Thực nghiệm

Tiến thành thực nghiệm với nhiều kịch bản sử dụng các qdisc, congesion control, khác nhau, giữa các kịch bản bị giới hạn bandwidth và có impairments với các kịch bản không bị giới hạn bandwidth và không có impairments, chạy với số flow khác nhau.

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

cài 3 máy ảo ubuntu 22.04 tương ứng với client, bottleneck, server.

Trên 3 máy cài các công cụ sau:

Update: Đảm bảo cài công cụ không bị lỗi

|  |
| --- |
| sudo apt -y update |

iperf3: Tạo và đo traffic TCP/UDP để kiểm tra băng thông, jitter, packet loss.

|  |
| --- |
| sudo apt -y install iperf3 |

iproute2: Cung cấp lệnh tc để cấu hình qdisc (fq\_codel, tbf...) và ss để xem thông tin kết nối TCP/UDP.

|  |
| --- |
| sudo apt install iproute2 |

tcpdump: Ghi lại toàn bộ gói tin mạng thành file .pcap để phân tích chi tiết.

|  |
| --- |
| sudo apt-get install tcpdump |

tshark: Phân tích và trích xuất dữ liệu từ file .pcap bằng dòng lệnh.

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

ifstat: Theo dõi tốc độ gửi/nhận thực tế trên các giao diện mạng.

|  |
| --- |
| sudo apt-get install ifstat |

python3, pandas, matplotlib (chỉ cài trên client): Xử lý và trực quan hóa dữ liệu đo lường (vẽ biểu đồ, tính trung bình...).

|  |
| --- |
| sudo apt install python3-pip -y  python3 -m pip install pandas  pip3 install matplotlib |

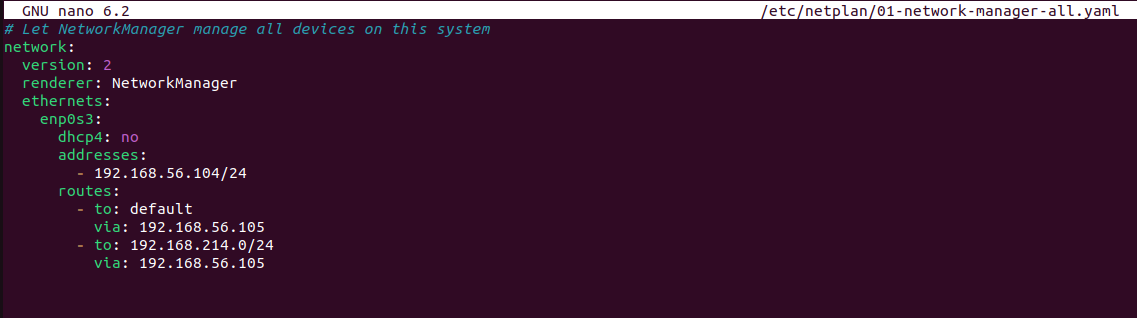
Cấu hình topology thực nghiệm

Topology:

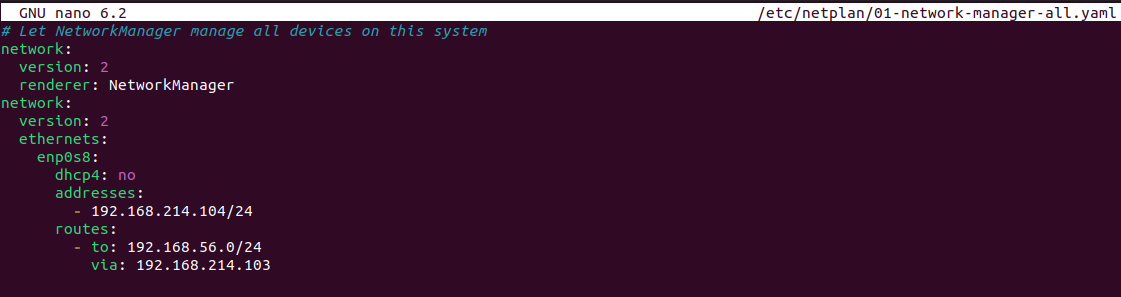
client(192.168.56.104) -> enps03(192.168.56.105) -> bottleneck -> enps08(192.168.214.103) -> server(192.168.214.104)

