Adaptive Condition-Aware Packet Handling in Linux-Based Networks

Amritha S
Dept. of ECE, SENSE
VIT Chennai, India
amritha.s2023@vitstudent.ac.in

Yugeshwaran P
Dept. of ECE, SENSE
VIT Chennai, India
yugeshwaran.p2023@vitstudent.ac.in

Deepti Annuncia
Dept. of ECE, SENSE
VIT Chennai, India
deepti.annucia2023@vitstudent.ac.in

***Abstract*—Linux-based networking systems employ sophisticated packet handling mechanisms such as queueing disciplines and schedulers to manage latency, throughput, and fairness. Although modern Linux kernels support advanced queue management techniques, most mechanisms remain statically configured and react only after congestion occurs. This work presents an adaptive, condition-aware packet handling framework that dynamically adjusts Linux packet handling behavior at runtime based on observed queue and traffic conditions. By leveraging existing traffic control mechanisms without kernel modification, the proposed approach improves latency stability and congestion responsiveness under dynamic traffic scenarios. Experimental results demonstrate the effectiveness of runtime adaptation compared to static configurations, highlighting the benefits of native adaptivity within the Linux networking stack.**

***Index Terms*—Linux networking, packet scheduling, traffic control, queueing disciplines, adaptive systems**

I. Problem Statement

Despite the availability of advanced queueing disciplines and schedulers in modern Linux kernels, packet handling behavior remains largely static and reactive. Fixed parameter configurations are unable to respond effectively to dynamic traffic conditions, leading to suboptimal latency, unstable queue behavior, and unnecessary packet loss under varying load. Existing adaptive solutions either operate at higher layers, require kernel modifications, or rely on specialized programmable infrastructure, limiting their practicality for widespread deployment.

II. Objective

The objective of this work is to design and evaluate a **lightweight, adaptive, condition-aware packet handling framework** that dynamically adjusts Linux packet handling behavior at runtime based on observed queue and traffic conditions, while:

- operating entirely within existing Linux traffic control mechanisms,
- requiring no kernel source modifications,
- and remaining deployable on commodity systems.

III. Proposed Solution

This work proposes a **user-space adaptive control layer** that continuously monitors live Linux kernel queue behavior and dynamically reconfigures packet handling parameters using standard traffic control interfaces. Rather than introducing new schedulers or queueing disciplines, the system adapts existing mechanisms (e.g., fq_codel) in response to real-time conditions. The proposed solution bridges the gap between **static kernel mechanisms** and **dynamic traffic behavior**, enabling runtime-aware packet handling within Linux.

A. Kernel-Level Observability

- Continuous monitoring of Linux queue disciplines using:
  - `tc`
  - `pyroute2`
- Extracted metrics include:
  - packet and byte counts
  - queue backlog
  - drop statistics
  - fq_codel parameters (target, interval, limit)
- Extends monitoring approaches used in fq_codel and Linux kernel studies

B. Traffic-Aware Measurement Pipeline

- Controlled traffic generation using:
  - `iperf3` (TCP and UDP)
  - `ping` (RTT measurement)
- Enables reproducible congestion and stress scenarios.
- Consistent with experimental setups used in fq_codel, FQ-PIE, and SCRR evaluations

C. Condition Detection Layer

- Periodic evaluation of queue and traffic state:
  - backlog growth trends
  - drop rate changes
  - RTT inflation indicators
- Enables early detection of congestion tendencies.
- Existing AQMs react only after threshold violations

D. Adaptive Packet Handling Controller (Primary Novelty)

- A user-space controller dynamically adjusts:
  - qdisc parameters
  - queue limits
  - delay targets
- Uses simple, interpretable control logic (rule-based, not ML).

