Muğla Sıtkı Koçman University

Software Engineering Department

PROJECT PROPOSAL

Dynamic, Persona-Aware QoS Management for OpenWrt

Enhancing Home Network Performance through Adaptive Quality of Service

Student:	Barış Can Ataklı
Student ID:	210717014
Advisor:	Cihat Çetinkaya
Date:	September 2025

Muğla, Türkiye

2025

Abstract

This proposal presents a comprehensive research and development project aimed at extending OpenWrt's Quality of Service (QoS) capabilities with a novel framework: **dynamic, persona-aware, and service-aware QoS management**. Building upon the strengths of CAKE/fq_codel and existing Smart Queue Management (SQM), the project will design and implement a new daemon (`qosd`) capable of adaptive decision-making, device-role assignment (personas), and real-time policy adjustment.

The system addresses critical limitations in current home networking solutions: static configuration, lack of application awareness, and poor user interfaces. By introducing persona-based device classification (gaming, streaming, VoIP, IoT) and adaptive bandwidth management, the proposed solution aims to reduce latency for real-time applications while maintaining fairness for bulk transfers.

A modern, responsive dashboard will be developed to improve user experience, supporting both a lightweight LuCI application for resource-constrained devices and an optional standalone web interface with advanced monitoring capabilities. The final deliverable will be a production-ready, open-source package submitted to the OpenWrt community, with demonstrated improvements in network performance metrics and user satisfaction.

Keywords: OpenWrt, Quality of Service, CAKE, Traffic Classification, Adaptive Networking, Home Router, Bufferbloat, Network Management

Contents

1 Introduction and Motivation **5**
1.1 Background . . . 5
1.2 Problem Statement . . . 6
1.3 Research Motivation . . . 6

2 Literature Review and Related Work **7**
2.1 Quality of Service Foundations . . . 7
2.2 Traffic Classification Techniques . . . 7
2.3 Adaptive QoS Systems . . . 8
2.4 OpenWrt QoS Ecosystem . . . 8

3 Problem Definition and Requirements Analysis **9**
3.1 Detailed Problem Analysis . . . 9
3.1.1 Technical Limitations . . . 9
3.1.2 User Experience Issues . . . 9
3.2 Requirements Specification . . . 10
3.2.1 Functional Requirements . . . 10
3.2.2 Non-Functional Requirements . . . 10
3.2.3 Hardware Target Specifications . . . 11

4 Proposed Solution Architecture **12**
4.1 System Overview . . . 12
4.2 Core QoS Daemon (qosd) . . . 13
4.2.1 Architecture Design . . . 13
4.2.2 Configuration Schema . . . 13
4.2.3 ubus API Design . . . 14
4.3 Classification Engine . . . 14
4.3.1 Multi-Layer Classification . . . 14
4.3.2 Service Categories . . . 15
4.4 Policy Engine . . . 15

4.4.1 Adaptive Decision Making 15
4.4.2 Policy Templates 16
4.4.3 Hysteresis and Stability 16
4.5 Management Interface 17
4.5.1 LuCI Integration 17
4.5.2 Standalone Dashboard 17

5 Implementation Methodology 18
5.1 Development Approach 18
5.1.1 Phase 1: Minimum Viable Product (Months 1-2) 18
5.1.2 Phase 2: Beta Release (Months 3-4) 19
5.1.3 Phase 3: Release Candidate (Months 5-6) 19
5.2 Development Tools and Environment 20
5.2.1 Build Environment 20
5.2.2 Testing Infrastructure 20
5.3 Quality Assurance 20
5.3.1 Code Quality Standards 20
5.3.2 Testing Strategy 20

6 Evaluation Plan 22
6.1 Performance Metrics 22
6.1.1 Technical Performance Indicators 22
6.1.2 User Experience Metrics 23
6.2 Experimental Design 23
6.2.1 Test Scenarios 23
6.2.2 Baseline Comparisons 24
6.2.3 Data Collection and Analysis 24

7 Project Plan and Timeline 25
7.1 Work Breakdown and Timeline 25
7.2 Risk Management 27

8 Conclusion and Future Work 28
8.1 Conclusion 28
8.2 Future Work 29

List of Figures

4.1 System Architecture Overview 12

List of Tables

3.1 Xiaomi AX3000T Specifications and Requirements 11

4.1 Service Categories and QoS Requirements 15

7.1 Project Timeline and Milestones 26
7.2 Risk Analysis and Mitigation Plan 27

Chapter 1

Introduction and Motivation

1.1 Background

Modern home and small office networks have evolved into complex ecosystems hosting an unprecedented diversity of connected devices. A typical household network now includes traditional computing devices (PCs, laptops, tablets), entertainment systems (smart TVs, gaming consoles, streaming devices), communication tools (VoIP phones, video conferencing systems), and an expanding array of Internet of Things (IoT) devices (smart speakers, security cameras, home automation systems).