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

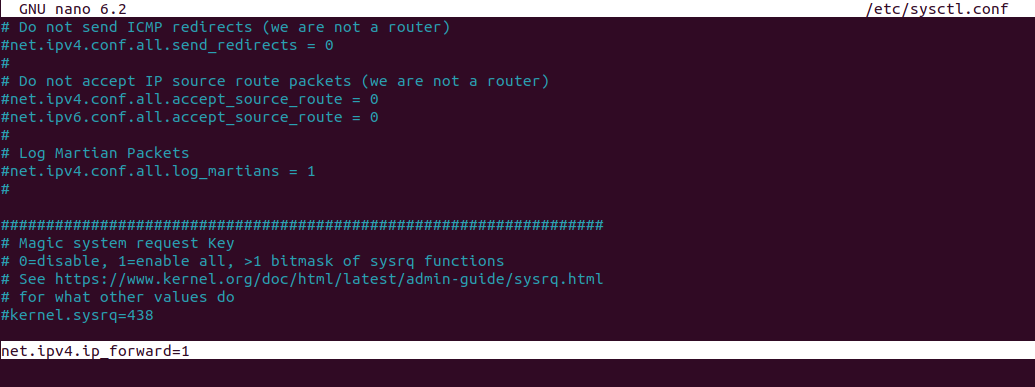
Client:



Server:



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



Cấu hình script mô phỏng và code phân tích dữ liệu cho thực nghiệm:

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.

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"; } |

* #!/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.

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 |

* 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.

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}:"${REMOTE\_TMP}/ifstat\_bottleneck\_${BOTTLENECK\_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." |

* 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

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.

**summary.py:**

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

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" |

* #!/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.

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 |

* Dùng regex để tìm số lượng flow trong tên scenario (ví dụ bw3\_5 → 5).
* Nếu không tìm được, mặc định trả về 1.

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 {} |

* 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 {}.

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) |

* 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).

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)) |

* Tính chỉ số công bằng Jain để đo độ chia sẻ băng thông giữa các flow theo công thức: fairness =
* Nếu danh sách rỗng hoặc tổng bằng 0 thì trả về 0.

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 |

* 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.

Hàm summarize\_scenario(path, 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 |

* 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).

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() |

* 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.**

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.

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.

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" |

* + #!/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.

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() |

* 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.

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 |

* 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.

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 |

* 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).

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 |

* 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.

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() |

* 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.

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.

plot\_result.py

File code python này dùng để vẽ biểu 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.

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") |

* #!/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.

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() |

* 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.

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 |

* 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).

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") |

* 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.

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") |

* 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.

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 |

* + 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.

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.

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 |

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 |

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> |

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> |

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

|  |
| --- |
| bash run\_experiments.sh |

Chạy summary.py:

|  |
| --- |
| python3 summary.py |

Chạy pcap\_summary.py:

|  |
| --- |
| python3 pcap\_summary.py |

Chạy plot\_result.py:

|  |
| --- |
| python3 plot\_results.py |

Phân tích dữ liệu các kịch bản

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 |

Kịch bản bandwidth bình thường với 1 flow chạy trên congesion control CUBIC và qdisc fq\_codel

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (CUBIC + fq\_codel) đạt throughput trung bình khoảng 778 Mbps, với giá trị tcp\_fairness = 1, hoàn toàn phù hợp với kịch bản chỉ có 1 flow, không có sự cạnh tranh băng thông. Điều này cho thấy CUBIC tận dụng tốt dung lượng đường truyền trong điều kiện băng thông bình thường, trong khi fq\_codel kiểm soát hàng đợi ổn định, giúp duy trì độ trễ thấp mà không gây queuebloat đáng kể.

