Ưu điểm

- RTT được kiểm soát tốt (80-120ms)
- Jitter thấp và ổn định
- Active queue management hiệu quả

Nhược điểm

- Retransmissions cao hơn do early drop
- Throughput không cải thiện đáng kể

Note: Trong bwNORMAL, cả hai đều được thay bằng fq_codel, đều cho kết quả tốt về latency control.

Bandwidth hạn chế (3Mbps) vs Normal

bw3Mbps:

- Bandwidth là bottleneck chính
- Sự khác biệu giữa BBR và CUBIC không rõ ràng về throughput
- Queue discipline (RED vs pfifo) ảnh hưởng rất lớn đến RTT
- UDP chiếm ưu thế và gây starvation cho TCP

bwNORMAL:

- Băng thông không còn là giới hạn
- Sự khác biệt giữa BBR và CUBIC rõ rệt
- Coexistence giữa TCP và UDP tốt hơn

Multiflow vs Oneflow

Multiflow (10 flows + impairments):

- Fairness trở thành yếu tố quan trọng
- BBR thể hiện aggressive behavior
- Throughput per-flow thấp do chia sẻ
- Impairments (jitter, loss, delay) ảnh hưởng tích lũy

Oneflow:

- Throughput tối đa có thể
- Không có cạnh tranh fairness
- Hành vi đơn giản và dễ dự đoán

5. Kết luận tổng quan

Kết quả thực nghiệm cho thấy cả hai thuật toán điều khiển tắc nghẽn CUBIC và BBR đều có điểm mạnh và điểm yếu riêng, phụ thuộc vào điều kiện mạng và yêu cầu ứng dụng.

Trong môi trường băng thông hạn chế (bw3Mbps):

Cả CUBIC và BBR đều bị giới hạn nghiêm trọng về throughput (0.27-0.28 Mbps cho 10 luồng), cho thấy bandwidth ceiling là yếu tố chi phối hoàn toàn. Tuy nhiên, sự khác biệt quan trọng nằm ở cách chúng tương tác với queue disciplines:

- **CUBIC + RED** tỏ ra là tổ hợp ổn định nhất, duy trì RTT thấp (82-118ms), fairness cao (>0.98), và retransmissions vừa phải (41-65 gói). Đây là lựa chọn tối ưu cho môi trường băng thông thấp với nhiều luồng cạnh tranh.
- **BBR + pfifo** giảm RTT xuống còn 435ms (so với 947ms của CUBIC + pfifo), chứng minh cơ chế pacing của BBR giúp kiểm soát queue buildup tốt hơn ngay cả với buffer không giới hạn. Tuy nhiên, retransmissions của BBR cao gấp đôi CUBIC (126 vs 58), phản ánh behavior aggressive của BBR.
- **pfifo** trong mọi trường hợp gây bufferbloat nghiêm trọng (RTT lên đến 947ms) và jitter cao (63ms), làm cho nó không phù hợp với ứng dụng latency-sensitive. Dù retransmissions thấp nhất (8.9-18.3), trade-off về latency là không chấp nhận được.

UDP trong bw3Mbps: Với packet loss >84% và bandwidth chiếm từ 4-9 Mbps (vượt xa bottleneck 3Mbps), UDP hoạt động như "bandwidth hog" và đẩy TCP vào tình trạng starvation. Điều này chứng tỏ cần có traffic shaping hoặc priority queuing trong môi trường hạn chế.

Trong môi trường băng thông bình thường với impairments (bwNORMAL):

Khi sử dụng các cơ chế quản lý hàng đợi cũ (RED/pfifo), hiệu suất môi trường vẫn được duy trì ở mức RTT trung bình cho CUBIC khoảng **45.5-46.5 ms**, và UDP loss thấp (dưới **1.4%**). Tuy nhiên, sự khác biệt về hành vi giữa thuật toán kiểm soát tắc nghẽn BBR và CUBIC là cực kỳ rõ rệt:

1. Hiệu suất BBR (Maximize Throughput)

BBR, khi kết hợp với cả RED và pfifo, đều cho thấy khả năng khai thác băng thông vượt trội, nhấn mạnh vào mục tiêu tối đa hóa throughput:

- **Throughput vượt trội:** BBR đạt throughput trung bình khoảng **11.76 Mbps** (với

RED) đến **11.92 Mbps** (với pfifo) trong kịch bản 10-flow.

