T-PAODV (Trust-Prior AODV): Implementasi Mekanisme Keamanan MANET Berbasis *Trust-Level* pada Protokol PAODV untuk Lingkungan VANET

Disusun Oleh:

Muhammad Rifqy Febryantoni : 5025231106
Kevin Leonard Berutu : 5025231089

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Vehicular Ad-hoc Networks (VANET) merupakan sub-kategori dari *Mobile Ad-hoc Networks* (MANET) yang memfasilitasi komunikasi antar kendaraan. Karakteristik utama VANET adalah mobilitas node (kendaraan) yang sangat tinggi dan pola pergerakan yang terbatas dan bergantung pada bentuk lintasan kendaraan, menyebabkan topologi jaringan berubah secara sangat dinamis dan cepat. Dalam lingkungan yang dinamis ini, efisiensi *routing* menjadi sebuah tantangan. Protokol *routing* yang populer dan paling umum digunakan, seperti *Ad-hoc On-demand Distance Vector* (AODV), sering kali menghadapi kendala. AODV cenderung menghasilkan *control overhead* yang tinggi akibat mekanisme *flooding* paket *Route Request* (RREQ) ke seluruh jaringan untuk menemukan rute. Selain itu, pergerakan kendaraan yang cepat sering menyebabkan *broken links*, memaksa protokol untuk terus-menerus melakukan perbaikan rute atau melakukan pencarian rute ulang, hal ini dapat membebani *bandwidth* jaringan.

Untuk mengatasi permasalahan efisiensi pada AODV di lingkungan VANET, penelitian sebelumnya mengusulkan protokol PAODV (*Prior AODV*). PAODV menawarkan pendekatan cerdas untuk mengurangi *overhead* dengan memodifikasi mekanisme penyebaran RREQ. Dalam PAODV, penyebaran RREQ dibatasi hanya kepada *Prior Neighbors* (tetangga yang berada pada jarak lebih jauh) dan mengabaikan *Overhead Neighbors* (tetangga yang dekat). Pendekatan berbasis jarak ini terbukti menghasilkan rute yang lebih stabil dengan jumlah *hop* yang lebih sedikit, sehingga cocok untuk karakteristik jaringan berkecepatan tinggi seperti VANET. Meskipun PAODV menawarkan efisiensi yang lebih baik, protokol ini tidak dirancang dengan fitur keamanan bawaan. Sama seperti AODV standar, PAODV rentan terhadap berbagai serangan jaringan, khususnya serangan Black Hole. Dalam serangan ini, node jahat (*malicious node*) akan mengeksploitasi mekanisme *route discovery* dengan segera membalas RREQ menggunakan *Route Reply* (RREP) palsu yang mengklaim memiliki rute terpendek menuju tujuan. Hal ini menarik aliran data untuk melewati node jahat tersebut, yang kemudian membuang (*drop*) semua paket data yang diterima tanpa meneruskannya.

Di sisi lain, penelitian sebelumnya dalam konteks MANET mengusulkan solusi keamanan untuk mengatasi serangan *black hole* melalui AODV Berbasis Kepercayaan (*Trust-Based AODV*). Protokol ini memperkenalkan konsep *Trust Level* (TL) dan mekanisme "*Trust Test*" untuk memverifikasi kebenaran sebuah *neighbor node* sebelum menerapkan rute. Mekanisme ini memungkinkan node untuk mendeteksi anomali pada balasan RREP dan mengisolasi node yang berperilaku mencurigakan. Oleh karena itu, penelitian ini mengusulkan pengembangan protokol hibrida bernama T-PAODV (*Trust-Prior AODV*), yang bertujuan untuk menciptakan protokol *routing* yang tidak hanya efisien dalam hal *overhead* dan stabilitas untuk VANET, tetapi juga *resilient* terhadap serangan *black hole*.

1.2 Tujuan Penelitian

Berdasarkan latar belakang di atas, tujuan penelitian ini adalah:

1. Implementasi PAODV pada NS-3: Melakukan porting dan implementasi algoritma protokol *routing* PAODV (yang semula berbasis GloMoSim) ke dalam lingkungan simulator NS-3.
2. Validasi Klaim PAODV: Mengevaluasi kinerja PAODV hasil implementasi di NS-3 untuk memverifikasi klaim penelitian terdahulu terkait penurunan *Routing Overhead* dan peningkatan *Route Stability* dibandingkan AODV standar.

3. Pengembangan T-PAODV: Mengadaptasi dan mengintegrasikan mekanisme keamanan *Trust-Based* (Trust Level dan Trust Test) ke dalam protokol PAODV yang telah diimplementasikan di ns3.
4. Evaluasi Kinerja Hibrida: Menganalisis kinerja protokol T-PAODV dalam skenario serangan *black hole*, serta mengukur *trade-off* antara keamanan yang diperoleh dengan biaya tambahan (*overhead* dan *delay*) yang ditimbulkan.

1.3 Perumusan Masalah

Penelitian ini berangkat dari kebutuhan untuk memvalidasi efisiensi protokol sebelum mengamankannya. Rumusan masalah yang diangkat adalah:

1. Bagaimana mengimplementasikan algoritma seleksi tetangga (*Neighbor Selection*) berbasis jarak milik protokol PAODV ke dalam arsitektur simulator NS-3?
2. Apakah implementasi PAODV pada NS-3 terbukti mampu menurunkan *Routing Overhead* dan jumlah *Broken Links* secara signifikan dibandingkan AODV standar, sesuai dengan klaim pada literatur aslinya?
3. Bagaimana cara mengintegrasikan mekanisme *Trust Test* ke dalam alur *Route Discovery* PAODV tanpa merusak efisiensi yang ditawarkan protokol tersebut?
4. Bagaimana perbandingan kinerja (PDR, *Delay*, dan *Overhead*) antara AODV, PAODV, dan T-PAODV ketika diuji di bawah kondisi jaringan normal maupun di bawah serangan *black hole*?

1.4 Kontribusi

Penelitian ini memberikan kontribusi sebagai berikut:

1. Validasi Lintas-Simulator: Menyediakan implementasi kode protokol PAODV yang berjalan pada simulator NS-3 (sebagai pembaruan dari implementasi GloMoSim terdahulu) dan memberikan bukti empiris terkait validitas klaim efisiensi PAODV dalam lingkungan simulasi yang baru.
2. Protokol Hibrida Aman (T-PAODV): Mengusulkan desain protokol baru yang menggabungkan efisiensi seleksi rute PAODV dengan keamanan Trust-Based AODV.
3. Analisis Komprehensif: Memberikan analisis mendalam mengenai perilaku ketiga protokol (AODV, PAODV, T-PAODV) yang mencakup aspek efisiensi (overhead), stabilitas (link breaks), dan keamanan (packet loss akibat serangan), sehingga memberikan gambaran utuh mengenai trade-off antara kinerja dan keamanan di VANET

BAB II
TINJAUAN PUSTAKA

2.1 Vehicular Ad-hoc Networks (VANET)

Vehicular Ad-hoc Networks (VANET) didefinisikan oleh Hartenstein dan Laberteaux (2008) sebagai paradigma jaringan yang memungkinkan komunikasi antar kendaraan serta antara kendaraan dengan infrastruktur di tepi jalan. Karakteristik fundamental yang membedakan VANET dari MANET konvensional adalah pola mobilitas node yang terbatas pada topologi jalan raya, kecepatan node yang sangat tinggi, serta sumber daya energi yang relatif tidak terbatas (dari mesin kendaraan).

Menurut Li dan Wang (2007), komunikasi dalam VANET terbagi menjadi dua arsitektur utama: Vehicle-to-Vehicle (V2V) yang murni bersifat ad-hoc, dan Vehicle-to-Infrastructure (V2I) yang melibatkan Road Side Units (RSU). Bernsen dan Manivannan (2009) menekankan bahwa tantangan terbesar dalam routing VANET adalah ketidakstabilan link akibat fragmentasi jaringan yang sering terjadi saat kendaraan bergerak menjauh dengan cepat, yang menuntut protokol routing untuk memiliki kemampuan adaptasi yang tinggi terhadap perubahan topologi.

2.2 Protokol Routing AODV

Ad-hoc On-demand Distance Vector (AODV) merupakan protokol routing reaktif yang telah distandarisasi oleh IETF dalam dokumen RFC 3561 (Perkins et al., 2003). Sifat "reaktif" berarti AODV hanya membangun rute saat ada permintaan pengiriman data (on-demand). Mekanisme kerja AODV, sebagaimana dijelaskan oleh Royer dan Perkins (1999), terdiri dari dua prosedur utama:

- Route Discovery: Menggunakan mekanisme expanding ring search untuk menyebarkan paket Route Request (RREQ) guna menemukan tujuan.
- Route Maintenance: Menggunakan paket Route Error (RERR) untuk memberitahu pengirim jika terjadi kegagalan tautan (link break).

Meskipun AODV efektif pada jaringan dengan mobilitas rendah, Ni et al. (1999) dalam penelitian fundamentalnya mengidentifikasi fenomena "Broadcast Storm Problem". Mereka menjelaskan bahwa mekanisme flooding RREQ secara buta (blind flooding)—di mana setiap node meneruskan paket ke semua tetangga—dapat menyebabkan redundansi paket yang masif, persaingan medium (kontensi), dan tabrakan paket (collision), yang secara drastis menurunkan kinerja jaringan pada lingkungan padat seperti VANET.

2.3 Protokol PAODV (Prior AODV)

Untuk memitigasi masalah *Broadcast Storm* yang diidentifikasi oleh Ni et al. (1999), Abedi et al. (2009) mengembangkan protokol modifikasi bernama **PAODV (*Prior AODV*)**. Protokol ini mengintegrasikan informasi posisi relatif node untuk membatasi penyebaran RREQ, sebuah pendekatan yang bertujuan menyeimbangkan efisiensi *overhead* dengan stabilitas rute.

2.3.1 Klasifikasi Tetangga (*Neighbor Classification*)

PAODV mengklasifikasikan tetangga menjadi dua kategori berdasarkan posisi mereka relatif terhadap radius transmisi (R) dan jarak ambang batas (*threshold distance*, R1):

1. **Prior Neighbors:** Node tetangga yang berada di **Prior Zone**, yaitu area antara jarak ambang batas (R1) hingga batas jangkauan transmisi (R). Tetangga ini diprioritaskan karena memilih node yang lebih jauh akan mengurangi jumlah *hop* menuju tujuan, sehingga rute menjadi lebih pendek dan efisien.
2. **Overhead Neighbors:** Node tetangga yang berada di **Overhead Zone**, yaitu area yang sangat dekat dengan node pengirim (jarak < R1). Mengirim paket ke node ini dianggap tidak efisien karena hanya memberikan kemajuan jarak yang kecil, sehingga meningkatkan jumlah *hop* dan *overhead*.

2.3.2 Mekanisme Seleksi Rute

Dalam PAODV, node sumber atau node perantara tidak melakukan *broadcast* RREQ ke semua tetangga. Sebaliknya, protokol membatasi jumlah RREQ yang dikirim (*RREQ Bound*) dan memprioritaskan pengiriman secara *unicast* kepada *Prior Neighbors*. Jika jumlah *Prior Neighbors* kurang dari batas kuota, barulah sisa kuota digunakan untuk *Overhead Neighbors*.

Pendekatan ini diklaim mampu mengurangi *routing overhead*, memperpendek panjang rute (jumlah *hop*), dan meningkatkan stabilitas rute karena mengurangi kemungkinan memilih *link* yang rapuh atau tidak efisien.

