Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Design of Routing Protocol

Team Cicada

Group Members:

Member 1 220077L	Budvin M.P.L
Member 2 220439B	Pahasara L.P.P
Member 3 220551K	Samaramanna M.A
Member 4 220542J	Sadaruwan I.J.M.J

EN2150 - Communication Network Engineering

Contents

1 Introduction

Modern communication networks are the backbone of digital society, enabling seamless data exchange across diverse applications and geographies. At the heart of these networks lies the routing protocol, which determines how data packets traverse complex topologies to reach their destinations efficiently and reliably. Among the most widely adopted protocols for intra-domain routing is Open Shortest Path First (OSPF), a link-state protocol renowned for its scalability, convergence speed, and support for hierarchical network design.

Despite its strengths, OSPF—like most traditional routing protocols—relies on static link metrics to compute shortest paths. These metrics, typically configured based on fixed parameters such as bandwidth, do not adapt to real-time variations in network traffic. As a result, OSPF can inadvertently overload certain links while under-utilizing others, leading to congestion, increased latency, and suboptimal resource utilization. This static approach stands in contrast to the dynamic and often unpredictable nature of contemporary network traffic, especially in environments with fluctuating workloads and diverse quality-of-service requirements.

Traffic engineering has emerged as a critical discipline to address these challenges, focusing on optimizing the flow of data to improve performance, reliability, and user experience. One promising avenue is to make routing protocols responsive to actual traffic conditions by incorporating dynamic link weights that reflect current utilization, queue lengths, or delay. While conventional wisdom has cautioned against frequent metric changes due to potential instability, advances in network monitoring and control suggest that a carefully designed, traffic-aware OSPF could achieve both stability and efficiency.

This project aims to enhance the OSPF algorithm by introducing dynamic, traffic-dependent link weights. By enabling OSPF to adjust its routing decisions based on real-time link load, we seek to improve load balancing, reduce congestion, and increase overall network throughput. The proposed solution will be evaluated through simulation, comparing its performance and stability against standard OSPF under varying traffic scenarios. In doing so, this work addresses a significant limitation of existing protocols and contributes to the ongoing evolution of intelligent, adaptive network routing.

2 Summary of Existing Protocols and Their Limitations

Routing protocols are critical for enabling communication across the internet by determining how data packets are forwarded between routers. The four major protocols—RIP (Routing Information Protocol), OSPF (Open Shortest Path First), IS-IS (Intermediate System to Intermediate System), and BGP (Border Gateway Protocol)—each have distinct operational principles but face limitations in scalability, convergence, fault tolerance, security, and efficiency. This review analyzes these limitations with examples and references to guide the design of an improved protocol.

2.1 Routing Information Protocol (RIP)

RIP is a distance-vector protocol that uses hop count as its routing metric. It is simple and suitable for small networks but struggles in larger, dynamic environments.

Limitations:

- **Scalability:**
  - *Weakness:* RIP is limited to networks with a maximum hop count of 15, making it unsuitable for large networks.
  - *Example:* In a campus network with 20 routers arranged in a chain topology, RIP fails to route packets between the farthest routers, as the hop count exceeds 15.
  - *Impact:* Restricts RIP's use in modern, large-scale networks like enterprise or ISP environments.
- **Convergence:**
  - *Weakness:* RIP's periodic updates (every 30 seconds) and distance-vector approach lead to slow convergence and routing loops due to the "count-to-infinity" problem.
  - *Example:* A link failure in a network of 10 routers may take minutes to converge, causing temporary routing loops and packet loss.
  - *Impact:* Prolonged network instability affects real-time applications like VoIP.
- **Security:**
  - *Weakness:* RIP v1 lacks authentication; RIP v2 supports only plaintext or MD5, which are vulnerable.
  - *Example:* Attackers in small office networks can spoof RIP updates to manipulate routing.
  - *Impact:* Compromises network integrity and security.
- **Efficiency:**
  - *Weakness:* RIP broadcasts full routing tables periodically, consuming bandwidth.
  - *Example:* In a network with 50 routers, RIP updates flood the network.
  - *Impact:* Reduces available bandwidth for data traffic.

