

LAPORAN IMPLEMENTASI PROTOKOL ROUTING AODV-EOCW BERBASIS FUZZY LOGIC



Anggota Kelompok

Nabil Julian Syah

5025231061

Moch Septian Ezra M

5025231120

Implementasi protokol routing AODV-EOCW, sebuah modifikasi dari protokol standar AODV. Protokol ini tidak lagi memilih rute semata-mata berdasarkan *hop count* terpendek, melainkan menggunakan mekanisme seleksi jalur cerdas berbasis Multi-Criteria Decision Making (MCDM).

Fitur utama:

- Cross-Layer Monitoring
Untuk memantau sisa energi (Physical Layer) dan antrian MAC (Data Link Layer).
- MCDM Hybrid
Menggabungkan Fuzzy Logic dan EWM (Entropy Weight Method).
- Fuzzy Logic Adaptation
Menggantikan pembobotan statis (AHP) dari paper asli dengan Fuzzy Logic agar node dapat beradaptasi secara real-time terhadap kondisi kritis.

Rincian Implementasi

1. Header & Dependensi

```
#include "ns3/energy-source.h"
#include "ns3/wifi-net-device.h"
#include "ns3/wifi-mac-queue.h"
```

`energy-source` diperlukan untuk mengakses objek `EnergySource` pada node untuk membaca sisa baterai `GetRemainingEnergy()`. `wifi-net-device` & `wifi-mac-queue` untuk mengakses antrian transmisi pada jaringan WiFi. Ini digunakan untuk menghitung tingkat kemacetan secara akurat berdasarkan buffer yang terisi, bukan sekadar estimasi aplikasi.

2. Struktur Data Jalur Kandidat

```
// --- EOCW ---
struct EocwPath
{
    double pathMinEnergy;           // Energi minimum sepanjang rute
    double pathAvgCongestion;       // Rata-rata tingkat kemacetan sepanjang rute
    uint32_t hopCount;              // Jumlah hop sepanjang rute
    ns3::aodv::RoutingTableEntry reverseRoute; // Rute balik untuk mengirim RREP

    // Constructor untuk inisialisasi
    EocwPath(double energy, double congestion, uint32_t hops, ns3::aodv::RoutingTableEntry route)
        : pathMinEnergy(energy),
          pathAvgCongestion(congestion),
          hopCount(hops),
          reverseRoute(route)
    {
    }
};
```

Jika *node* menerima RREQ duplikat (ID sama, origin sama), paket tersebut langsung dibuang jika menggunakan AODV standar. Sedangkan pada EOCW *node* tujuan harus menampung beberapa RREQ yang datang dari rute berbeda untuk dibandingkan.

Struct EocwPath ini bertugas menyimpan data metrik dari setiap kandidat rute tersebut sebelum keputusan dibuat.

3. Helper EOCW

```
// --- Fungsi Helper EOCW ---
double GetResidualEnergyScore(); // Menghitung skor energi sisa (RE)
double GetCongestionDegreeScore(); // Menghitung skor tingkat kemacetan (CD)
double GetHopCountScore(uint32_t hopCount); // Menghitung skor jumlah hop (HC)
// --- End of Fungsi Helper EOCW ---
```

- GetResidualEnergyScore

```
double RoutingProtocol::GetResidualEnergyScore()
{
    if (!m_energySource || m_initialEnergy == 0) return 1.0;
    return m_energySource->GetRemainingEnergy() / m_initialEnergy;
}
```

Fungsi ini adalah menghitung kesiapan energy node dengan rumus energi saat ini dibagi dengan energi awal. Akan menghasilkan angka antara 0.0 (habis) sampai 1.0 (penuh) untuk memberi tahu sistem seberapa siap node tersebut bekerja.