Tuy throughput cao, số lượng retransmission trung bình khoảng 1537 gói vẫn là dấu hiệu cần chú ý. Tuy nhiên, do RTT và jitter thấp, phần lớn các lần tái truyền này nhiều khả năng là fast retransmit do fq\_codel chủ động drop nhẹ để điều tiết hàng đợi. Cơ chế này giúp TCP phục hồi nhanh, không ảnh hưởng đáng kể đến thông lượng tổng thể.

Đối với UDP, throughput trung bình chỉ đạt ~661 Mbps, thấp hơn TCP, với tỷ lệ mất gói khoảng 2.77%. Điều này phản ánh cơ chế drop có kiểm soát của fq\_codel khi tải vượt ngưỡng, đồng thời jitter rất nhỏ (~0.02 ms) chứng minh độ ổn định cao trong việc xử lý gói tin và duy trì thời gian truyền nhất quán.

Dữ liệu từ pcap cho thấy RTT và cwnd gần như không đổi, còn các chỉ số gap và ack\_interval nhỏ thể hiện luồng truyền liên tục, phù hợp với kịch bản tải đơn.

Kịch bản bandwidth giới hạn với 1 flow chạy trên congesion control CUBIC và qdisc fq\_codel

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (CUBIC + fq\_codel) trong kịch bản băng thông giới hạn với 1 flow đạt throughput trung bình chỉ khoảng 1.47 Mbps, phản ánh rõ ràng giới hạn của đường truyền. Giá trị tcp\_fairness = 1 hoàn toàn phù hợp với trường hợp chỉ có một luồng TCP, cho thấy không có hiện tượng cạnh tranh chia sẻ băng thông. Tuy băng thông thấp, hệ thống vẫn duy trì truyền tải ổn định nhờ cơ chế điều tiết của CUBIC và fq\_codel.

Số lượng retransmission trung bình khoảng 73 gói/run là mức thấp và hợp lý trong môi trường hạn chế băng thông, có thể chủ yếu do fast retransmit hơn là mất gói thực sự. RTT trung bình tại nút bottleneck chỉ khoảng 0.47–0.58 ms, cho thấy độ trễ đường truyền rất thấp và ổn định. Các giá trị gap và ack\_interval cao (~12–14 ms) phản ánh tốc độ trao đổi gói chậm hơn, phù hợp với đường truyền hẹp.

Trong khi đó, UDP đạt throughput trung bình ~1.0 Gbps, nhưng có tỷ lệ mất gói cực cao (~98.9%), nghĩa là phần lớn gói tin bị drop do vượt quá giới hạn băng thông. Điều này chứng tỏ fq\_codel vẫn hoạt động đúng vai trò, ưu tiên kiểm soát độ trễ và giảm queuebloat bằng cách loại bỏ gói khi hàng đợi đầy. Tuy nhiên, jitter chỉ ~1.2 ms, cho thấy độ biến thiên thời gian giữa các gói vẫn được kiểm soát tốt.

Kịch bản bandwidth bình thường với 5 flow chạy trên congesion control CUBIC và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifsstat



Dữ liệu phần pcap và ss



TCP (CUBIC + fq\_codel) trong kịch bản 5 flow có impairments đạt throughput trung bình chỉ ~2.7 Mbps/flow, thấp hơn đáng kể so với kịch bản không giới hạn băng thông. Giá trị tcp\_fairness ≈ 0.94 cho thấy mức chia sẻ băng thông giữa các luồng tương đối đồng đều, dù có sự ảnh hưởng của nhiễu và mất gói. Điều này phản ánh khả năng điều tiết công bằng của fq\_codel vẫn được duy trì khi có nhiều luồng hoạt động song song.

