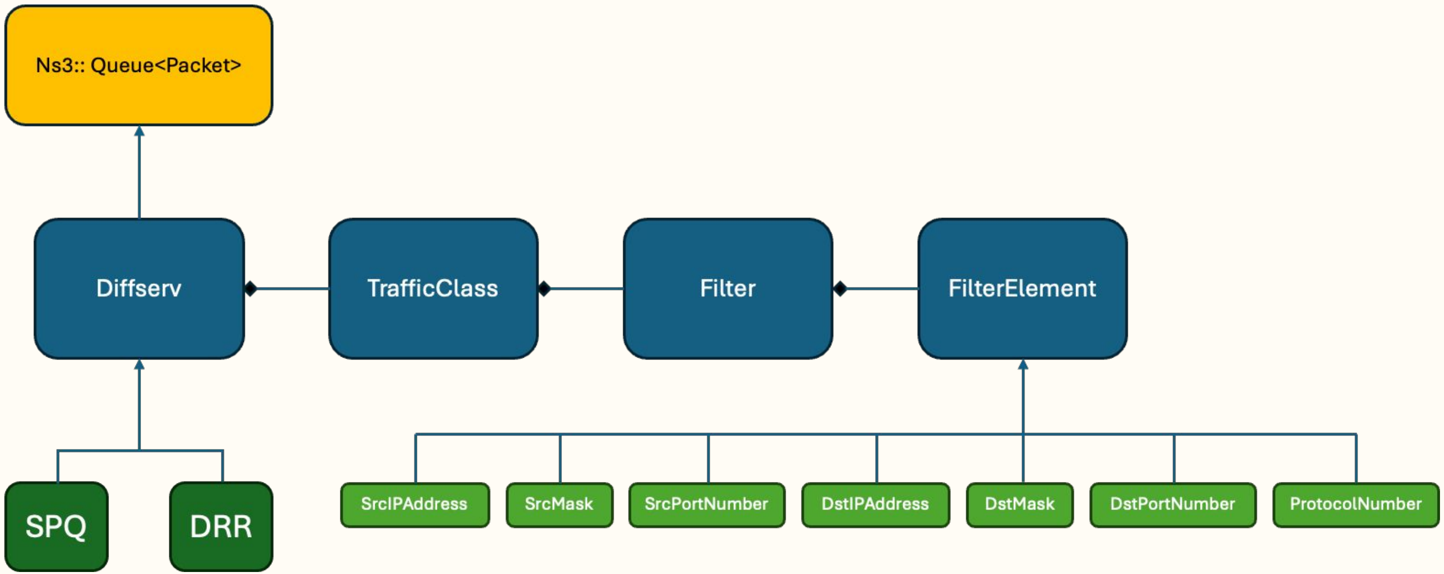
Extending DiffServ in NS-3: SPQ and DRR Implementation

Jinglin Zhou, Shijie Xu

Project Design



DiffServ
Base Class Design

DiffServ extends ns3::Queue<Packet> and acts as a reusable QoS base class.

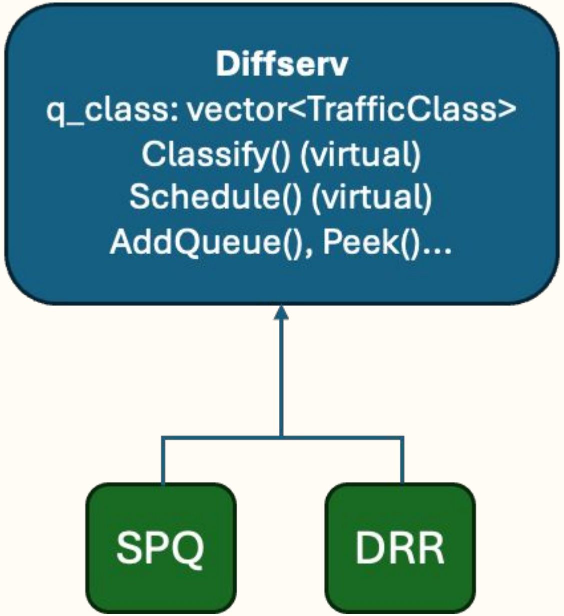
It stores multiple TrafficClass queues in q_class.

Provides key virtual methods:

- Classify(Packet) → Assigns packet to matching queue.
- Schedule() → Selects queue to serve.

Implements enqueue/dequeue logic via DoEnqueue(), DoDequeue(), etc.

New queues can be dynamically added with AddQueue().



SPQ Implementation

`SPQ` is a subclass of `DiffServ` with a simple priority-based scheduler.

`Schedule()` iterates through non-empty queues and selects the one with the highest priority value.

`Classify()` assigns packets to queues based on filter matches.

Uses destination port to differentiate traffic (e.g., port 9000 = high priority).