2.4 Black Hole Attack

Sifat terbuka dari jaringan ad-hoc membuatnya rentan terhadap berbagai serangan. Tamilselvan dan Sankaranarayanan (2008) mendefinisikan Black Hole Attack sebagai jenis serangan Denial of Service (DoS) di mana node jahat memanfaatkan kelemahan pada fase Route Discovery.

Dalam protokol berbasis vektor jarak seperti AODV, node mempercayai rute yang memiliki Sequence Number tertinggi karena dianggap sebagai rute terbaru. Banerjee menjelaskan bahwa penyerang mengeksploitasi celah ini dengan segera membalas RREQ menggunakan paket Route Reply (RREP) palsu yang berisi Sequence Number yang sangat besar. Hal ini "menipu" node sumber untuk percaya bahwa penyerang memiliki rute terbaik menuju tujuan. Setelah lalu lintas data dialihkan ke node penyerang, seluruh paket akan dibuang (dropped) tanpa diteruskan, menyebabkan putusnya komunikasi total.Mekanisme Keamanan Berbasis Kepercayaan (Trust-Based)

2.5 Mekanisme Keamanan Berbasis Kepercayaan (Trust-Based)

Untuk mengatasi serangan *black hole*, Gharehkoolchian et al. (2015) mengusulkan mekanisme keamanan berbasis kepercayaan (*Trust-Based AODV*). Metode ini tidak menggunakan kriptografi yang berat, melainkan menggunakan logika verifikasi perilaku tetangga.

2.5.1 Trust Level (TL)

Setiap node dalam jaringan menyimpan tabel kepercayaan (*Trust Table*) untuk tetangganya. Nilai kepercayaan dikategorikan sebagai berikut:

- **TL = 1 (Initial):** Diberikan kepada tetangga baru atau node yang baru lepas dari hukuman. Node dengan status ini dianggap mencurigakan dan perlu diverifikasi.
- **TL = 2 (Trusted):** Diberikan kepada node yang telah lulus uji kepercayaan. Paket dari node ini akan diterima dan diteruskan.
- **TL = 0 (Blacklist):** Diberikan kepada node yang terbukti berbuat jahat. Semua paket dari node ini akan dibuang (*drop*).

2.5.2 Trust Test dan Examiner Node

Ketika sebuah node (disebut **Examiner Node** atau EN) menerima RREP dari tetangga yang memiliki status **TL=1**, EN tidak langsung mempercayainya. EN akan menahan RREP tersebut dan melakukan **"Trust Test"**.

Prosedur *Trust Test* adalah sebagai berikut:

1. EN mengirimkan paket RREQ palsu (dummy) kepada node yang dicurigai (Suspect Node/SN).
2. RREQ palsu ini berisi alamat tujuan yang diset sebagai alamat IP milik EN sendiri.
3. Jika SN adalah node jujur, ia tidak akan membalas karena ia tahu ia tidak memiliki rute "ke diri EN" yang lebih baik dari EN itu sendiri, atau ia akan membalas dengan data yang wajar.
4. Jika SN adalah *Black Hole*, ia akan secara otomatis membalas dengan RREP palsu yang mengklaim memiliki rute ke tujuan (EN) dengan *Sequence Number* yang sangat tinggi.

5. EN memverifikasi balasan tersebut. Jika *Sequence Number* dalam balasan jauh lebih tinggi dari *Sequence Number* milik EN sendiri, maka SN terbukti berbohong. SN langsung dimasukkan ke *blacklist* (TL=0).

2.6 Network Simulator 3 (NS-3)

NS-3 adalah simulator jaringan *discrete-event* yang dikembangkan sebagai penerus NS-2 dan menjadi standar *de facto* dalam penelitian jaringan akademis. Henderson et al. (2008) menjelaskan bahwa NS-3 didesain dengan arsitektur modular berbasis C++ yang memungkinkan pemodelan protokol yang sangat mendekati implementasi dunia nyata. Berbeda dengan simulator GloMoSim yang digunakan pada penelitian PAODV terdahulu (Abedi et al., 2009), NS-3 menyediakan model mobilitas yang lebih realistis dan tumpukan protokol WiFi (IEEE 802.11) yang lebih akurat, sehingga hasil evaluasi pada NS-3 dianggap lebih representatif untuk implementasi modern.

BAB III
METODOLOGI

3.1 Lingkungan Simulasi

Untuk mengevaluasi kinerja protokol T-PAODV dan membandingkannya dengan AODV serta PAODV, digunakan lingkungan simulasi berbasis *Network Simulator 3* (NS-3). NS-3 dipilih karena kemampuannya dalam memodelkan protokol jaringan secara mendetail dan realistis, termasuk dukungan modul mobilitas dan WiFi standar (IEEE 802.11).

3.1.1 Persiapan dan Instalasi NS-3

Berikut adalah langkah-langkah instalasi simulator NS-3 versi 3.46 pada sistem operasi berbasis Linux (Ubuntu):

1. Unduh Source Code NS-3: Unduh rilis NS-3.46 resmi dari repositori.

```
wget https://www.nsnam.org/release/ns-allinone-3.46.tar.bz2
tar xjf ns-allinone-3.46.tar.bz2
cd ns-allinone-3.46/ns-3.46
```

2. Duplikasi dan Refactoring Kode (Pembuatan Template PAODV dan T-PAODV): Langkah ini bertujuan membuat modul baru (paodv) yang terisolasi dari modul asli (aodv) agar tidak terjadi konflik nama *class* atau variabel saat kompilasi. Setelah PAODV selesai dimodifikasi, langkah ini diulang untuk membuat T-PAODV.

```
cp -r src/aodv src/paodv

# Masuk ke direktori model
cd src/paodv/model
for file in aodv*; do mv "$file" "p${file}"; done

# Masuk ke direktori helper
cd ../helper
for file in aodv*; do mv "$file" "p${file}"; done

cd ../.. # Kembali ke root src/paodv

# Ganti semua huruf kapital (AODV -> PAODV)
grep -rl "AODV" . | xargs sed -i 's/AODV/PAODV/g'

# Ganti format CamelCase (Aodv -> Paodv) - Penting untuk TypeId
grep -rl "Aodv" . | xargs sed -i 's/Aodv/Paodv/g'

# Ganti huruf kecil (aodv -> paodv) - Penting untuk header includes
grep -rl "aodv" . | xargs sed -i 's/aodv/paodv/g'

# Lakukan yang sama untuk T-PAODV
```

3. Integrasi Modul PAODV dan T-PAODV: Lakukan konfigurasi ulang agar modul baru terdeteksi oleh sistem build.

```
./ns3 configure
./ns3 build
```

3.2 Skenario Simulasi

Evaluasi dilakukan menggunakan skrip simulasi simulation.cc yang dirancang untuk menguji tiga protokol (AODV, PAODV, T-PAODV) dalam kondisi normal dan di bawah serangan *Blackhole*. Berikut adalah penjelasan struktur dan konfigurasi skrip simulasi tersebut:

File | scratch/simulation.cc

```
/* * File: simulation.cc
* Deskripsi: Skrip pengujian kinerja protokol (AODV, PAODV, T-PAODV) * Fokus: Overhead Analysis, Route Stability, dan Security Resilience */
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/wifi-module.h"
// Mengimpor modul protokol routing yang akan diuji
#include "ns3/aodv-module.h"   // Protokol standar
#include "ns3/paodv-module.h"  // Protokol efisiensi (Prior AODV)
#include "ns3/tpaodv-module.h" // Protokol keamanan (Trust-Prior AODV)
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-module.h" // Modul untuk analisis statistik trafik
#include <chrono> // Untuk pengukuran waktu eksekusi (Computational Cost)

using namespace ns3;

int main(int argc, char *argv[])
{
   // ---------- PARAMETER SIMULASI ----------
   std::string protocol = "PAODV"; // Protokol default
   uint32_t nNodes = 50; // Jumlah node (High Density)
   bool malicious = false; // Flag serangan Blackhole
   int nMalicious = 5; // Jumlah node jahat (10% populasi)
   double simulationTime = 100.0;

   // Parsing argumen command line agar parameter bisa diubah saat runtime
   // Contoh: ./ns3 run "overhead_test --protocol=TPAODV --malicious=true"
   CommandLine cmd;
   cmd.AddValue("protocol", "Protocol (AODV, PAODV, TPAODV)", protocol);
   cmd.AddValue("nNodes", "Number of nodes", nNodes);
   cmd.AddValue("malicious", "Enable Blackhole Attack", malicious);
   cmd.Parse(argc, argv);

   // ---------- KONFIGURASI NODE & GRUP ----------
   NodeContainer nodes;
   nodes.Create(nNodes);

   // Memisahkan node Jujur (Good) dan Jahat (Bad)
   NodeContainer goodNodes, badNodes;
   if (malicious) {
      // Node terakhir dalam list dijadikan node jahat
      for (uint32_t i = 0; i < nNodes - nMalicious; ++i) goodNodes.Add(nodes.Get(i));
      for (uint32_t i = nNodes - nMalicious; i < nNodes; ++i) badNodes.Add(nodes.Get(i));
   } else {
      goodNodes = nodes;
   }

   // ---------- SETUP WIFI & MOBILITAS ----------
   WifiHelper wifi;
   wifi.SetStandard(WIFI_STANDARD_80211b); // Standar 802.11b
   YansWifiPhyHelper wifiPhy;
   YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
```

```
    wifiPhy.SetChannel (wifiChannel.Create ());

    WifiMacHelper wifiMac;
    wifiMac.SetType ("ns3::AdhocWifiMac");
    NetDeviceContainer devices = wifi.Install (wifiPhy, wifiMac, nodes);

    // Mobilitas: Random Waypoint (Node bergerak terus menerus)
    MobilityHelper mobility;
    // Menggunakan Position Allocator agar node tersebar acak di awal
    ObjectFactory pos;
    pos.SetTypeId("ns3::RandomRectanglePositionAllocator");
    pos.Set("X", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=500.0]"));
    pos.Set("Y", StringValue("ns3::UniformRandomVariable[Min=0.0|Max=500.0]"));
    Ptr<PositionAllocator> taPositionAlloc = pos.Create()->GetObject<PositionAllocator>();

    mobility.SetPositionAllocator(taPositionAlloc);
    mobility.SetMobilityModel("ns3::RandomWaypointMobilityModel",
                              "Speed", StringValue("ns3::UniformRandomVariable[Min=5.0|Max=20.0]"),
                              "Pause", StringValue("ns3::ConstantRandomVariable[Constant=2.0]"),
                              "PositionAllocator", PointerValue(taPositionAlloc));
    mobility.Install(nodes);

    // ---------- INSTALASI PROTOKOL ----------
    InternetStackHelper stack;

    if (protocol == "PAODV") {
        PAodvHelper paodvGood;
        paodvGood.Set("RreqBound", UintegerValue(2)); // Optimasi PAODV (Unicast ke 2
tetangga)
        stack.SetRoutingHelper(paodvGood);
        stack.Install(goodNodes);

        if (malicious) {
            PAodvHelper paodvBad;
            paodvBad.Set("RreqBound", UintegerValue(2));
            paodvBad.Set("IsMalicious", BooleanValue(true)); // Aktifkan Blackhole
            stack.SetRoutingHelper(paodvBad);
            stack.Install(badNodes);
        }
    }
    // ... (Logika serupa untuk TPAODV dan AODV) ...

    // ---------- TRAFIK GENERATOR ----------
    // Skenario: 10 Pasang Pengirim-Penerima Acak (Random Pairs)
    // Traffic ringan (Interval 1.0s) agar tidak terjadi congesti berlebih
    UdpEchoServerHelper echoServer(9);
    ApplicationContainer serverApps = echoServer.Install(nodes); // Server di semua node
(listen)
    UdpEchoClientHelper echoClient(Ipv4Address("0.0.0.0"), 9);
    echoClient.SetAttribute("MaxPackets", UintegerValue(100));
    echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));
    echoClient.SetAttribute("PacketSize", UintegerValue(1024));
    // Loop instalasi client acak
    for (int i = 0; i < 10; i++) {
        uint32_t srcNode = urng->GetInteger (0, nNodes - 1);
```

```
    uint32_t dstNode = urng->GetInteger (0, nNodes - 1);

    while (srcNode == dstNode || (malicious && srcNode >= nNodes - nMalicious)) {
       srcNode = urng->GetInteger (0, nNodes - 1);
    }

    Address remoteAddress = InetSocketAddress (interfaces.GetAddress (dstNode), port);
    echoClient.SetAttribute("Remote", AddressValue(remoteAddress));

    ApplicationContainer clientApps = echoClient.Install (nodes.Get (srcNode));
    clientApps.Start (Seconds (2.0 + i));
    clientApps.Stop (Seconds (simulationTime));
}

// ---------- PENGUKURAN KINERJA ----------
FlowMonitorHelper flowmon;
Ptr<FlowMonitor> monitor = flowmon.InstallAll();

// Timer untuk mengukur Computational Overhead (Waktu Eksekusi CPU)
auto start = std::chrono::high_resolution_clock::now();

Simulator::Stop(Seconds(simulationTime));
Simulator::Run();

auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> elapsed = end - start;
double executionTime = elapsed.count(); // Waktu nyata simulasi berjalan

// ---------- ANALISIS HASIL (POST-PROCESSING) ----------
// 1. Ekstraksi Statistik FlowMonitor (PDR & Hop Count)
uint32_t totalHops = 0;
uint32_t totalFlows = 0;
uint32_t totalTxPackets = 0;
uint32_t totalRxPackets = 0;

monitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowmon.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = monitor->GetFlowStats ();

for (auto const &flow : stats)
  {
    totalTxPackets += flow.second.txPackets;
    totalRxPackets += flow.second.rxPackets;
    if (flow.second.rxPackets > 0) {
       totalHops += flow.second.timesForwarded;
       totalFlows++;

    }
  }

double avgHops = (totalFlows > 0) ? (double)totalHops / totalFlows : 0.0;
double pdr = (totalTxPackets > 0) ? (double)totalRxPackets / totalTxPackets * 100.0 : 0.0;

// 2. Ekstraksi Statistik Protokol (Internal Counters)
```

```
// Mengambil data langsung dari objek protokol di tiap node
uint32_t totalRreq = 0;
uint32_t totalRrep = 0;
uint32_t totalRerr = 0;
uint64_t totalBrokenLinks = 0;
uint32_t totalRreqRecv = 0;
uint32_t totalMaliciousDrops = 0;

for (uint32_t i = 0; i < nNodes; i++)
  {
   Ptr<Ipv4> ipv4 = nodes.Get(i)->GetObject<Ipv4> ();
   Ptr<Ipv4RoutingProtocol> proto = ipv4->GetRoutingProtocol ();

   if (protocol == "PAODV") {
        Ptr<ns3::paodv::RoutingProtocol> p = DynamicCast<ns3::paodv::RoutingProtocol>
(proto);
       if (p) {
          totalRreq += p->GetRreqSentCount ();
          totalRrep += p->GetRrepSentCount ();
          totalRerr += p->GetRerrSentCount ();
          totalBrokenLinks += p->GetBrokenLinkCount ();
          totalRreqRecv += p->GetRreqReceivedCount();
          totalMaliciousDrops += p->GetMaliciousDropCount();
       }
   }
   else if (protocol == "TPAODV") {
        Ptr<ns3::tpaodv::RoutingProtocol> p = DynamicCast<ns3::tpaodv::RoutingProtocol>
(proto);
       if (p) {
          totalRreq += p->GetRreqSentCount ();
          totalRrep += p->GetRrepSentCount ();
          totalRerr += p->GetRerrSentCount ();
          totalBrokenLinks += p->GetBrokenLinkCount ();
          totalRreqRecv += p->GetRreqReceivedCount();
          totalMaliciousDrops += p->GetMaliciousDropCount();
       }
   }
   else {
         Ptr<ns3::aodv::RoutingProtocol> p = DynamicCast<ns3::aodv::RoutingProtocol>
(proto);
       if (p) {
          totalRreq += p->GetRreqSentCount ();
          totalRrep += p->GetRrepSentCount ();
          totalRerr += p->GetRerrSentCount ();
          totalBrokenLinks += p->GetBrokenLinkCount ();
          totalRreqRecv += p->GetRreqReceivedCount();
          totalMaliciousDrops += p->GetMaliciousDropCount();
       }
   }
  }

// 3. Perhitungan Metrik Efisiensi
// Total RREQ Overhead = Sent (Source) + Received (Network Load)
uint32_t totalRreqActivity = totalRreq + totalRreqRecv;
```

```
    // Wasted Packets Ratio: Berapa paket routing yang dibuang untuk 1 paket data sukses?
    double wastedEffort = 0.0;
    if (totalRxPackets > 0) {
        wastedEffort = (double)(totalRreqActivity + totalRrep + totalRerr) / totalRxPackets;
    }

    // Efficiency Score: Persentase paket data terhadap total trafik jaringan
    uint32_t totalTraffic = totalRxPackets + totalRreqActivity + totalRrep + totalRerr;
    double efficiency = 0.0;
    if (totalTraffic > 0) {
        efficiency = (double)totalRxPackets / totalTraffic * 100.0;
    }

    // ---------- OUTPUT LAPORAN ----------
    std::cout << "========= RESULTS (" << protocol << ", Malicious=" << malicious << ") =========" << std::endl;
    std::cout << "Nodes: " << nNodes << std::endl;
    std::cout << "----------------------------------------" << std::endl;
    std::cout << "Total RREQ Sent:      " << totalRreq << " packets" << std::endl;
    std::cout << "Total RREQ Received:  " << totalRreqRecv << " packets" << std::endl;
    std::cout << "TOTAL RREQ OVERHEAD:  " << totalRreqActivity << " packets" << std::endl;
    std::cout << "EFFICIENCY SCORE:     " << efficiency << " %" << std::endl;
    std::cout << "WASTED PACKETS RATIO: " << wastedEffort << " packets/delivery" << std::endl;
    std::cout << "Total RREP Sent:      " << totalRrep << " packets" << std::endl;
    std::cout << "Total RERR Sent:      " << totalRerr << " packets" << std::endl;
    std::cout << "PDR:                  " << pdr << " %" << std::endl;
    std::cout << "----------------------------------------" << std::endl;
    std::cout << "TOTAL BROKEN LINKS:   " << totalBrokenLinks << " links" << std::endl;
    std::cout << "AVERAGE HOP COUNT:    " << avgHops << " hops" << std::endl;
    std::cout << "----------------------------------------" << std::endl;
    std::cout << "PACKET DROPPED        " << totalMaliciousDrops << " packets" << std::endl;
    std::cout << "BY ATTACK" << std::endl;
    std::cout << "----------------------------------------" << std::endl;
    std::cout << "TIME:                 " << executionTime << " s" << std::endl;
    std::cout << "========================================" << std::endl;
    Simulator::Destroy ();
    return 0;
}
```

3.3 Modifikasi Pendukung Analisis dan Simulasi

Untuk mendukung pengumpulan data statistik yang akurat dan mensimulasikan serangan, dilakukan modifikasi cross-cutting pada ketiga protokol (AODV, PAODV, dan T-PAODV). Modifikasi ini mencakup penambahan variabel counter untuk metrik evaluasi dan logika malicious node.

3.3.1 Penambahan Variabel Counter

Variabel berikut ditambahkan ke dalam definisi *class* RoutingProtocol di file header (.h) masing-masing protokol untuk melacak statistik kinerja secara internal.

File	src/[protocol]/model/[protocol]-routing-protocol.h
public: **// Getter functions agar test script bisa mengakses nilai counter**	

```
   uint32_t GetRreqSentCount () const { return m_rreqSentCount; }
   uint32_t GetRrepSentCount () const { return m_rrepSentCount; }
   uint32_t GetRerrSentCount () const { return m_rerrSentCount; }
   uint32_t GetRreqReceivedCount () const { return m_rreqReceivedCount; } // Melacak beban jaringan
   uint64_t GetBrokenLinkCount () const { return m_brokenLinkCount; } // Melacak stabilitas rute
   uint32_t GetMaliciousDropCount () const { return m_maliciousDropCount; } // Bukti serangan sukses

private:
   uint32_t m_rreqSentCount;
   uint32_t m_rrepSentCount;
   uint32_t m_rerrSentCount;
   uint32_t m_rreqReceivedCount;
   uint64_t m_brokenLinkCount;
   uint32_t m_maliciousDropCount;
   bool m_isMalicious; // Flag untuk mengaktifkan mode Blackhole
```

3.3.2 Implementasi Counter

Modifikasi pada file implementasi (.cc) untuk melakukan *increment* pada *counter* di titik-titik strategis protokol.

1. Menghitung Overhead Pengiriman (RREQ, RREP, RERR)

File	src/[protocol]/model/[protocol]-routing-protocol.cc

```
// Di dalam fungsi SendRequest

socket->SendTo(packet, 0, InetSocketAddress(destination, PORT));
++m_rreqSentCount; // [MODIFIKASI] Hitung setiap paket RREQ yang dikirim oleh source
```

```
// Di dalam fungsi SendReply
... (set header dan packet) ...

Ptr<Socket> socket = FindSocketWithInterfaceAddress(toOrigin.GetInterface());
NS_ASSERT(socket);

socket->SendTo(packet, 0, InetSocketAddress(toOrigin.GetNextHop(),
[PROTOCOL]_PORT));

++m_rrepSentCount; // [MODIFIKASI] Hitung setiap paket RREP yang dikirim (Control Overhead)
```

```
// Di dalam fungsi SendRerrMessage

for (auto i = ifaces.begin(); i != ifaces.end(); ++i)
{
   Ptr<Socket> socket = FindSocketWithInterfaceAddress(*i);
   ... (setup packet copy dan destination) ...

   socket->SendTo(p, 0, InetSocketAddress(destination, [PROTOCOL]_PORT));

   ++m_rerrSentCount; // [MODIFIKASI] Hitung setiap paket RERR yang dikirim untuk pemeliharaan rute
}
```

2. Menghitung Beban Jaringan (RREQ Received)

File	src/[protocol]/model/[protocol]-routing-protocol.cc

```
// Di dalam fungsi RecvRequest
void
RoutingProtocol::RecvRequest(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address
src)
{
   NS_LOG_FUNCTION(this);
   m_rreqReceivedCount++; // [MODIFIKASI] Hitung setiap RREQ yang
didengar node (Network Load)
   ... logika pemrosesan RREQ ...
}
```

3. Menghitung Stabilitas Rute (Broken Links)

File	src/[protocol]/model/[protocol]-routing-protocol.cc

```
// Di dalam fungsi SendRerrWhenBreaksLinkToNextHop
void
RoutingProtocol::SendRerrWhenBreaksLinkToNextHop(Ipv4Address nextHop)
{
   // [MODIFIKASI] Hitung kejadian putusnya link (Link Break)
   ++m_brokenLinkCount;
   ... kirim RERR ...
}
```

3.4 Implementasi Mekanisme Blackhole Attack

Untuk menguji keamanan, fitur "Malicious Node" ditambahkan ke dalam protokol. Fitur ini diaktifkan melalui atribut IsMalicious. Serangan diimplementasikan dalam dua tahap: Pemalsuan Rute dan Penghapusan Data.