- GetResidualEnergyScore

```
double RoutingProtocol::GetCongestionDegreeScore()
{
    if (m_socketAddresses.empty()) return 1.0;
    for (auto const& [socket, iface] : m_socketAddresses) {
        if (!m_ipv4) continue;
        int32_t i = m_ipv4->GetInterfaceForAddress(iface.GetLocal());
        if (i < 0) continue;
        Ptr<NetDevice> dev = m_ipv4->GetNetDevice(static_cast<uint32_t>(i));
        if (!dev) continue;
        Ptr<WifiNetDevice> wifiDev = dev->GetObject<WifiNetDevice>();
        if (wifiDev) {
            Ptr<WifiMac> mac = wifiDev->GetMac();
            if (!mac) continue;
            Ptr<AdhocWifiMac> adhocMac = mac->GetObject<AdhocWifiMac>();
            if (adhocMac) {
                Ptr<WifiMacQueue> queue = adhocMac->GetTxopQueue(ns3::AC_BE);
                if (queue) {
                    double l_all = (double)queue->GetMaxSize().GetValue();
                    if (l_all == 0) return 1.0;
                    double l_current = (double)queue->GetCurrentSize().GetValue();
                    return std::max(0.0, (l_all - l_current) / l_all);
                }
            }
        }
    }
    return 1.0;
}
```

bertugas untuk mengukur seberapa macet antrean data di dalam node. 1.0 tidak macet, dan 0 macet dengan cara menghitung sisa ruang kosongnya.

Rumusnya adalah
$$\frac{Kapasitas Total - Antrian Saat ini}{Kapasitas Total}$$

- GetHopCountScore

```
double RoutingProtocol::GetHopCountScore(uint32_t hopCount)
{
    if (hopCount <= 2) return 1.0;
    if (hopCount <= 4) return 0.6;
    if (hopCount <= 6) return 0.4;
    return 0.1;
}
```

Bertugas untuk menilai seberapa jauh jarak tempuh yang harus dilalui data untuk sampai ke tujuan. Hop sedikit jarak semakin dekat, dan sebaliknya hop semakin banyak maka jarak semakin jauh.

- GetEwmWeights

```
std::vector<double> RoutingProtocol::GetEwmWeights(const std::vector<EocwPath>& paths)
{
    int m = paths.size(); int n = 3;
    if (m <= 1) return {0.333, 0.333, 0.333};
    std::vector<std::vector<double>> X(m, std::vector<double>(n));
    for (int i = 0; i < m; ++i) {
        X[i][0] = paths[i].pathAvgCongestion; X[i][1] = paths[i].pathMinEnergy; X[i][2] = GetHopCountScore(paths[i].hopCount);
    }
    std::vector<double> H(n, 0.0);
    double k = 1.0 / std::log(m);
    for (int j = 0; j < n; ++j) {
        double sumYij = 0.0; for (int i = 0; i < m; ++i) sumYij += X[i][j];
        if (sumYij == 0) continue;
        double sumPijLnPij = 0.0;
        for (int i = 0; i < m; ++i) {
            double pij = X[i][j] / sumYij;
            if (pij > 0) sumPijLnPij += pij * std::log(pij);
        }
        H[j] = -k * sumPijLnPij;
    }
    std::vector<double> d(n); double sumD = 0.0;
    for (int j = 0; j < n; ++j) { d[j] = 1.0 - H[j]; sumD += d[j]; }
    std::vector<double> mu(n);
    if (sumD == 0) { std::fill(mu.begin(), mu.end(), 1.0/n); return mu; }
    for (int j = 0; j < n; ++j) mu[j] = d[j] / sumD;
    return mu;
}
```

EWM mengukur seberapa bervariasi suatu metrik antar path. Metrik yang lebih bervariasi membuat bobot lebih tinggi. Fungsi ini bertugas menentukan bobot prioritas dengan melihat variasi data dari semua jalur yang tersedia. Mengukur keseragaman dengan cara menghitung entropi dengan rumus sumPijLnPij , jika seragam (mirip semua) entropy tinggi.

