# High Performance Multi-Mobile Node Routing Communication Protocol Analysis

## Executive Summary

This report analyzes various routing protocols for multi-mobile node networks, with a focus on reliable active node-based approaches. The research integrates concepts from traditional routing protocols with modern implementations that prioritize packet forwarding rate, energy efficiency, end-to-end delay minimization, and network node lifetime extension. The analysis includes comparisons between different routing mechanisms, ARQ protocols, and network simulation results for various network topologies.

## 1. Introduction to Multi-Mobile Node Routing Protocols

Modern Internet applications have expanded to include diverse services such as information distribution, distance learning, and video conferencing. These applications share a common requirement for reliable one-to-many or many-to-many data communication. Multicast communication technology has emerged as a solution, but it faces challenges including feedback information explosion, data loss, and difficulties in recovering lost data.

### 1.1 Need for Enhanced Routing Protocols

Traditional protocols such as OSPF (Open Shortest Path First) have limitations in network reliability as nodes do not participate actively in reliability work. This leads to:

- Low packet forwarding rates
- Long average end-to-end delays
- High energy consumption
- Short lifetime of network nodes

To address these issues, high-performance multi-mobile node routing communication protocols based on reliable active nodes have been developed.

## 2. Reliable Active Node Framework

### 2.1 Role Classification

The reliable active node approach divides network nodes into two categories:

1. **Forward Active Nodes (FAN)**:
    - Similar to traditional Internet network nodes
    - Handle basic packet forwarding duties

  - Lower energy availability
  - Primarily concerned with packet transit rather than error detection
  - Minimal caching capabilities
  - Typically located in network periphery
  - Support simple path discovery and maintenance
2. **Reliable Active Nodes (RAN)**:
  - Handle reliability work through programmable network services
  - Actively detect errors in communication data packages using CRC or hash functions
  - Process data loss packet retransmission applications with priority handling
  - Implement communication data caching with optimized buffer management
  - Use local multicast for packet loss retransmission requests to reduce network overhead
  - Higher energy availability and computational resources
  - Positioned at critical network junctions
  - Maintain state information about ongoing transmissions
  - Implement congestion control mechanisms

### 2.2 Hierarchical Model

The protocol follows a layered architecture:

- **Application Layer**: Dynamic configuration of multicast communication service
- **RM Management Sub-layer**: Establishes and releases session connections
- **RM Reliability Sub-layer**: Responsible for reliable data transmission
- **IP Layer**: Network interconnection
- **Network Interface Layer**: Physical connectivity

### 2.3 Active Network Programming

Active networks provide programmability by allowing packets to carry not just data but also code. This is implemented through two approaches:

1. **Capsule Approach**: Each packet carries its own processing code that executes at each node
2. **Programmable Switch Approach**: Nodes can be programmed remotely to handle specific packet types

For the reliable active node framework, the programmable switch approach is typically used where nodes are configured with specialized reliability functions based on their role (FAN vs RAN). This allows for:

- Dynamic adaptation to network conditions
- Custom error recovery schemes
- Localized retransmission strategies
- Application-specific reliability requirements
- Load balancing across multiple paths

### 2.4 Node State Management

Each node maintains state information including:

- Energy level monitoring
- Current role status (FAN/RAN)
- Neighbor node capabilities
- Link quality measurements
- Traffic statistics
- Buffer occupation levels
- Forwarding history

This state information is periodically exchanged between neighboring nodes to ensure optimal routing decisions based on current network conditions.