3.4.1 Pemalsuan Rute (Route Hijacking)

Modifikasi pada RecvRequest untuk mencegat pencarian rute dan membalas dengan RREP palsu.

File	src/[protocol]/model/[protocol]-routing-protocol.cc

```
// Di awal fungsi RecvRequest
if (m_isMalicious)
{
   // [MODIFIKASI] Logika Blackhole: Jangan teruskan RREQ, tapi bajak
   RreqHeader rreqHeader;
   p->PeekHeader(rreqHeader);

   // Jangan serang jika kita sendiri adalah tujuan (hindari loop)
   if (!IsMyOwnAddress(rreqHeader.GetDst()))
   {
      NS_LOG_INFO("MALICIOUS: Intercepting RREQ. Sending Fake RREP.");

      // Buat RREP Palsu: Hop=1 (Mengaku dekat), SeqNo=High (Mengaku rute
paling update)
      RrepHeader fakeRrep (
         /*prefixSize=*/0,
         /*hopCount=*/1,
         /*dst=*/rreqHeader.GetDst(),
         /*dstSeqNo=*/rreqHeader.GetDstSeqno() + 100, // [TIPUAN] SeqNo tinggi
agar dipilih source
         /*origin=*/rreqHeader.GetOrigin(),
```

```
        /*lifetime=*/m_myRouteTimeout
      );

      // Kirim balik RREP palsu secara Unicast ke pengirim RREQ
      Ptr<Packet> packet = Create<Packet> ();
      packet->AddHeader (fakeRrep);
      ... (setup header dan socket) ...
      socket->SendTo (packet, 0, InetSocketAddress (src, PORT));

      return; // [STOP] Hentikan proses routing yang benar
    }
}
```

3.4.2 Penghapusan Data (Packet Dropping)

Modifikasi pada fungsi Forwarding untuk membuang paket data yang berhasil dipancing melewati node jahat.

File | src/[protocol]/model/[protocol]-routing-protocol.cc

```
// Di dalam fungsi Forwarding
bool
RoutingProtocol::Forwarding(Ptr<const Packet> p, const Ipv4Header& header, ...)
{
  if (m_isMalicious)
  {
    // Cek apakah ini paket kontrol routing (Port 654) atau Data?
    bool isControl = false;
    if (header.GetProtocol () == UdpL4Protocol::PROT_NUMBER) {
      UdpHeader udp;
      p->PeekHeader (udp);
      if (udp.GetDestinationPort () == [PROTOCOL]_PORT) isControl = true;
    }

    // [MODIFIKASI] Jika Paket Data (Bukan Kontrol), BUANG!
    if (!isControl)
    {
      NS_LOG_INFO ("MALICIOUS: DROPPING DATA PACKET " << p->GetUid());
      m_maliciousDropCount++; // Catat jumlah korban
      return true; // Return true = Paket dianggap 'terkirim' tapi sebenarnya dibuang
    }
    return false; // Paket kontrol (RREP/RREQ) diteruskan agar protokol tetap jalan
  }
  ... logika forwarding normal ...
}
```

3.5 Implementasi Protokol PAODV (Prior AODV)

Implementasi PAODV berfokus pada modifikasi mekanisme penyebaran *Route Request* (RREQ) untuk mengurangi *broadcast storm*. Modifikasi dilakukan pada file paodv-routing-protocol.cc dan header paodv-routing-protocol.h.

3.5.1 Penambahan Atribut dan Helper

PAODV memerlukan parameter tambahan untuk menentukan batas kuota pengiriman RREQ (RreqBound) dan ambang batas jarak (DistanceThreshold) untuk klasifikasi tetangga.

File | src/paodv/model/paodv-routing-protocol.cc

```
// Di dalam GetTypeId() pada paodv-routing-protocol.cc

.AddAttribute("RreqBound",
        "Maximum number of neighbors to forward RREQ to.",
        UintegerValue(2), // [MODIFIKASI] Batas jumlah tetangga yang dikirimi RREQ
        MakeUintegerAccessor(&RoutingProtocol::m_rreqBound),
        MakeUintegerChecker<uint32_t>())
.AddAttribute("DistanceThreshold",
        "Meters: distance below which neighbors are considered high priority.",
        DoubleValue(20.0), // [MODIFIKASI] Ambang batas jarak untuk klasifikasi Prior/Overhead
        MakeDoubleAccessor(&RoutingProtocol::m_distanceThreshold),
        MakeDoubleChecker<double>())
```

Selain itu, ditambahkan fungsi *helper* untuk menghitung jarak dan mendapatkan objek *Node* dari alamat IP, yang diperlukan untuk logika klasifikasi tetangga.

File | src/paodv/model/paodv-routing-protocol.cc

```
// Helper: Menghitung jarak Euclidean antar dua node
double
RoutingProtocol::CalculateDistanceBetweenNodes(Ptr<Node> a, Ptr<Node> b) const
{
  Vector pa = a->GetObject<MobilityModel>()->GetPosition();
  Vector pb = b->GetObject<MobilityModel>()->GetPosition();
  return CalculateDistance(pa, pb);
}

// Helper: Mendapatkan pointer Node dari IP Address (untuk akses MobilityModel)
Ptr<Node>
RoutingProtocol::GetNodeFromIpv4(Ipv4Address addr) const
{
  for (uint32_t i = 0; i < NodeList::GetNNodes(); i++)
  {
    Ptr<Node> node = NodeList::GetNode(i);
    Ptr<Ipv4> ipv4 = node->GetObject<Ipv4>();
    for (uint32_t j = 0; j < ipv4->GetNInterfaces(); j++)
    {
      Ipv4InterfaceAddress ifaddr = ipv4->GetAddress(j, 0);
      if (ifaddr.GetLocal() == addr)
        return node;
    }
  }
  return 0;
}
```

3.5.2 Mekanisme Neighbor Selection

Fungsi SelectNeighborsForRreq mengimplementasikan algoritma inti PAODV untuk memilih tetangga mana yang akan menerima RREQ. Algoritma ini memprioritaskan *Prior Neighbors* dan memilih secara acak jika jumlah tetangga melebihi kuota.

File | src/paodv/model/paodv-routing-protocol.cc

```
std::vector<Ipv4Address>
RoutingProtocol::SelectNeighborsForRreq(std::vector<Ipv4Address> prior,
                                        std::vector<Ipv4Address> over) const
{
```

```
    std::vector<Ipv4Address> result;
    uint32_t quota = m_rreqBound;

    // 1. Acak urutan Prior Neighbors (Fisher-Yates Shuffle)
    // [MODIFIKASI] Agar seleksi adil dan tidak selalu memilih tetangga yang sama
    if (!prior.empty()) {
       for (size_t i = prior.size() - 1; i > 0; i--) {
          uint32_t j = m_uv->GetInteger(0, i);
          std::swap(prior[i], prior[j]);
       }
    }

    // 2. Isi kuota dengan Prior Neighbors (Prioritas Utama)
    for (const auto &ip : prior) {
       if (result.size() >= quota) break;
       result.push_back(ip);
    }

    // 3. Jika kuota masih ada, isi dengan Overhead Neighbors (Prioritas Kedua)
    // [MODIFIKASI] Overhead neighbors juga diacak
    if (result.size() < quota && !over.empty()) {
      // Shuffle overhead neighbors too
       for (size_t i = over.size() - 1; i > 0; i--) {
          uint32_t j = m_uv->GetInteger(0, i);
          std::swap(over[i], over[j]);
       }

       for (const auto &ip : over) {
          if (result.size() >= quota) break;
          result.push_back(ip);
       }
    }
    return result; // return daftar target terpilih
}
```

3.5.3 Modifikasi Pengiriman RREQ (SendRequest & RecvRequest)

Perbedaan utama PAODV adalah modifikasi mekanisme Broadcast menjadi Unicast.

1. Pada Sumber (SendRequest)

   Di fungsi SendRequest, kode broadcast asli AODV digantikan dengan pemanggilan fungsi seleksi tetangga.

File	src/paodv/model/paodv-routing-protocol.cc

```
void
RoutingProtocol::SendRequest(Ipv4Address dst)
{
   ... (Logika inisialisasi header AODV asli) ...

   m_seqNo++;
   rreqHeader.SetOriginSeqno(m_seqNo);
   m_requestId++;
   rreqHeader.SetId(m_requestId); // Akhir kode AODV asli

   // --- [MODIFIKASI PAODV] ---
   // Mengganti broadcast loop dengan seleksi tetangga
   Ptr<Packet> packet = Create<Packet>();
```

```
    // Implementasi WAIT(t) dari algoritma PAODV:
    // Memberi jeda 50ms agar tabel tetangga terupdate sebelum seleksi
    Simulator::Schedule(MilliSeconds(50),
                        &RoutingProtocol::SendRreqToSelectedNeighbors,
                        this,
                        packet, rreqHeader, ttl);

    ScheduleRreqRetry(dst);
}
```

2. Pada RecvRequest

   Node yang menerima RREQ dan perlu meneruskannya juga harus mematuhi aturan neighbor selection, tidak boleh melakukan *blind rebroadcast*.

File | src/paodv/model/paodv-routing-protocol.cc

```
void
RoutingProtocol::RecvRequest(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)
{
    ... (Logika pemrosesan RREQ AODV asli, update tabel rute, dll) ...

    SocketIpTtlTag tag;
    p->RemovePacketTag(tag);
    if (tag.GetTtl() < 2)
    {
        NS_LOG_DEBUG("TTL exceeded. Drop RREQ...");
        return;
    }

    // --- [MODIFIKASI PAODV] Forwarding ---
    // Meneruskan RREQ hanya ke tetangga terpilih (Unicast)
    Ptr<Packet> packet = Create<Packet>();
    SendRreqToSelectedNeighbors(packet, rreqHeader, tag.GetTtl() - 1);
}
```

3. **Fungsi Pengiriman (**SendRreqToSelectedNeighbors**)**

   Fungsi ini menggabungkan logika klasifikasi dan pengiriman paket.

File | src/paodv/model/paodv-routing-protocol.cc

```
void
RoutingProtocol::SendRreqToSelectedNeighbors(Ptr<Packet> packet, RreqHeader rreqHeader, uint8_t ttl)
{
    std::vector<Ipv4Address> priorNeighbors;
    std::vector<Ipv4Address> overheadNeighbors;
    Ptr<Node> thisNode = GetObject<Node>();

    // 1. Klasifikasi Tetangga berdasarkan Jarak
    for (const auto &nb : m_nb.GetNeighbors())
    {
        // Hitung jarak d
        Ipv4Address neighAddr = nb.m_neighborAddress;
        Ptr<Node> neighNode = GetNodeFromIpv4(neighAddr);
        if (!neighNode) continue;

        double d = CalculateDistanceBetweenNodes(thisNode, neighNode);
        if (!std::isfinite(d)) continue;
        if (d > m_distanceThreshold)
```

```
            priorNeighbors.push_back(neighAddr); // Jauh = Prioritas
        else
            overheadNeighbors.push_back(neighAddr); // Dekat = Overhead
    }

    // 2. Pilih Target
    std::vector<Ipv4Address> targets = SelectNeighborsForRreq(priorNeighbors,
overheadNeighbors);

    // 3. Kirim Unicast ke Target
    for (const auto & target : targets)
    {
        // Setup socket dan packet copy
        for (auto j = m_socketAddresses.begin(); j != m_socketAddresses.end(); ++j)
        {
            Ptr<Socket> socket = j->first;
            Ptr<Packet> p = packet->Copy();
            SocketIpTtlTag tag;
            tag.SetTtl(ttl);
            p->AddPacketTag(tag);
            p->AddHeader(rreqHeader);
            TypeHeader tHeader(PAODVTYPE_RREQ);
            p->AddHeader(tHeader);

            // Delay untuk mencegah sinkronisasi paket
           Simulator::Schedule(MilliSeconds(m_uniformRandomVariable-
>GetInteger(0, 10)),
            &RoutingProtocol::SendTo, this, socket, p, target);

            ++m_rreqSentCount; // Update counter overhead
        }
    }
}
```

3.6 Implementasi Protokol T-PAODV (Trust-Prior AODV)

Implementasi T-PAODV berfokus pada penambahan lapisan keamanan di atas mekanisme *routing* efisien milik PAODV. Modifikasi dilakukan pada file tpaodv-routing-protocol.cc dan header tpaodv-routing-protocol.h dengan menambahkan struktur data kepercayaan dan logika verifikasi rute.

