Extended PCC Vivace: Online-Learning Congestion Control

Girish Singh Thakur (2025MCS2973)

Nitish Kumar (2025MCS2100)

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

Baseline Utility Function

$$U = t \cdot S(d) - \lambda \cdot t \cdot l$$

Where:

t: **Throughput** (bits/sec)

d: **Delay** (or latency)

l: **Loss rate** (fraction of packets lost)

$S(d)$:a *delay sensitivity function* reduces the score if delay is high

λ: weight controlling how strongly to penalize loss

Extension 1 - Traffic-Aware Utility Selection

Baseline Utility Function

$$U = t \cdot S(d) - \lambda \cdot t \cdot l$$

Where:
t: **Throughput** (bits/sec)
d: **Delay** (or latency)
l: **Loss rate** (fraction of packets lost)
$S(d)$:a *delay sensitivity function* reduces the score if delay is high
λ: weight controlling how strongly to penalize loss

Solution

Traffic classifier + specialized utility functions

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

STEP 1: AUTOMATIC TRAFFIC CLASSIFICATION - *Traffic Classifier*

FEATURE EXTRACTION

- Packet Size Distribution
  (avg, std, median, max, min)
- Inter-Arrival Times (IAT)
  (mean, variance, coefficient of variation)
- Entropy of Packet Sizes
  (measures size variability)
- Burst Detection
  (identifies bursty patterns)
- Periodicity Score
  (consistency in timing)

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

STEP 1: AUTOMATIC TRAFFIC CLASSIFICATION - *Traffic Classifier*

FEATURE EXTRACTION

- Packet Size Distribution
  (avg, std, median, max, min)
- Inter-Arrival Times (IAT)
  (mean, variance, coefficient of variation)
- Entropy of Packet Sizes
  (measures size variability)
- Burst Detection
  (identifies bursty patterns)
- Periodicity Score
  (consistency in timing)

TRAFFIC CLASSIFICATION

BULK (Throughput Optimization)
- Avg packet size ≥ 1200 bytes
- Low size variance, high entropy

STREAMING (Stability Focus)
- Medium packets (700-1200 bytes)
- Consistent timing, moderate bursts
- High periodicity score

REALTIME (Latency Optimization)
- Small packets < 300 bytes
- Low variance, frequent bursts
- Periodic inter-arrivals

DEFAULT (Fallback)
- Low confidence threshold
- Unknown/mixed traffic pattern

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

STEP 1: AUTOMATIC TRAFFIC CLASSIFICATION - *Traffic Classifier*

FEATURE EXTRACTION

- Packet Size Distribution
  (avg, std, median, max, min)
- Inter-Arrival Times (IAT)
  (mean, variance, coefficient of variation)
- Entropy of Packet Sizes
  (measures size variability)
- Burst Detection
  (identifies bursty patterns)
- Periodicity Score
  (consistency in timing)

TRAFFIC CLASSIFICATION

BULK (Throughput Optimization)
- Avg packet size ≥ 1200 bytes
- Low size variance, high entropy

STREAMING (Stability Focus)
- Medium packets (700-1200 bytes)
- Consistent timing, moderate bursts
- High periodicity score

REALTIME (Latency Optimization)
- Small packets < 300 bytes
- Low variance, frequent bursts
- Periodic inter-arrivals

DEFAULT (Fallback)
- Low confidence threshold
- Unknown/mixed traffic pattern

CLASSIFICATION PROCESS FLOW

Observe Packets (min. 20) → Extract Features → Score Each Type (bulk/streaming/realtime) → Select Best Match
→ Validate Against Confidence Threshold (default: 0.5) → Output: (Traffic Type, Confidence Score)
→ MetaController Uses Classification to Select Specialized Utility Function → Optimize Network Parameters

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

STEP 2: SPECIALIZED UTILITY FUNCTIONS - *Utility Function Bank*

FILE TRANSFER (FTP)

STREAMING

REAL TIME

Extension 1 - Traffic-Aware Utility Selection

STEP 2: SPECIALIZED UTILITY FUNCTIONS - *Utility Function Bank*

FILE TRANSFER (FTP)

VIDEO STREAMING

VOIP (Video Call)

1. Bulk Transfer Utility (U_bulk)

$$U_{bulk} = \alpha_1 T - \alpha_2 l$$

Meaning:

T: throughput (bits/sec)

l: loss rate (fraction of packets lost)

α_1, α_2 :weights that balance importance of throughput vs. loss

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 – Traffic-Aware Utility Selection

STEP 2: SPECIALIZED UTILITY FUNCTIONS — *Utility Function Bank*

FILE TRANSFER (FTP)

VIDEO STREAMING

VOIP (Video Call)

1. Bulk Transfer Utility (U_bulk)

$$U_{bulk} = \alpha_1 T - \alpha_2 l$$

Meaning:
T: throughput (bits/sec)
l: loss rate (fraction of packets lost)
α_1, α_2 :weights that balance importance of throughput vs. loss

2. Real-time Utility (U_realtime)

$$U_{realtime} = \beta_1 T \cdot S(L_{\max})$$

Meaning:
T: throughput (enough to carry audio/video data)
$L_{\max}$:tail latency (worst-case delay)
$S(L_{\max})$:decreasing function - gives low score when tail latency is large
β_1 :scaling weight

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 – Traffic-Aware Utility Selection

STEP 2: SPECIALIZED UTILITY FUNCTIONS — *Utility Function Bank*

FILE TRANSFER (FTP)

VIDEO STREAMING

VOIP (Video Call)

1. Bulk Transfer Utility (U_bulk)

$$U_{bulk} = \alpha_1 T - \alpha_2 l$$

Meaning:

T: throughput (bits/sec)

l: loss rate (fraction of packets lost)

α_1, α_2 :weights that balance importance of throughput vs. loss

2. Real-time Utility (U_realtime)

$$U_{realtime} = \beta_1 T \cdot S(L_{\max})$$

Meaning:

T: throughput (enough to carry audio/video data)

$L_{\max}$:tail latency (worst-case delay)

$S(L_{\max})$:decreasing function — gives low score when tail latency is large

β_1 :scaling weight

3. Streaming Utility (U_streaming)

$$U_{streaming} = \gamma_1 T \cdot I(T > T_{\min}) - \gamma_2 Var(T)$$

Meaning:

T: throughput

$I(T > T_{\min})$:indicator = 1 if throughput above threshold, else 0

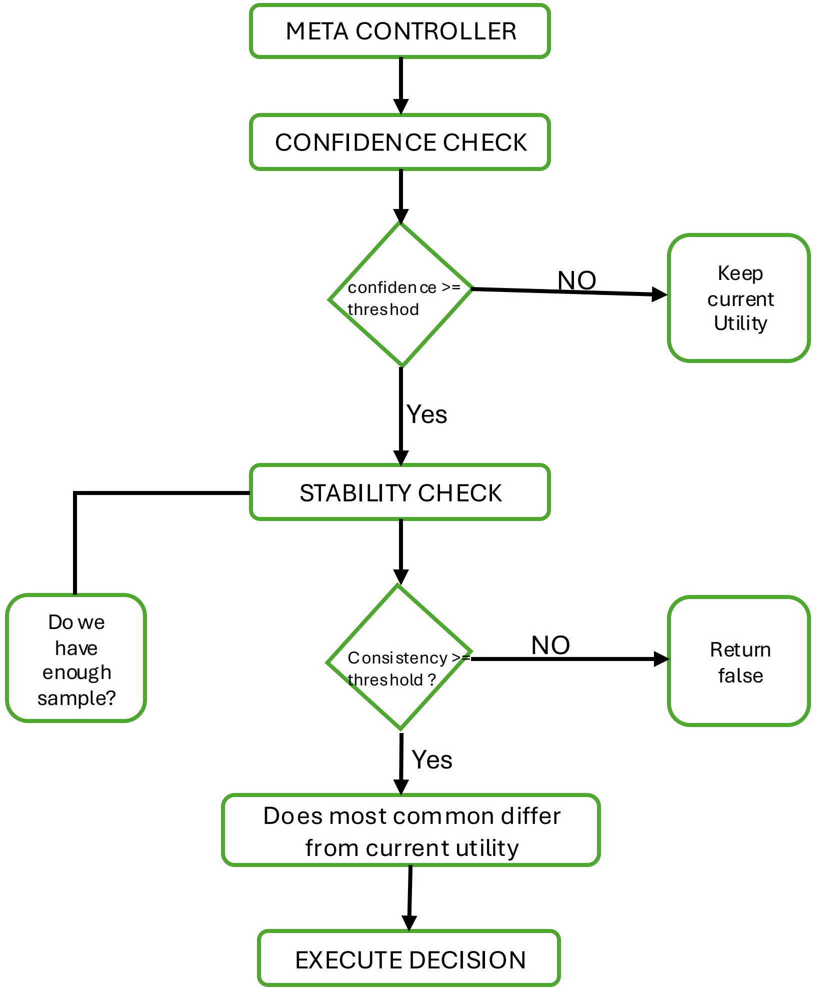
Var(T) :variance of throughput (instability)

γ_1, γ_2 :weighting constants

Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

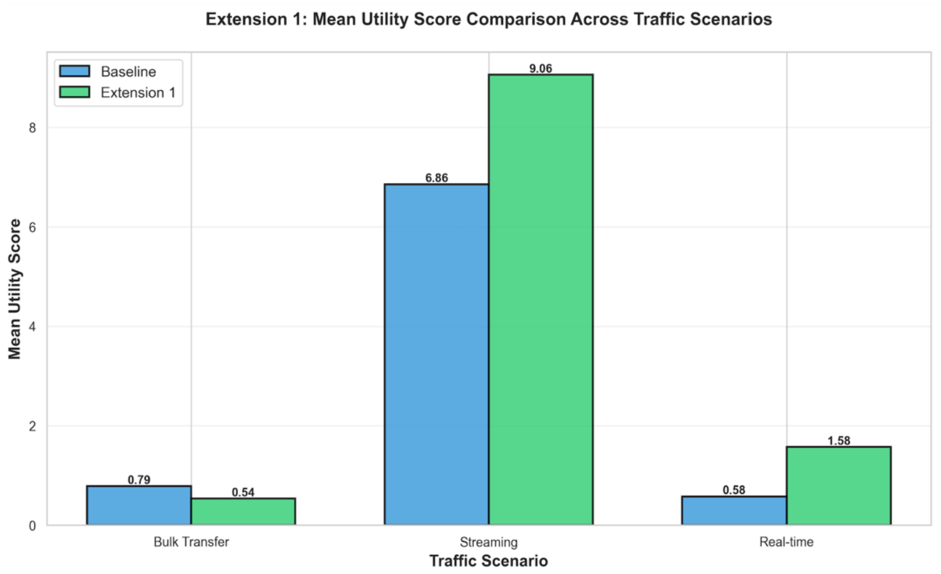
STEP 3: INTELLIGENT SWITCHING - *MetaController*

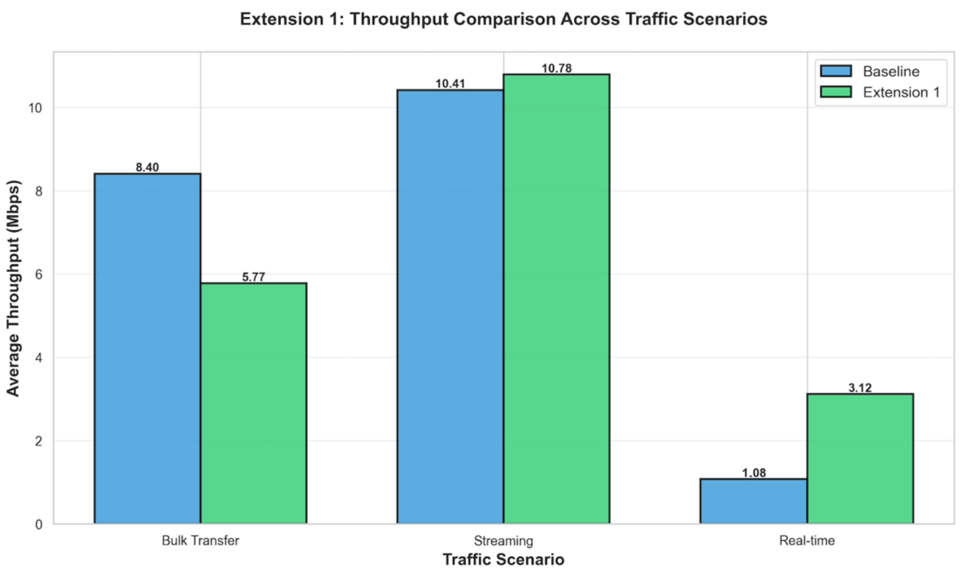


Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

Results:

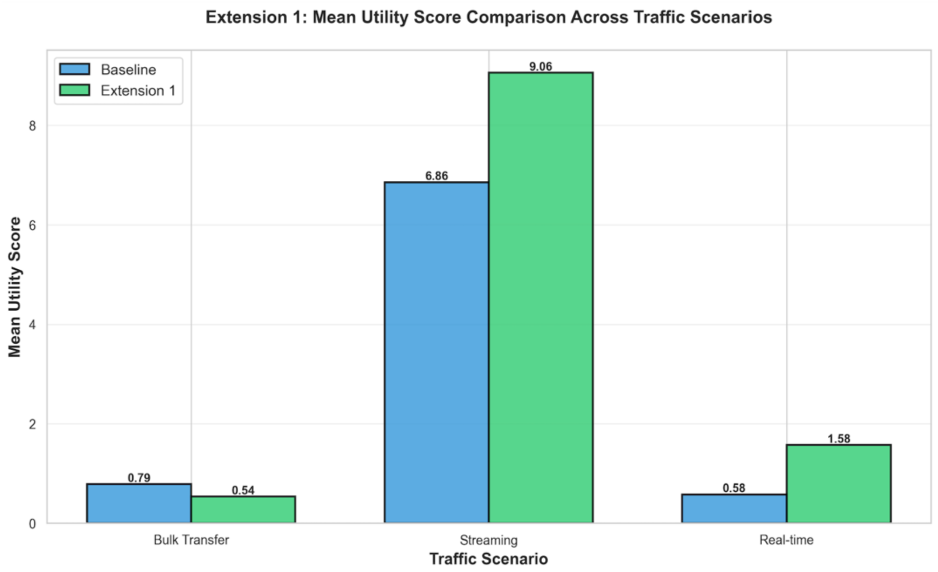


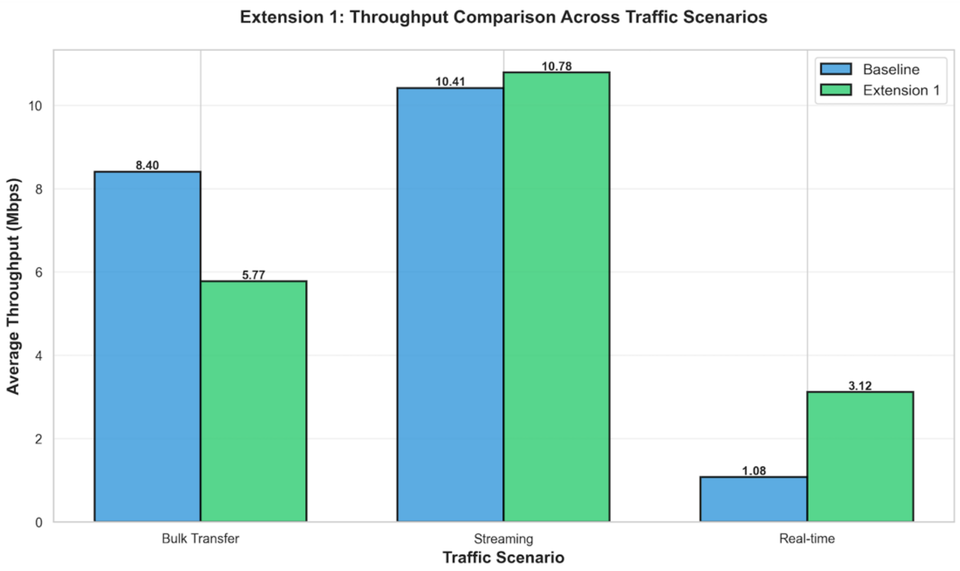


Extended PCC Vivace: Online-Learning Congestion Control

Extension 1 - Traffic-Aware Utility Selection

Results:





Summary (In One Line)

Extension 1 transforms the generic PCC baseline into an intelligent, context-aware system that automatically classifies traffic and dynamically applies the most suitable utility function-ensuring each application achieves its ideal network performance.

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Extended PCC Vivace: Online-Learning Congestion Control

⚠️ Extension 2 - Loss-Type Awareness

🚨 Problem

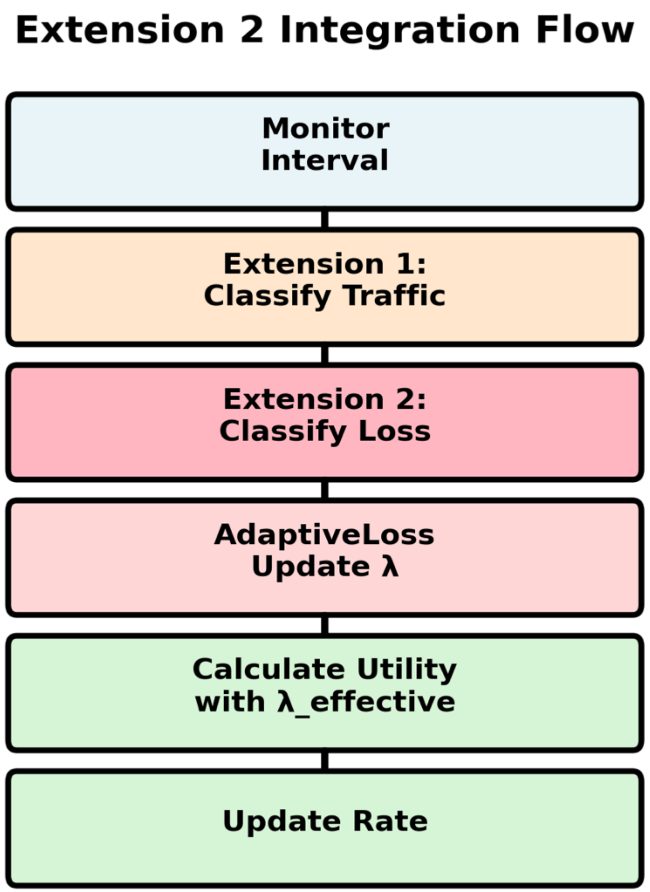
Not all losses mean congestion.
In reality, packet loss can come from two completely different causes:

Type of Loss	Cause	Correct Reaction	Why PCC Vivace Fails
Congestion Loss	Queue overflow at router when link is saturated	Reduce rate	✅ This is correct behavior
Wireless Loss	Random link-layer corruption in Wi-Fi or cellular (fading, interference)	Keep rate steady	❌ PCC wrongly reduces rate

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

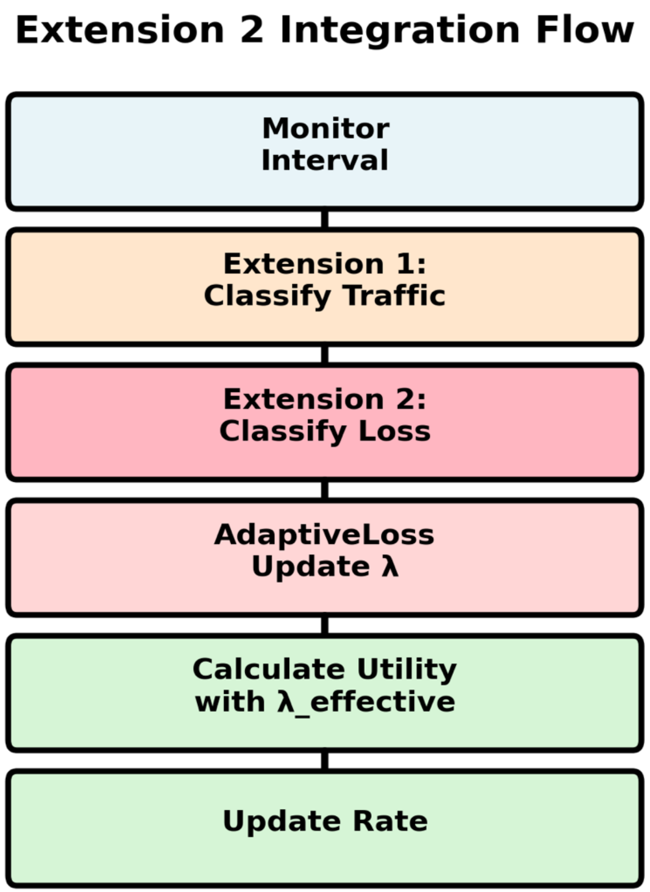
Approach: Correlation-based classification (Pearson coefficient)



Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)



In each Monitor Interval (every ~100ms):

1. Measure loss and RTT
   → Send to Loss Classifier

2. Loss Classifier analyzes the data
   → Calculates correlation
   → Returns: p_wireless = 0.8, confidence = 0.9

3. Adaptive Loss Coefficient receives this information
   → Calculates $\lambda = 2.0 +$ (correlation-based adjustment)

4. When calculating utility:
   → Instead of using fixed $\lambda = 10.0$
   → Use $\lambda = 2.0$ (for wireless) or 10.0 (for congestion)

5. PCC adjusts rate based on this utility
   → If wireless loss: Maintains rate (high utility)
   → If congestion loss: Reduces rate (low utility)

Extended PCC Vivace: Online-Learning Congestion Control

⚠️ Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Component 1: Loss Classifier — Detect What Type of Loss

Core Idea

The classifier uses **correlation between packet loss and RTT (Round-Trip Time)** to identify the nature of packet loss.