Tuy nhiên, số lượng retransmission trung bình khoảng 99 gói/flow và RTT trung bình lên tới 58–80 ms (tại bottleneck) cho thấy đường truyền bị ảnh hưởng mạnh bởi độ trễ và packet loss. fq\_codel có thể đã chủ động drop gói để hạn chế queuebloat, nhưng khi kênh có sẵn impairments, việc này dẫn đến TCP CUBIC phải tái truyền thường xuyên hơn. Mặc dù vậy, các giá trị gap và ack\_interval ổn định (~2 ms) cho thấy kết nối vẫn duy trì luồng dữ liệu đều đặn.

Đối với UDP, throughput trung bình chỉ ~185 Mbps và tỷ lệ mất gói cực cao (~87%) phản ánh rõ tác động tiêu cực của impairments. Dù vậy, jitter ~12.8 ms chứng minh fq\_codel vẫn kiểm soát tốt độ ổn định thời gian gói, giảm biến thiên trễ trong điều kiện mất mát nặng.

Kịch bản bandwidth giới hạn với 5 flow chạy trên congesion control CUBIC và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (CUBIC + fq\_codel) trong kịch bản băng thông giới hạn với 5 flow và có impairments đạt throughput trung bình chỉ khoảng 0.27 Mbps/flow, giảm mạnh so với trường hợp 1 flow do phải chia sẻ băng thông và chịu ảnh hưởng từ mất mát, trễ và jitter. Giá trị tcp\_fairness ≈ 0.99 cho thấy các luồng TCP vẫn chia sẻ công bằng, chứng tỏ fq\_codel và CUBIC duy trì ổn định truyền tải dù môi trường bất lợi.

Số lượng retransmission trung bình ~56 gói/run phản ánh hiện tượng mất gói nhẹ do impairments, chủ yếu dẫn đến fast retransmit. RTT trung bình tại nút bottleneck tăng cao (55–86 ms) với độ lệch lớn (~45–54 ms), cho thấy độ trễ hàng đợi và biến thiên thời gian truyền cao hơn nhiều so với kịch bản không impairments. Cwnd nhỏ (~13–23 kB) chứng tỏ CUBIC phải liên tục giảm tốc độ tăng cửa sổ truyền để thích nghi với điều kiện mạng xấu.

Trong khi đó, UDP đạt throughput trung bình ~223 Mbps nhưng có tỷ lệ mất gói rất cao (~94.8%), nghĩa là phần lớn gói tin bị drop khi hàng đợi đầy. Jitter ~24.6 ms thể hiện biến động lớn trong thời gian giữa các gói, phù hợp với môi trường có mất mát và trễ cao.

Kịch bản bandwidth bình thường với 10 flow chạy trên congesion control CUBIC và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (CUBIC + fq\_codel) trong kịch bản băng thông bình thường với 10 flow và có impairments đạt throughput trung bình khoảng 2.62 Mbps/flow, tổng hợp lại vẫn thể hiện khả năng tận dụng tốt đường truyền trong điều kiện có chia sẻ băng thông. Giá trị tcp\_fairness ≈ 0.97 cho thấy các luồng TCP phân chia băng thông khá đồng đều, phản ánh hiệu quả cân bằng của fq\_codel và khả năng điều tiết cửa sổ truyền của CUBIC ngay cả khi có mất mát và trễ.

Số lượng retransmission trung bình ~84 gói/run ở mức vừa phải, cho thấy mạng có xuất hiện mất gói hoặc trễ ngẫu nhiên do impairments, nhưng không ảnh hưởng nghiêm trọng đến tính ổn định của luồng TCP. RTT tại nút bottleneck cao (~83–90 ms) với độ lệch lớn (~41–63 ms), phản ánh rõ tác động của độ trễ mạng và biến thiên thời gian truyền. Cwnd nhỏ (~16–20 kB) cho thấy CUBIC đang điều chỉnh tốc độ tăng cửa sổ cẩn trọng để thích nghi với điều kiện không ổn định.

Trong khi đó, UDP đạt throughput trung bình ~78 Mbps nhưng mất gói rất cao (~84%), tức phần lớn gói tin bị drop do vượt quá khả năng xử lý hàng đợi. Jitter trung bình ~15 ms chứng tỏ biến thiên thời gian gói khá lớn, phù hợp với đặc tính của mạng có impairments và tải cao.