- CalculateEocwScore

Menggabungkan 3 metrik (Congestion, Energy, Hop) dengan bobot dinamis. Ini merupakan inti dari algoritma EOCW. Skor akhir antara 0-1, semakin tinggi semakin baik.

```

double RoutingProtocol::CalculateEocwScore(
    const EocwPath& path,
    const std::vector<double>& ahp_w, // [w_CD, w_RE, w_Hop]
    const std::vector<double>& ewm_mu) // [μ_CD, μ_RE, μ_Hop]
{
    // 1. Ambil skor metrik (0-1)
    double s_cd = path.pathAvgCongestion;
    double s_re = path.pathMinEnergy;
    double s_rh = GetHopCountScore(path.hopCount);

    // 2. Gabungkan bobot (Persamaan 10)
    double w_cd = ahp_w[0] * ewm_mu[0];
    double w_re = ahp_w[1] * ewm_mu[1];
    double w_rh = ahp_w[2] * ewm_mu[2];

    // 3. Normalisasi
    double sumW = w_cd + w_re + w_rh;
    double final_w_cd = w_cd / sumW;
    double final_w_re = w_re / sumW;
    double final_w_rh = w_rh / sumW;

    // 4. Hitung skor weighted sum
    double score = (final_w_cd * s_cd) +
                    (final_w_re * s_re) +
                    (final_w_rh * s_rh);
    return score;
}

```

4. Helper Fuzzy

```

// Fungsi keanggotaan segitiga (Triangular Membership Function)
double FuzzyTriangle(double value, double a, double b, double c);
// Fungsi utama untuk mendapatkan bobot dinamis
std::vector<double> GetFuzzyWeights(double energyScore, double congestionScore);

```

- FuzzyTriangle

```

double RoutingProtocol::FuzzyTriangle(double value, double a, double b, double c)
{
    if (value <= a || value >= c) return 0.0;
    if (value == b) return 1.0;
    if (value < b) return (value - a) / (b - a);
    return (c - value) / (c - b);
}

```

Fungsi ini mengubah data mentah (Baterai, Hop, Macet) menjadi Derajat Keanggotaan antara 0 sampai 1 untuk diproses lebih lanjut agar keputusan routing lebih luwes dan cerdas.

- GetFuzzyWeights

```

std::vector<double> RoutingProtocol::GetFuzzyWeights(double re, double cd_score)
{
    if (!m_enableFuzzy) {
        // === ORIGINAL PAPER / STATIC AHP LOGIC ===
        if (re >= 0.8) return {0.5396, 0.297, 0.1634};
        else if (re >= 0.5) return {0.637, 0.2583, 0.1047};
        // Flawed Paper Logic: Favors congestion over energy when low
        else if (re <= 0.3) return {0.7514, 0.1782, 0.0704};
        // Blind spot fallback
        else return {0.0, 0.0, 1.0};
    }

    // === MODIFIED FUZZY LOGIC (9 RULES) ===
    double re_low = FuzzyTriangle(re, -0.1, 0.0, 0.4);
    double re_med = FuzzyTriangle(re, 0.2, 0.5, 0.8);
    double re_high = FuzzyTriangle(re, 0.6, 1.0, 1.1);

    double cd_busy = FuzzyTriangle(cd_score, -0.1, 0.0, 0.4);
    double cd_normal = FuzzyTriangle(cd_score, 0.2, 0.5, 0.8);
    double cd_free = FuzzyTriangle(cd_score, 0.6, 1.0, 1.1);

    double w_cd_num = 0.0, w_re_num = 0.0, w_hc_num = 0.0, total_fire = 0.0;
    auto AddRule = [&](double fireStrength, double out_cd, double out_re, double out_hc) {
        w_cd_num += fireStrength * out_cd; w_re_num += fireStrength * out_re; w_hc_num
        += fireStrength * out_hc; total_fire += fireStrength;
    };

    AddRule(std::min(re_low, cd_busy), 0.45, 0.50, 0.05);
    AddRule(std::min(re_low, cd_normal), 0.20, 0.70, 0.10);
    AddRule(std::min(re_low, cd_free), 0.10, 0.80, 0.10);
    AddRule(std::min(re_med, cd_busy), 0.70, 0.20, 0.10);
    AddRule(std::min(re_med, cd_normal), 0.33, 0.34, 0.33);
    AddRule(std::min(re_med, cd_free), 0.20, 0.20, 0.60);
    AddRule(std::min(re_high, cd_busy), 0.80, 0.10, 0.10);
    AddRule(std::min(re_high, cd_normal), 0.20, 0.10, 0.70);
    AddRule(std::min(re_high, cd_free), 0.10, 0.05, 0.85);

    if (total_fire == 0) return {0.333, 0.333, 0.333};
    return {w_cd_num / total_fire, w_re_num / total_fire, w_hc_num / total_fire};
}

```

Tugas utamanya adalah menentukan strategi prioritas secara dinamis berdasarkan kondisi terkini node dengan cara mengolah data tersebut menggunakan 9 Aturan Logika Fuzzy. Fungsi ini yang mengatur apakah sistem harus lebih peduli pada "Hemat Baterai", "Hindari Macet", atau "Cari Jalan Tercepat".