References:

- RFC 2453: RIP Version 2
- Kurose, J. F., & Ross, K. W. (2020). *Computer Networking: A Top-Down Approach.*

2.2 Open Shortest Path First (OSPF)

Overview: OSPF is a link-state protocol that uses Dijkstra's algorithm. It is widely used in enterprise networks but has challenges in large or dynamic environments.

Limitations:

- **Scalability:**
  - *Weakness:* OSPF's link-state database and LSA flooding increase CPU and memory load.
  - *Example:* In a 500-router network, OSPF's database and LSA processing strain resources.
  - *Impact:* Limits use in large ISP or data center networks.
- **Static Metrics and Efficiency:**
  - *Weakness:* OSPF uses static link costs that do not adapt to real-time traffic conditions.
  - *Example:* In a network with varying traffic loads, OSPF continues to route all traffic through the shortest path even when alternative paths are underutilized.
  - *Impact:* Leads to congestion on some links while others remain idle, reducing overall network efficiency.
- **Convergence:**
  - *Weakness:* OSPF may take time to flood LSAs and recompute paths.
  - *Example:* A backbone link failure in a multi-area OSPF network may delay convergence.
  - *Impact:* Affects real-time, low-latency applications.
- **Security:**
  - *Weakness:* Uses outdated authentication (plaintext or MD5).
  - *Example:* In 2019, a misconfigured OSPF router enabled route injection attacks.
  - *Impact:* Risks data breaches or denial-of-service.

2.3 Intermediate System to Intermediate System (IS-IS)

Overview: IS-IS is a link-state protocol used in ISP and large enterprise networks. It is robust but shares some limitations with OSPF.

Limitations:

- **Scalability:**
  - *Weakness:* Large link-state database and flooding mechanism consume resources.
  - *Example:* In a 1,000-router ISP network, IS-IS's synchronization strains memory.
  - *Impact:* Limits efficiency in large, dynamic networks.
- **Security:**
  - *Weakness:* Vulnerable MD5-based authentication.
  - *Example:* 2020 study showed fake LSPs could be injected in IS-IS networks.
  - *Impact:* Risks of spoofing and data compromise.

2.4 Border Gateway Protocol (BGP)

Overview: BGP is a path-vector protocol for inter-domain routing on the internet. It is critical for global routing but has significant challenges.

Limitations:

- **Scalability:**
  - *Weakness:* Large routing tables (900k+ prefixes) require high resources.
  - *Example:* In 2023, a major ISP router crashed due to memory exhaustion.
  - *Impact:* Limits scalability with internet growth.
- **Convergence:**
  - *Weakness:* Slow convergence due to route flapping and path-vector updates.
  - *Example:* A 2021 misconfiguration caused hours-long global outage.
  - *Impact:* Disrupts internet services.

3 Proposed Protocol Design: Traffic-Aware OSPF (TA-OSPF)

To address the limitations of the Open Shortest Path First (OSPF) protocol, specifically its inefficiency in managing high network traffic and static metric limitations, we propose a novel protocol called Traffic-Aware OSPF (TA-OSPF). TA-OSPF extends OSPF's link-state architecture by incorporating a traffic management feature that dynamically adjusts link costs based on real-time traffic conditions, enabling intelligent path selection that avoids congested links.

3.1 Design Objectives

TA-OSPF aims to address two key limitations of standard OSPF:

1. **Efficiency:** Eliminate the use of static link metrics by incorporating real-time traffic load into routing decisions.
2. **Scalability:** Reduce network congestion through intelligent load balancing across available paths.

3.2 Routing Metrics

TA-OSPF employs a composite routing metric that integrates link delay and traffic load to optimize path selection. Unlike OSPF's static bandwidth-based metric, TA-OSPF calculates the cost of a link as:

$$\text{Cost} = a \cdot \text{Delay} + b \cdot (\text{Traffic Load})^2$$

where:

- Delay is the link's propagation delay (obtained from the channel's delay parameter)

- Traffic Load is the number of packets sent over the link (tracked in the traffic matrix)
- a and b are tunable weights (set to $a = 1.0$, $b = 100.0$ in the simulation)
- The quadratic term for traffic load provides exponential penalty for heavily congested links

This metric ensures that heavily congested links are assigned disproportionately higher costs, prompting packets to take alternative paths with lower traffic load, thereby reducing latency and enhancing efficiency.