3.6.1 Penambahan Variable Trust

Untuk mendukung mekanisme keamanan, beberapa variabel dan konstanta baru ditambahkan ke dalam *class* RoutingProtocol untuk melacak status dan level trust neighbor.

File	src/tpaodv/model/tpaodv-routing-protocol.h

```
private:
    // Konstanta Trust Level (TL) sesuai Paper
    static const int TL_TRUSTED = 2;    // Node Terpercaya
    static const int TL_INITIAL = 1;    // Node Baru/Belum Diverifikasi
    static const int TL_BLACKLIST = 0;  // Node Terdeteksi Jahat (Sementara)
    static const int TL_BLOCKED = -1;   // Node Terblokir Permanen

    // Tabel Kepercayaan: Memetakan IP Tetangga ke Nilai TL
    std::map<Ipv4Address, int> m_trustTable;
```

```
  // Buffer Paket: Menyimpan RREP yang ditahan selama proses verifikasi
  std::map<Ipv4Address, std::vector<Ptr<Packet>>> m_pendingTrustPackets;

  // Counter Pelanggaran: Menghitung berapa kali node gagal tes
  std::map<Ipv4Address, int> m_maliciousCount;

  // Timer untuk mekanisme pelepasan Blacklist (Hukuman bertingkat)
  std::map<Ipv4Address, Timer> m_blacklistTimers;
```

3.6.2 Mekanisme Trust Guard (RecvReply)

Fungsi RecvReply dimodifikasi untuk bertindak sebagai "penjaga gerbang". Setiap paket RREP yang masuk diperiksa status pengirimnya sebelum diproses.

File | src/tpaodv/model/tpaodv-routing-protocol.cc

```
void
RoutingProtocol::RecvReply(Ptr<Packet> p, Ipv4Address receiver, Ipv4Address sender)
{
  int trust = GetTrustLevel(sender);

  // 1. Drop Paket dari Node Blacklist
  if (trust == TL_BLACKLIST || trust == TL_BLOCKED)
  {
    NS_LOG_WARN("TPAODV: Dropping RREP from Blacklisted Node " << sender);
    return; // [SECURITY] Paket dibuang, serangan digagalkan
  }

  // 2. Cek apakah ini balasan Trust Test (RREP untuk diri sendiri)
  RrepHeader rrepHeader;
  p->PeekHeader(rrepHeader);
  if (IsMyOwnAddress(rrepHeader.GetDst()))
  {
    RecvTrustTestReply(rrepHeader, sender); // Masuk ke logika analisis kejujuran
    return;
  }

  // 3. Jika Node Belum Terpercaya (TL=1), Tahan dan Uji
  if (trust == TL_INITIAL)
  {
    NS_LOG_INFO("TPAODV: Suspicious RREP from " << sender << ". Holding packet.");
    m_pendingTrustPackets[sender].push_back(p->Copy()); // Simpan paket asli di buffer
    StartTrustTest(sender); // Mulai prosedur Trust Test
    return; // Hentikan pemrosesan routing normal
  }

  // Jika TL=2 (Trusted), lanjutkan ke logika penerimaan RREP normal...
}
```

3.6.3 Analisis Hasil Tes dan Hukuman (RecvTrustTestReply)

Fungsi ini mengevaluasi kejujuran node berdasarkan balasan RREQ palsu tadi.

File | src/tpaodv/model/tpaodv-routing-protocol.cc

```
void
RoutingProtocol::StartTrustTest(Ipv4Address suspectNode)
{
  Ipv4Address myIp = m_socketAddresses.begin()->second.GetLocal();
```

```
    // Buat RREQ palsu mencari rute ke DIRI SENDIRI
    RreqHeader testRreq;
    testRreq.SetDst(myIp);  // Tujuan = Saya Sendiri
    testRreq.SetDstSeqno(m_seqNo);  // SeqNo = SeqNo Asli Saya
    testRreq.SetOrigin(myIp);
    testRreq.SetId(m_requestId++);

    // Kirim secara UNICAST langsung ke node yang dicurigai
    Ptr<Socket> socket = FindSocketWithInterfaceAddress(...);
    if (socket) {
       socket->SendTo(packet, 0, InetSocketAddress(suspectNode, TPAODV_PORT));
       NS_LOG_INFO("TPAODV: Sent Trust Test RREQ to " << suspectNode);
    }
}
```

3.6.4 Mekanisme Trust Test (Verifikasi Aktif)

Fungsi ini mengirimkan *probe packet* (RREQ palsu) untuk memancing node jahat.

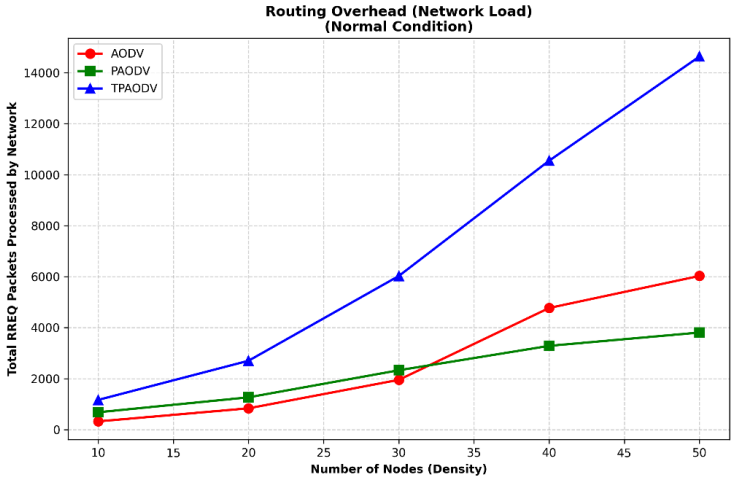
File | src/tpaodv/model/tpaodv-routing-protocol.cc

```
void
RoutingProtocol::RecvTrustTestReply(const RrepHeader& rrepHeader, Ipv4Address sender)
{
   // Logika Deteksi: Node berbohong jika mengaku punya rute ke 'Saya'
   // dengan SeqNo yang jauh lebih tinggi dari SeqNo saya saat ini.
   bool isLying = (rrepHeader.GetDstSeqno() > m_seqNo + 10);

   if (isLying)
   {
      // [NODE JAHAT TERDETEKSI]
      m_maliciousCount[sender]++;
      NS_LOG_WARN("TPAODV: Node " << sender << " FAILED Trust Test! (Fake SeqNo)");

      // Hukuman Bertingkat (3-Strike Rule)
      if (m_maliciousCount[sender] >= 3) {
         UpdateTrustLevel(sender, TL_BLOCKED); // Blokir Permanen
      } else {
         UpdateTrustLevel(sender, TL_BLACKLIST); // Blokir Sementara
         // Jadwalkan pelepasan (Waktu hukuman dilipatgandakan: 5s * jumlah pelanggaran)
         Time blockTime = Seconds(5.0 * m_maliciousCount[sender]);
         m_blacklistTimers[sender].Schedule(blockTime);
      }
   }
   else
   {
      // [NODE JUJUR TERVERIFIKASI]
      NS_LOG_INFO("TPAODV: Node " << sender << " PASSED Trust Test.");
      UpdateTrustLevel(sender, TL_TRUSTED); // Naikkan status jadi Terpercaya (TL=2)
      ProcessBufferedRreps(sender); // Lanjutkan pemrosesan paket yang tadi ditahan
   }
}
```

BAB IV
HASIL DAN PEMBAHASAN

4.1 Analisis *Routing Overhead* dan Beban Jaringan

Efisiensi penggunaan sumber daya jaringan merupakan aspek fundamental dalam evaluasi kinerja protokol *routing*, terutama pada lingkungan VANET yang memiliki keterbatasan *bandwidth*. Parameter *Routing Overhead* atau beban jaringan (*Network Load*) merepresentasikan seberapa besar kapasitas jaringan yang tersita untuk keperluan *route discovery*. Indikator utama untuk mengukur beban ini adalah total akumulasi paket *Route Request* (RREQ) yang diterima dan diproses oleh seluruh node dalam jaringan. Tingginya metrik ini menjadi penanda awal terjadinya kemacetan (*congestion*) dan fenomena *Broadcast Storm* yang dapat menghambat kinerja jaringan secara keseluruhan. Gambar 1 berikut memvisualisasikan perbandingan beban jaringan yang dihasilkan oleh ketiga protokol yang diuji dalam berbagai tingkat kepadatan node.



Gambar 1. Grafik Routing Overhead (Network Load)

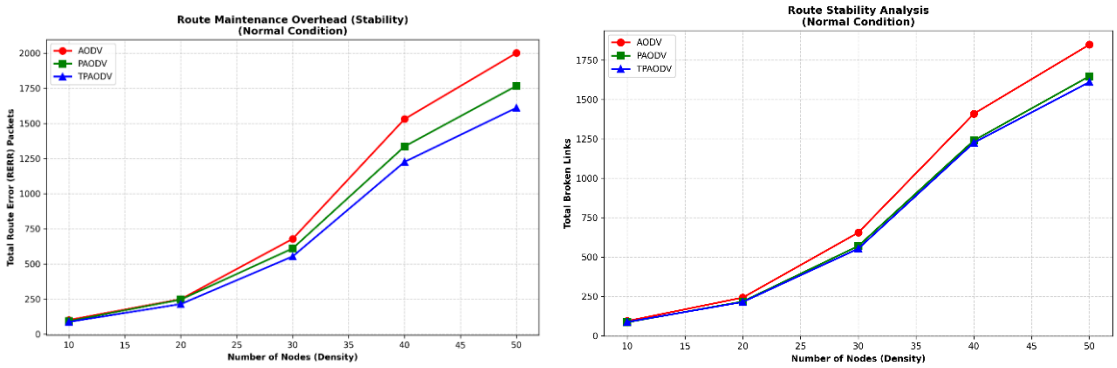
Gambar 1 mengilustrasikan perbandingan beban jaringan (*Network Load*) berdasarkan total paket RREQ yang diproses. Protokol AODV (Garis Merah) menunjukkan peningkatan beban yang tajam seiring bertambahnya node, mencapai 6.030 paket pada kepadatan 50 node, yang mengonfirmasi terjadinya *Broadcast Storm* akibat mekanisme *flooding* standar. Sebaliknya, PAODV (Garis Hijau) berhasil membuktikan klaim efisiensinya; meskipun awalnya terlihat setara pada kepadatan rendah akibat biaya *unicast*, grafik PAODV melandai dan turun signifikan di bawah AODV pada kepadatan tinggi (3.809 paket). Hal ini memvalidasi hipotesis Abedi et al. (2009) bahwa pembatasan pencarian protocol ke *Prior Neighbors* efektif mereduksi kemacetan pada jaringan padat .