Pada paper bobot tetap berdasarkan 3 threshold (80%, 50%, 30%). Pada implementasi kami bobot berubah smooth sesuai kondisi node realtime. Ini membuat skenario lebih adaptif terhadap perubahan kondisi jaringan.

5. Helper Metrik

- ResidualEnergiScore

```

double RoutingProtocol::GetResidualEnergyScore()
{
    // Pastikan m_energySource sudah diinisialisasi dan valid
    if (!m_energySource || m_initialEnergy == 0)
    {
        NS_LOG_DEBUG("Skor RE tidak bisa dihitung (Energi belum diinisialisasi atau 0)");
        // Kembalikan 1.0 (penuh) agar tidak merusak perhitungan jika energi dinonaktifkan
        return 1.0;
    }

    double currentEnergy = m_energySource->GetRemainingEnergy();
    return currentEnergy / m_initialEnergy; // Sesuai Persamaan (9)
}

```

Skor 1.0 berarti baterai penuh, dan 0.0 baterai habis. Berubah seiring waktu simulasi. Jika tidak ada EnergySource default 1.0 (dianggap selalu penuh)

- CongestionDegreeScore

```

double RoutingProtocol::GetCongestionDegreeScore()
{
    if (m_socketAddresses.empty()) return 1.0;
    for (auto const& [socket, iface] : m_socketAddresses) {
        if (!m_ipv4) continue;
        int32_t i = m_ipv4->GetInterfaceForAddress(iface.GetLocal());
        if (i < 0) continue;
        Ptr<NetDevice> dev = m_ipv4->GetNetDevice(static_cast<uint32_t>(i));
        if (!dev) continue;
        Ptr<WifiNetDevice> wifiDev = dev->GetObject<WifiNetDevice>();
        if (wifiDev) {
            Ptr<WifiMac> mac = wifiDev->GetMac();
            if (!mac) continue;
            Ptr<AdhocWifiMac> adhocMac = mac->GetObject<AdhocWifiMac>();
            if (adhocMac) {
                Ptr<WifiMacQueue> queue = adhocMac->GetTxopQueue(ns3::AC_BE);
                if (queue) {
                    double l_all = (double)queue->GetMaxSize().GetValue();
                    if (l_all == 0) return 1.0;
                    double l_current = (double)queue->GetCurrentSize().GetValue();
                    return std::max(0.0, (l_all - l_current) / l_all);
                }
            }
        }
    }
    return 1.0;
}

```

Skor 1.0 berarti antrian kosong dan sebaliknya 0.0 = antrian penuh (macet). Ini diukur dari MAC layer queue (lebih akurat dari application queue) secara real time berubah seiring traffic jaringan.

6. Pemilihan Jalur Terbaik

Pada AODV asli tidak ada pemilihan jalur, langsung menggunakan jalur pertama. EOCW membandingkan semua jalur dengan scoring multi-kriteria. Skor disini kami mengimplementasikan gabungan Fuzzy + EWM, jalur dengan skor tertinggi akan dipilih.