DRR Implementation

DRR also derives from DiffServ and ensures fair bandwidth sharing.

Each queue has a weight (quantum) and deficit counter.

Schedule() selects a queue only if its deficit ≥ packet size.

Dequeue() updates the deficit after a packet is removed.

Classify and filters work the same as in SPQ.

```
For each round:
  deficit[i] += quantum[i]
  if deficit[i] >= packet_size:
    serve packet
    deficit[i] -= packet_size
```

Design Challenges & Limitations

Implementing the DRR Schedule() method was challenging due to the need to fairly allocate bandwidth using deficit counters while managing complex round-robin traversal across multiple queues.

We had to carefully handle edge cases like empty queues and insufficient deficits, avoid infinite loops, and maintain correct state across iterations.

Integration with ns-3's object model and ensuring accurate logging for debugging further increased implementation complexity.

Design Challenges & Limitations

Limited Filtering Flexibility: Filters currently support only basic header fields. Supporting DSCP or custom application-level criteria would improve classification precision.

Small-Scale Simulation: Simulations were conducted on a minimal 3-node topology. Larger-scale networks could reveal hidden bottlenecks or fairness issues.

Future Work

- Add minimum bandwidth guarantees in SPQ to prevent starvation.
- Optimize DRR’s deficit management logic to reduce complexity and improve maintainability.

Recommendation to improve Diffserv

Modular Scheduling: Introduce an IScheduler interface to abstract the scheduling strategy. DiffServ can then accept any scheduler object (SPQ, DRR, WFQ), enabling runtime selection without subclassing.

Enhanced Filters: Extend FilterElement to support classification based on DSCP, QoS markings, or protocol-specific flags for more realistic traffic differentiation.

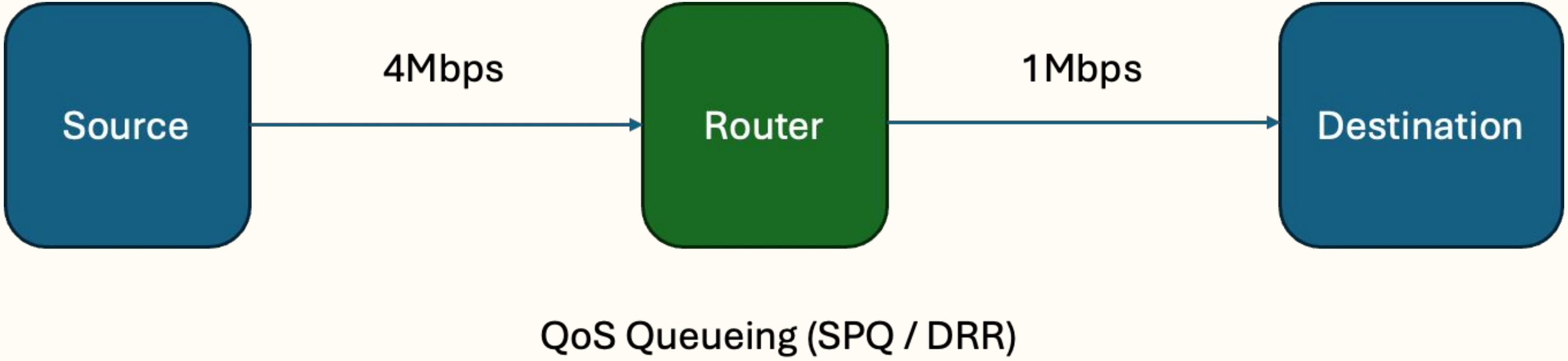
Recommendation to improve Diffserv

Add a PeekSchedule() Method: Separate peek-only scheduling from state-updating Schedule(). This would improve testing reliability and avoid unintended state changes when probing the next queue.

Built-in Statistics Collection: Add queue-level metrics (latency, drop count, dequeue count) within TrafficClass or DiffServ to support debugging and performance reporting.

Network Topology

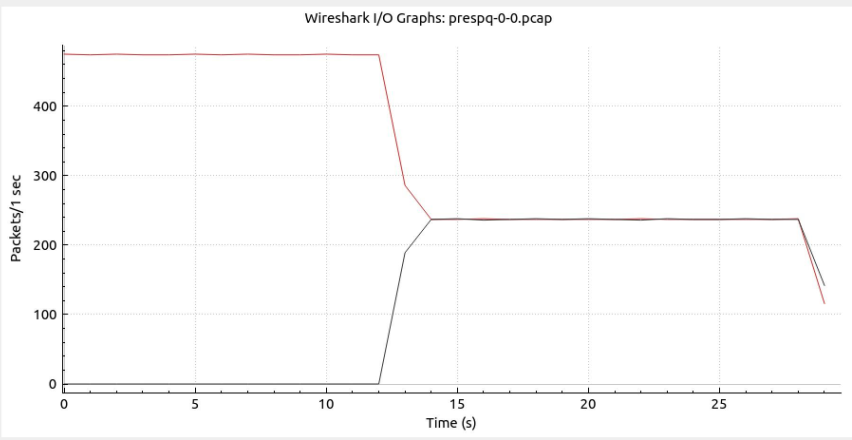
This simulation uses a simple 3-node topology, where packets flow from a Source node to a Destination node through a Router equipped with a QoS scheduler (SPQ or DRR).