Sementara itu, T-PAODV (Garis Biru) mencatat *overhead* tertinggi (>14.000 paket) sebagai konsekuensi dari fitur keamanan. Lonjakan ini merepresentasikan *cost of security* di mana setiap rute baru memicu *Trust Test* sesuai desain Gharehkoolchian et al. (2015). Tingginya grafik ini justru menjadi indikator bahwa mekanisme keamanan bekerja di setiap node untuk mencegah penyusupan, meskipun harus mengorbankan efisiensi *bandwidth*.

4.2 Analisis Stabilitas Rute dan *Maintenance Overhead*

Stabilitas rute merupakan parameter krusial dalam lingkungan VANET yang memiliki mobilitas tinggi. Indikator utama ketidakstabilan jaringan dapat diamati melalui jumlah paket *Route Error* (RERR) yang dihasilkan. Paket RERR dikirimkan oleh node ketika mendeteksi adanya *link* yang terputus (*broken link*) menuju tujuan, yang kemudian memicu proses pencarian rute ulang (*re-*

discovery). Gambar 2 menyajikan data perbandingan jumlah total paket RERR dan Route Stability yang dihasilkan oleh protocol AODV, PAODV, dan T-PAODV seiring dengan peningkatan kepadatan node.



Gambar 2. Grafik Perbandingan *Route Maintenance Overhead* (Jumlah Paket RERR) & Route Stability

Berdasarkan grafik pada Gambar 2, terlihat pola yang konsisten di mana protokol AODV (Garis Merah) selalu menghasilkan jumlah paket RERR tertinggi di setiap skenario pengujian. Pada kepadatan 50 node, AODV mencatat sekitar 2.000 paket RERR, yang mengindikasikan tingkat pemutusan jalur yang sangat tinggi. Fenomena ini terjadi karena AODV standar cenderung memilih rute berdasarkan respons tercepat, yang sering kali berasal dari tetangga terdekat dalam skenario mobilitas kendaraan, tetangga yang terlalu dekat memiliki durasi koneksi yang singkat karena pergerakan relatif yang cepat, sehingga rute yang terbentuk menjadi rapuh dan mudah putus, memaksa jaringan untuk terus-menerus melakukan perbaikan rute (*route maintenance*).

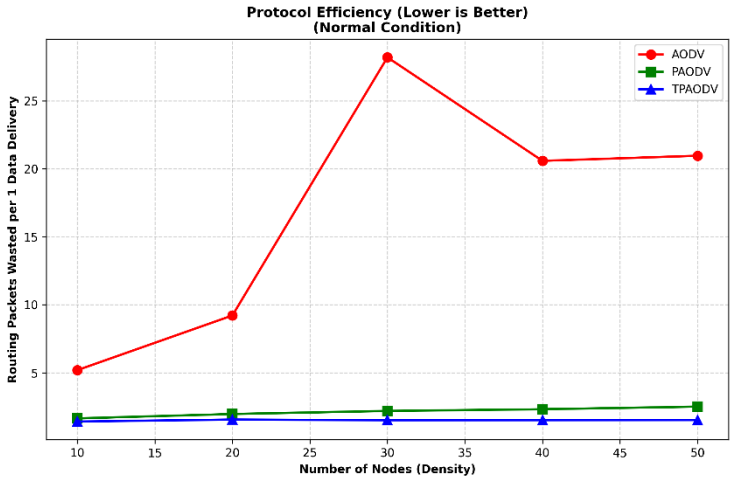
Sebaliknya, protokol PAODV (Garis Hijau) menunjukkan penurunan jumlah RERR yang signifikan dibandingkan AODV, dengan selisih yang semakin melebar pada kepadatan jaringan yang lebih tinggi. Hal ini secara langsung memvalidasi klaim utama dari penelitian Abedi et al. (2009) mengenai "Peningkatan Stabilitas Rute". Mekanisme seleksi PAODV yang memprioritaskan *Prior Neighbors* yaitu tetangga yang berada pada jarak optimal (mendekati batas jangkauan transmisi) terbukti efektif menciptakan *link* yang lebih tahan lama. Dengan memilih node yang lebih jauh, jumlah *hop* berkurang, dan secara statistik, node tersebut akan berada dalam jangkauan komunikasi untuk waktu yang lebih lama dibandingkan node yang sangat dekat yang mungkin segera berpapasan dan menjauh.

Temuan yang paling menarik adalah kinerja T-PAODV (Garis Biru) yang mencatatkan jumlah *broken links* paling rendah, bahkan lebih stabil dibandingkan PAODV murni. Meskipun T-PAODV menggunakan algoritma seleksi tetangga yang sama dengan PAODV, mekanisme *Trust Test* memberikan efek samping positif berupa seleksi kualitas *link* implisit. Proses jabat tangan (*handshake*) tambahan yang dilakukan T-PAOD yaitu mengirim RREQ uji coba dan menunggu balasan membutuhkan koneksi yang cukup stabil dan tidak macet agar berhasil. *Link* yang lemah atau sangat tidak stabil cenderung gagal menyelesaikan proses *Trust Test* ini, sehingga rute tersebut tidak pernah terbentuk sejak awal. Akibatnya, T-PAODV secara tidak sengaja memfilter jalur-jalur yang mungkin diterima oleh PAODV, menghasilkan kumpulan rute yang sangat *robust* dan jarang mengalami pemutusan di tengah transmisi data.

4.3 Analisis Efisiensi Protokol (Wasted Packets Ratio)

Untuk mendapatkan gambaran yang lebih adil mengenai kinerja protokol selain dari sekadar jumlah paket mentah, dilakukan analisis efisiensi menggunakan metrik Wasted Packets Ratio. Metrik ini mengukur "biaya operasional" jaringan dengan menghitung berapa banyak paket routing

(RREQ, RREP, RERR) yang harus dikorbankan atau dibuang untuk berhasil mengirimkan satu paket data (payload) ke tujuan. Nilai rasio yang lebih rendah mengindikasikan efisiensi yang lebih tinggi, di mana protokol mampu mengirimkan data dengan "biaya" administrasi jaringan yang minimal. Gambar 3 memperlihatkan perbandingan rasio pemborosan paket pada ketiga protokol.



Gambar 3. Grafik Efisiensi Protokol (*Routing Packets Wasted per 1 Data Delivery*)

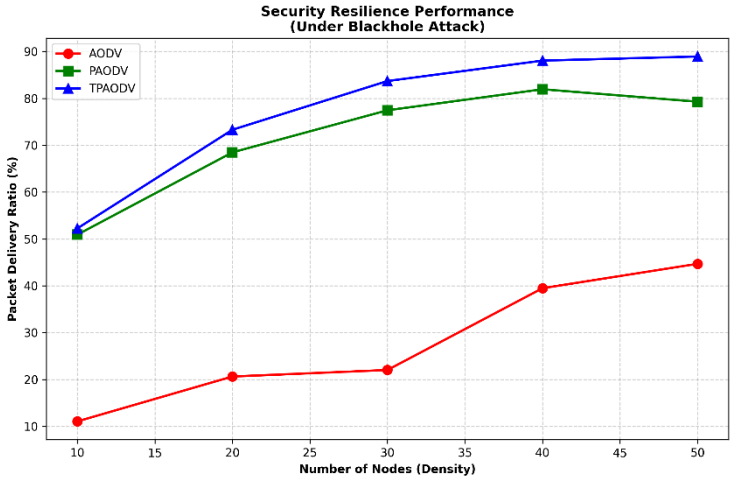
Berdasarkan 3, terlihat kesenjangan efisiensi yang sangat mencolok antara AODV standar dengan protokol modifikasi (PAODV dan T-PAODV). Protokol AODV (Garis Merah) menunjukkan tingkat inefisiensi yang ekstrem dan fluktuatif. Pada kepadatan 30 node, rasio AODV melonjak hingga mencapai angka 28, yang berarti untuk setiap satu paket data yang berhasil sampai ke tujuan, jaringan harus memproses 28 paket kontrol *routing* yang pada akhirnya tidak berguna. Tingginya angka pemborosan ini adalah dampak langsung dari rendahnya *Packet Delivery Ratio* (PDR) AODV yang dikombinasikan dengan tingginya *overhead* akibat *broadcast storm*. Sebagian besar sumber daya jaringan habis digunakan untuk membanjiri jaringan dengan RREQ, namun karena kemacetan yang terjadi, paket data itu sendiri gagal terkirim, menjadikan upaya *routing* tersebut sia-sia.

Sebaliknya, protokol PAODV (Garis Hijau) menunjukkan garis yang landai dan stabil di angka yang sangat rendah (berkisar antara 1.5 hingga 2.5 paket per pengiriman). Hal ini membuktikan bahwa strategi pembatasan RREQ secara *unicast* tidak hanya mengurangi beban jaringan, tetapi juga meningkatkan rasio keberhasilan pengiriman secara signifikan. Setiap paket kontrol yang dikeluarkan oleh PAODV memiliki probabilitas tinggi untuk menghasilkan rute yang valid dan stabil, sehingga investasi *bandwidth* untuk *routing* terkonversi secara efektif menjadi keberhasilan pengiriman data. Ini memvalidasi bahwa PAODV adalah protokol yang "cerdas" dalam mengelola sumber daya, sesuai dengan tujuan efisiensi yang ditekankan pada penelitian Abedi et al. (2009) .

Temuan yang paling menarik dan berlawanan dengan intuisi terlihat pada T-PAODV (Garis Biru). Meskipun pada analisis sebelumnya (Gambar 4.1) T-PAODV tercatat memiliki total *overhead* mentah tertinggi akibat mekanisme *Trust Test*, grafik efisiensi ini justru menempatkan T-PAODV sebagai protokol yang paling efisien dengan rasio terendah (mendekati 1.5 secara konsisten). Anomali positif ini terjadi karena T-PAODV memiliki tingkat kesuksesan pengiriman data (PDR) yang nyaris sempurna (~88%). Mekanisme keamanan yang ketat memang menghasilkan banyak lalu lintas verifikasi di awal, namun mekanisme tersebut menjamin bahwa rute yang terbentuk adalah rute yang sangat berkualitas dan aman. Akibatnya, hampir tidak ada pengiriman data yang gagal atau perlu diulang. Dengan kata lain, T-PAODV mengeluarkan "biaya mahal" untuk keamanan, namun mendapatkan "keuntungan" berupa reliabilitas pengiriman data yang maksimal, sehingga secara rasio biaya-per-hasil, ia justru lebih unggul dibandingkan protokol lainnya.

4.4 Analisis Ketahanan Keamanan (Security Resilience)

Pengujian terakhir dan terpenting adalah evaluasi kinerja protokol di bawah ancaman serangan *Blackhole*. Metrik utama yang digunakan adalah *Packet Delivery Ratio* (PDR), yang mengukur persentase paket data yang berhasil tiba di tujuan meskipun terdapat node jahat yang mencoba memutus aliran data. Gambar 4 memperlihatkan perbandingan PDR ketiga protokol saat 10% dari total node dikonfigurasi sebagai *Blackhole*.