This heterogeneity creates highly variable and often conflicting traffic patterns that challenge traditional network management approaches. Each device type generates traffic with distinct characteristics:

- **Gaming devices** require ultra-low latency ($<$ 50ms RTT) and minimal jitter for competitive online gaming
- **Video streaming** services demand consistent bandwidth but can tolerate moderate latency due to buffering
- **VoIP and video conferencing** need balanced low-latency and moderate bandwidth with strict jitter requirements
- **File downloads and cloud backups** utilize maximum available bandwidth but are latency-tolerant
- **IoT devices** typically generate low-bandwidth, periodic traffic with varying latency sensitivities

1.2 Problem Statement

Current OpenWrt QoS implementations, while functional, exhibit several critical limitations that prevent optimal network performance in modern heterogeneous environments:

1. **Static Configuration Paradigm:** Existing SQM implementations operate with fixed bandwidth ceilings and static priority assignments, unable to adapt to changing network conditions or traffic patterns.
2. **Limited Application Awareness:** Current solutions cannot effectively differentiate between traffic types, leading to suboptimal resource allocation where latency-sensitive applications compete with bulk transfers.
3. **Inadequate User Interface:** The existing LuCI-based interface provides only basic configuration forms with minimal real-time visibility into network performance and traffic patterns.
4. **Lack of Device Context:** No mechanism exists for assigning contextual roles or "personas" to network devices, preventing intelligent, use-case-aware traffic management.
5. **Poor Adaptability:** Systems cannot dynamically respond to congestion events, changing user priorities, or varying application demands.

These limitations result in suboptimal user experiences: online gaming sessions suffer from lag spikes during concurrent video streaming, VoIP calls experience quality degradation when file transfers are active, and users lack visibility into network performance issues.

1.3 Research Motivation

The motivation for this research stems from the gap between the sophisticated traffic control capabilities available in modern Linux kernels (particularly CAKE and fq_codel) and the simplified, static management interfaces typically available to end users. While the underlying technology can provide excellent QoS when properly configured, the complexity and inflexibility of current management tools prevent most users from realizing these benefits.

Furthermore, the increasing prevalence of encrypted traffic (HTTPS, QUIC, VPNs) makes traditional port-based classification less effective, necessitating more sophisticated classification approaches that consider flow characteristics, temporal patterns, and contextual information.

Chapter 2

Literature Review and Related Work

2.1 Quality of Service Foundations

Quality of Service in IP networks has been extensively studied, with foundational work by Blake et al. [1] establishing the Differentiated Services (DiffServ) architecture that forms the basis for modern QoS implementations. The evolution from IntServ's per-flow reservations to DiffServ's traffic class-based approach enables scalable QoS deployment in edge networks.

Recent research has focused on addressing bufferbloat, a phenomenon where excessive buffering in network devices causes increased latency. Nichols and Jacobson's CoDel algorithm [2] and its fair queuing variant fq_codel provide active queue management that maintains low latency while preserving throughput. The CAKE (Common Applications Kept Enhanced) algorithm extends these concepts with integrated shaping and enhanced fairness mechanisms.

2.2 Traffic Classification Techniques

Traditional traffic classification relies heavily on port numbers and protocol identifiers. However, the increasing use of encryption and port randomization has reduced the effectiveness of these approaches. Moore and Zuev [3] demonstrated that statistical flow features could provide robust classification even for encrypted traffic.

Machine learning approaches to traffic classification have shown promise, with Nguyen and Armitage [4] providing a comprehensive survey of techniques. However, the computational requirements of these approaches often exceed the capabilities of typical home router hardware.