- Runtime adaptation of existing Linux packet handling mechanisms without modifying kernel code

Example logic:

- Increasing backlog → tighten delay target or queue limit
- Stable conditions → relax parameters to improve throughput

E. Runtime Reconfiguration without Kernel Modification

- Dynamic reconfiguration using: `tc qdisc change dev root fq_codel ...`
- No traffic interruption
- No kernel rebuild or module insertion
- Immediately deployable on real Linux systems

IV. Dataset Description

The dataset used in this work is **original and system-generated**, consisting of:

- real-time kernel telemetry
- collected during controlled experiments
- structured as time-series data

Example format:

`time` packets — bytes — backlog — drops — RTT—

V. Novelty

While prior work focuses on designing new schedulers, AQMs, or programmable data planes, this work demonstrates that meaningful latency and stability improvements can be achieved by **dynamically adapting existing Linux packet handling mechanisms based on real-time conditions**, without kernel modifications or specialized infrastructure.

VI. Literature Survey

The literature related to packet handling in Linux-based networks can be broadly classified into two categories. Section VII presents core works that directly influence packet scheduling and queue management within the Linux kernel. Section VIII discusses supplementary works that motivate adaptive packet handling by exposing limitations of static configurations and exploring alternative architectures. SCRR decides *when* to serve packets; we decide *how the scheduler itself adapts* based on observed conditions.

VII. Core Literature

A. Self-Clocked Round-Robin Packet Scheduling

Authors: Erfan Sharafzadeh et al.
Year: 2025
Venue: USENIX NSDI 2025

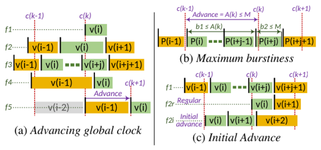

Fig. 1. Virtual time advancement mechanisms in Self-Clocked Round-Robin (SCRR) scheduling

1) Problem Statement: Traditional packet schedulers such as Deficit Round Robin (DRR) assume stable packet size distributions and long-lived backlogged flows. Modern Internet traffic is dominated by short, latency-sensitive and bursty flows, leading to excessive queuing delay and CPU overhead.

2) Objective: To design a scalable packet scheduler that preserves fairness while reducing latency for short flows without manual parameter tuning.

3) Proposed Solution: Self-Clocked Round-Robin (SCRR) dynamically advances virtual time based on packet service rather than fixed quanta. The scheduler is implemented as a Linux queuing discipline.

4) Evaluation Metrics and Results: Metrics include flow completion time, tail latency, CPU overhead, and fairness. SCRR achieves up to 71% latency reduction and approximately 23% lower CPU overhead compared to DRR.

5) Relevance to This Work: SCRR demonstrates the effectiveness of service-driven virtual time advancement in reducing latency for short and bursty flows within the Linux kernel. However, SCRR remains a fixed scheduling policy and does not adapt its behavior based on real-time congestion signals or system-level conditions. This limitation motivates our work on **adaptive, condition-aware packet handling mechanisms that dynamically adjust scheduling behavior at runtime while operating natively within the Linux packet processing path**.

B. P4TC: Programmable Packet Handling in the Linux Kernel

Authors: P4TC Development Team
Year: 2023
Venue: ACM CoNEXT 2023

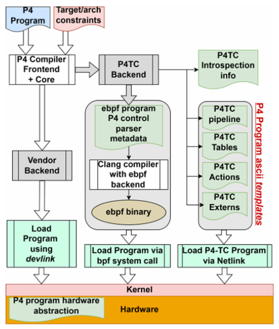
P4 Program
Target/arch constraints
P4 Compiler Frontend + Core
P4TC Backend
P4TC Introspection info
ebpf program P4 control parser metadata
Clang compiler with ebpf backend
ebpf binary
Vendor Backend
P4TC pipeline
P4TC Tables
P4TC Actions
P4TC Externs
P4 Program ascii templates
Load Program using devlink
Load Program via bpf system call
Load P4-TC Program via Netlink
Kernel
P4 program hardware abstraction
Hardware

Fig. 2. P4TC compilation and execution pipeline showing integration of P4 programs with Linux traffic control via eBPF

1) Problem Statement: Linux packet processing pipelines are rigid and difficult to extend without kernel modification.

2) Objective: To enable programmable packet processing directly within the Linux kernel.

3) Proposed Solution: P4TC integrates P4 with Linux traffic control using eBPF, enabling programmable parsing and actions at tc hooks.