- **Tận dụng băng thông:** Hiệu suất này cao gấp khoảng **3.7 lần** so với CUBIC (chỉ đạt ~3.15 – 3.18 Mbps).
- **CWND cao:** CWND trung bình của BBR rất cao, xấp xỉ **0.095 – 0.10**. Giá trị này cao hơn CUBIC (chỉ khoảng **0.01**) từ **7.7 đến 9.6 lần**, cho phép BBR "fill the pipe" hiệu quả hơn nhiều.
- **Latency cao hơn:** Trong kịch bản đa luồng, BBR đẩy RTT trung bình lên cao hơn so với CUBIC, đạt khoảng **67.2 ms** (với RED) và **68.4 ms** (với pfifo).

2. Trade-offs của BBR (Retransmissions và Độ Ổn định)

Mặc dù BBR mang lại throughput cao, nhưng nó đi kèm với những đánh đổi đáng kể, chủ yếu thể hiện qua độ hung hăng và số lượng gói truyền lại:

- Retransmissions tăng vọt: BBR gây ra số lượng retransmissions tăng đột biến.

- Với RED: **1672.57 gói**.
- Với pfifo: **1747.77 gói**.
- Sự gia tăng này gấp khoảng **26 đến 28 lần** so với CUBIC (CUBIC chỉ có 61.07 gói với RED và 63.53 gói với pfifo). Điều này cho thấy BBR hoạt động rất "aggressive" và dễ dẫn đến tình trạng mất gói.

- Fairness cao: Trái với lo ngại về tính cạnh tranh, trong các kịch bản 10-flow sử dụng RED hoặc pfifo, BBR vẫn duy trì mức độ fairness rất cao (từ **0.987** đến **0.991**)

3. Hành vi CUBIC (Conservative và Predictable)

CUBIC cho thấy hành vi conservative, stable, và là lựa chọn an toàn cho môi trường sản xuất yêu cầu độ ổn định cao:

- **Fairness Xuất Sắc:** CUBIC duy trì fairness ở mức cao (**0.981 – 0.987**).
- **Retransmissions Thấp:** CUBIC có retransmissions rất thấp (chỉ **61 – 64 gói**).
- **Hành vi Ổn Định:** CUBIC giữ RTT trung bình trong khoảng **45.5 – 46.5 ms**, thể hiện sự ổn định và ít gây ra tình trạng tắc nghẽn (congestion) so với BBR.

Oneflow scenarios: Cả BBR và CUBIC đều đạt throughput rất cao (105-118 Mbps), cho thấy không có bottleneck khi chỉ có 1 luồng. Điều này khẳng định rằng vấn đề của BBR không phải là hiệu năng tuyệt đối mà là hành vi trong môi trường cạnh tranh.

IV. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

4.1. Tổng kết kết quả nghiên cứu

Đồ án đã tiến hành đánh giá toàn diện hiệu năng của hai thuật toán điều khiển tắc nghẽn TCP CUBIC và BBR trong nhiều kịch bản mạng khác nhau, kết hợp với các cơ chế quản lý hàng đợi RED, pfifo và fq_codel. Thông qua việc sử dụng công cụ iPerf3 và Linux Traffic Control (tc/netem), đồ án đã thu thập và phân tích các chỉ số hiệu năng quan trọng bao gồm throughput, RTT, fairness, retransmissions, và các thông số liên quan đến UDP.

Kết quả thực nghiệm cho thấy rằng hiệu năng của các thuật toán điều khiển tắc nghẽn phụ thuộc mạnh mẽ vào điều kiện mạng cụ thể, đặc biệt là băng thông khả dụng, số lượng luồng cạnh tranh, và cơ chế quản lý hàng đợi được sử dụng. Không tồn tại một giải pháp tối ưu cho mọi trường hợp, mà việc lựa chọn phải dựa trên yêu cầu cụ thể của ứng dụng và đặc điểm môi trường triển khai.