2.3 Adaptive QoS Systems

Commercial implementations of adaptive QoS exist in enterprise and service provider environments. Cisco's Application Visibility and Control (AVC) and Qualcomm's StreamBoost technology demonstrate the feasibility of dynamic, application-aware QoS. However, these solutions are proprietary and not available for open-source router firmware.

Research by Chen et al. [5] explored reinforcement learning for adaptive QoS, showing potential for automated policy adjustment. However, the complexity and computational requirements of these approaches have limited their practical deployment in resource-constrained environments.

2.4 OpenWrt QoS Ecosystem

OpenWrt's current QoS capabilities center around the sqm-scripts package, which provides a simplified interface to Linux traffic control. While effective for basic bandwidth management, it lacks the sophistication needed for modern heterogeneous networks. The luci-app-sqm package provides a web-based configuration interface but offers limited real-time monitoring and no adaptive capabilities.

Recent community efforts have explored enhanced QoS solutions, including the qosify project, which attempts to provide more sophisticated traffic classification. However, these efforts remain fragmented and lack the integrated approach proposed in this research.

Chapter 3

Problem Definition and Requirements Analysis

3.1 Detailed Problem Analysis

Based on the literature review and analysis of current OpenWrt implementations, we can categorize the key problems into several domains:

3.1.1 Technical Limitations

- **Static Bandwidth Allocation:** Current SQM implementations require manual bandwidth ceiling configuration and cannot adapt to varying connection speeds (common with ISP throttling or variable wireless connections).
- **Primitive Traffic Classification:** Reliance on simple port-based rules fails to accurately identify modern applications that use dynamic ports, CDNs, or encryption.
- **No Feedback Mechanisms:** Absence of latency monitoring and quality metrics prevents adaptive responses to network congestion.
- **Limited Fairness Models:** Current per-host fairness doesn't account for different device types or user priorities.

3.1.2 User Experience Issues

- **Configuration Complexity:** Users must understand technical concepts like bandwidth ceilings, queue types, and DSCP markings.
- **Lack of Visibility:** No real-time insight into which devices or applications are consuming bandwidth or experiencing quality issues.

- **No Contextual Control:** Cannot easily prioritize traffic based on user activity (gaming session, work meeting, entertainment).

3.2 Requirements Specification

3.2.1 Functional Requirements

1. **FR1 - Persona Management:** The system shall allow users to assign personas (Gaming, Streaming, VoIP, IoT, Bulk) to network devices.
2. **FR2 - Dynamic Classification:** The system shall classify network traffic by application type using multiple indicators (ports, flow characteristics, DNS queries).
3. **FR3 - Adaptive Policies:** The system shall automatically adjust QoS parameters based on real-time network conditions and congestion feedback.
4. **FR4 - Policy Templates:** The system shall provide pre-configured policy templates for common use cases (Gaming Household, Remote Work, Entertainment).
5. **FR5 - Real-time Monitoring:** The system shall provide real-time visibility into bandwidth usage, latency metrics, and QoS decisions.
6. **FR6 - Manual Override:** Users shall be able to manually boost or throttle specific devices or applications temporarily.
7. **FR7 - Historical Analytics:** The system shall maintain historical data on network performance and usage patterns.

3.2.2 Non-Functional Requirements

1. **NFR1 - Performance:** CPU overhead shall not exceed 20% on the Xiaomi AX3000T's 1.3GHz ARM64 processor during normal operation.
2. **NFR2 - Memory Usage:** RAM usage shall not exceed 64MB on the AX3000T's 256MB system memory (25% maximum utilization).
3. **NFR3 - Reliability:** System shall gracefully degrade to standard SQM operation if advanced features fail, maintaining router stability.
4. **NFR4 - Compatibility:** Solution shall work on OpenWrt 23.05+ specifically targeting MediaTek MT7981B chipset architecture.
5. **NFR5 - Usability:** Configuration tasks shall be completable by non-technical users within 5 minutes using the web interface.
6. **NFR6 - Maintainability:** Code shall follow OpenWrt packaging standards for MediaTek platforms and include comprehensive documentation.
7. **NFR7 - Storage Efficiency:** Complete package size shall not exceed 8MB on the AX3000T's 128MB NAND flash storage.