Kịch bản bandwidth bình thường với 10 flow chạy trên congesion control BBR và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (BBR + fq\_codel) trong kịch bản băng thông bình thường có impairments với 10 flow đạt throughput trung bình khoảng 12.0 Mbps, thể hiện khả năng khai thác băng thông tốt của BBR trong điều kiện mạng có nhiễu và tổn thất gói. Tuy nhiên, giá trị tcp\_fairness ≈ 0.69 cho thấy mức độ công bằng giữa các luồng chưa cao — một số flow có thể chiếm ưu thế về băng thông, phản ánh đặc trưng của BBR khi có nhiều luồng cùng cạnh tranh.

Số lượng retransmission trung bình ~1589 gói/run là tương đối cao, cho thấy hiện tượng mất gói và/hoặc fast retransmit xảy ra thường xuyên do tác động của impairments và cạnh tranh băng thông giữa các flow. Dù vậy, nhờ cơ chế pacing và điều chỉnh cwnd theo ước lượng băng thông–RTT, BBR vẫn duy trì throughput ở mức khá ổn định qua ba lần chạy (dao động 7.5–18 Mbps).

Tại nút bottleneck, RTT trung bình dao động 83–110 ms, cao hơn đáng kể so với đường truyền bình thường, phản ánh ảnh hưởng rõ rệt của impairments và khả năng queueing tại hàng đợi. Độ lệch chuẩn RTT (rtt\_std\_ms ~40–62 ms) cho thấy độ trễ biến động mạnh, phù hợp với đặc tính của mạng có mất gói ngẫu nhiên và trễ biến thiên. Giá trị cwnd trung bình rất thấp (~0.02 kB) tại bottleneck không thực sự phản ánh kích thước cửa sổ thực tế mà có thể do phương pháp đo chưa tương thích với pacing của BBR (BBR không tăng cwnd theo độ trễ như CUBIC).

Về UDP, throughput trung bình đạt ~74 Mbps nhưng tỷ lệ mất gói cực cao (~84%), nghĩa là phần lớn gói UDP bị drop khi cạnh tranh với các luồng TCP trong hàng đợi fq\_codel. Điều này đúng với đặc tính của fq\_codel — ưu tiên kiểm soát độ trễ bằng cách chủ động loại bỏ gói khi queue đầy. Dù mất nhiều gói, jitter trung bình chỉ khoảng 16 ms, cho thấy fq\_codel vẫn giữ được độ ổn định thời gian giữa các gói.

Kịch bản bandwidth giới hạn với 10 flow chạy trên congesion control CUBIC và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và sss



TCP (CUBIC + fq\_codel) trong kịch bản băng thông giới hạn (3 Mbps) với 10 flow và có impairments đạt throughput trung bình chỉ khoảng 0.16 Mbps/flow, cho thấy băng thông thực tế bị chia nhỏ đáng kể giữa các luồng TCP và chịu ảnh hưởng mạnh bởi giới hạn đường truyền. Giá trị tcp\_fairness ≈ 0.99 chứng tỏ CUBIC duy trì mức độ công bằng rất cao giữa các luồng, không có luồng nào chiếm ưu thế rõ rệt.

Số lượng retransmission trung bình khoảng 58 gói/run, là mức thấp và hợp lý trong môi trường có băng thông hạn chế, cho thấy mất gói không nghiêm trọng, phần lớn retransmission có thể đến từ fast retransmit hơn là mất gói vật lý. Mặc dù throughput thấp, CUBIC vẫn duy trì ổn định truyền tải nhờ cơ chế tăng cwnd tuyến tính và giảm chậm sau khi phát hiện tắc nghẽn.