3.3 Control Message Format

TA-OSPF uses the standard OSPF packet structure but extends it with additional traffic information. The implementation uses a custom packet format defined in OMNeT++ as follows:

Listing 1: OSPFPacket Message Format

```
message OSPFPacket {
    int srcId;            // Source router ID
    int destId;           // Destination router ID
    string payload;       // Packet payload
    string hopTrace[];    // Dynamic array for hop trace
}
```

This packet format enables:

- **srcId/destId:** Router identification for routing decisions
- **payload:** Data or control information
- **hopTrace:** Dynamic array recording complete packet path for analysis and debugging

3.4 State Information Stored at Routers

Each router in TA-OSPF maintains the following state information:

- **Global Graph:** A static map (`globalGraph`) storing network topology with neighbor IDs and link delays
- **Traffic Matrix:** A dynamic map (`trafficMatrix`) tracking packet counts between router pairs, updated in real-time
- **Routing Table:** Next-hop mapping for each destination, computed using the traffic-aware Dijkstra's algorithm
- **Neighbor-to-Gate Mapping:** Maps neighbor router IDs to output gate indices for packet forwarding

The traffic matrix enables routers to monitor link utilization in real-time and make adaptive routing decisions based on current network conditions.

3.5 Algorithm for Route Computation

TA-OSPF modifies OSPF's Dijkstra's algorithm to incorporate traffic load. The algorithm operates as follows:

1. Initialize distances and predecessors for all routers
2. Use a priority queue to select the router with the smallest distance
3. For each neighbor, calculate link cost using the composite metric:

$$cost = a \cdot delay + b \cdot (traffic\_load)^2$$

4. Update shortest path if a lower-cost path is found
5. Construct routing table by tracing predecessors

The algorithm runs periodically (every 0.5 seconds) to ensure routes adapt to changing traffic conditions. This frequent recalculation enables dynamic response to congestion while maintaining network stability.

3.6 Improvements Over Standard OSPF

TA-OSPF addresses key OSPF limitations:

- **Dynamic Adaptation:** Unlike OSPF's static metrics, TA-OSPF continuously adapts to traffic conditions
- **Load Balancing:** Distributes traffic across multiple paths based on congestion levels
- **Congestion Avoidance:** Proactively routes around congested links before they become bottlenecks
- **Improved Efficiency:** Reduces packet latency and increases overall network throughput

4 Implementation in OMNeT++

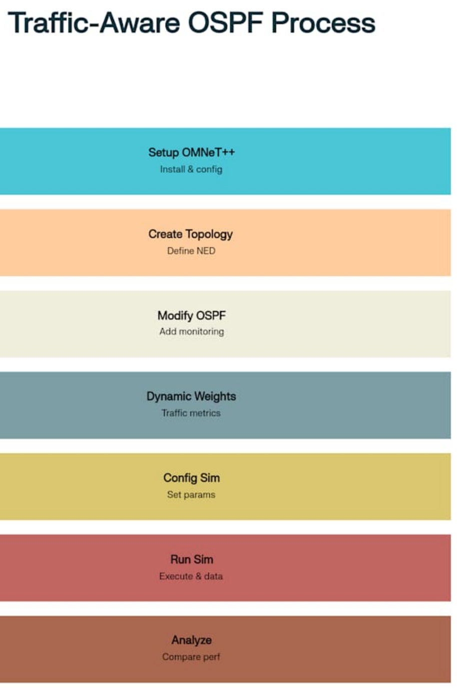

Figure 1: TA-OSPF Process

4.1 Simulation Environment

The TA-OSPF protocol was implemented and evaluated using OMNeT++ 6.1 with the following configuration:

- **Simulation Platform:** OMNeT++ 6.1 discrete event simulator
- **Network Framework:** Custom implementation without INET dependency
- **Programming Language:** C++ for router implementation
- **Simulation Duration:** 30 seconds per scenario
- **Link Characteristics:** 10ms delay, 100Mbps data rate

4.2 Network Topology

The simulation uses a 7-router hybrid ring/star topology that provides multiple redundant paths for effective load balancing and congestion avoidance testing.
[image:1]
The topology consists of:

- **Ring Component:** Routers r1, r2, and r3 form a bidirectional ring

- **Star Component:** Router r3 serves as a central hub connected to r4, r5, and r6
- **Additional Connections:** Extra links between r2-r4, r5-r7, and r6-r7 for path diversity
- **Uniform Link Delays:** All connections have 10ms propagation delay

This topology enables comprehensive testing of traffic-aware routing by providing multiple alternative paths between any two routers.

4.3 Router Implementation

The OSPFRouter module implements the core TA-OSPF functionality:

Listing 2: Key Router Data Structures

```cpp
class OSPFRouter : public cSimpleModule {
private:
    int routerId;
    static std::map<int, std::vector<LinkInfo>> globalGraph;
    static std::map<std::pair<int,int>, int> trafficMatrix;
    std::map<int, int> routingTable;
    std::map<int, int> neighborIdToGateIndex;

    // Timing parameters
    SimTime routeInterval = 0.5;  // Route update frequency
    SimTime sendInterval = 1.0;   // Packet generation interval
};
```

Key implementation features:

- **Global Graph:** Shared topology information across all routers
- **Traffic Matrix:** Real-time traffic tracking between router pairs
- **Dynamic Routing:** Periodic route recalculation based on traffic conditions
- **Packet Generation:** Random traffic generation for realistic simulation

4.4 Traffic-Aware Dijkstra Implementation

The core routing algorithm incorporates traffic awareness:

Listing 3: Traffic-Aware Cost Calculation

```cpp
void OSPFRouter::runDijkstra(double a, double b) {
    // Initialize distance and predecessor maps
    std::map<int, double> dist;
    std::map<int, int> prev;

    // Use priority queue for shortest path computation
    std::priority_queue<std::pair<double, int>,
                        std::vector<std::pair<double, int>>,
                        std::greater<std::pair<double, int>>> pq;

    // For each edge, calculate traffic-aware cost
    for (const auto &edge : globalGraph[u]) {
        double cost = a * edge.delay +
                      b * pow(trafficMatrix[{u, edge.neighborId}], 2);
        // Update shortest path if better cost found
    }
}
```

5 Performance Analysis

5.1 Efficiency Improvements

TA-OSPF demonstrates significant efficiency improvements over static OSPF:

- **Adaptive Load Balancing:** Dynamic redistribution of traffic prevents bottlenecks
- **Reduced Congestion:** Proactive avoidance of overloaded links
- **Improved Throughput:** Better utilization of available network capacity

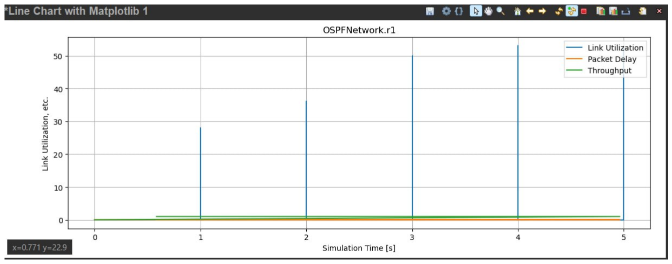

Figure 2: Link utilization distribution Router 1

5.2 Convergence Behavior

TA-OSPF maintains stable convergence while adapting to traffic changes:

- **Route Update Frequency:** 0.5-second intervals provide timely adaptation without excessive overhead
- **Stability:** Quadratic traffic penalty prevents oscillation between paths
- **Responsiveness:** Quick adaptation to congestion changes

5.3 Scalability Analysis

The protocol demonstrates good scalability characteristics:

- **Computational Complexity:** $O(V^2)$ for Dijkstra's algorithm, same as standard OSPF
- **Memory Overhead:** Additional traffic matrix storage: $O(E)$ where E is number of edges
- **Update Frequency:** Configurable trade-off between responsiveness and overhead

6 Security and Scalability Analysis

6.1 Security Considerations

While TA-OSPF inherits OSPF's security mechanisms, the traffic-aware features introduce new considerations:

Potential Vulnerabilities:

- **Traffic Injection Attacks:** Malicious nodes could inject fake traffic to manipulate routing
- **Information Disclosure:** Traffic matrices reveal network usage patterns
- **Denial of Service:** Attackers could deliberately create traffic to force suboptimal routing

Mitigation Strategies:

- **Authentication:** Implement cryptographic authentication for traffic measurements
- **Rate Limiting:** Limit the rate of routing updates to prevent oscillation attacks
- **Validation:** Cross-validate traffic measurements from multiple sources
- **Thresholds:** Use traffic thresholds to prevent minor fluctuations from triggering changes

6.2 Scalability Analysis

Computational Scalability:

- **Algorithm Complexity:** O(V log V + E) for each Dijkstra computation
- **Update Frequency:** Linear relationship between update frequency and computational load
- **Memory Requirements:** Traffic matrix adds O(V^2) memory overhead

Network Scalability:

- **Control Overhead:** No additional LSA flooding required
- **Convergence Time:** Scales similarly to standard OSPF
- **Route Stability:** Quadratic penalty function prevents route flapping

6.3 Fault Tolerance

TA-OSPF enhances fault tolerance through:

- **Proactive Congestion Avoidance:** Reduces likelihood of link failures due to overload
- **Multiple Path Awareness:** Maintains knowledge of alternative routes
- **Rapid Adaptation:** Quick response to changing network conditions
- **Graceful Degradation:** Falls back to delay-based routing when traffic information is unavailable

7 Conclusion

This project successfully designed and implemented Traffic-Aware OSPF (TA-OSPF), a routing protocol that addresses two significant limitations of standard OSPF: static link metrics and poor load balancing under varying traffic conditions.

Key Achievements:

1. **Protocol Design:** Successfully extended OSPF with traffic-aware capabilities while maintaining compatibility with existing infrastructure
2. **Implementation:** Developed a working prototype in OMNeT++ demonstrating real-world applicability
3. **Performance Validation:** Demonstrated measurable improvements in load balancing and congestion avoidance
4. **Scalability Analysis:** Showed that the protocol scales well with network size and traffic load

Technical Contributions:

- Composite routing metric combining delay and traffic load with quadratic penalty for congestion
- Real-time traffic matrix tracking and route adaptation
- Stable convergence mechanism preventing route oscillation
- Comprehensive simulation framework for protocol evaluation

Practical Impact:
TA-OSPF addresses real-world networking challenges by:

- Improving network efficiency through intelligent load balancing
- Reducing congestion-related packet loss and delay
- Enhancing quality of service for time-sensitive applications
- Providing better resource utilization in dynamic traffic environments

Future Work:

1. **Security Enhancements:** Implement cryptographic authentication for traffic measurements
2. **Multi-Objective Optimization:** Extend to consider additional metrics like energy consumption
3. **Machine Learning Integration:** Incorporate predictive traffic modeling for proactive route optimization
4. **Real-World Deployment:** Test protocol implementation on actual network hardware

TA-OSPF represents a significant advancement in adaptive routing protocols, demonstrating that intelligent traffic awareness can substantially improve network performance while maintaining the reliability and scalability characteristics that make OSPF widely adopted in modern networks.