3.2.3 Hardware Target Specifications

The solution targets the Xiaomi AX3000T router as the primary development and testing platform:

Table 3.1: Xiaomi AX3000T Specifications and Requirements

Component	**Xiaomi AX3000T Specs**	**Project Requirements**
CPU	MediaTek MT7981B @ 1.3GHz ARM64	Sufficient for classification algorithms
RAM	256MB DDR4	Target < 64MB usage (25% utilization)
Storage	128MB NAND Flash	< 8MB package size
Network	Gigabit Ethernet, Wi-Fi 6	Full line-rate QoS processing
OpenWrt	Supported since 23.05	Target 23.05+ compatibility
Architecture	aarch64 (ARM64)	Native compilation support

Performance Targets for Xiaomi AX3000T:

- **CPU Usage:** < 20% during normal operation, < 40% under peak load
- **Memory Footprint:** < 64MB total RAM usage including classification tables
- **Throughput:** Support full gigabit speeds with QoS active
- **Latency Impact:** < 2ms additional latency from QoS processing
- **Power Consumption:** No significant increase from baseline

Chapter 4

Proposed Solution Architecture

4.1 System Overview

The proposed solution consists of four main components working together to provide intelligent, adaptive QoS management:

1. **QoS Daemon (`qosd`):** Core service managing traffic control and policy enforcement
2. **Classification Engine:** Multi-layer traffic identification and categorization system
3. **Policy Engine:** Adaptive decision-making system with feedback loops
4. **Management Interface:** Modern web-based dashboard with real-time monitoring

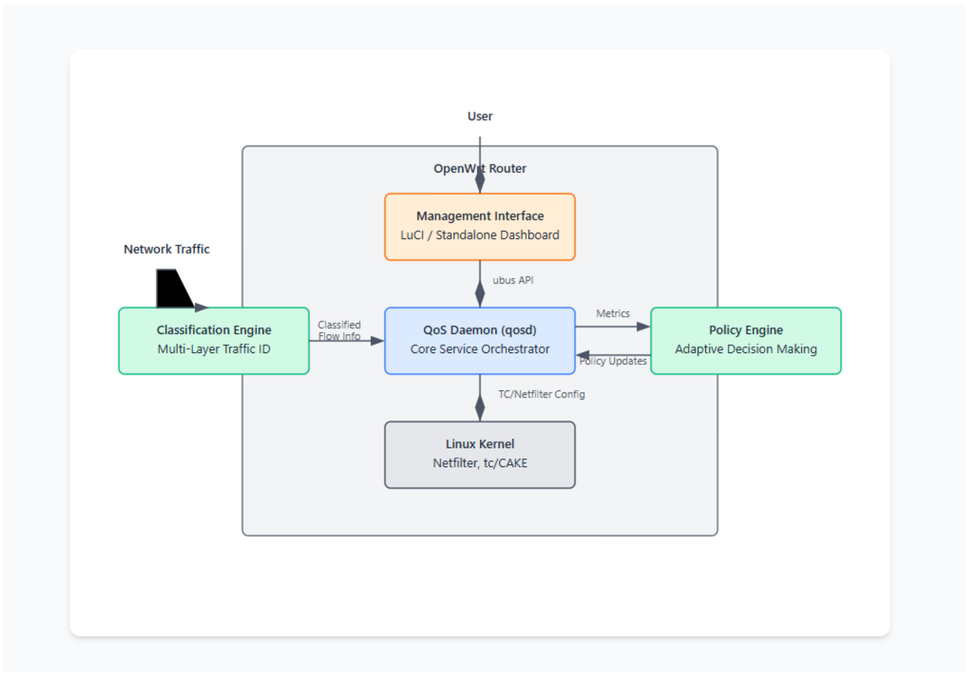

Figure 4.1: System Architecture Overview

4.2 Core QoS Daemon (qosd)

4.2.1 Architecture Design

The `qosd` daemon serves as the central orchestrator, implementing a modular architecture that separates concerns while maintaining tight integration with OpenWrt's system services.