## 3. Routing Protocol Design

### 3.1 Routing Delay Estimation

The protocol calculates packet transmission delay time between adjacent nodes using:

```
Delay(Ps) = Tc + Tt + Tp + Tq + Ts = (T2 - T1)/2
```

Where:

- Tc: Communication channel competition time - time spent waiting to access the shared medium
- Tt: Transmission time - physical time to transmit all bits of the packet
- Tp: Processing time - time required for packet header processing and routing decisions
- Tq: Queue delay time - time spent in transmission buffers
- Ts: Node sleep time - delay due to energy conservation modes
- T1, T2: Packet entry and receipt timestamps

Specifically:

- Transmission time (Tt) = packet size / link bandwidth
- Processing time (Tp) depends on node CPU speed and current load
- Queue delay (Tq) varies with network congestion levels

The protocol updates packet transmission delay in real-time using weighted average:

```
Delay_t(Ps) = αE(S,D)Delay_t-1(Ps) + (1-α)∑Delay_k(Ps)
```

Where:

- $\alpha$ is the weighting factor ($0 < \alpha < 1$)
- E(S,D) represents the energy state function between nodes S and D
- Delay_t-1(Ps) is the previous delay measurement
- ∑Delay_k(Ps) represents historical delay measurements

The value of $\alpha$ is dynamically adjusted based on delay variance:

- High variance (rapidly changing network) → larger $\alpha$ (0.7-0.9)
- Low variance (stable network) → smaller $\alpha$ (0.3-0.5)

### 3.2 Link Quality Assessment

Link quality is measured by packet arrival success rate:

```
PRR = 1/16 ∑(j=1 to 16) ((-1)^j × exp(2^j × γ(d)/j - 1))
```

Where $\gamma(d)$ is the signal-to-noise ratio of active network communication.

The PRR (Packet Reception Rate) calculation incorporates:

- Physical layer bit error rate (BER)
- Signal attenuation based on distance
- Interference from concurrent transmissions
- Environmental noise factors
- Hardware-specific reception characteristics

Different network environments require PRR threshold adjustments:

- Indoor environments: PRR > 0.85 considered good quality

- Outdoor urban environments: PRR > 0.75 acceptable
- Industrial settings: PRR > 0.90 recommended due to interference

### 3.3 Next Hop Selection

The protocol calculates a forwarding adaptation index θ for each potential next-hop node:

```
θs→i = α × sink/∑(v(s→i)(Ps)) + β × Delay(s→i)(Ps) + γ × PRR
```

Where:

- α, β, γ are weight coefficients satisfying α + β + γ = 1
- PRR is the packet reception rate
- sink represents distance to destination
- Delay represents transmission delay

The values of α, β, and γ are context-dependent:

- For delay-sensitive applications: β > α ≈ γ (e.g., β=0.6, α=γ=0.2)
- For reliability-focused transmissions: γ > α ≈ β (e.g., γ=0.6, α=β=0.2)
- For energy-constrained networks: values adjusted to favor energy-efficient paths

The node with the highest θ value is selected as the next forwarding node.

### 3.4 Adaptive Path Maintenance

The protocol implements adaptive path maintenance through:

1. **Proactive monitoring**: Periodic link quality assessment
2. **Reactive updates**: Immediate recalculation of forwarding indices when conditions change
3. **Backup path preparation**: Alternative paths kept ready for immediate failover
4. **Gradient-based routing**: Multiple potential paths maintained toward destination

## 4. ARQ Protocol Analysis

### 4.1 Go-Back-N ARQ

Go-Back-N is an Automatic Repeat reQuest protocol that:

- Operates with a sliding window of packets
- Sender can transmit multiple frames without waiting for acknowledgment

- Uses cumulative ACKs
- On error, retransmits all frames in the window
- Offers simplicity but may be inefficient with high error rates

Implementation reveals:

- Lower efficiency in high-error environments
- Higher total transmissions compared to Selective Repeat
- Simpler implementation with less overhead

### 4.2 Selective Repeat ARQ

The Selective Repeat ARQ protocol:

- Resends only the specific frames that are lost/corrupted
- Requires buffers at both sender and receiver
- Maintains individual acknowledgment for each packet
- Demonstrates higher efficiency in lossy networks
- Has lower total transmissions but more complex implementation

### 4.3 Protocol Comparison

Based on simulations with a 20% error rate, Selective Repeat consistently outperforms Go-Back-N in terms of retransmission efficiency, especially as network complexity increases.