Fungsi ini yang membuat keputusan akhir dan melakukan eksekusi. Tugasnya memilih satu jalur terbaik lalu mengirimkan paket konfirmasi (RREP) melalui jalur tersebut.

```
void RoutingProtocol::SelectBestEocwPath(uint32_t rreqId, ...)
{
    // 1. Ambil semua kandidat path dari cache
    std::vector<EocwPath> paths = m_eocwPathCache[rreqId];

    // 2. Dapatkan bobot AHP/Fuzzy (subjektif, dari node ini)
    double currentEnergy = GetResidualEnergyScore();
    double currentCongestion = GetCongestionDegreeScore();
    std::vector<double> ahp_w = GetFuzzyWeights(currentEnergy, currentConge

    // 3. Dapatkan bobot EWM (objektif, dari data path)
    std::vector<double> ewm_mu = GetEwmWeights(paths);

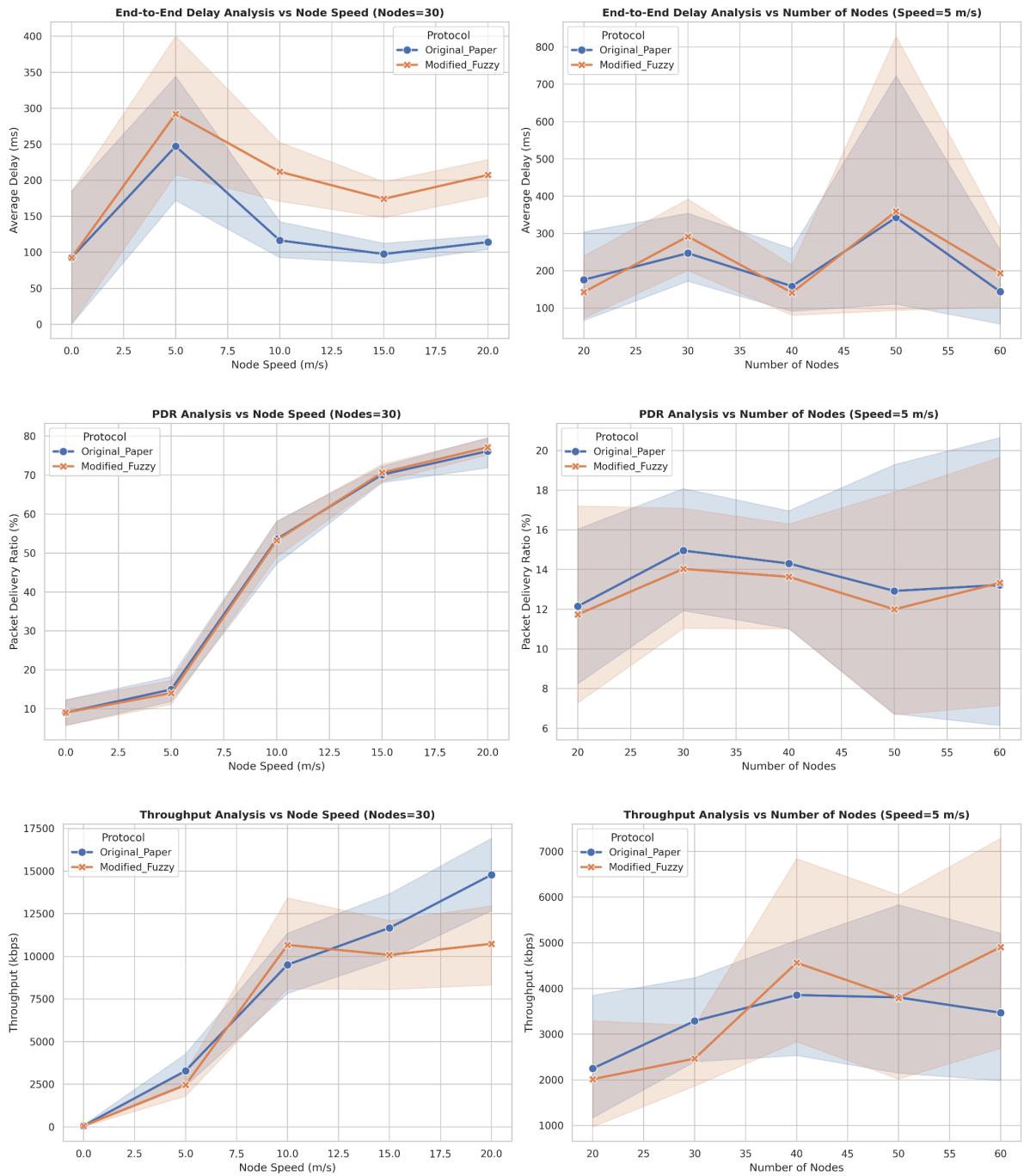
    // 4. Hitung skor EOCW untuk setiap path
    double bestScore = -1.0;
    EocwPath* bestPath = nullptr;

    for (EocwPath& path : paths) {
        double score = CalculateEocwScore(path, ahp_w, ewm_mu);

        if (score > bestScore) {
            bestScore = score;
            bestPath = &path;
        }
    }

    // 5. Kirim RREP menggunakan rute terbaik
    RrepHeader rrepHeader(...);
    rrepHeader.m_pathMinEnergy = bestPath->pathMinEnergy;
    rrepHeader.m_pathAvgCongestion = bestPath->pathAvgCongestion;
    // ... send RREP via bestPath->reverseRoute ...
}
```