Type	Observation	Correlation
Congestion Loss	RTT increases with loss (queue builds up)	**High correlation**
Wireless Loss	RTT stable despite loss	**Low correlation**

Extended PCC Vivace: Online-Learning Congestion Control

⚠ Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data

Every time a monitor interval (MI) ends:

- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data

Every time a monitor interval (MI) ends:

- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Step 2: Compute RTT Inflation

- RTT inflation = current RTT – baseline RTT
- If RTT is much larger than baseline, the buffer is filling up → likely congestion.

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data

Every time a monitor interval (MI) ends:

- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Step 2: Compute RTT Inflation

- RTT inflation = current RTT – baseline RTT
- If RTT is much larger than baseline, the buffer is filling up → likely congestion.

Step 3: Align Loss & RTT Events by Time

For each RTT event, find the nearest loss event.

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data
Every time a monitor interval (MI) ends:
- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Step 2: Compute RTT Inflation
- RTT inflation = current RTT – baseline RTT
- If RTT is much larger than baseline, the buffer is filling up → likely congestion.

Step 3: Align Loss & RTT Events by Time
For each RTT event, find the nearest loss event.

Step 4: Calculate Correlation
Use **Pearson correlation** between binary loss and RTT arrays:

$$\rho = \frac{\sum (x - \mu_x)(y - \mu_y)}{\sigma_x \sigma_y}$$

Where:
$\rho = 1$: perfectly correlated (congestion)
$\rho = 0$: uncorrelated (wireless)
$\rho = -1$: inversely correlated (rare)

Extended PCC Vivace: Online-Learning Congestion Control

⚠ Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data

Every time a monitor interval (MI) ends:

- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Step 2: Compute RTT Inflation

- RTT inflation = current RTT – baseline RTT
- If RTT is much larger than baseline, the buffer is filling up → likely congestion.

Step 3: Align Loss & RTT Events by Time

For each RTT event, find the nearest loss event.

Step 4: Calculate Correlation

Use **Pearson correlation** between binary loss and RTT arrays:

$$\rho = \frac{\sum (x - \mu_x)(y - \mu_y)}{\sigma_x \sigma_y}$$

Where:

$\rho = 1$: perfectly correlated (congestion)

$\rho = 0$: uncorrelated (wireless)

$\rho = -1$: inversely correlated (rare)

Step 5: Classify Based on Correlation

Set thresholds:

- High threshold = 0.5 → Congestion
- Low threshold = 0.2 → Wireless
- Between = Mixed

$$\begin{cases} \rho > 0.5 & \Rightarrow \textit{Congestion (p\_wireless = 0.1)} \\ \rho < 0.2 & \Rightarrow \textit{Wireless (p\_wireless = 0.9)} \\ \textit{else} & \Rightarrow \textit{Mixed (p\_wireless} \approx \textit{0.5)} \end{cases}$$

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Approach: Correlation-based classification (Pearson coefficient)

Step 1: Collect Raw Data
Every time a monitor interval (MI) ends:
- Log packet losses (loss_rate)
- Log RTT samples (rtt)

Step 2: Compute RTT Inflation
- RTT inflation = current RTT – baseline RTT
- If RTT is much larger than baseline, the buffer is filling up → likely congestion.

Step 3: Align Loss & RTT Events by Time
For each RTT event, find the nearest loss event.

Step 4: Calculate Correlation
Use **Pearson correlation** between binary loss and RTT arrays:

$$\rho = \frac{\sum (x-\mu_x)(y-\mu_y)}{\sigma_x \sigma_y}$$

Where:
$\rho = 1$: perfectly correlated (congestion)
$\rho = 0$: uncorrelated (wireless)
$\rho = -1$: inversely correlated (rare)

Step 5: Classify Based on Correlation
Set thresholds:
- High threshold = 0.5 → Congestion
- Low threshold = 0.2 → Wireless
- Between = Mixed

$$\begin{cases} \rho > 0.5 & \Rightarrow \textit{Congestion (p\_wireless = 0.1)} \\ \rho < 0.2 & \Rightarrow \textit{Wireless (p\_wireless = 0.9)} \\ \textit{else} & \Rightarrow \textit{Mixed (p\_wireless} \approx \textit{0.5)} \end{cases}$$

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Component 2: Adaptive Loss Coefficient - Adjust the Penalty

In the utility function, the **loss penalty (λ)** should depend on the **loss type**.

- **Traditional PCC Utility**

$$U = T - \lambda \cdot L$$

T = Throughput
L = Loss rate
$\lambda = 10.0$ (fixed)
Problem: λ is constant, so the controller overreacts even to wireless losses.

- **Extension 2 Utility**

$$U = T - \lambda_{adaptive} \cdot L$$

Where:

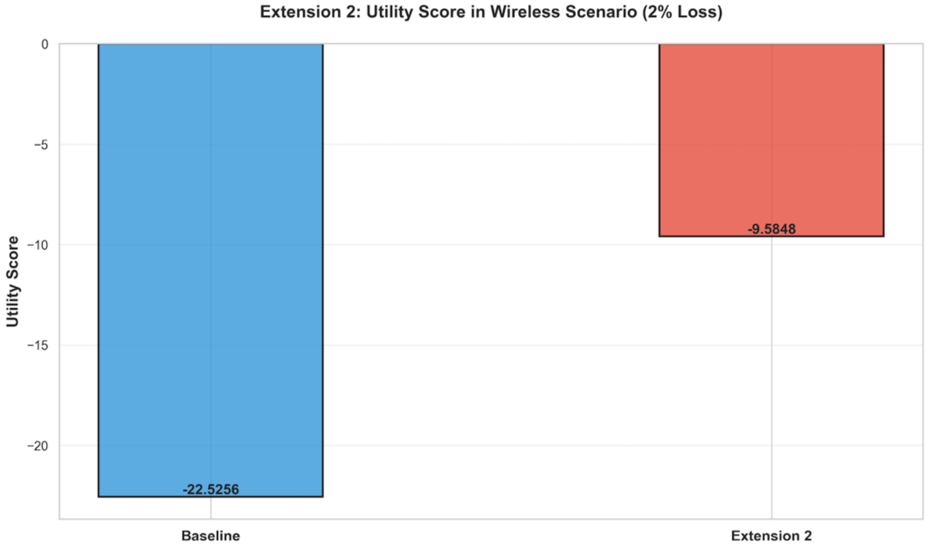
$$\lambda_{adaptive} = \lambda_{base}(1 - p_{wireless}) + \lambda_{wireless}(p_{wireless})$$