Tại nút bottleneck, RTT trung bình 80–86 ms với độ lệch chuẩn khá cao (~40–52 ms) phản ánh ảnh hưởng của trễ mạng và queueing delay do impairments. Giá trị cwnd trung bình rất nhỏ (~0.02–0.13 kB) phù hợp với băng thông thấp, cho thấy mỗi luồng chỉ có thể giữ một lượng nhỏ dữ liệu đang chờ ACK. Khoảng cách giữa các gói (gap ~10–12 ms) và ack\_interval tương ứng (~11–12 ms) cho thấy tốc độ trao đổi gói tương đối chậm, đúng với đặc tính của đường truyền hẹp và giới hạn tốc độ.

Trong khi đó, UDP đạt throughput trung bình ~87 Mbps, cao hơn nhiều so với TCP, nhưng tỷ lệ mất gói cực cao (~97%), nghĩa là phần lớn gói UDP bị drop khi vượt quá dung lượng hàng đợi. Điều này thể hiện rõ cơ chế hoạt động của fq\_codel — ưu tiên giảm queuebloat và giữ độ trễ thấp bằng cách loại bỏ gói sớm khi hàng đợi đầy. Dù vậy, UDP jitter trung bình ~26 ms vẫn ở mức chấp nhận được trong môi trường nhiễu, cho thấy fq\_codel vẫn duy trì độ ổn định tương đối của khoảng thời gian giữa các gói còn lại.

Kịch bản bandwidth giới hạn với 10 flow chạy trên congesion control BBR và qdisc fq\_codel có impairments

Dữ liệu phần iperf3 và ifstat



Dữ liệu phần pcap và ss



TCP (BBR + fq\_codel) trong kịch bản băng thông giới hạn (3 Mbps) với 10 flow và có impairments đạt throughput trung bình khoảng 0.25 Mbps/flow, cao hơn nhẹ so với CUBIC trong cùng điều kiện, cho thấy khả năng BBR tận dụng băng thông tốt hơn dù vẫn bị giới hạn bởi dung lượng đường truyền. Chỉ số tcp\_fairness ≈ 0.998 chứng minh các luồng BBR chia sẻ băng thông rất đồng đều, phản ánh đặc tính ổn định và công bằng cao của thuật toán điều khiển tắc nghẽn dựa trên tốc độ.

Số lượng retransmission trung bình ~97 gói/run, nhỉnh hơn so với CUBIC (~58 gói), phù hợp với cơ chế hoạt động của BBR — vốn không dựa trên mất gói để điều chỉnh tốc độ, nhưng có thể sinh thêm một số retransmission nhỏ do đo đạc sai lệch tạm thời của RTT hoặc bandwidth estimation khi có jitter và impairments. Nhìn chung, mức này vẫn được xem là thấp, thể hiện BBR duy trì truyền tải hiệu quả và không bị tắc nghẽn nghiêm trọng.