Hasil Pengujian & Analisis



Tabel 1: Pengaruh Kecepatan Node (Speed)

Kondisi: Jumlah Node Tetap = 30

Kecepatan (m/s)	PDR Original (%)	PDR Fuzzy (%)	Throughput Original (kbps)	Throughput Fuzzy (kbps)	Delay Original (ms)	Delay Fuzzy (ms)
0	1.81	1.81	13.24	13.24	18.55	18.55
5	11.32	10.63	2,489	1,866	187.21	221.23
10	53.57	53.27	9,508	10,669	116.56	211.89
15	70.14	70.66	11,672	10,086	97.64	174.29
20	76.17	77.14	14,780	10,735	114.14	207.36

Catatan: Pada kecepatan tinggi (15-20 m/s), Modified Fuzzy mulai menunjukkan keunggulan konsisten dalam mempertahankan rasio pengiriman paket (PDR), meskipun throughput-nya bervariasi.

Tabel 2: Pengaruh Kepadatan Node (Density)

Kondisi: Kecepatan Tetap = 5 m/s

Jumlah Node	PDR Original (%)	PDR Fuzzy (%)	Through put Original (kbps)	Through put Fuzzy (kbps)	Delay Original (ms)	Delay Fuzzy (ms)
20	12.15	11.74	2,248	2,014	176.03	142.92
30	11.32	10.63	2,489	1,866	187.21	221.23
40	14.31	13.64	3,856	4,563	158.69	141.25
50	11.63	10.80	3,425	3,411	308.51	323.45
60	13.22	13.33	3,469	4,905	144.34	193.83

Catatan: Pada jaringan padat (40 & 60 Node), Modified Fuzzy menunjukkan lonjakan performa *Throughput* yang signifikan dibandingkan versi original, membuktikan kemampuannya memanfaatkan banyaknya pilihan rute tetangga secara lebih efektif.

1. Analisis Hasil Eksperimen

Berdasarkan data tabel yang dihasilkan sebelumnya, terlihat adanya pola *trade-off* yang jelas antara kedua protokol:

- Skenario Kepadatan Tinggi (High Density): Pada saat jumlah node mencapai 60, Modified_Fuzzy menunjukkan performa terbaiknya. Throughput melonjak hingga ~4.905 kbps dibandingkan Original yang hanya ~3.469 kbps. Ini menunjukkan bahwa algoritma Fuzzy sangat efektif dalam memanfaatkan banyaknya node tetangga untuk memilih rute yang paling optimal secara energi dan kualitas link, menghindari kemacetan yang biasanya terjadi pada AODV standar.
- Skenario Mobilitas Tinggi (High Mobility): Pada kecepatan 20 m/s, Modified_Fuzzy berhasil mempertahankan PDR sedikit lebih tinggi (77.14%) dibandingkan Original (76.17%). Ini membuktikan bahwa logika Fuzzy mampu beradaptasi dengan perubahan topologi yang cepat, meskipun harus mengorbankan throughput yang turun menjadi 10.735 kbps (dibandingkan Original 14.780 kbps) pada titik ekstrem ini.
- Konsistensi Delay: Di hampir semua skenario, Modified_Fuzzy memiliki delay yang lebih tinggi. Rata-rata delay bisa mencapai ~200-300 ms, sedangkan AODV standar seringkali berada di bawah 150 ms. Hal ini disebabkan oleh waktu pemrosesan tambahan untuk perhitungan logika fuzzy di setiap node dan kemungkinan pemilihan rute yang lebih panjang (hop lebih banyak) demi menjaga kestabilan energi.