Key Components:

- **UCI Configuration Handler:** Manages configuration persistence and validation
- **ubus API Server:** Provides D-Bus-style IPC for UI and other system components
- **CAKE Interface Module:** Manages tc/CAKE queue configuration and monitoring
- **Netfilter Integration:** Handles packet marking and connection tracking
- **Metrics Collection:** Gathers performance data and system telemetry

4.2.2 Configuration Schema

The UCI configuration schema provides structured, hierarchical configuration:

```
# Device persona configuration
config persona 'gaming_pc'
    option device '192.168.1.100'
    option mac 'aa:bb:cc:dd:ee:ff'
    option type 'gaming'
    option priority 'high'
    option bandwidth_share '30'

# Policy template configuration
config policy 'gaming_household'
    option name 'Gaming Optimized'
    option gaming_priority 'highest'
    option streaming_priority 'medium'
    option bulk_priority 'lowest'
    option adaptive 'true'

# Service classification rules
config service 'steam'
    option name 'Steam Gaming'
    option ports '27015,27036,27037'
    option protocol 'udp'
    option category 'gaming'
    option dscp_mark 'ef'
```

Listing 4.1: UCI Configuration Example

4.2.3 ubus API Design

The ubus API provides a clean, documented interface for management applications:

```
{
  "qos.status": {
    "description": "Get current QoS status and statistics",
    "returns": {
      "active_policy": "string",
      "devices": "array",
      "bandwidth_utilization": "object",
      "latency_metrics": "object"
    }
  },
  "qos.personas": {
    "methods": {
      "list": "Get all configured personas",
      "add": "Add new device persona",
      "update": "Modify existing persona",
      "delete": "Remove persona configuration"
    }
  },
  "qos.policies": {
    "methods": {
      "get_active": "Get current active policy",
      "set_active": "Switch active policy template",
      "boost_device": "Temporarily prioritize device"
    }
  }
}
```

Listing 4.2: ubus API Methods

4.3 Classification Engine

4.3.1 Multi-Layer Classification

The classification engine employs a hierarchical approach combining multiple identification methods:

Layer 1 - Protocol Analysis:

- Port-based rules for well-known services

- Protocol field inspection (TCP flags, UDP patterns)
- Payload size distribution analysis

Layer 2 - Flow Characteristics:

- Packet size histograms
- Inter-arrival time analysis
- Burst pattern detection
- Connection duration and volume

Layer 3 - Context Integration:

- DNS query correlation
- Device persona weighting
- Time-based usage patterns
- Historical classification learning

4.3.2 Service Categories

Traffic is classified into predefined service categories, each with associated QoS requirements:

Table 4.1: Service Categories and QoS Requirements

Category	**Latency**	**Jitter**	**Bandwidth**	**DSCP**
Gaming	< 50ms	< 10ms	Low-Medium	EF (46)
VoIP/Video	< 150ms	< 30ms	Medium	AF31 (26)
Streaming	< 500ms	Variable	High	AF23 (20)
Web/Interactive	< 300ms	Variable	Medium	AF13 (14)
Bulk Transfer	Best Effort	Variable	Variable	BE (0)
IoT/Background	Best Effort	Variable	Low	CS1 (8)

4.4 Policy Engine

4.4.1 Adaptive Decision Making

The policy engine implements a feedback-driven control system that continuously adjusts QoS parameters based on measured network conditions:

Control Loop Components:

1. **Measurement:** Collect latency, jitter, loss, and utilization metrics

2. **Analysis:** Compare metrics against target thresholds for each service class
3. **Decision:** Determine required parameter adjustments using control algorithms
4. **Action:** Apply changes to CAKE tin configurations and DSCP markings
5. **Verification:** Monitor impact of changes and adjust feedback parameters

4.4.2 Policy Templates

Pre-configured policy templates provide optimized settings for common scenarios:

Gaming Household:

- Gaming traffic: 40% minimum bandwidth guarantee
- Ultra-low latency target (< 30ms)
- Aggressive bulk traffic throttling during gaming

Remote Work:

- VoIP/Video conferencing priority
- Consistent bandwidth allocation
- Background update throttling during business hours

Entertainment:

- Streaming service optimization
- Balanced bandwidth sharing
- Peak hour adaptive scheduling

4.4.3 Hysteresis and Stability

To prevent oscillation and ensure stable operation, the adaptive system implements hysteresis in decision making:

- **Threshold Bands:** Different thresholds for increasing vs. decreasing priority
- **Dampening:** Gradual parameter adjustments rather than sudden changes
- **Stability Timers:** Minimum intervals between major policy changes
- **Load Averaging:** Smoothed metrics over configurable time windows

4.5 Management Interface

4.5.1 LuCI Integration

A lightweight LuCI application provides basic configuration and monitoring for resource-constrained devices:

Features:

- Device persona assignment interface
- Policy template selection
- Real-time bandwidth and latency display
- Basic troubleshooting tools

4.5.2 Standalone Dashboard

An optional standalone web application provides advanced management capabilities:

Advanced Features:

- Interactive network topology visualization
- Detailed traffic analytics with historical trending
- Custom policy creation and testing
- Advanced diagnostic tools and packet capture
- Mobile-responsive design with progressive web app capabilities

The dashboard uses modern web technologies (React/Vue.js, WebSocket for real-time updates, Chart.js for visualization) while maintaining compatibility with router hardware limitations.