## 5. Network Simulation Results

### 5.1 Random Tree Network Topology

Simulations on random tree networks with varying node counts (10, 30, 50, 100) revealed:

- As network size increases, average efficiency decreases
- Shortest path length (hop count) directly impacts performance
- Error probability compounds with each hop
- Network diameter plays a crucial role in performance

### 5.2 K-Connected Graph Networks

Simulations on k-vertex-connected graphs showed:

- Higher connectivity (k) improves resilience to node failures

- Additional random edges enhance routing options
- Multiple potential paths between source and destination increase reliability
- Network congestion occurs less frequently than in tree topologies

### 5.3 Energy Consumption Analysis

The simulations tracked node energy consumption, showing:

- Energy depletion occurs unevenly across the network
- High-traffic nodes deplete energy more quickly
- Nodes near the source/destination experience higher energy drain
- Role switching between FAN and RAN based on energy levels extends network lifetime

## 6. Performance Evaluation

### 6.1 Packet Forwarding Rate

The implementation of the reliable active node approach significantly improved packet forwarding rates compared to traditional protocols. This improvement stems from:

- Active participation of nodes in reliability work
- Selection of higher quality links for transmission
- Real-time adaptation to network conditions
- Balanced load distribution across network nodes

### 6.2 Energy Consumption

Energy consumption was notably reduced through:

- Strategic selection of forwarding paths
- Dynamic role assignment based on energy levels
- Efficient retransmission policies
- Reduced overhead in packet headers

### 6.3 End-to-End Delay

Average end-to-end delay was shortened by:

- Accurate estimation of transmission delays
- Selection of optimal paths based on delay metrics
- Reduced retransmission frequency

- More efficient error recovery mechanisms

### 6.4 Network Lifetime

The protocol extended network node lifetime through:

- Balanced energy consumption across nodes
- Reduced unnecessary retransmissions
- Adaptive forwarding index calculation
- Energy-aware routing decisions

## 7. Implementation Considerations

### 7.1 Node Classification Algorithm

The protocol implementation requires sophisticated algorithms to classify nodes as FAN or RAN based on:

- Current energy levels (threshold at 60%)
- Historical reliability metrics
- Position in the network topology
- Processing capabilities

python

```python
def update_role(self):
    # Basic implementation
    self.role = "RAN" if self.energy > 60 else "FAN"

    # Enhanced implementation considering additional factors
    if self.energy > 60:
        if self.get_centrality_score() > 0.7:  # Topological importance
            self.role = "RAN"
            self.activate_reliability_functions()
        elif self.processing_power > threshold:
            self.role = "RAN"
            self.activate_reliability_functions(limited=True)
        else:
            self.role = "FAN"
    else:
        self.role = "FAN"
        self.deactivate_reliability_functions()
```

The centrality score calculation determines a node's importance in the network:

python

```python
def get_centrality_score(self):
    # Count number of shortest paths passing through this node
    path_count = 0
    total_paths = 0

    for source in self.network.nodes:
        for dest in self.network.nodes:
            if source != dest and source != self.id and dest != self.id:
                all_shortest_paths = self.network.get_all_shortest_paths(source, dest)
                total_paths += len(all_shortest_paths)

                for path in all_shortest_paths:
                    if self.id in path:
                        path_count += 1

    return path_count / total_paths if total_paths > 0 else 0
```

## 7.2 Forwarding Index Calculation

The forwarding index calculation integrates multiple factors:

python

```python
def calculate_forwarding_index(self, delay, link_quality):
    # Basic index calculation
    basic_index = (1 / delay) * link_quality * (self.energy / 100)

    # Enhanced calculation incorporating congestion and node role
    congestion_factor = 1.0
    if self.buffer_occupancy > 0.8:  # 80% buffer full
        congestion_factor = 0.5

    role_bonus = 1.2 if self.role == "RAN" else 1.0

    # Distance to destination factor (if known)
    distance_factor = 1.0
    if self.known_destinations:
        if destination in self.known_destinations:
            hops_to_dest = self.known_destinations[destination]
            distance_factor = 1 + (1 / (hops_to_dest + 1))

    self.forwarding_index = basic_index * congestion_factor * role_bonus * distance_factor
```