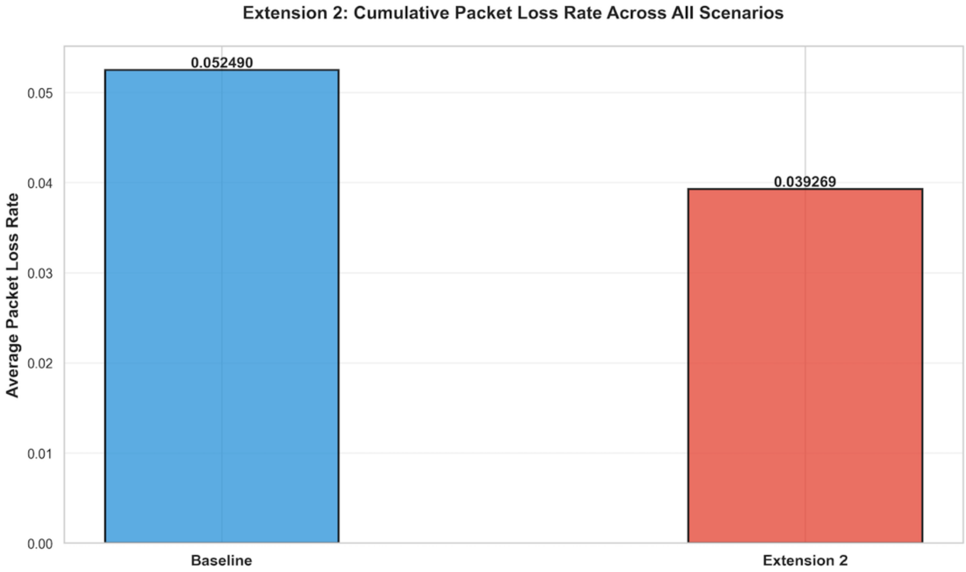
Parameters:
$\lambda_{base} = 10.0$ →congestion penalty
$\lambda_{wireless} = 2.0$ →wireless penalty
$p_{wireless}$ →detected wireless fraction

Extended PCC Vivace: Online-Learning Congestion Control

Extension 2 - Loss-Type Awareness

Results:





Summary (In One Line)

Extension 2 makes PCC Vivace loss-type aware by detecting whether losses stem from congestion or wireless errors and adaptively adjusting the loss penalty, preventing unnecessary rate drops

Extension 3 - Distributed Fairness Control

Baseline Problem: Unfair Bandwidth Sharing

⚠️ Why It Happens - Gradient Oscillation Problem

PCC Vivace adjusts rate via *gradient feedback* (utility changes)

When multiple flows explore simultaneously:

- Their actions interfere
- Gradients oscillate (+, −, +, −)
- No stable convergence

The more aggressive flow keeps increasing → monopolizes capacity Other flows keep backing off → starve

Result: Persistent unfairness & slow convergence.

Extension 3 - Distributed Fairness Control

Baseline Problem: Unfair Bandwidth Sharing

Goal: Achieve *Distributed Fairness*
→ Each flow detects contention, cooperates, and balances bandwidth.

Main Components:

- **Contention Detector** -- Detect if other flows are competing
- **Cooperative Explorer** -- Coordinate exploration (take turns)
- **Virtual Queue Estimator** -- Monitor queue depth from RTT
- **Fairness Controller** -- Penalize greedy flows in utility

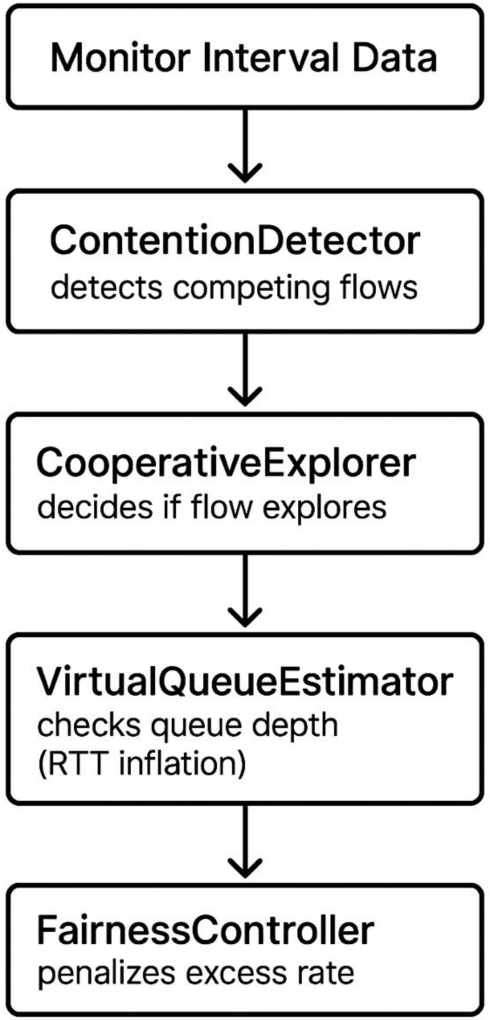
Extension 3 – Distributed Fairness Control

Baseline Problem: Unfair Bandwidth Sharing

Goal: Achieve *Distributed Fairness*
→ Each flow detects contention, cooperates, and balances bandwidth.

Main Components:

- ❖ **Contention Detector** -- Detect if other flows are competing
- ❖ **Cooperative Explorer** -- Coordinate exploration (take turns)
- ❖ **Virtual Queue Estimator** -- Monitor queue depth from RTT
- ❖ **Fairness Controller** -- Penalize greedy flows in utility



Extension 3 – Distributed Fairness Control

❖ **Contention Detector - Detect Competition**

Problem:

Flows don’t know when others are present.

Insight:

Gradient sign pattern reveals contention:

Solo: monotonic (+ + + +)

Competing: oscillating (+ − + −)

Algorithm:

Count gradient sign changes over last 5 intervals

Compute ratio & volatility

Classify as SOLO / LIGHT / MODERATE / HEAVY contention

Example:

Gradients = [+0.5, −0.8, −0.2, +0.5, −0.7]

→ 4 sign changes → **HEAVY contention** (4–5 flows)

Extended PCC Vivace: Online-Learning Congestion Control

Extension 3 – Distributed Fairness Control

❖ **Contention Detector - Detect Competition**

Problem:
Flows don't know when others are present.

Insight:
Gradient sign pattern reveals contention:
Solo: monotonic (+ + + +)
Competing: oscillating (+ − + −)

Algorithm:
Count gradient sign changes over last 5 intervals
Compute ratio & volatility
Classify as SOLO / LIGHT / MODERATE / HEAVY contention

Example:
Gradients = [+0.5, −0.8, −0.2, +0.5, −0.7]
→ 4 sign changes → **HEAVY contention** (4–5 flows)

❖ **Cooperative Explorer — Avoid Collisions**

Problem:
All flows explore together → collisions → oscillations.

Solution:
Hash-based turn-taking (no communication):

```
slot = current_time // cycle
if hash(flow_id, slot) % 2 == 0:
   explore = True
```

Effect:
Only one flow explores per slot
Others wait, then take their turn
Smooth convergence & no interference

Result:
→ Turn-based exploration → stability achieved!

Extension 3 – Distributed Fairness Control

❖ **Virtual Queue Estimator — Detect Monopolization**

Problem:

Dominant flows fill queues without noticing.

Idea:

Estimate queue depth from RTT inflation:

$$Queue\ Delay = RTT_{current} - RTT_{baseline}$$

Baseline RTT = 5th percentile of last samples

Queue fills → larger inflation → congestion detected

Action:

If queue depth > threshold → reduce rate.