SPQ Validation

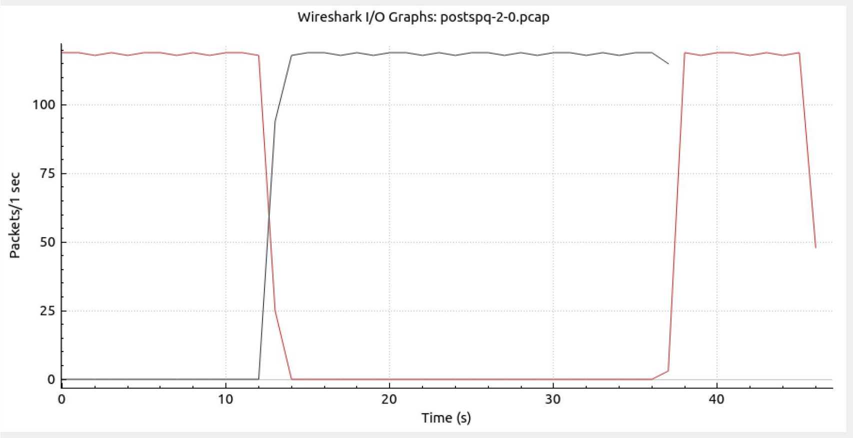
Pre

Before SPQ is enabled, both low- and high-priority traffic share the bandwidth equally once both flows are active.



Post

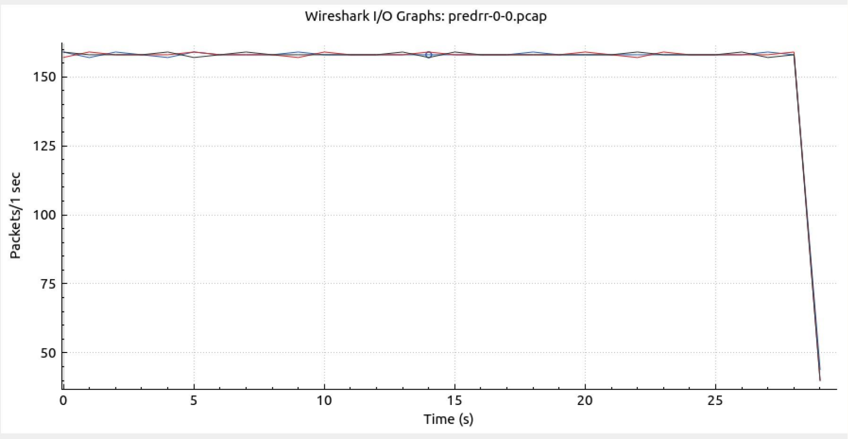
After SPQ is enabled, the high-priority traffic completely dominates bandwidth when active, and low-priority traffic resumes only when high-priority flow stops.



DRR Validation

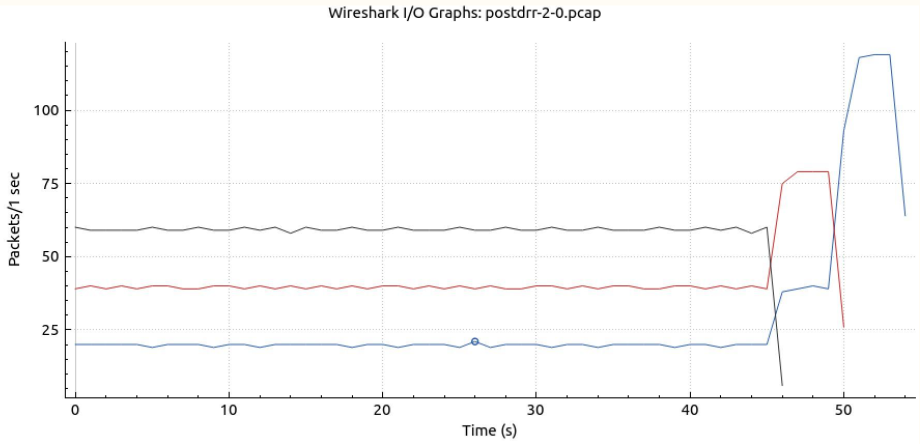
Pre

Before DRR is applied, all traffic flows share the bandwidth equally regardless of their assigned weights.



Post

After DRR is enabled, bandwidth is distributed proportionally based on queue weights, demonstrating a clear 3:2:1 packet ratio between flows.



Thank you