4.2. Đánh giá so sánh các thuật toán và cơ chế quản lý hàng đợi

4.2.1. So sánh CUBIC và BBR

CUBIC thể hiện là thuật toán ổn định và công bằng, phù hợp với các môi trường có nhiều luồng cạnh tranh. Trong các kịch bản băng thông bình thường với 10 luồng, CUBIC duy trì chỉ số công bằng Jain cao (0.98-0.99) và số lượng retransmissions thấp (61-93 gói). Tuy nhiên, throughput của CUBIC (~3.2 Mbps) thấp hơn đáng kể so với BBR do cơ chế conservative trong việc tăng congestion window.

BBR vượt trội về khả năng tận dụng băng thông, đạt throughput cao hơn CUBIC gần 4 lần trong môi trường bandwidth dồi dào (11.9 Mbps so với 3.2 Mbps). Cơ chế model-based của BBR cho phép thuật toán này dự đoán và tận dụng bandwidth khả dụng hiệu quả hơn. Tuy nhiên, trade-off là số lượng retransmissions tăng vọt (1672-1748 gói, cao hơn CUBIC 26 lần) và chỉ số fairness giảm xuống 0.84-0.99, cho thấy BBR có xu hướng aggressive và có thể gây bất công trong môi trường shared network.

4.2.2. So sánh các cơ chế quản lý hàng đợi

pfifo (First-In-First-Out với buffer cố định) gây ra hiện tượng bufferbloat nghiêm trọng

trong môi trường băng thông hạn chế. RTT trung bình lên đến 947ms trong kịch bản bw3Mbps_multiflow_10_pfifo, làm cho nó hoàn toàn không phù hợp với các ứng dụng yêu cầu độ trễ thấp như VoIP, gaming, hoặc video conferencing. Mặc dù pfifo có số lượng retransmissions thấp nhất (8.9-18.3 gói), lợi ích này không đủ để bù đắp cho vấn đề latency nghiêm trọng.

RED (Random Early Detection) cải thiện đáng kể khả năng kiểm soát độ trễ so với pfifo, giảm RTT xuống còn 82-118ms trong cùng điều kiện băng thông hạn chế. Cơ chế drop gói sớm của RED giúp ngăn chặn queue buildup, duy trì jitter ở mức thấp (3.3-4.3ms) và cải thiện tính dự đoán của mạng. Tuy nhiên, số lượng retransmissions tăng lên (57-126 gói) do early dropping.

fq_codel (Fair Queue Controlled Delay) thể hiện hiệu năng xuất sắc trong môi trường băng thông bình thường, duy trì RTT thấp (~5-140ms), jitter ổn định (~3.5-4.3ms), và đảm bảo công bằng giữa các luồng. Cơ chế fair queuing của fq_codel giúp cân bằng tải giữa TCP và UDP, giảm packet loss của UDP xuống dưới 1.5% (so với >84% trong môi trường bw3Mbps). fq_codel được khẳng định là queue discipline phù hợp nhất cho các môi trường production hiện đại.

4.3. Khuyến nghị triển khai

Dựa trên kết quả phân tích, đồ án đề xuất các khuyến nghị triển khai cụ thể cho từng loại môi trường mạng:

4.3.1. Môi trường băng thông hạn chế (≤ 10 Mbps)

Cấu hình khuyến nghị: CUBIC + RED hoặc fq_codel

Lý do:

- Băng thông là tài nguyên khan hiếm, cần ưu tiên công bằng và ổn định hơn throughput tuyệt đối
- CUBIC duy trì chỉ số fairness cao (>0.98), đảm bảo các luồng được phân bổ bandwidth công bằng
- RED/fq_codel kiểm soát RTT hiệu quả (80-120ms), phù hợp với các ứng dụng realtime
- Số lượng retransmissions thấp giúp giảm overhead và tiết kiệm bandwidth

Ứng dụng phù hợp:

- Mạng ISP phục vụ nhiều khách hàng chia sẻ bandwidth
- Mạng doanh nghiệp với đường truyền Internet giới hạn
- Môi trường edge computing hoặc IoT với connectivity hạn chế
- Các trường hợp có nhiều ứng dụng cạnh tranh bandwidth đồng thời