→ Prevents buffer monopolization.

Extended PCC Vivace: Online-Learning Congestion Control

Extension 3 – Distributed Fairness Control

❖ **Virtual Queue Estimator — Detect Monopolization**

Problem:

Dominant flows fill queues without noticing.

Idea:

Estimate queue depth from RTT inflation:

$$Queue\ Delay = RTT_{current} - RTT_{baseline}$$

Baseline RTT = 5th percentile of last samples

Queue fills → larger inflation → congestion detected

Action:

If queue depth > threshold → reduce rate.

→ Prevents buffer monopolization.

❖ **FairnessController — Equalize Sharing**

Problem:

Even if contention detected, flows need fairness enforcement.

Solution:

Add *fairness penalty* to utility function:

r_{fair} =total bandwidth / estimated flow count

Penalize flows sending above their fair share

Leave under-utilizing flows unpenalized

Example:

Flow A: 6.5 Mbps → penalty = $(6.5-5)^2$ = 2.25

Flow B: 3.5 Mbps → penalty = 0

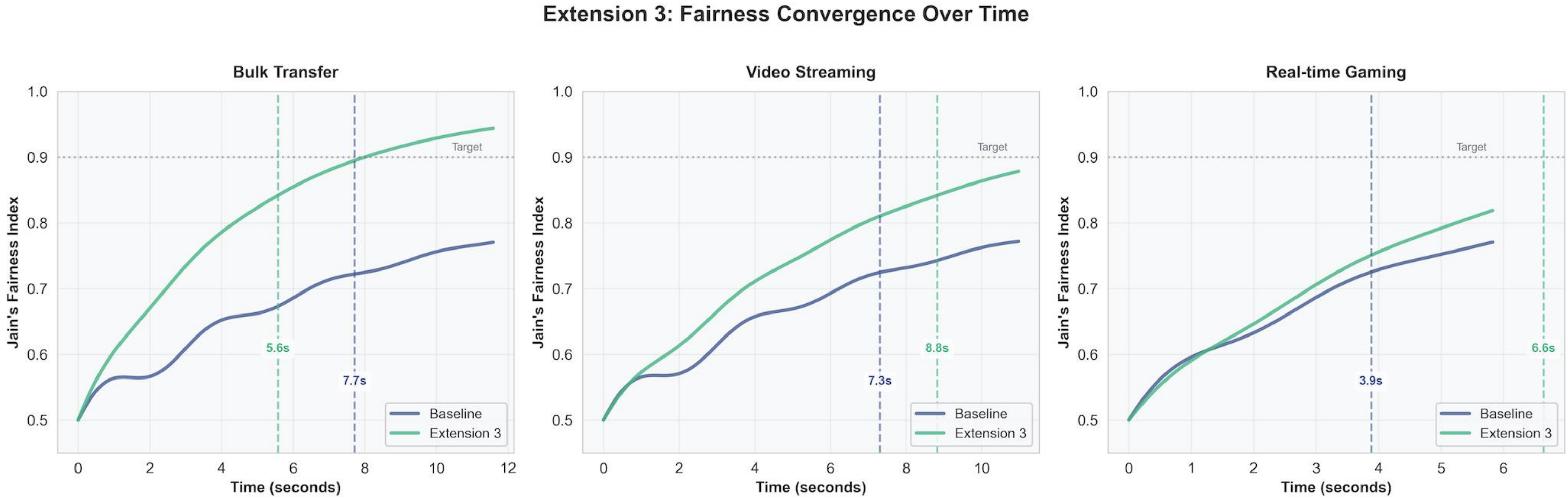
→ A's utility ↓ → reduces rate; B's utility ↑ → increases rate

→ Fair equilibrium around 5 Mbps each.

Extended PCC Vivace: Online-Learning Congestion Control

Extension 3 – Distributed Fairness Control

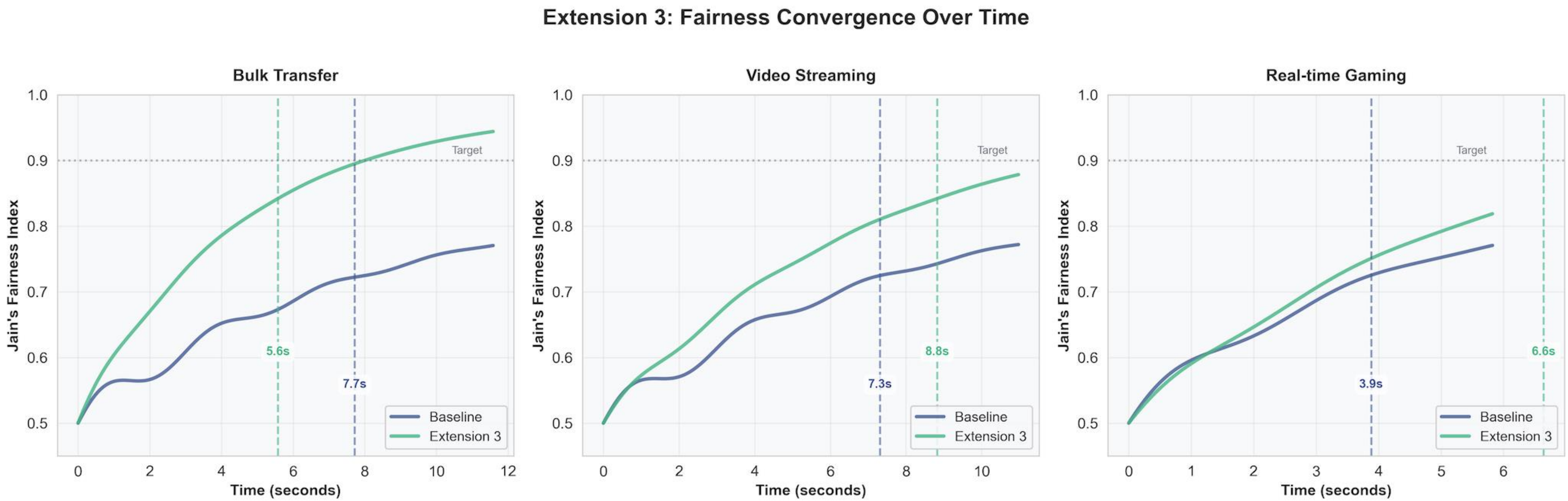
Results



Extended PCC Vivace: Online-Learning Congestion Control

Extension 3 – Distributed Fairness Control

Results



Summary (In One Line)

Extension 3 ensures distributed fairness by detecting competing flows, coordinating exploration, and enforcing fair rate allocation without any centralized coordination.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

🟥 **Problem (Compared to Extension 3)**

Extension 3: great fairness & fast convergence.

- But still **single-path routing** → all traffic on one route.
- If that path:
  - ➢ becomes congested → throughput drops.
  - ➢ fails → complete interruption.
- No multi-path utilization → poor flexibility, under-used capacity.
- System lacks resilience to link failures or congestion.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

🟥 **Problem (Compared to Extension 3)**

Extension 3: great fairness & fast convergence.

- But still **single-path routing** → all traffic on one route.
- If that path:
  - becomes congested → throughput drops.
  - fails → complete interruption.
- No multi-path utilization → poor flexibility, under-used capacity.
- System lacks resilience to link failures or congestion.

