

## **BAB III**

### **DESAIN DAN PERANCANGAN**

Pada bab ini akan dijelaskan mengenai hal-hal berkaitan dengan perancangan sistem yang akan dibuat. Perancangan tersebut mencakup deskripsi umum aplikasi, arsitektur sistem, model fungsional, diagram alir aplikasi serta antarmuka aplikasi.

#### **3.1 Deskripsi Umum Sistem**

Aplikasi yang dibuat pada Tugas Akhir ini adalah aplikasi pendeteksi intrusi pada lalu lintas jaringan yang berbasis anomali dengan menggunakan metode *n-gram* dan *Incremental Learning*. Aplikasi ini ditempatkan pada *router* dengan sistem operasi Linux. Aplikasi ini akan bertugas sebagai *sniffer*, dimana akan melakukan penangkapan paket data secara terus menerus. Paket yang ditangkap oleh aplikasi ini hanya paket yang menggunakan protokol TCP dan UDP dengan memanfaatkan *library* Jpcap [3]. Paket yang ditangkap akan di rekonstruksi terlebih dahulu dengan mengelompokkan paket yang memiliki IP asal, Port asal, IP tujuan dan Port tujuan yang sama. Setelah selesai direkonstruksi, selanjutnya adalah menghitung distribusi *byte* karakter pada setiap paket menggunakan metode *n-gram* [4] yang kemudian hasil dari proses tersebut akan ditampung kedalam *array*.

Paket-paket yang telah melewati proses perhitungan *n-gram* selanjutnya adalah proses menghitung jarak mahalanobis paket. Menghitung jarak mahalanobis paket menggunakan metode mahalanobis *distance* [5] antara paket dengan model yang ada. Model yang ada merupakan hasil dari pengolahan paket-paket yang berasal dari DARPA IDS Data Set [7].

Hasil dari perhitungan tersebut akan menghasilkan sebuah nilai. Nilai tersebut akan dibandingkan dengan *threshold* yang sudah ditentukan sebelumnya. Jika nilai tersebut lebih besar dari *threshold* yang ditentukan, maka paket tersebut dapat dikategorikan sebagai paket yang tidak normal. Hasil dari

perhitungan ini kemudian disimpan dalam sebuah *file log* yang akan dibedakan filenya setiap jam.

## 3.2 Perancangan

Subbab perancangan akan membahas garis besar dan detail dari sistem yang dibangun. Garis besar dan detail dari sistem akan dijelaskan menggunakan diagram alir untuk mempermudah dalam memahami alur kerja sistem.

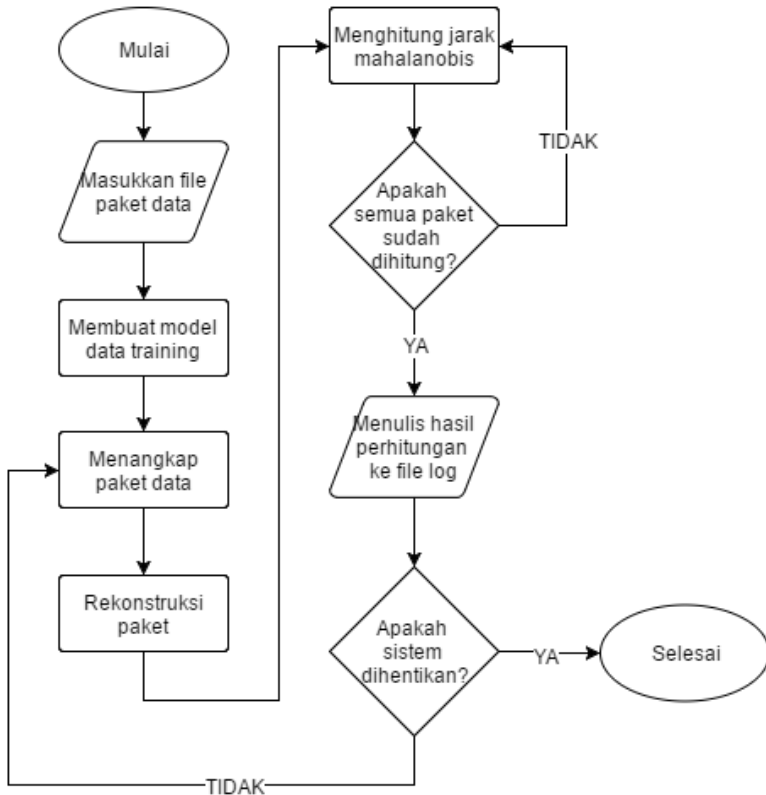
### 3.2.1 Alur Kerja Sistem Secara Umum

Secara umum sistem yang dibangun terdiri dari tiga proses utama, yaitu proses *training* data set, proses *sniffing* dan proses identifikasi serangan. Alur kerja sistem secara umum dapat dilihat pada Gambar 3.1.