## 7.3 Next Hop Selection

The next hop selection process prioritizes nodes with higher forwarding indices:

python

```python
def select_next_hop(self, current_id, destination_id=None, packet_type="DATA"):
    # Get all potential next hops with their forwarding indices
    neighbors = [(j, self.nodes[j].forwarding_index) for j in range(len(self.nodes))
                if j != current_id]

    # Filter by activity status and minimum energy requirements
    viable_neighbors = [(nid, idx) for nid, idx in neighbors
                        if self.nodes[nid].active and self.nodes[nid].energy > 0]

    if not viable_neighbors:
        return None  # No viable next hop

    if packet_type == "HIGH_PRIORITY":
        # For high priority packets, filter to include only RAN nodes if possible
        ran_neighbors = [(nid, idx) for nid, idx in viable_neighbors
                        if self.nodes[nid].role == "RAN"]
        if ran_neighbors:
            viable_neighbors = ran_neighbors

    # Sort by forwarding index (highest first)
    viable_neighbors.sort(key=lambda x: x[1], reverse=True)

    # Select best next hop
    return viable_neighbors[0][0]
```

## 7.4 Packet Queue Management

Implementing priority queues for different packet types enhances protocol performance:

python

```python
class PacketQueue:
    def __init__(self, max_size=100):
        self.control_queue = []  # Highest priority
        self.reliability_queue = []  # Medium priority
        self.data_queue = []  # Normal priority
        self.max_size = max_size

    def enqueue(self, packet):
        if packet.type == "CONTROL":
            if len(self.control_queue) < self.max_size:
                self.control_queue.append(packet)
                return True
        elif packet.type == "RELIABILITY":
            if len(self.reliability_queue) < self.max_size:
                self.reliability_queue.append(packet)
                return True
        else:  # DATA
            if len(self.data_queue) < self.max_size:
                self.data_queue.append(packet)
                return True
        return False  # Queue full

    def dequeue(self):
        if self.control_queue:
            return self.control_queue.pop(0)
        elif self.reliability_queue:
            return self.reliability_queue.pop(0)
        elif self.data_queue:
            return self.data_queue.pop(0)
        return None  # All queues empty
```

## 7.5 Role Transition Management

Smooth transitions between FAN and RAN roles prevent network disruptions:

python

```python
def transition_role(self, new_role):
    if self.role == new_role:
        return  # No change needed

    if new_role == "RAN":
        # Allocate resources for reliability functions
        self.allocate_buffer_space(size="LARGE")
        self.enable_error_detection()
        self.advertise_capabilities_to_neighbors()
        self.role = "RAN"

    elif new_role == "FAN":
        # Graceful degradation
        self.transfer_buffered_packets()
        self.notify_neighbors_of_role_change()
        self.deallocate_reliability_resources()
        self.role = "FAN"
```

## 8. Conclusion and Future Work

The high-performance multi-mobile node routing communication protocol based on reliable active nodes demonstrates significant improvements over traditional approaches. The protocol effectively addresses key challenges in multi-mobile node networks by:

1. Enhancing packet forwarding rates through intelligent next-hop selection
2. Reducing overall energy consumption via energy-aware routing decisions
3. Shortening average end-to-end delay through optimized path selection
4. Extending network node lifetime via balanced energy utilization

Future work should focus on:

- Adaptive parameter tuning for different network environments
- Machine learning integration for predictive routing decisions
- Security enhancements for protection against routing attacks
- Further optimization for extremely large network scales
- Integration with emerging IoT and 5G/6G standards

The protocol offers a promising foundation for addressing the increasing demands of modern network applications while efficiently managing limited network resources.