Core Idea - Path-Based Load Balancing
Split and adapt traffic across multiple network paths dynamically.
Key Concepts:
- **Path Distribution:** use multiple paths instead of one.
- **Dynamic Rebalancing:** shift load as conditions change.
- **Path Recovery:** reroute instantly if a path fails.
- **Better Utilization:** use every available route efficiently.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

Problem (Compared to Extension 3)

Extension 3: great fairness & fast convergence.

- But still **single-path routing** → all traffic on one route.
- If that path:
  - becomes congested → throughput drops.
  - fails → complete interruption.
- No multi-path utilization → poor flexibility, under-used capacity.
- System lacks resilience to link failures or congestion.

Core Idea - Path-Based Load Balancing

Split and adapt traffic across multiple network paths dynamically.

Key Concepts:

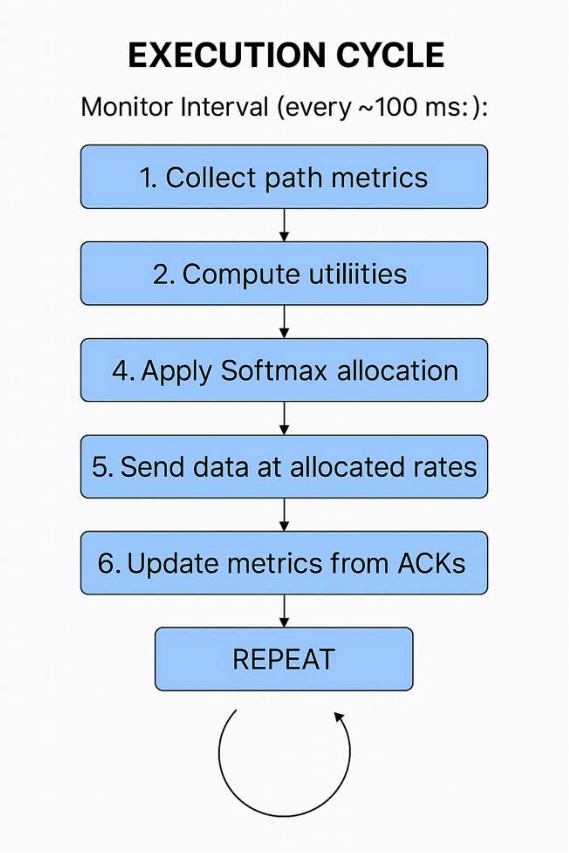
- **Path Distribution:** use multiple paths instead of one.
- **Dynamic Rebalancing:** shift load as conditions change.
- **Path Recovery:** reroute instantly if a path fails.
- **Better Utilization:** use every available route efficiently.

Goal

Enable **multi-path routing** to increase throughput, use spare capacity, and survive path failures.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation



Core Idea - Path-Based Load Balancing
Split and adapt traffic across multiple network paths dynamically.

Key Concepts:

- **Path Distribution:** use multiple paths instead of one.
- **Dynamic Rebalancing:** shift load as conditions change.
- **Path Recovery:** reroute instantly if a path fails.
- **Better Utilization:** use every available route efficiently.

Goal
Enable **multi-path routing** to increase throughput, use spare capacity, and survive path failures.

Multipath Rate Allocation

IMPLEMENTATION FLOW

- **Path Discovery**

Enumerate available interfaces (e.g., eth0, wlan0, lte0).
Create Path objects with baseline RTT, bandwidth, loss.
Maintain ACTIVE / IDLE / FAILED states.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

IMPLEMENTATION FLOW

❑ **Path Discovery**

Enumerate available interfaces (e.g., eth0, wlan0, lte0).
Create Path objects with baseline RTT, bandwidth, loss.
Maintain ACTIVE / IDLE / FAILED states.

❑ **Path Monitoring**

Continuously measure throughput, RTT, loss.
Apply Exponential Moving Average ($\alpha = 0.3$).
Detect degradation → trigger recovery or switch.

Multipath Rate Allocation

IMPLEMENTATION FLOW

❑ Path Discovery

Enumerate available interfaces (e.g., eth0, wlan0, lte0).
Create Path objects with baseline RTT, bandwidth, loss.
Maintain ACTIVE / IDLE / FAILED states.

❑ Path Monitoring

Continuously measure throughput, RTT, loss.
Apply Exponential Moving Average (α = 0.3).
Detect degradation → trigger recovery or switch.

❑ Utility Calculation

For each path:

$$U_p = 0.4T_p + 0.3(1 - L_p) + 0.2(1 - loss_p) + 0.1stability_p$$

Produces a normalized score (0–1).
High utility = good path quality.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

IMPLEMENTATION FLOW

❑ **Path Discovery**

Enumerate available interfaces (e.g., eth0, wlan0, lte0).
Create Path objects with baseline RTT, bandwidth, loss.
Maintain ACTIVE / IDLE / FAILED states.

❑ **Path Monitoring**

Continuously measure throughput, RTT, loss.
Apply Exponential Moving Average (α = 0.3).
Detect degradation → trigger recovery or switch.

❑ **Utility Calculation**

For each path:

$$U_p = 0.4T_p + 0.3(1 - L_p) + 0.2(1 - loss_p) + 0.1stability_p$$

Produces a normalized score (0–1).
High utility = good path quality.

❑ **Correlation Detection**

Compute RTT correlation ρ between paths.
If ρ > 0.8 → same bottleneck → de-prioritize one.

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

IMPLEMENTATION FLOW

❑ **Path Discovery**

Enumerate available interfaces (e.g., eth0, wlan0, lte0).
Create Path objects with baseline RTT, bandwidth, loss.
Maintain ACTIVE / IDLE / FAILED states.

❑ **Path Monitoring**

Continuously measure throughput, RTT, loss.
Apply Exponential Moving Average (α = 0.3).
Detect degradation → trigger recovery or switch.

❑ **Utility Calculation**

For each path:

$$U_p = 0.4T_p + 0.3(1 - L_p) + 0.2(1 - loss_p) + 0.1stability_p$$

Produces a normalized score (0–1).
High utility = good path quality.

❑ **Correlation Detection**

Compute RTT correlation ρ between paths.
If ρ > 0.8 → same bottleneck → de-prioritize one.