Cara kerja sistem yang lebih detail yaitu, membaca *file* data set, membuat model data *training*, menyimpan model data *training*, menangkap paket, memproses paket dan membandingkan jarak mahalanobis paket dengan model serta pengambilan keputusan terhadap hasil perbandingan. Membuat model data *training* adalah proses dimana membaca *file* paket data dan ditampung pada *array of object*. *File* yang dapat dibaca hanya *file* yang berekstensi *\*.cap*, *\*.pcap*, *\*.tcpdump* dengan memanfaatkan *library* Jpcap. Proses selanjutnya adalah menangkap *packet* dari *network* interface dengan bantuan *library* Jpcap lalu menyimpannya pada *array of object*. Selanjutnya adalah proses membandingkan jarak mahalanobis *packet* dengan model data. *Packet* yang dihitung hanya *packet* yang memiliki port tujuan yang kurang dari 1024. Setelah terdapat *packet* yang memenuhi syarat tersebut akan dilakukan proses perhitungan, mulai dari perhitungan rata-rata dan standar deviasi dari *packet*. Dan selanjutnya proses pemanggilan fungsi Mahalanobis *Distance* untuk menghitung jarak mahalanobis *packet* dengan model data *training*. Setelah mendapatkan jarak mahalanobis, lalu dibandingkan dengan nilai *threshold* yang sudah ditentukan sebelumnya. Nilai *threshold*

setiap port memiliki besaran yang berbeda. Jika jarak mahalanobis *packet* melebihi nilai *threshold*, maka *packet* tersebut dapat dikategorikan paket yang tidak normal.

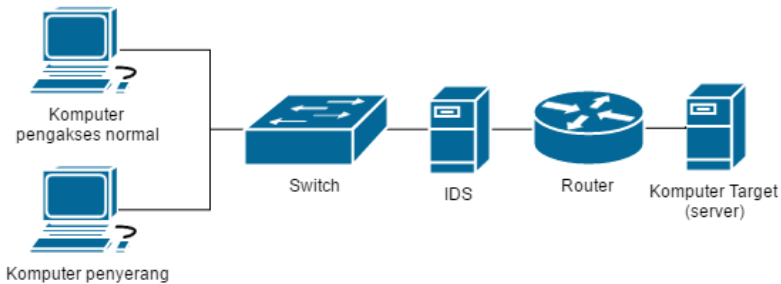
Proses diatas diulangi sampai aplikasi dihentikan oleh pengguna dan menulis hasil keputusan terhadap proses perbandingan ke sebuah *file log*.



**Gambar 3.1 Diagram Alir kerja sistem secara umum**

### 3.2.2 Perancangan Arsitektur Jaringan

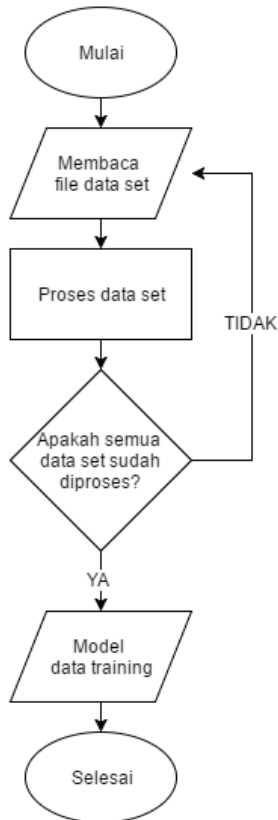
Sistem pendeteksi ini bekerja pada suatu arsitektur jaringan dengan multi subnet yang diilustrasikan pada Gambar 3.2. IDS akan menangkap paket data yang datang maupun keluar dari internet. IDS bekerja sesuai dengan konfigurasi yang dilakukan oleh pengguna. Konfigurasi tersebut disimpan dalam sebuah *file text*.