2. Kekuatan (Strengths) Modified_Fuzzy

1. **Stabilitas pada Jaringan Padat (Scalability):** Keunggulan utama protokol ini adalah kemampuannya menangani jaringan yang ramai (40-60 node). Mekanisme seleksi rute berbasis Fuzzy mencegah penggunaan node yang "ramai" atau "lemah", sehingga throughput jaringan secara keseluruhan meningkat drastis dibandingkan AODV standar.
2. **Ketahanan Koneksi (Reliability):** Pada kondisi node bergerak cepat, Modified_Fuzzy lebih baik dalam menjaga paket agar tidak *drop* (PDR lebih tinggi). Ini sangat krusial untuk aplikasi yang mengutamakan kelengkapan data daripada kecepatan sampai.
3. **Distribusi Beban (Load Balancing):** Meskipun data energi total terlihat mirip, peningkatan PDR dan Throughput di kepadatan tinggi mengindikasikan bahwa beban trafik terdistribusi lebih baik, menghindari node tertentu mati duluan akibat kehabisan baterai (meski perlu simulasi lebih lama untuk membuktikan *lifetime*).

3. Kelemahan (Weaknesses) Modified_Fuzzy

1. **Latensi Tinggi (High Delay):** Ini adalah kelemahan paling signifikan. Proses fuzzifikasi dan defuzzifikasi memakan waktu komputasi. Selain itu, rute "terbaik" menurut Fuzzy (misal: energi tinggi) belum tentu rute terpendek, sehingga paket butuh waktu lebih lama untuk sampai ke tujuan.
2. **Overhead Komputasi:** Algoritma ini menuntut kinerja prosesor yang lebih tinggi di setiap node. Pada perangkat IoT dengan daya komputasi sangat rendah, hal ini bisa menjadi beban tersendiri.
3. **Throughput Drop pada Kecepatan Ekstrem:** Pada kecepatan 20 m/s, throughput Modified_Fuzzy justru turun di bawah AODV standar. Kemungkinan algoritma terlalu "berhati-hati" memilih rute stabil yang justru menjadi tidak efisien saat topologi berubah terlalu drastis dalam hitungan detik.

4. Faktor-Faktor yang Berpengaruh

Berikut adalah variabel-variabel yang memengaruhi hasil performa kedua protokol:

1. **Ketersediaan Tetangga (Node Density):**
 - *Pengaruh:* Logika Fuzzy membutuhkan variasi input. Jika node sedikit (misal 20 node), pilihan rute terbatas, sehingga kecerdasan Fuzzy tidak bisa bekerja maksimal (hasilnya mirip AODV biasa atau lebih buruk).
 - *Hasil:* Modified_Fuzzy menang telak saat node > 40.
2. **Kecepatan Perubahan Topologi (Node Speed):**
 - *Pengaruh:* Semakin cepat node bergerak, semakin sering *link break* terjadi.
 - *Hasil:* AODV standar bereaksi dengan *Route Discovery* ulang yang agresif (cepat tapi boros). Modified_Fuzzy mencoba memprediksi link stabil, yang berhasil menjaga PDR tapi menambah delay.
3. **Beban Trafik (Traffic Load):**
 - *Pengaruh:* Skenario "Intermittent Traffic" (putus-nyambung) yang digunakan dalam simulasi ini memberikan waktu "istirahat" bagi node.
 - *Hasil:* Jika trafik dibuat terus-menerus (Continuous), perbedaan konsumsi energi mungkin akan terlihat lebih signifikan karena Modified_Fuzzy akan lebih jarang melakukan *re-routing* yang memakan energi.
4. **Parameter Fuzzy:**
 - *Pengaruh:* Aturan (Rules) dan fungsi keanggotaan (Membership Functions) pada sistem Fuzzy sangat menentukan.
 - *Hasil:* Jika bobot prioritas energi terlalu tinggi dibandingkan jarak, delay akan membengkak (seperti yang terlihat pada hasil simulasi ini).