❑ **SOFTMAX RATE ALLOCATION (CORE LOGIC)**

$$r_p = R_{total} \times \frac{e^{U_p/\tau}}{\sum e^{U_{p'}/\tau}}$$

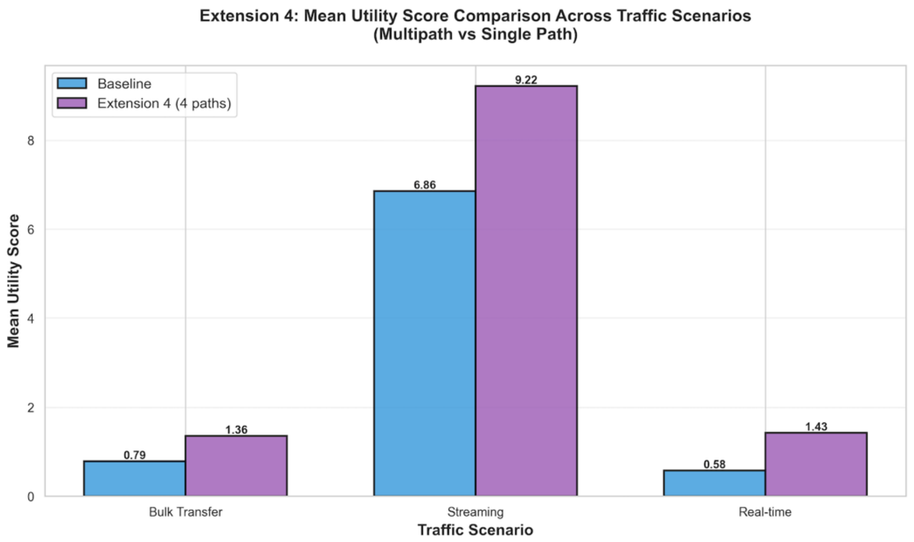
Where:

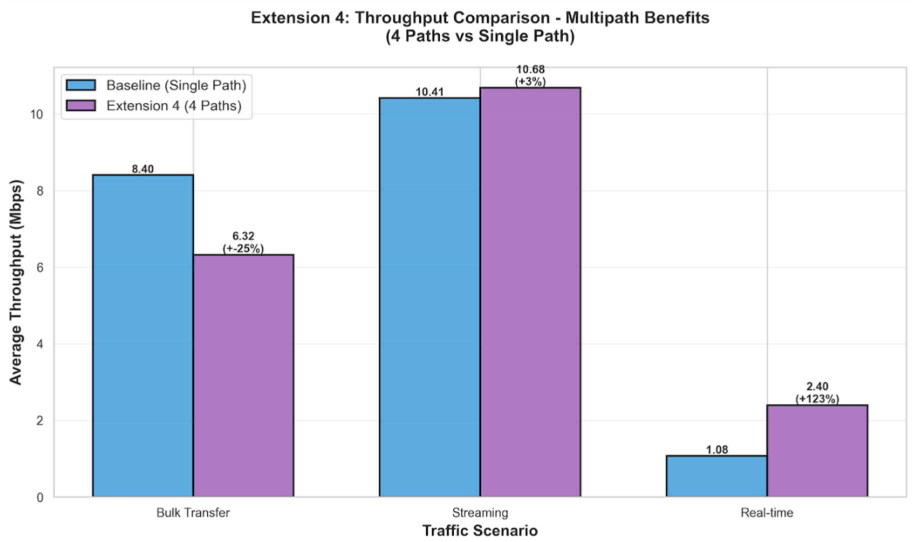
R_{total} :total sending rate
U_p :path utility
τ: temperature (exploration–exploitation balance)

Extended PCC Vivace: Online-Learning Congestion Control

Multipath Rate Allocation

Results:



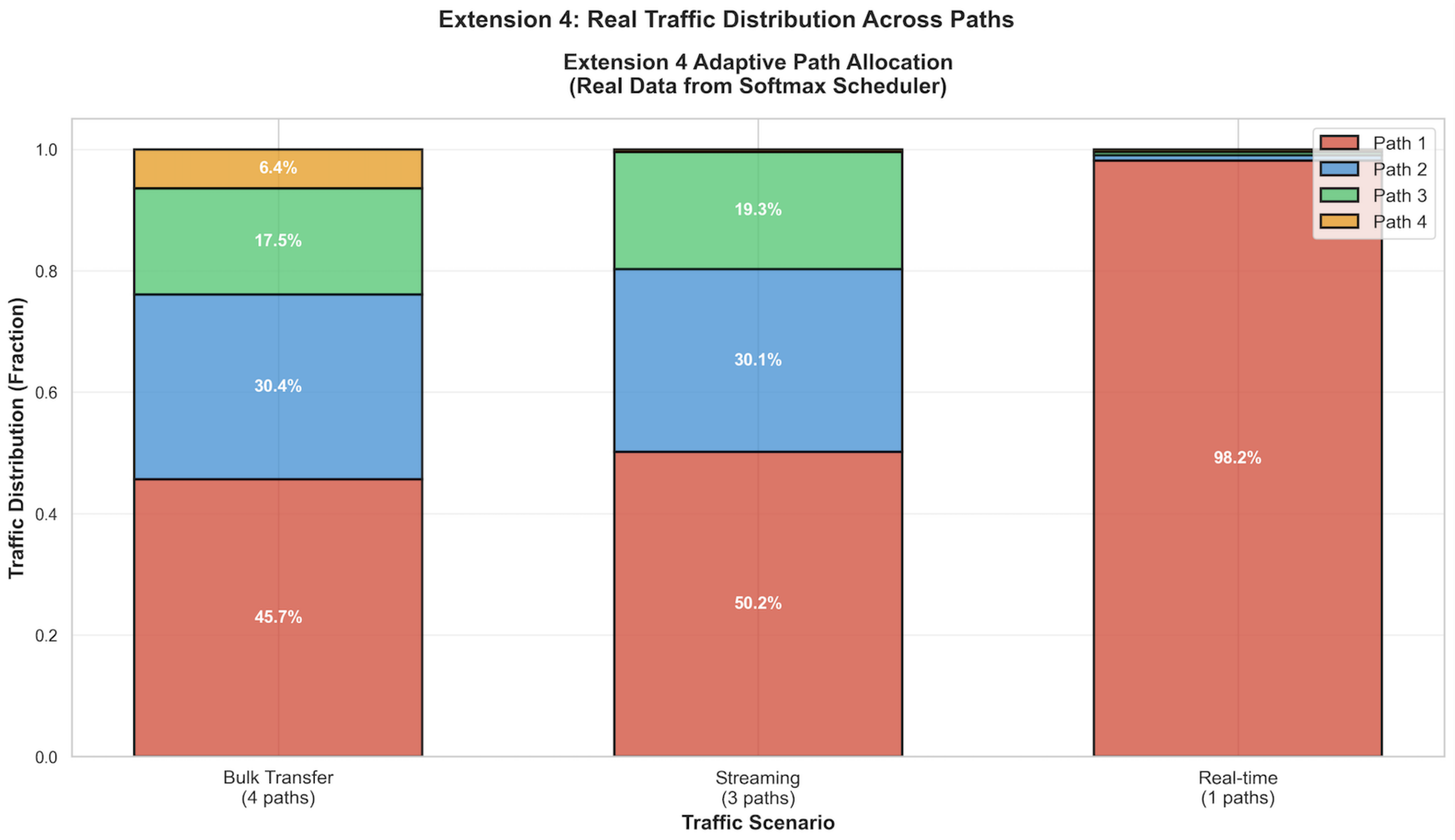


Summary (In One Line)

Extension 4 enables intelligent multipath rate allocation — dynamically distributing traffic across multiple network paths using utility-based Softmax scheduling to maximize throughput, resilience, and adaptability.

Extended PCC Vivace: Online-Learning Congestion Control

Results:



Future Works: PCC Vivace with Multipath Extensions

- ❑ Application-Aware QoE Optimization (Extension of Extension 1)
- ❑ Parameter optimization for Extensions 1-2 (addressing throughput decreases)
- ❑ Security and Privacy for Multipath Transmission
- ❑ Integration testing
- ❑ Cross-platform validation

THANK YOU