**Gambar 3.2** Topologi jaringan yang akan digunakan

### 3.2.3 Perancangan Proses *Training* Data Set

Pada bagian ini akan dijelaskan cara program membuat model data *training*. Proses *training* data set merupakan proses yang pertama kali harus dilakukan ketika aplikasi di jalankan. Proses ini merupakan proses yang terpenting pada palikasi yang akan dibangun, karena data set merupakan data yang digunakan sebagai data pembanding dengan paket data yang baru. Pada proses ini aplikasi akan menyimpan model-model paket data normal. Data set yang digunakan merupakan data set dari DARPA tahun 1999 yang merupakan kumpulan paket-paket yang didapat dari kegiatan simulasi pada sebuah arsitektur jaringan yang di desain oleh DARPA sendiri. Untuk alur proses dapat dilihat pada Gambar 3.3.

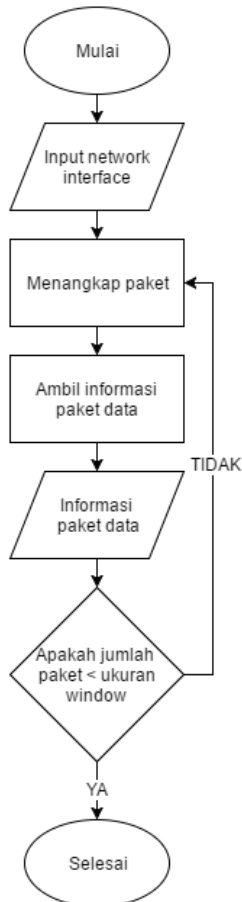


**Gambar 3.3** Proses training data set

### 3.2.4 Perancangan Proses *Sniffing*

Pada bagian ini akan dijelaskan cara program melakukan proses *sniffing*. Proses *sniffing* baru dapat dilakukan ketika sudah ada model data *training*. Jika model data *training* masih kosong sistem akan meminta pengguna untuk menjalankan *training* data set terlebih dahulu. Sniffer ini akan dijalankan pada komputer yang berfungsi sebagai router. Sniffer hanya menangkap paket data yang menggunakan protokol TCP dan UDP, selain itu paket data tidak ditangkap. Sebelum proses *sniffing* berjalan pengguna akan

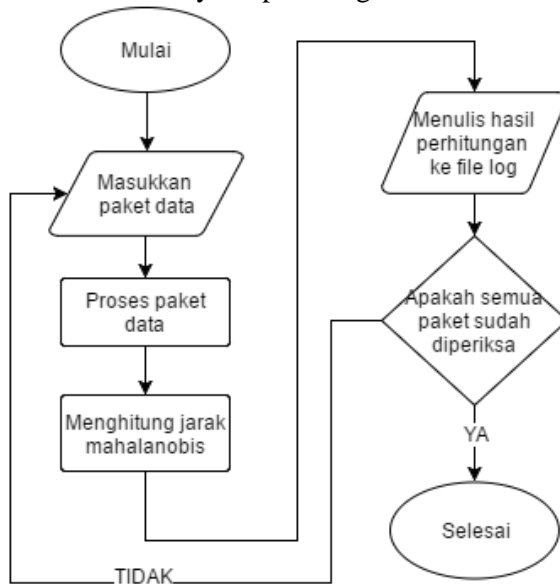
memilih network interface yang akan dimonitor. Selajutnya sistem akan membaca konfigurasi yang terdapat pada file *conf*. Pada file konfigurasi terdapat jumlah paket data yang akan ditangkap oleh sniffer. Jika sniffer sudah menangkap paket data sejumlah yang ditentukan pada file konfigurasi, sniffer akan berhenti menangkap paket lalu mengolah paket data tersebut.



**Gambar 3.4** Proses *sniffing*

### 3.2.5 Perancangan Proses Identifikasi Intrusi

Pada bagian ini akan dijelaskan cara program melakukan deteksi intrusi. Proses deteksi dilakukan setelah proses *training* data set dan *sniffing* selesai. Didalam proses deteksi terdapat sejumlah proses perhitungan data hasil *sniffing* yang harus dilakukan sebelum akhirnya dapat menghasilkan sebuah laporan.