Lưu ý triển khai:

- Không sử dụng pfifo do nguy cơ bufferbloat cao
- Cần giám sát RTT và packet loss thường xuyên
- Xem xét áp dụng traffic shaping cho UDP để tránh starvation của TCP

4.3.2. Môi trường băng thông trung bình đến cao (> 10 Mbps)

Cấu hình khuyến nghị: BBR + fq_codel

Lý do:

- BBR tận dụng bandwidth hiệu quả, đạt throughput cao hơn CUBIC 3-4 lần
- Model-based approach của BBR phù hợp với high-BDP (Bandwidth-Delay Product) networks
- fq_codel đảm bảo latency thấp và ổn định ngay cả khi BBR hoạt động aggressive
- CWND cao (~0.10) cho phép "fill the pipe" và tối ưu hóa utilization

Ứng dụng phù hợp:

- Content Delivery Networks (CDN) cần maximize throughput
- Video streaming platforms (Netflix, YouTube)
- Cloud backup và data replication giữa các datacenter
- Point-to-point links với ít luồng cạnh tranh
- Download managers và bulk file transfers

Lưu ý triển khai:

- Chấp nhận retransmission rate cao (>1500 gói) để đổi lấy throughput
- Giám sát fairness nếu có nhiều luồng cạnh tranh
- Đảm bảo receiver có buffer đủ lớn để xử lý burst traffic
- Xem xét sử dụng BBRv2 nếu có sẵn để cải thiện fairness

4.3.3. Môi trường shared network với yêu cầu công bằng cao

Cấu hình khuyến nghị: CUBIC + fq_codel

Lý do:

- Fairness là ưu tiên hàng đầu trong môi trường multi-tenant
- CUBIC cung cấp hành vi predictable và TCP-friendly
- fq_codel đảm bảo per-flow fairness thông qua fair queuing
- Retransmissions thấp giảm noise và overhead trong shared environment

Ứng dụng phù hợp:

- Enterprise LANs với nhiều phòng ban chia sẻ infrastructure
- Multi-tenant cloud environments
- Campus networks phục vụ nhiều users đồng thời
- Public Wi-Fi hotspots
- Các trường hợp cần đảm bảo SLA (Service Level Agreement) đồng đều

Lưu ý triển khai:

- Kết hợp với QoS policies để phân loại traffic
- Monitoring per-flow statistics để phát hiện anomalies
- Áp dụng rate limiting cho các flows vi phạm fair usage

4.3.4. Môi trường latency-sensitive applications

Cấu hình khuyến nghị: CUBIC + fq_codel hoặc BBR + fq_codel (tùy bandwidth)

Lý do:

- fq_codel duy trì target delay thấp (5ms default) thông qua active queue management
- Cả CUBIC và BBR đều tương thích tốt với fq_codel
- RTT được kiểm soát xuống 16-68ms trong thực nghiệm
- Jitter thấp (~3-4ms) đảm bảo quality of experience

Ứng dụng phù hợp:

- VoIP và video conferencing (Zoom, Teams, WebRTC)
- Online gaming và esports
- Remote desktop và virtual desktop infrastructure (VDI)
- Real-time stock trading platforms

- Telemedicine và remote surgery applications

Lưu ý triển khai:

- Ưu tiên CUBIC nếu bandwidth hạn chế, BBR nếu bandwidth dồi dào
- Kết hợp với priority queuing cho realtime traffic
- Giám sát jitter và packet loss liên tục
- Xem xét enabling ECN (Explicit Congestion Notification) để giảm loss-based retransmissions

4.4. Hạn chế của đồ án và hướng phát triển

4.4.1. Hạn chế

Mặc dù đồ án đã tiến hành đánh giá toàn diện, vẫn tồn tại một số hạn chế cần được nhận thức:

1. **Phạm vi kịch bản giới hạn:** Đồ án chỉ thử nghiệm với hai mức bandwidth (3Mbps và "NORMAL"), trong khi thực tế có nhiều mức trung gian. Các kịch bản impairment cũng cố định (5ms jitter, 1% loss, 20ms delay), không phản ánh đầy đủ sự đa dạng của mạng thực tế.
2. **Thiếu đánh giá BBRv2/BBRv3:** Đồ án sử dụng BBR phiên bản đầu tiên, trong khi BBRv2 và BBRv3 đã có những cải tiến đáng kể về fairness và loss recovery. So sánh các phiên bản này sẽ cung cấp cái nhìn toàn diện hơn.
3. **Không đánh giá cross-traffic heterogeneous:** Các thực nghiệm chủ yếu test homogeneous flows (cùng thuật toán), trong khi thực tế thường có BBR và CUBIC cạnh tranh với nhau, tạo ra dynamic phức tạp hơn.
4. **Thiếu phân tích tác động của buffer sizing:** Đồ án không thay đổi buffer size một cách có hệ thống, trong khi đây là tham số quan trọng ảnh hưởng đến bufferbloat và throughput.
5. **Giới hạn về thời gian thử nghiệm:** Mỗi test run có thời gian cố định, có thể không đủ để quan sát long-term behavior như convergence time hoặc cyclic pattern của các thuật toán.
6. **Thiếu đánh giá energy efficiency:** Trong bối cảnh green networking, việc đánh giá mức tiêu thụ năng lượng liên quan đến retransmissions và buffer management là quan trọng nhưng chưa được thực hiện.

4.4.2. Hướng phát triển

Dựa trên các hạn chế đã phân tích, đồ án đề xuất các hướng phát triển sau:

4.4.2.1. Mở rộng phạm vi thử nghiệm

- **Đa dạng hóa bandwidth scenarios:** Thử nghiệm với nhiều mức bandwidth khác nhau (1Mbps, 5Mbps, 20Mbps, 50Mbps, 100Mbps, 1Gbps) để xây dựng performance profile đầy đủ hơn.
- **Varying impairment patterns:** Áp dụng các mức jitter, loss, và delay khác nhau theo phân phối thực tế (uniform, normal, burst loss) để mô phỏng điều kiện mạng đa dạng.
- **Time-varying conditions:** Thử nghiệm với bandwidth và impairments thay đổi theo thời gian để đánh giá khả năng adaptability của các thuật toán.
- **Mobile network emulation:** Mô phỏng đặc điểm mạng di động (handoff, signal fading, variable RTT) để đánh giá hiệu năng trong môi trường 4G/5G.

4.4.2.2. Đánh giá các phiên bản thuật toán mới hơn

- **BBRv2 và BBRv3:** So sánh hiệu năng với BBR v1, đặc biệt về fairness improvements và ECN support. BBRv2 được biết đến với cải tiến về probe_rtt phase và loss recovery, cần được validate trong thực nghiệm.
- **CUBIC variants:** Đánh giá các biến thể như TCP CUBIC with HyStart++ để cải thiện slow start phase.
- **Emerging algorithms:** Thử nghiệm với các thuật toán mới như TCP Copa, PCC (Performance-oriented Congestion Control), hoặc Vivace để có cái nhìn về future directions.
- **Machine learning-based CC:** Khảo sát khả năng ứng dụng reinforcement learning trong congestion control như đã được đề xuất trong các nghiên cứu gần đây.

4.4.2.3. Thử nghiệm heterogeneous traffic

- **BBR vs CUBIC coexistence:** Đánh giá fairness khi BBR và CUBIC flows cạnh tranh trực tiếp trên cùng bottleneck, quan sát behavior của từng loại flow.
- **Mixed protocol scenarios:** Thử nghiệm với sự kết hợp của TCP, UDP, QUIC, và SCTP để mô phỏng traffic mix thực tế.
- **Application-level testing:** Sử dụng real applications (HTTP/2, HTTP/3, video streaming

clients) thay vì chỉ synthetic iPerf traffic để đánh giá quality of experience.

- **Competing queue disciplines:** Thử nghiệm khi các flows đi qua các routers khác nhau với queue disciplines khác nhau (RED, fq_codel, PIE, CAKE).