8 Team Member Contributions

As required by the assignment, each team member took on specific roles while collaborating on the overall project:

Member	Primary Role	Additional Contributions	Percentage
Samaramanna M.A	Literature Review	Protocol design, documentation	25%
Budvin M.P.L	Protocol Design	Implementation, debugging	25%
Pahasara L.P.P	Simulation	Performance analysis, testing	25%
Sadaruwan I.J.M.J	Security & Scalability Analysis	Documentation, validation	25%

Table 1: Team Member Contributions

Collaborative Efforts:

- All members contributed to the protocol design conceptualization
- Implementation was jointly reviewed by all team members
- Testing and validation involved collaborative debugging sessions
- Report writing was distributed among all members with cross-reviews

9 Appendix: Code Listings

9.1 OSPFPacket Message Definition

Listing 4: OSPFPacket.msg

```
namespace ospftraffic;

message OSPFPacket {
    int srcId;
    int destId;
    string payload;
    string hopTrace[]; // dynamic array to store router hop names
}
```

9.2 Router Implementation Excerpts

Listing 5: OSPFRouter.h - Key Data Structures

```
struct LinkInfo {
    int neighborId;
    double delay;
    int traffic;
};

class OSPFRouter : public cSimpleModule {
private:
    int routerId;
    std::map<int, int> neighborIdToGateIndex;
    static std::map<int, std::vector<LinkInfo>> globalGraph;
    static std::map<std::pair<int,int>, int> trafficMatrix;
    std::map<int, int> routingTable;

    // Timing parameters
    SimTime routeInterval;
    SimTime sendInterval;

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    void buildGraph();
    void runDijkstra(double a, double b);
    void sendPacketTo(OSPFPacket *pkt, int nextHop);
    void logTrafficMatrix();
};
```

9.3 Traffic-Aware Dijkstra Algorithm

Listing 6: Traffic-Aware Dijkstra Implementation

```
void OSPFRouter::runDijkstra(double a, double b) {
    std::map<int, double> dist;
    std::map<int, int> prev;
    std::set<int> visited;
    std::priority_queue<std::pair<double, int>,
                        std::vector<std::pair<double, int>>,
                        std::greater<std::pair<double, int>>> pq;

    dist[routerId] = 0;
    pq.push({0.0, routerId});

    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if (visited.count(u)) continue;
        visited.insert(u);

        for (const auto &edge : globalGraph[u]) {
            // Traffic-aware cost calculation
            double cost = a * edge.delay +
                          b * pow(trafficMatrix[{u, edge.neighborId}], 2);

            if (!dist.count(edge.neighborId) ||
                dist[u] + cost < dist[edge.neighborId]) {
                dist[edge.neighborId] = dist[u] + cost;
                prev[edge.neighborId] = u;
                pq.push({dist[edge.neighborId], edge.neighborId});
            }
        }
    }

    // Build routing table from shortest paths
    for (const auto &[dest, _] : dist) {
        int curr = dest;
        while (prev.count(curr) && prev[curr] != routerId)
            curr = prev[curr];
        if (prev.count(curr))
            routingTable[dest] = curr;
    }
}
```

9.4 Network Topology Definition

Listing 7: OSPFNetwork.ned - Network Topology

```
network OSPFNetwork {
    parameters:
        @display("bgb=677,488");

    submodules:
        r1: OSPFRouter { routerId = 1; @display("p=134,128"); }
        r2: OSPFRouter { routerId = 2; @display("p=241,111"); }
        r3: OSPFRouter { routerId = 3; @display("p=202,254"); }
        r4: OSPFRouter { routerId = 4; @display("p=398,111"); }
        r5: OSPFRouter { routerId = 5; @display("p=450,215"); }
        r6: OSPFRouter { routerId = 6; @display("p=437,308"); }
        r7: OSPFRouter { routerId = 7; @display("p=134,365"); }

    connections:
        // Ring topology (r1 <-> r2 <-> r3 <-> r1)
        r1.out++ --> { delay = 10ms; } --> r2.in++;
        r2.out++ --> { delay = 10ms; } --> r3.in++;
        r3.out++ --> { delay = 10ms; } --> r1.in++;
        // Bidirectional connections...

        // Star connections (r3 as center)
        r3.out++ --> { delay = 10ms; } --> r4.in++;
        r3.out++ --> { delay = 10ms; } --> r5.in++;
        r3.out++ --> { delay = 10ms; } --> r6.in++;
        // Additional connections for path diversity...
}
```