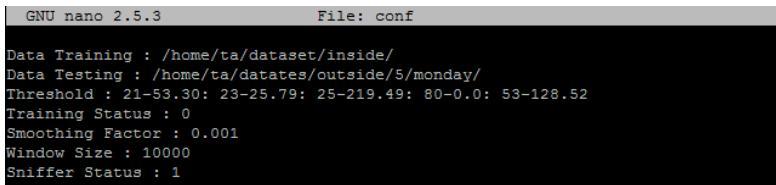
**Gambar 3.5 Proses Identifikasi Intrusi**

Pada Gambar 3.5 menjabarkan mengenai alur kerja proses deteksi intrusi. Pertama akan dijalankan pemeriksaan apakah Port tujuan paket sama dengan Port tujuan data *training*. Lalu mengambil rata-rata dan standar deviasi yang ada pada model sesuai dengan port paket. Proses selanjutnya adalah pemanggilan fungsi Mahalanobis *Distance* dengan mengirimkan parameter *n-gram* dari paket, *n-gram* dari data *training*, standar deviasi dari data *training* dan nilai *smoothing factor*. Proses selanjutnya adalah membandingkan nilai Mahalanobis *Distance* dengan nilai

*threshold* yang telah ditentukan sebelumnya. Jika nilai Mahalanobis *Distance* dari paket dan data set tersebut lebih besar dari nilai *threshold* maka paket tersebut dapat dikatakan sebagai sebuah intrusi. Lalu program akan menulis informasi paket ke sebuah file log.

### 3.2.6 Rancangan Antarmuka

Sistem pendeteksi intrusi berbasis anomali ini merupakan sistem yang bekerja dibelakang layar. Hal ini menyebabkan tidak ada antarmuka yang digunakan. Untuk menjembatani konfigurasi sistem yang dilakukan oleh pengguna, maka dibuatlah sebuah *filetext* seperti Gambar 3.6 untuk konfigurasi umum yang biasanya akan dilakukan pengguna.



```

GNU nano 2.5.3                               File: conf
Data Training : /home/ta/dataset/inside/
Data Testing  : /home/ta/datasets/outside/5/monday/
Threshold : 21-53.30: 23-25.79: 25-219.49: 80-0.0: 53-128.52
Training Status : 0
Smoothing Factor : 0.001
Window Size : 10000
Sniffer Status : 1
  
```

Gambar 3.6 Contoh *file* konfigurasi

### 3.2.7 Rancangan Luaran Sistem

Ketika sistem selesai mengolah data hasil *sniffing*, maka akan dihasilkan luaran berupa *log* yang didalamnya terdapat data mengenai olahan data dan identifikasi dari data *sniffing*. Pada Gambar 3.7 ditunjukkan mengenai elemen-elemen yang ada pada *log* nantinya.

```

+++++
# Start time : waktu pada saat dijalankan #
+++++
Protokol | Date | Source | Destination | Keterangan
-----
Konten paket data yang berupa serangan
  
```

Gambar 3.7 Contoh log hasil luaran sistem



## **BAB IV IMPLEMENTASI**

Bab ini membahas implementasi perancangan perangkat lunak dari aplikasi yang merupakan penerapan data, kebutuhan dan alur sistem yang mengacu pada desain dan perancangan yang telah dibahas sebelumnya. Selain itu, bab ini juga membahas lingkungan pembangunan perangkat lunak yang menjelaskan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam pembangunan sistem.

### **4.1 Lingkungan Implementasi**

Lingkungan implementasi dan pengembangan dibagi menjadi dua bagian, meliputi perangkat lunak dan perangkat keras.

#### **4.1.1 Perangkat Lunak**

Lingkungan implementasi dan pengembangan dilakukan menggunakan perangkat lunak sebagai berikut:

- Sistem Operasi Linux Ubuntu Desktop 16.04 LTS 64-bit sebagai lingkungan pengembangan sistem secara keseluruhan
- Netbeans IDE 8.1 sebagai IDE utama pembangunan dan pengembangan sistem
- Oracle Java Development Kit (JDK) 1.8 (64-bit) untuk aplikasi penangkapan dan pengolahan paket data
- Jpcap 0.7 untuk library penangkapan dan pengolahan paket data

#### **4.1.2 Perangkat Keras**

Lingkungan perangkat keras yang digunakan selama proses pengerjaan Tugas Akhir adalah sebagai berikut:

- Laptop dengan *processor* Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz 2.40GHz, *Installed Memory* (RAM) 8.00 GB dan sistem operasi Linux Ubuntu Desktop 16.04 LTS untuk pengembangan sistem.
- Komputer Aspire M3970 dengan *processor* Intel(R) Core(TM) i3-2120 2.40GHz 2.40GHz, *Installed Memory* (RAM) 8 GB dan sistem operasi Linux Ubuntu Desktop 16.04 LTS untuk uji coba sistem.