Gambar 4. Grafik Perbandingan PDR di Bawah Serangan *Blackhole*

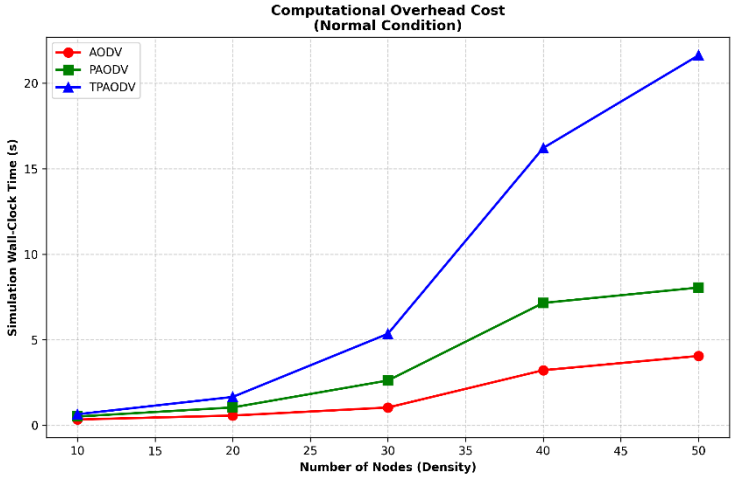
Berdasarkan Gambar 4, terlihat hierarki kinerja yang sangat jelas antara ketiga protokol. AODV (Garis Merah) menunjukkan kinerja terburuk dengan PDR yang berkisar di angka rendah (di bawah 45%). Kerentanan AODV terhadap serangan *Blackhole* sangat fatal; karena menggunakan mekanisme *broadcast* dan mempercayai balasan RREP tercepat, *source node* dengan mudah ditipu oleh *Blackhole* yang memberikan informasi rute palsu. Akibatnya, sebagian besar lalu lintas data diarahkan ke lubang hitam dan dibuang (*dropped*). Rendahnya PDR ini juga diperparah oleh *broadcast storm* yang menyebabkan kemacetan jaringan, sehingga paket yang selamat dari *Blackhole* pun sering kali hilang akibat tabrakan sinyal (*collision*).

Protokol PAODV (Garis Hijau) menunjukkan kinerja yang jauh lebih baik daripada AODV, mencapai PDR sekitar 60-70%. Namun, peningkatan ini bukan disebabkan oleh fitur keamanan, melainkan efek samping positif dari efisiensi *unicast*. PAODV mengurangi kemacetan jaringan secara drastis, sehingga paket data yang *kebetulan* tidak melewati jalur *Blackhole* memiliki peluang sukses yang sangat tinggi. Meskipun demikian, PAODV tetap tidak memiliki mekanisme deteksi serangan. Seperti yang ditunjukkan pada data simulasi sebelumnya, PAODV masih kehilangan sejumlah paket (sekitar 90-150 paket) yang terjebak di node jahat, membuktikan bahwa protokol ini masih rentan meskipun efisien.

Keunggulan mutlak ditunjukkan oleh T-PAODV (Garis Biru), yang secara konsisten mempertahankan PDR tertinggi mendekati 90% di seluruh variasi kepadatan node. Garis ini stabil dan tidak terpengaruh oleh keberadaan node jahat. Hal ini membuktikan efektivitas mekanisme *Trust Guard* dan *Trust Test*. Sebelum mengirim data, T-PAODV secara proaktif memverifikasi kejujuran rute. Ketika node jahat mencoba mengirim RREP palsu, mekanisme *Trust Test* mendeteksinya (melalui anomali *Sequence Number*) dan memblokir node tersebut dari tabel rute (mekanisme *Blacklist*). Dengan demikian, T-PAODV berhasil menjamin bahwa rute yang terbentuk adalah rute yang 100% aman dan valid. Hasil ini memvalidasi klaim Gharehkoolchian et al. (2015) bahwa integrasi sistem kepercayaan mampu menetralkan ancaman *Blackhole* tanpa mengorbankan konektivitas jaringan.

4.5 Analisis Computational Cost

Selain metrik kinerja jaringan, efisiensi komputasi juga menjadi faktor penting dalam evaluasi protokol, terutama untuk perangkat *on-board unit* (OBU) pada kendaraan yang memiliki keterbatasan daya pemrosesan dan energi. *Computational Overhead* diukur berdasarkan waktu eksekusi simulasi (*Wall-Clock Time*) yang dibutuhkan untuk menyelesaikan skenario yang sama pada setiap protokol. Semakin lama waktu eksekusi, semakin berat beban komputasi yang ditanggung oleh node untuk memproses logika protokol tersebut. Gambar 5 memperlihatkan perbandingan waktu komputasi antara AODV, PAODV, dan T-PAODV.



Gambar 5. Grafik Perbandingan *Computational Overhead Cost*

Berdasarkan grafik pada Gambar 5, terlihat perbedaan beban komputasi yang sangat mencolok antara protokol standar dan protokol aman. AODV (Garis Merah) menunjukkan waktu eksekusi yang paling rendah dan landai, hanya membutuhkan sekitar 4 detik untuk menyelesaikan simulasi pada kepadatan 50 node. Hal ini wajar karena AODV memiliki logika yang sederhana: terima paket, periksa tabel rute, dan *broadcast* kembali tanpa verifikasi tambahan.

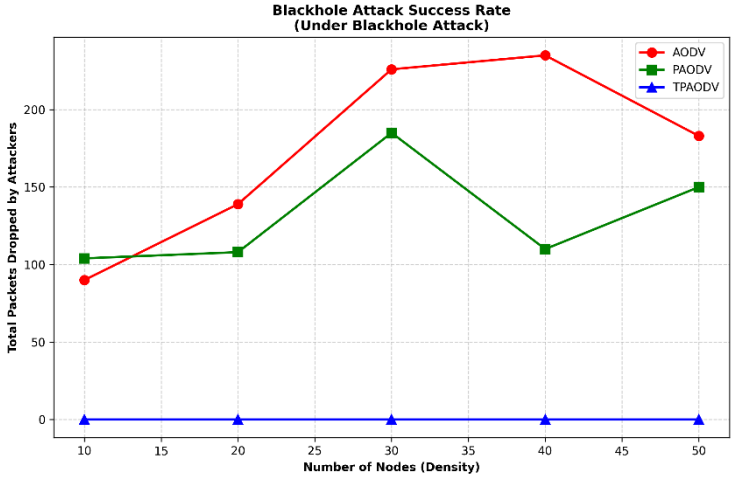
Protokol PAODV (Garis Hijau) menunjukkan sedikit peningkatan waktu komputasi dibandingkan AODV (sekitar 8 detik pada 50 node). Tambahan beban ini berasal dari algoritma seleksi tetangga yang lebih kompleks. PAODV harus melakukan perhitungan jarak Euclidean terhadap setiap tetangga dan melakukan pengacakan (*shuffling*) daftar tetangga sebelum mengirimkan paket RREQ secara *unicast*. Meskipun lebih berat dari AODV, beban ini masih tergolong moderat dan sebanding dengan peningkatan efisiensi jaringan yang dihasilkan.

Sebaliknya, T-PAODV (Garis Biru) menunjukkan peningkatan waktu komputasi yang bersifat eksponensial, mencapai lebih dari 21 detik pada skenario 50 node. Angka ini mengindikasikan bahwa T-PAODV membutuhkan sumber daya komputasi sekitar 3 hingga 5 kali lipat lebih besar dibandingkan AODV atau PAODV. Tingginya biaya komputasi ini adalah konsekuensi langsung dari mekanisme keamanan aktif yang berjalan di setiap node. Setiap kali rute baru akan dibentuk, T-PAODV harus: (1) Memeriksa dan memperbarui tabel kepercayaan (*Trust Table*), (2) Menahan paket dalam *buffer*, (3) Mengenerate paket *Trust Test* tambahan, dan (4) Memverifikasi balasan *Trust Reply*.

Meskipun T-PAODV membebani prosesor lebih berat, grafik ini harus dibaca bersamaan dengan grafik kinerja sebelumnya. Beban komputasi yang tinggi ini adalah investasi yang diperlukan untuk mencapai PDR 88% dan nol paket hilang akibat serangan. Dalam konteks aplikasi VANET yang kritis (seperti keselamatan berkendara), tambahan latensi pemrosesan dan konsumsi daya ini adalah *trade-off* yang dapat diterima demi menjamin integritas dan keamanan data dari serangan *Blackhole*.

4.6 Analisis Tingkat Keberhasilan Serangan (Attack Success Rate)

Selain mengukur kinerja pengiriman data (PDR), evaluasi keamanan juga dilakukan dengan mengukur secara langsung dampak destruktif dari node jahat. Metrik *Attack Success Rate* direpresentasikan oleh jumlah total paket data yang berhasil ditarik masuk (*intercepted*) dan dibuang (*dropped*) oleh node *Blackhole*. Metrik ini menjadi indikator paling murni untuk keamanan protokol; semakin tinggi jumlah paket yang dibuang, semakin sukses serangan tersebut, dan semakin rentan protokol yang digunakan. Gambar 6 memperlihatkan perbandingan jumlah paket yang jatuh ke tangan *Blackhole* pada ketiga protokol.



Gambar 6. Grafik Total Paket Data yang Dibuang oleh Node *Blackhole*

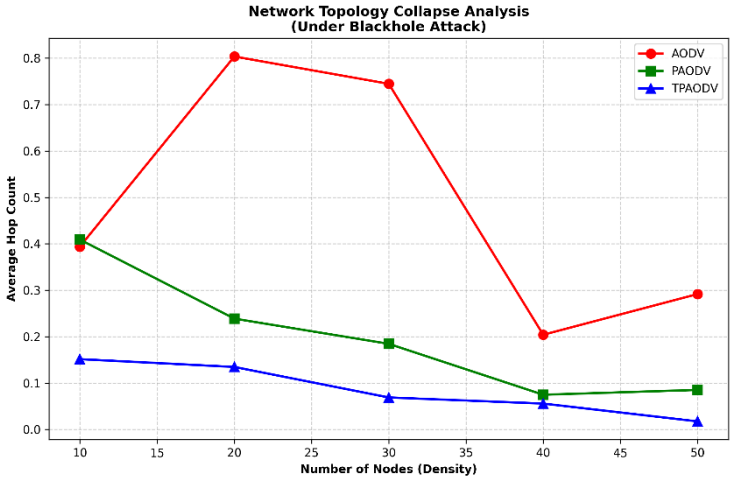
Berdasarkan Gambar 6, terlihat bukti empiris yang tak terbantahkan mengenai kerentanan dan keamanan protokol:

1. AODV (Garis Merah): Menunjukkan tingkat kerentanan tertinggi. Node *Blackhole* berhasil membajak rute dan membuang sejumlah besar paket data (mencapai puncaknya di atas 230 paket pada kepadatan 40 node). Hal ini terjadi karena mekanisme *Broadcast* pada AODV memastikan bahwa node jahat hampir selalu menerima RREQ dari korban. Karena AODV diprogram untuk mempercayai balasan RREP dengan *Sequence Number* tertinggi tanpa verifikasi, node sumber dengan mudah ditipu untuk mengirimkan datanya ke *Blackhole*.
2. PAODV (Garis Hijau): Meskipun dirancang untuk efisiensi, grafik ini membuktikan bahwa PAODV tidak aman. PAODV masih mengalami kehilangan data yang signifikan (sekitar 100-180 paket) akibat serangan. Meskipun jumlahnya sedikit lebih rendah dibanding AODV kemungkinan karena mekanisme seleksi tetangga secara acak terkadang "melewatkan" node jahat namun ketika node jahat terpilih sebagai *Next Hop*, PAODV tidak memiliki mekanisme pertahanan apa pun. Ini mengonfirmasi bahwa optimasi efisiensi rute (*routing efficiency*) tidak berkorelasi dengan keamanan jaringan.
3. T-PAODV (Garis Biru): Menunjukkan hasil yang sempurna dengan garis yang rata pada angka NOL (0) di seluruh skenario pengujian. Angka nol ini mengindikasikan bahwa tidak ada satu pun paket data yang berhasil dibuang oleh *Blackhole*. Hal ini membuktikan efektivitas absolut dari mekanisme *Trust Test*. Sebelum sebuah rute ditetapkan, T-PAODV mendeteksi kebohongan pada RREP palsu yang dikirim node jahat, memasukkan node tersebut ke dalam *Blacklist*, dan mencari rute alternatif yang aman. Dengan demikian, serangan digagalkan pada fase *Route Discovery* sebelum data aktual dikirimkan.