4) Evaluation Metrics and Results: Evaluated using throughput, packet processing latency, and kernel overhead.

5) Relevance to This Work: P4TC provides programmability but not autonomous adaptivity.

C. fq_codel: Fair Queueing with Controlled Delay

Authors: Eric Dumazet, Kathleen Nichols, Van Jacobson
Year: 2014
Venue: ACM Queue / Linux Kernel

```
                  +----------------+
                  | root qdisc HTB |
                  +----------------+
                          |
                     +---------+
                     | class 1 |
                     +---------+
                          |
                   +------------+
                   | qdisc PRIO |
                   +------------+
                 /        |        \
+-------------+ +-------------+ +-------------+
| class prio1 | | class prio2 | | class prio3 |
+-------------+ +-------------+ +-------------+
       |               |               |
 +-----------+   +-----------+   +-----------+
 | qdisc SFQ |   | qdisc SFQ |   | qdisc SFQ |
 +-----------+   +-----------+   +-----------+
```

Fig. 3. Hierarchical Linux traffic control configuration illustrating fq_codel-style per-flow queueing under class-based schedulers

1) Problem Statement: Bufferbloat results in excessive latency under congestion.

2) Proposed Solution: fq_codel combines per-flow fair queueing with delay-based packet dropping.

3) Relevance to This Work: fq_codel effectively mitigates bufferbloat through per-flow queueing and delay-based packet drops, but its response is inherently reactive and driven by static delay thresholds. This motivates the need for proactive and adaptive scheduling mechanisms that can manage latency before queues build up.

D. FQ-PIE Queue Discipline

Authors: Gautam Ramakrishnan et al.
Year: 2019
Venue: IEEE LCN Symposium

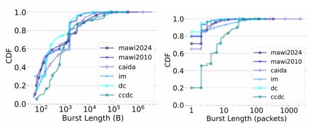

Fig. 4. CDF of traffic burst lengths in bytes and packets across real-world traces, illustrating bursty flow behavior targeted by FQ-PIE

1) Contribution: Combines PIE with per-flow queueing to improve fairness.

2) Relevance to This Work: FQ-PIE improves fairness under bursty traffic by combining per-flow queueing with PIE's delay-based control, but it remains statically configured and reactive to observed congestion. This motivates adaptive schedulers that can proactively adjust behavior based on runtime traffic dynamics.

E. BBR Congestion Control

Authors: Neal Cardwell et al.
Year: 2016
Venue: ACM Queue

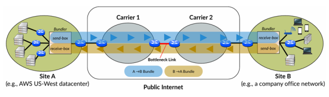
Carrier 1
Carrier 2
Bottleneck Link
Site A
(e.g., AWS US-West datacenter)
Public Internet
Site B
(e.g., a company office network)

Fig. 5. End-to-end path model used by BBR showing bottleneck bandwidth and round-trip propagation delay across the public Internet

1) Contribution: Model-based congestion control operating at the transport layer.

2) Relevance: Operates above packet scheduling layer. BBR demonstrates the benefits of proactive, model-based control at the transport layer, but it operates above the packet scheduling layer and does not influence in-network queuing or per-flow scheduling decisions. This motivates complementary mechanisms at the scheduler and data-plane level for fine-grained latency control.

VIII. Supplementary Literature

A. MoonEm: High-Precision Network Emulation

Authors: Sung Bhin Oh et al.
Year: 2025
Venue: ACM CoNEXT

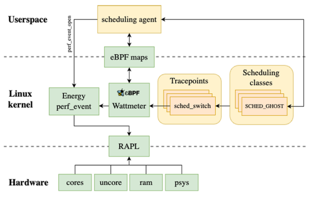

Fig. 6. MoonEm architecture and experimental setup illustrating DPDK-based packet timing control compared to Linux tc and NetEm

1) Problem: Linux traffic control and netem introduce timing inaccuracies and packet burstiness during emulation.

2) Method: MoonEm bypasses the Linux networking stack using DPDK to precisely control packet timing.

3) Results: Demonstrates significantly lower timing error and reduced burst artifacts compared to Linux tc-based emulation.