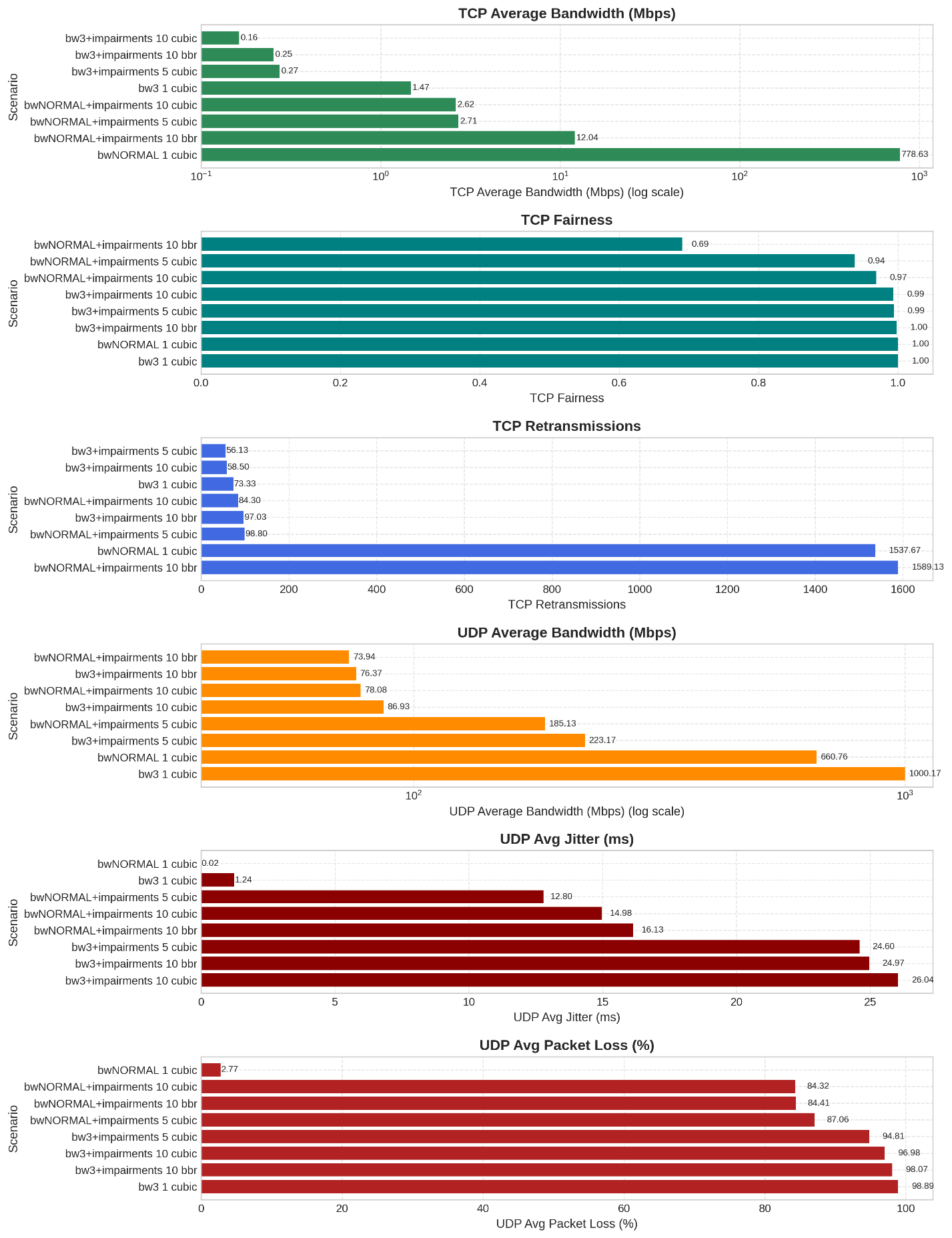
Tại nút bottleneck, RTT trung bình 92–106 ms với độ lệch chuẩn khá cao (~65–70 ms), phản ánh ảnh hưởng đáng kể của độ trễ hàng đợi và jitter do impairments gây ra. Cwnd trung bình chỉ khoảng 0.02–2.68 kB, rất nhỏ, phù hợp với thiết kế của BBR khi giới hạn cwnd dựa trên bandwidth-delay product thực tế. Các giá trị gap và ack\_interval khoảng 5–6 ms thể hiện tốc độ phản hồi nhanh và ổn định của cơ chế pacing trong BBR, giúp giảm hiện tượng bursty traffic.

Đối với UDP, throughput trung bình đạt ~76 Mbps, cao hơn TCP nhiều lần nhưng đi kèm tỷ lệ mất gói cực cao (~98%), cho thấy fq\_codel đã chủ động loại bỏ phần lớn gói UDP để duy trì độ trễ thấp và ngăn queuebloat. Tuy nhiên, UDP jitter trung bình chỉ ~25 ms, tương đối thấp so với mức mất gói, cho thấy fq\_codel vẫn kiểm soát được độ biến thiên thời gian giữa các gói còn lại, giúp hệ thống duy trì ổn định tương đối trong điều kiện tải cao.