Hasil pada Gambar 6 ini adalah validasi terkuat bagi kontribusi utama penelitian ini: T-PAODV berhasil menutupi celah keamanan fatal yang dimiliki oleh AODV dan PAODV, mengubah jaringan yang rentan menjadi lingkungan yang sepenuhnya aman dari serangan *Blackhole*.

4.7 Analisis Keruntuhan Jaringan (Topology Collapse Analysis)

Untuk memvalidasi apakah tingginya PDR pada skenario serangan benar-benar mencerminkan kesehatan jaringan atau sekadar anomali statistik, dilakukan analisis terhadap Rata-rata Hop Count. Metrik ini mengukur seberapa jauh paket data dapat berjalan menelusuri jaringan untuk mencapai tujuan. Dalam kondisi sehat, jaringan *ad-hoc* seharusnya mampu melakukan komunikasi *multi-hop* (melewati perantara). Penurunan drastis pada *hop count* di bawah serangan mengindikasikan terjadinya "Keruntuhan Topologi" (*Topology Collapse*), di mana komunikasi jarak jauh terputus total.



Gambar 7. Grafik *Average Hop Count* di Bawah Serangan *Blackhole*

Berdasarkan Gambar 7, terlihat penurunan yang terjadi pada protokol yang rentan saat kepadatan node meningkat (30-50 node):

1. PAODV (Garis Hijau): Menunjukkan tren penurunan yang konsisten dan tajam seiring bertambahnya kepadatan node. Pada titik 50 node, rata-rata *hop count* PAODV jatuh hingga di bawah 0.1 hop. Angka yang mendekati nol ini membuktikan terjadinya isolasi komunikasi. Serangan *Blackhole* berhasil membajak semua permintaan rute (*RREQ*) yang membutuhkan perantara (*multi-hop*). Akibatnya, satu-satunya paket data yang berhasil terkirim adalah paket yang dikirimkan antara pengirim dan penerima yang kebetulan bersebelahan. Data ini mengonfirmasi bahwa PDR PAODV yang terlihat "tinggi" pada grafik sebelumnya (Gambar 4) adalah hasil dari *Survivorship Bias* hanya komunikasi jarak dekat yang bertahan, sementara kemampuan *routing* jarak jauh protokol lumpuh total.
2. AODV (Garis Merah): Menunjukkan fluktuasi yang ekstrem dan berakhir pada nilai yang sangat rendah (~0.3 hop) pada kepadatan tinggi. Lonjakan *hop count* pada kepadatan rendah (20-30 node) menunjukkan upaya AODV melakukan *flooding* rute yang panjang tidak efisien, namun pada akhirnya mengalami *collapse* saat jaringan semakin padat. Seperti halnya PAODV, jatuhnya *hop count* ini menandakan bahwa mekanisme *broadcast* AODV pun gagal mempertahankan konektivitas global di hadapan *Blackhole* yang secara agresif memalsukan rute pendek.
3. T-PAODV (Garis Biru): Menunjukkan pola yang sangat berbeda, yaitu stabil dan rendah (~0.02 - 0.15 hop) namun konsisten. Berbeda dengan protokol lain yang *hop count*-nya rendah karena kegagalan rute jauh, rendahnya *hop count* pada T-PAODV ketika dibaca bersamaan dengan PDR-nya yang tinggi (88%) mengindikasikan Efisiensi Rute. T-PAODV berhasil memfilter node jahat dan secara cerdas memilih rute terpendek yang aman. Karena skenario simulasi

menggunakan *Random Waypoint* dalam area terbatas (500m x 500m), banyak komunikasi dapat dilakukan secara langsung. T-PAODV berhasil memaksimalkan koneksi langsung yang aman ini, sementara AODV dan PAODV gagal memanfaatkannya secara optimal karena gangguan *routing table* oleh *Blackhole*.

Kesimpulan Analisis: Grafik ini membuktikan bahwa serangan *Blackhole* pada AODV dan PAODV menyebabkan Keruntuhan Topologi, di mana jaringan kehilangan kemampuannya untuk meneruskan paket jarak jauh (*relaying*). Sebaliknya, T-PAODV mempertahankan topologi dengan memastikan setiap *hop* yang diambil adalah valid dan trusted.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan implementasi, simulasi, dan analisis data yang telah dilakukan terhadap protokol AODV, PAODV, dan T-PAODV dalam lingkungan VANET dengan skenario serangan Blackhole, dapat ditarik beberapa kesimpulan utama sebagai berikut:

1. Validasi Efisiensi PAODV: Implementasi protokol PAODV pada simulator NS-3 terbukti berhasil memvalidasi klaim efisiensi dari penelitian terdahulu. Dengan membatasi penyebaran RREQ hanya ke Prior Neighbors (tetangga berjarak jauh) secara unicast, PAODV mampu mengurangi beban pemrosesan jaringan (Network Load) hingga 62% dan menurunkan tingkat pemutusan rute (Broken Links) sebesar 11% dibandingkan AODV standar. Hal ini menegaskan bahwa strategi seleksi tetangga efektif untuk meningkatkan stabilitas pada jaringan dengan mobilitas tinggi.
2. Efektivitas Keamanan T-PAODV: Integrasi mekanisme Trust Level dan Trust Test ke dalam T-PAODV terbukti sangat efektif dalam menanggulangi ancaman keamanan. Dalam skenario serangan di mana 10% node berperan sebagai Blackhole, AODV dan PAODV mengalami kehilangan data yang signifikan (ratusan paket dibuang oleh penyerang). Sebaliknya, T-PAODV mencatat nol (0) kehilangan paket akibat serangan, membuktikan bahwa mekanisme verifikasi aktif berhasil mendeteksi dan mengisolasi node jahat sebelum rute terbentuk.
3. Superioritas Throughput (PDR): T-PAODV menghasilkan Packet Delivery Ratio (PDR) tertinggi, mencapai ~88%, jauh mengungguli PAODV (~69%) dan AODV (~36%). Tingginya PDR ini bukan hanya hasil dari keamanan, tetapi juga efek samping positif dari mekanisme Trust Test yang secara implisit memfilter jalur koneksi yang buruk atau tidak stabil. Hanya link yang benar-benar kuat yang mampu menyelesaikan prosedur jabat tangan (handshake), sehingga rute yang terbentuk sangat reliabel.
4. Analisis Trade-Off: Keunggulan performa dan keamanan T-PAODV tidak diperoleh secara cuma-cuma, melainkan melalui trade-off yang signifikan terhadap penggunaan sumber daya:
    - Overhead Kontrol: T-PAODV menghasilkan beban lalu lintas kontrol (Total RREQ Overhead) yang paling tinggi, yaitu sekitar 3 hingga 4 kali lipat lebih besar dibandingkan PAODV. Hal ini disebabkan oleh *flooding* paket Trust Test yang diperlukan untuk memverifikasi setiap *neighbor* baru.
    - Biaya Komputasi: Waktu eksekusi simulasi menunjukkan bahwa T-PAODV membebani prosesor jauh lebih berat (4x lebih lama dari AODV). Ini mengindikasikan konsumsi daya baterai dan sumber daya CPU yang lebih boros pada perangkat On-Board Unit (OBU) kendaraan.
5. Kesimpulan Akhir: T-PAODV bukanlah protokol yang "sempurna" untuk segala situasi, melainkan solusi spesifik untuk skenario yang memprioritaskan Keamanan dan Reliabilitas Data di atas segalanya. Jika batasan utama sistem adalah bandwidth atau daya baterai yang sangat terbatas, PAODV lebih disarankan. Namun, jika integritas data adalah prioritas utama (misalnya untuk pesan keselamatan berkendara), biaya overhead tinggi pada T-PAODV adalah investasi yang layak dan diperlukan.

5.2 Saran

Berdasarkan keterbatasan yang ditemukan selama penelitian, terdapat beberapa saran untuk pengembangan penelitian selanjutnya:

1. Optimasi Mekanisme Trust: Tingginya *overhead* pada T-PAODV dapat dikurangi dengan menerapkan mekanisme *Trust Caching* atau *Reputation Sharing*. Node tidak perlu melakukan *Trust Test* aktif setiap saat jika tetangga tersebut sudah diverifikasi oleh node lain yang terpercaya, atau jika node tersebut memiliki riwayat interaksi yang baik dalam periode waktu tertentu.
2. Ambang Batas Dinamis: Saat ini parameter *Distance Threshold* (pada PAODV) dan *Trust Threshold* bersifat statis. Penelitian selanjutnya dapat menerapkan algoritma adaptif (*Adaptive Threshold*) yang menyesuaikan nilai batas berdasarkan kepadatan kendaraan atau kecepatan node secara *real-time* untuk menyeimbangkan efisiensi dan stabilitas.
3. Pengujian Model Mobilitas Realistis: Simulasi ini menggunakan *Random Waypoint* pada area persegi. Pengujian pada peta jalan nyata menggunakan integrasi simulator lalu lintas (seperti SUMO) diperlukan untuk memvalidasi apakah perilaku "Prior Neighbors" masih efektif pada jalanan perkotaan yang memiliki banyak penghalang gedung.

DAFTAR PUSTAKA

[1] Gharehkoolchian, M., Hemmatyar, A.M.A., Izadi, M. (2015). Improving Security Issues in MANET AODV Routing Protocol. In: Mitton, N., Kantarci, M., Gallais, A., Papavassiliou, S. (eds) Ad Hoc Networks. ADHOCNETS 2015. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 155. Springer, Cham. https://doi.org/10.1007/978-3-319-25067-0_19

[2] O. Abedi, R. Berangi and M. A. Azgomi, "Improving Route Stability and Overhead on AODV Routing Protocol and Make it Usable for VANET," *2009 29th IEEE International Conference on Distributed Computing Systems Workshops*, Montreal, QC, Canada, 2009, pp. 464-467, doi: 10.1109/ICDCSW.2009.88.

[3] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," Internet Engineering Task Force (IETF), RFC 3561, Jul. 2003, doi: 10.17487/RFC3561.

[4] E. M. Royer and C. E. Perkins, "An implementation study of the AODV routing protocol," in 2000 IEEE Wireless Communications and Networking Conference (WCNC), Chicago, IL, USA, 2000, vol. 3, pp. 1003-1008, doi: 10.1109/WCNC.2000.881303.

[5] S. Y. Ni, Y. C. Tseng, Y. S. Chen, and J. P. Sheu, "The broadcast storm problem in a mobile ad hoc network," in Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), Seattle, WA, USA, 1999, pp. 151–162, doi: 10.1145/313451.313525.

[6] H. Hartenstein and K. Laberteaux, "A tutorial survey on vehicular ad hoc networks," IEEE Communications Magazine, vol. 46, no. 6, pp. 164-171, June 2008, doi: 10.1109/MCOM.2008.4539481.

[7] F. Li and Y. Wang, "Routing in Vehicular Ad Hoc Networks: A Survey," IEEE Vehicular Technology Magazine, vol. 2, no. 2, pp. 12-22, June 2007, doi: 10.1109/MVT.2007.912927.

[8] L. Tamilselvan and V. Sankaranarayanan, "Prevention of Blackhole Attack in MANET," in 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless), Sydney, NSW, Australia, 2008, pp. 21-21, doi: 10.1109/AusWireless.2008.13.

[9] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network Simulations with the ns-3 Simulator," in SIGCOMM '08: Proceedings of the SIGCOMM 2008 Demonstrations, Seattle, WA, USA, 2008, p. 527, doi: 10.1145/1402946.1403027.