Chapter 5

Implementation Methodology

5.1 Development Approach

The project follows an iterative development methodology with clearly defined phases, allowing for incremental feature delivery and continuous validation:

5.1.1 Phase 1: Minimum Viable Product (Months 1-2)

Objectives:

- Establish basic daemon architecture and UCI integration
- Implement simple persona-based QoS mapping
- Create fundamental LuCI configuration interface
- Validate CAKE integration and basic traffic shaping

Deliverables:

- `qosd` daemon with basic persona support
- UCI configuration schema and validation
- ubus API framework
- Basic LuCI forms for device-persona assignment
- Simple policy application (Gaming vs. Default)

Success Criteria:

- Successful CAKE tin assignment based on device personas
- Measurable latency improvement for gaming devices
- Stable operation on target hardware for 48+ hours

5.1.2 Phase 2: Beta Release (Months 3-4)

Objectives:

- Implement comprehensive traffic classification engine
- Add adaptive policy adjustment based on network feedback
- Develop real-time monitoring dashboard
- Integrate DNS-based service identification

Deliverables:

- Multi-layer traffic classification system
- Adaptive control loop with latency/jitter feedback
- Real-time web dashboard with bandwidth/latency visualization
- Policy templates for common use cases
- Enhanced debugging and logging capabilities

Success Criteria:

- Accurate service classification (>85% accuracy on test traffic)
- Demonstrable adaptive behavior during congestion events
- User-friendly dashboard accessible via web browser

5.1.3 Phase 3: Release Candidate (Months 5-6)

Objectives:

- Optimize performance for low-end hardware
- Complete advanced policy templates and configuration options
- Prepare comprehensive documentation and packaging
- Conduct thorough testing across hardware platforms

Deliverables:

- Production-ready package for OpenWrt feeds
- Complete policy template library
- Comprehensive user and developer documentation
- Performance optimization for 128MB RAM devices
- Automated testing suite and CI/CD pipeline

Success Criteria:

- $< 15\%$ CPU overhead on minimum hardware specifications

- Successful community package review and acceptance
- Zero critical bugs in 7-day continuous operation test

5.2 Development Tools and Environment

5.2.1 Build Environment

- **OpenWrt SDK:** Cross-compilation toolchain for target architectures
- **Version Control:** Git with GitHub/GitLab for collaboration and CI/CD
- **Development Languages:** C for daemon core, shell scripts for integration, JavaScript/HTML for UI
- **Testing Framework:** Custom network simulation environment with controlled traffic generation

5.2.2 Testing Infrastructure

- **Hardware Lab:** Collection of target routers representing different performance tiers
- **Traffic Generation:** iperf3, flent, and custom tools for realistic traffic patterns
- **Monitoring Tools:** Wireshark, tc statistics, and custom latency measurement tools
- **Automation:** Python-based test orchestration for repeatable performance measurements

5.3 Quality Assurance

5.3.1 Code Quality Standards

- **Coding Standards:** Follow Linux kernel coding style for C code
- **Documentation:** Comprehensive inline documentation and API references
- **Static Analysis:** Regular use of cppcheck, valgrind, and static analyzers
- **Peer Review:** All code changes require review before integration

5.3.2 Testing Strategy

- **Unit Testing:** Individual component testing with mock interfaces
- **Integration Testing:** End-to-end testing of complete QoS scenarios
- **Performance Testing:** Latency, throughput, and resource usage validation

- **Stress Testing:** High-load scenarios with concurrent traffic flows
- **Compatibility Testing:** Validation across different hardware platforms and OpenWrt versions

Chapter 6

Evaluation Plan

6.1 Performance Metrics

6.1.1 Technical Performance Indicators

Latency Metrics:

- **Round-Trip Time (RTT):** Measured using ICMP and UDP probes to various targets
- **Jitter:** Standard deviation of RTT measurements over time windows
- **Packet Loss Rate:** Percentage of dropped packets during congestion events
- **Queue Delay:** Time packets spend in router queues before transmission

Throughput and Fairness:

- **Per-Device Throughput:** Bandwidth allocation across different personas
- **Application-Level Performance:** Gaming frame rates, video quality metrics, VoIP MOS scores
- **Fairness Index:** Jain's fairness index for bandwidth distribution
- **Adaptation Speed:** Time to detect and respond to congestion events

System Resource Usage:

- **CPU Utilization:** Average and peak CPU usage during various load conditions
- **Memory Footprint:** RAM usage including buffers and classification tables
- **Storage Requirements:** Configuration and log file space consumption
- **Network Overhead:** Additional control traffic generated by the system

6.1.2 User Experience Metrics

Application-Specific Quality:

- **Gaming Performance:** Lag compensation effectiveness, ping stability in competitive games
- **Video Streaming:** Startup delay, rebuffering events, quality adaptation smoothness
- **VoIP Quality:** Voice clarity (MOS), call setup time, dropout frequency
- **Web Browsing:** Page load times, perceived responsiveness

Management Interface Usability:

- **Configuration Time:** Time required for initial setup by non-technical users
- **Problem Resolution:** Time to identify and resolve network issues using dashboard
- **Interface Responsiveness:** Dashboard load times and real-time update latency

6.2 Experimental Design

6.2.1 Test Scenarios

Scenario 1 - Gaming Under Load:

- Primary traffic: Online gaming session (CS:GO, Valorant, or similar)
- Background load: Multiple concurrent video streams (Netflix, YouTube)
- Bulk transfer: Large file download/upload
- Metrics: Gaming latency, jitter, packet loss

Scenario 2 - Video Conferencing Priority:

- Primary traffic: Zoom/Teams video conference
- Competing load: Smart TV streaming, mobile device usage
- Background: Automated cloud backups
- Metrics: Video quality, audio dropout, connection stability

Scenario 3 - Mixed Household Usage:

- Simultaneous: Gaming, streaming, web browsing, IoT device communication
- Variable load: Peak usage periods, idle times, burst activities
- Metrics: Overall network satisfaction, fairness across device types

Scenario 4 - Adaptive Response Testing:

- Dynamic conditions: ISP throttling simulation, wireless interference
- Policy changes: User switching between Gaming and Work-from-Home modes
- Device changes: New devices joining network, device persona modifications
- Metrics: Adaptation speed, stability, configuration persistence

6.2.2 Baseline Comparisons

Performance will be compared against multiple baseline configurations:

- **No QoS:** Standard FIFO (First-In, First-Out) queuing without any traffic management. This represents the default state of many consumer routers and serves as the worst-case scenario for bufferbloat.
- **Standard SQM:** OpenWrt's ‘sqm-scripts‘ with CAKE and static bandwidth settings. This is the current best-practice for bufferbloat mitigation and serves as the primary benchmark.
- **Alternative Open Source QoS:** A comparable open-source solution like ‘qosify‘ to evaluate performance against other advanced classification-based systems.
- **Commercial Router QoS:** A popular consumer router with a "Gaming Mode" or adaptive QoS feature, to benchmark against proprietary, black-box solutions.

6.2.3 Data Collection and Analysis

Data will be collected systematically using a combination of automated tools and manual observation. The ‘flent‘ network testing tool will be used to generate controlled workloads and collect latency, throughput, and packet loss data. Custom scripts will be developed to monitor router CPU/memory usage and log policy decisions from the ‘qosd‘ daemon.

For each test scenario, multiple runs will be conducted to ensure statistical significance. The collected data will be analyzed using statistical methods (e.g., t-tests, ANOVA) to compare the performance of the proposed solution against the baselines. Results will be visualized using box plots, time-series graphs, and summary tables to clearly demonstrate the impact of the dynamic QoS system.

Chapter 7

Project Plan and Timeline

7.1 Work Breakdown and Timeline

The project is scheduled to be completed over an 8-month period, divided into four major phases. The following table outlines the key activities, deliverables, and timeline.