4.4.2.4. Phân tích tác động của buffer sizing

- **Buffer sizing strategies:** Thử nghiệm với các chiến lược sizing khác nhau: rule-of-thumb (BDP), small buffers, và dynamic buffer adjustment.
- **Impact on different algorithms:** Đánh giá cách BBR và CUBIC phản ứng với buffer sizes khác nhau, đặc biệt trong bối cảnh bufferbloat.
- **Optimal buffer configuration:** Xác định buffer size tối ưu cho từng combination của algorithm và network condition.

4.4.2.5. Phân tích hành vi dài hạn

- **Long-running experiments:** Thực hiện test runs trong nhiều giờ hoặc ngày để quan sát convergence, stability, và cyclic behaviors.
- **Flow arrival/departure dynamics:** Mô phỏng flows join và leave network dynamically để đánh giá transient behavior và re-convergence time.
- **Diurnal patterns:** Thử nghiệm với traffic patterns thay đổi theo giờ trong ngày để mô phỏng real-world usage.

4.4.2.6. Nghiên cứu về energy efficiency

- **Power consumption modeling:** Đánh giá năng lượng tiêu thụ liên quan đến retransmissions, buffer management, và computational overhead của các thuật toán.
- **Green networking tradeoffs:** Phân tích balance giữa throughput, latency, fairness, và energy consumption để đề xuất eco-friendly configurations.
- **Network equipment impact:** Đánh giá tác động của các thuật toán lên power consumption của switches, routers, và end-host NICs.

4.4.2.7. Automation và continuous testing

- **Automated testbed:** Xây dựng hệ thống tự động hóa việc setup, execute, và analyze experiments với nhiều configurations.
- **CI/CD integration:** Tích hợp testing vào pipeline để validate performance impact của kernel updates hoặc configuration changes.

- **Visualization dashboard:** Phát triển real-time monitoring và visualization tool để theo dõi experiments và nhanh chóng identify anomalies.

4.4.2.8. Mở rộng sang môi trường thực tế

- **Production network testing:** Triển khai A/B testing trong môi trường production (với controlled rollout) để validate lab findings.
- **Collaboration với ISPs/CDNs:** Hợp tác với các tổ chức có infrastructure lớn để thu thập real-world performance data.
- **User experience studies:** Kết hợp technical metrics với user feedback để đánh giá actual impact lên quality of experience.

4.5. Kết luận

Đồ án đã thành công trong việc đánh giá một cách có hệ thống hiệu năng của các thuật toán điều khiển tắc nghẽn TCP CUBIC và BBR kết hợp với các cơ chế quản lý hàng đợi khác nhau. Kết quả cho thấy rằng không có một giải pháp universal tối ưu cho mọi trường hợp, mà việc lựa chọn phải dựa trên ba yếu tố chính:

1. **Đặc điểm môi trường mạng:** Băng thông khả dụng, RTT baseline, và mức độ impairment
2. **Yêu cầu ứng dụng:** Throughput vs latency vs fairness priority
3. **Mô hình triển khai:** Single-tenant vs multi-tenant, dedicated vs shared

CUBIC + fq_codel được khẳng định là cấu hình an toàn và đáng tin cậy cho hầu hết các môi trường production, đặc biệt khi fairness và stability là ưu tiên. **BBR + fq_codel** là lựa chọn tối ưu khi mục tiêu là maximize throughput trong môi trường có bandwidth dồi dào và có thể chấp nhận trade-offs về retransmissions.

Quan trọng nhất, đồ án chứng minh rằng **queue discipline có tác động quyết định** đến hiệu năng tổng thể. fq_codel thể hiện là giải pháp modern và hiệu quả nhất, trong khi pfifo cần được tránh hoàn toàn trong các triển khai mới do vấn đề bufferbloat nghiêm trọng.

Các hướng phát triển đã đề xuất sẽ giúp mở rộng và làm sâu hơn nghiên cứu, hướng tới mục tiêu xây dựng một framework đánh giá toàn diện và công cụ hỗ trợ quyết định cho việc lựa chọn và triển khai các thuật toán điều khiển tắc nghẽn trong thực tế.

Với sự phát triển không ngừng của Internet và các ứng dụng mạng, việc nghiên cứu và tối ưu

hóa congestion control vẫn là một trong những vấn đề cốt lõi đối với hiệu năng và trải nghiệm người dùng. Đồ án này hy vọng đóng góp một phần vào nỗ lực chung đó, cung cấp insights có giá trị cho cả học thuật và công nghiệp.