So sánh tổng quan và kết luận

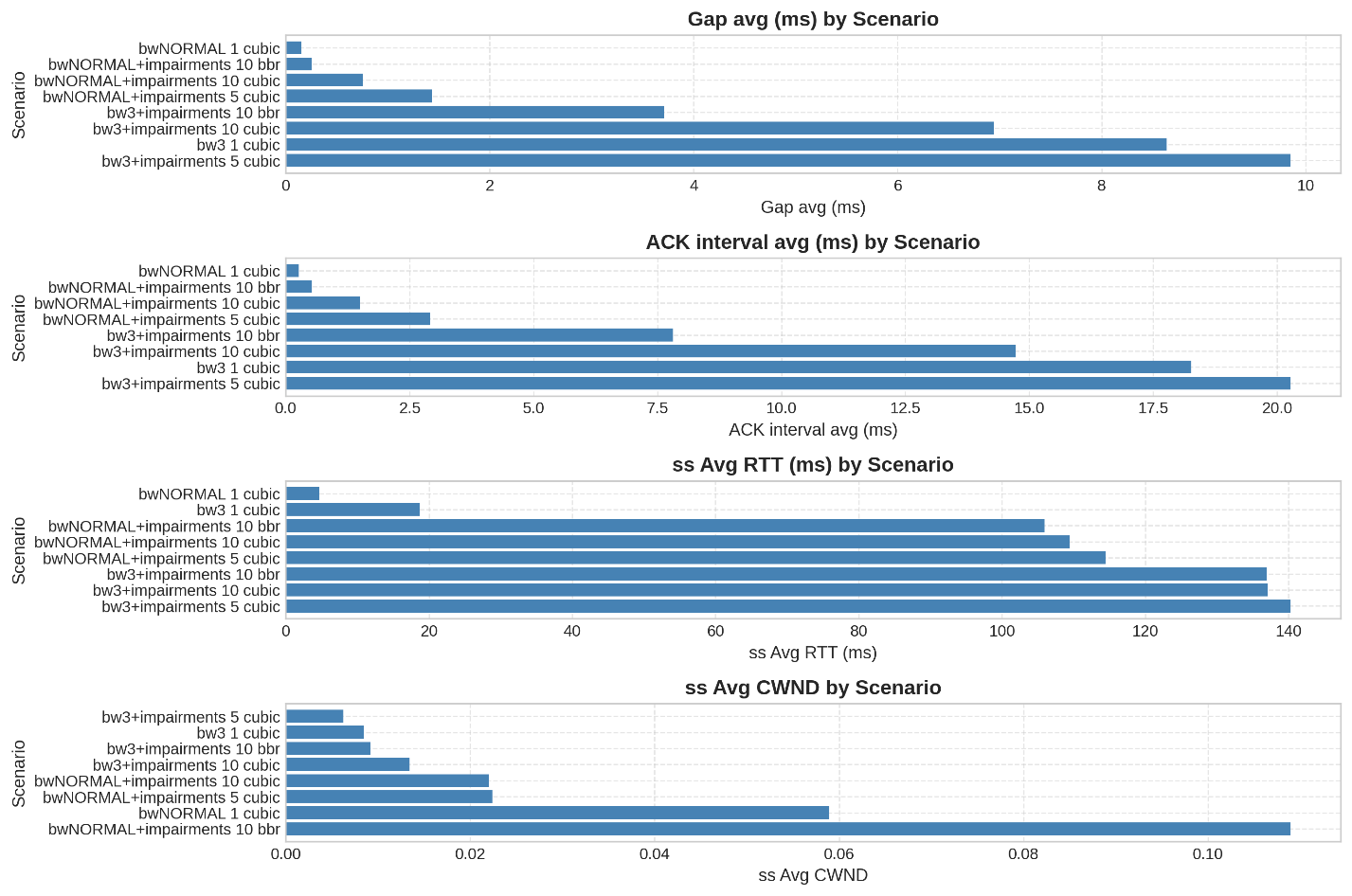
So sánh tổng quan phần iperf3





So sánh tổng quan phần pcap





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.