Table 7.1: Project Timeline and Milestones

Phase	**Duration**	**Key Activities**	**Milestones**
1. MVP	Months 1-2	- Daemon architecture design - UCI/ubus integration - Basic persona-to-CAKE mapping - LuCI configuration UI	MVP complete
2. Beta	Months 3-4	- Multi-layer classification engine - Adaptive policy feedback loop - Real-time monitoring dashboard - Policy template implementation	Beta version functional
3. Release	Months 5-6	- Performance profiling and optimization - Comprehensive documentation - OpenWrt packaging - Automated test suite development	Release candidate ready
4. Evaluation	Months 7-8	- Execute evaluation plan experiments - Analyze performance data - Write final thesis/report - Prepare for project defense	Final report submitted

7.2 Risk Management

A proactive approach to risk management will be adopted to identify and mitigate potential issues that could impact the project's success.

Table 7.2: Risk Analysis and Mitigation Plan

Risk Category	Description	Mitigation Strategy
Technical	Classification engine has excessive CPU/memory overhead on target hardware.	Profile performance early and continuously. Prioritize lightweight heuristics over complex algorithms. Implement a graceful degradation mode.
Hardware	Target hardware (Xiaomi AX3000T) shows unexpected limitations or becomes unavailable.	Identify a secondary, comparable hardware platform (e.g., another MT798x-based router) for parallel testing and as a fallback target.
Scope Creep	Pressure to add features beyond the defined requirements, delaying core development.	Strictly adhere to the phased development plan. Defer non-essential feature requests to a post-project "future work" list.
Dependency	Breaking changes in OpenWrt core, the Linux kernel, or other dependencies.	Target a specific stable OpenWrt release (e.g., 23.05). Monitor developer mailing lists and perform regular integration testing.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

This project proposes the design and implementation of a dynamic, persona-aware Quality of Service management system for OpenWrt. The current state of home network management, characterized by static configurations and a lack of application awareness, fails to meet the demands of modern, heterogeneous network environments. This leads to degraded performance for latency-sensitive applications like online gaming and video conferencing, resulting in a poor user experience.

The proposed solution, centered around a new daemon named `qosd`, will leverage the power of the CAKE queue management algorithm while adding a crucial layer of intelligence. By introducing device personas, multi-layer traffic classification, and an adaptive policy engine with a real-time feedback loop, the system will dynamically allocate network resources where they are needed most. This will ensure that high-priority, latency-sensitive traffic is shielded from the negative impacts of bulk transfers and other competing traffic.

The project will deliver a production-quality, open-source package for the OpenWrt community, complete with an intuitive web interface for configuration and monitoring. The expected outcome is a significant, measurable improvement in network latency, jitter, and overall application performance, validated through a rigorous evaluation plan. This work aims to bridge the gap between advanced kernel networking capabilities and user-friendly, effective home network management.

8.2 Future Work

While this project will deliver a comprehensive solution, several avenues for future research and development exist:

- **Machine Learning-Based Classification:** Integrating a lightweight, online machine learning model could enable the system to classify encrypted or unknown traffic flows with greater accuracy, adapting over time to new applications and user behaviors.
- **Cloud Integration:** A companion cloud service could be developed to offload heavy computational tasks, aggregate anonymized data for improved classification models, and allow users to manage multiple routers from a single dashboard.
- **Deeper Application Integration:** The system could be enhanced to interact directly with applications and services via APIs (e.g., Discord, GeForce NOW) to receive explicit QoS hints, further improving prioritization accuracy.
- **Security Integration:** The classification engine could be extended to identify and de-prioritize or block malicious or unwanted traffic (e.g., from IoT devices participating in a DDoS attack) as part of its QoS policy enforcement.

Bibliography

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, *An architecture for differentiated services.* RFC 2475, December 1998.

[2] K. Nichols and V. Jacobson, *Controlling queue delay.* Communications of the ACM, vol. 55, no. 7, pp. 42-50, 2012.

[3] A. W. Moore and D. Zuev, *Internet traffic classification using host behavior.* In Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement, pp. 11-11, 2005.

[4] T. T. T. Nguyen and G. Armitage, *A survey of techniques for internet traffic classification using machine learning.* IEEE Communications Surveys & Tutorials, vol. 10, no. 4, pp. 56-76, 2008.

[5] S. Chen, Y. Wen, and K. Chen, *Adaptive QoS for real-time applications in software-defined networks.* In 2015 IEEE International Conference on Communications (ICC), pp. 4843-4848, 2015.