4) Relation to This Work: MoonEm reveals fundamental timing and burstiness limitations in Linux tc-based packet handling under load, motivating improvements within the kernel packet scheduling and queuing framework rather than bypassing it via userspace fast paths.

B. INT-Assisted Adaptive Packet Scheduling

Authors: Zichen Xu et al.
Year: 2025
Venue: IEEE INFOCOM

1) Problem: Static scheduling fails to guarantee end-to-end latency under dynamic traffic.

2) Method: Uses in-band network telemetry (INT) to embed real-time latency feedback into packets and adapt scheduling decisions in programmable switches.

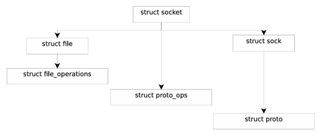

Fig. 7. INT-assisted adaptive packet scheduling using real-time latency feedback embedded within packets

3) Results: Improves end-to-end latency and reduces variance for delay-sensitive flows.

4) Relation to This Work: This work demonstrates the effectiveness of condition-aware scheduling using real-time telemetry in programmable networks, motivating similar adaptivity within Linux packet scheduling without requiring programmable switch support.

C. FIPO: Programmable Packet Scheduling Framework

Authors: Shang Liu et al.
Year: 2025
Venue: IEEE Systems Journal

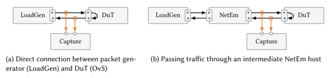
LoadGen
DuT
Capture
(a) Direct connection between packet generator (LoadGen) and DuT (OvS)
LoadGen
NetEm
DuT
Capture
(b) Passing traffic through an intermediate NetEm host

Fig. 8. Architecture of FIPO enabling programmable and extensible packet scheduling primitives

1) Problem: Existing schedulers lack flexibility for rapid experimentation.

2) Method: Introduces a lightweight, software-defined scheduling primitive for customized scheduling logic.

3) Results: Enables fast deployment of new schedulers with modest CPU overhead.

4) Relation to This Work: FIPO highlights the importance of flexible and programmable scheduling abstractions, motivating designs that combine programmability with runtime adaptivity and low overhead in Linux packet scheduling.

D. OpenSched: Centralized Packet Scheduling

Authors: Tian Pan et al.
Year: 2017
Venue: ACM/IEEE ANCS

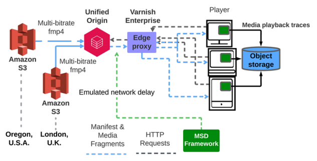

Fig. 9. OpenSched architecture illustrating centralized packet scheduling using SDN control

1) Problem: Distributed QoS mechanisms lack global visibility.

2) Method: Introduces centralized control over packet scheduling using SDN principles.

3) Results: Improves QoS control but relies on centralized infrastructure.

4) Relation to This Work: OpenSched contrasts with decentralized Linux-based approaches by relying on centralized control and global visibility, motivating adaptive packet scheduling mechanisms that operate locally within the Linux kernel.

IX. Research Gap

A comprehensive review of existing packet scheduling, queue management, and programmable networking literature—including fq_codel, FQ-PIE, SCRR, BBR, P4TC, OpenSched, MoonEm, and INT-assisted scheduling—reveals a consistent limitation: **Linux packet handling mechanisms are predominantly statically configured and reactive in nature**. Specifically:

- Active Queue Management (AQM) mechanisms such as fq_codel and FQ-PIE react only after congestion indicators (e.g., delay thresholds or drops) are exceeded.
- Scheduler designs such as SCRR improve fairness and latency but operate as fixed policies without runtime adaptivity.
- Transport-layer solutions like BBR adapt sending rates but remain decoupled from kernel queue behavior.
- Programmable frameworks (e.g., P4TC, OpenSched) either require specialized infrastructure or external control planes.
- Kernel-bypass approaches (e.g., MoonEm) improve precision but sacrifice realism and deployability.

Thus, a gap exists for a lightweight, deployable mechanism that enables runtime-adaptive, condition-aware packet handling natively within the Linux networking stack, without kernel modification or specialized hardware.