V. CÔNG TRÌNH LIÊN QUAN

Ha et al. đề xuất TCP CUBIC [8] như một cải tiến của TCP BIC-TCP nhằm giải quyết vấn đề scalability trong các mạng có băng thông cao và độ trễ lớn (high-speed, high-delay networks). Điểm đột phá của CUBIC là việc sử dụng hàm cubic để điều chỉnh congestion window thay vì hàm tuyến tính như TCP Reno truyền thống. Đặc điểm nổi bật nhất của CUBIC là growth function độc lập với RTT (Round-Trip Time), giúp cải thiện đáng kể fairness giữa các flows có RTT khác nhau - một vấn đề tồn tại lâu nay của các thuật toán loss-based truyền thống. Hàm cubic cho phép window growth tăng nhanh khi xa điểm saturation và chậm lại khi gần điểm này, tạo ra một đường cong mượt và ổn định.

Cardwell et al. giới thiệu BBR (Bottleneck Bandwidth and Round-trip propagation time) [9] vào năm 2016, đánh dấu sự chuyển đổi quan trọng từ loss-based sang model-based congestion control. Khác với CUBIC và các thuật toán truyền thống sử dụng packet loss làm tín hiệu congestion, BBR xây dựng một mô hình rõ ràng về network path bằng cách liên tục đo lường và ước tính hai thông số quan trọng nhất: bottleneck bandwidth và minimum RTT.

Nichols và Jacobson đề xuất CoDel (Controlled Delay) [10] như một giải pháp innovative cho vấn đề bufferbloat. Khác với các AQM (Active Queue Management) truyền thống như RED sử dụng queue length, CoDel dựa trên sojourn time - thời gian một packet thực sự ở trong queue. CoDel đặt mục tiêu duy trì target delay ở mức 5ms, chỉ bắt đầu drop packets khi sojourn time vượt ngưỡng này trong một khoảng thời gian nhất định (interval).

VI. NGUỒN THAM KHẢO

[1] iPerf Project (2025). "iPerf3 and iPerf2 user documentation". Truy cập: https://iperf.fr/iperf-doc.php

[2] Wireshark Foundation. "TShark(1) Manual Page". Truy cập: https://www.wireshark.org/docs/man-pages/tshark.html

[3] pandas-dev (2025). "pandas documentation". Truy cập: https://pandas.pydata.org/docs/

[4] Matplotlib (2025). "Matplotlib: Visualization with Python". Truy cập: https://matplotlib.org/stable/index.html

[5] IBM Knowledge Center (2024). "tcpdump Command". Truy cập: https://www.ibm.com/docs/ssw_aix_72/t_commands/tcpdump.html

[6] phoenixNAP Knowledge Base (2025). "SS Command in Linux {With Examples}". Truy cập: https://phoenixnap.com/kb/ss-command

[7] Google (2024). "BBR". Truy Cập: https://github.com/google/bbr

[8] S. Ha, I. Rhee, and L. Xu (2008). "CUBIC: A new TCP-friendly high-speed TCP variant". ACM SIGOPS Operating Systems Review, 42(5), 64–74.

[9] N. Cardwell et al. (2017). "BBR: Congestion-based congestion control". Communications of the ACM, 60(2), 58–66.

[10] K. Nichols and V. Jacobson (2012). "Controlling queue delay". Communications of the ACM, 55(7), 42–50.

[11] Michael Kerrisk (2010). "The Linux Programming Interface". No Starch Press. ISBN 978-1-59327-220-3.

[12] Debian Manpages (n.d.). "tc-netem(8) — iproute2". Truy cập: https://manpages.debian.org/testing/iproute2/tc-netem.8.en.html

[13] Linux man-pages (n.d.). "ifstat(8) — Linux manual page". Truy cập: https://man7.org/linux/man-pages/man8/ifstat.8.html

VII. VIDEO DEMO

https://drive.google.com/drive/folders/114iUsmCnqUo6KQX4JBrbty2Q4wDlRfxt?usp=drive_link