## 4.2 Implementasi Proses

Berikut adalah implementasi proses deteksi intrusi. Proses-proses akan dijelaskan sesuai dengan urutan berjalannya sistem. Proses dimulai dengan membuat model data *training*, membaca *packet*, memproses *packet*, dan membandingkan jarak mahalanobis *packet* dengan model data *training* yang menghasilkan keputusan terhadap hasil perbandingan. Kemudian hasil perbandingan dan informasi *packet* yang berupa intrusi ditulis ke sebuah file *log*.

### 4.2.1 Data set

Pada tugas akhir ini penulis menggunakan data set dari 1999 DARPA IDS Data Set sebagai data *training*. Data set yang digunakan terdiri dari beberapa *file* paket data yang berisi sejumlah paket-paket dan mempunyai ukuran *file* yang berbeda serta jumlah *packet* yang berbeda untuk tiap *file*. Data set yang digunakan sebagai data *training* adalah *file* paket data hasil tangkapan paket pada jaringan internal dari DARPA pada minggu pertama dan minggu ke-3. Pada minggu pertama terdapat lima *file* paket data dan minggu ke-3 terdapat tujuh *file* paket data. Keterangan mengenai *file* data set dapat dilihat pada Tabel 4.1.

**Tabel 4.1 Data set file Paket Data**

No.	Waktu Penangkapan Paket		Nama File	Ukuran File
	Minggu ke	Hari		
1	1	Senin	inside.tcpdump	325MB
2	1	Selasa	inside.tcpdump	325MB
3	1	Rabu	inside.tcpdump	367MB
4	1	Kamis	inside.tcpdump	527MB
5	1	Jumat	inside.tcpdump	294MB
6	3	Senin	inside.tcpdump	446MB
7	3	Selasa	inside.tcpdump	395MB
8	3	Rabu	inside.tcpdump	533MB
9	3	Kamis	inside.tcpdump	248MB
10	3	Jumat	inside.tcpdump	489MB
11	3	Senin	inside_extra.tcpdump	223MB
12	3	Selasa	inside_extra.tcpdump	438MB
13	3	Rabu	inside_extra.tcpdump	831MB

#### 4.2.2 Implementasi Proses Rekonstruksi Paket Data

Rekonstruksi paket data adalah proses pengelompokan paket data agar dapat digunakan pada proses selanjutnya. Proses rekonstruksi ini bertujuan untuk memisahkan paket yang berupa *request* dari klien dan paket yang berupa *response* dari *server*. Untuk melakukan proses rekonstruksi dibutuhkan beberapa bagian dari paket, bagian yang dibutuhkan dapat dilihat pada Tabel 4.2.

Setelah informasi-informasi didapatkan, program akan memulai proses rekonstruksi. Rekonstruksi yang dimaksud adalah mencocokkan informasi setiap paket yang ada pada *file* paket data. *Pseudocode* untuk rekonstruksi paket data dapat dilihat pada Gambar 4.1.

**Tabel 4.2 Daftar bagian paket yang dibutuhkan program**

No.	Bagian-bagian paket yang dibutuhkan program
1	Protokol paket
2	IP asal paket
3	Port asal paket
4	IP tujuan paket
5	Port tujuan paket

```

1 Deklarasi ArrayList<DataPacket> datasetTcp
2 Deklarasi ArrayList<DataPacket> datasetUdp
3 Terima paket menggunakan fungsi getPacket()
4 Jika paket termasuk TCP
5     Cocokkan IP asal, Port asal, IP tujuan dan
6     Port tujuan
7     Tambahkan paket ke datasetTcp
8 Jika paket termasuk UDP
9     Cocokkan IP asal, Port asal, IP tujuan dan
10    Port tujuan
11    Tambahkan paket ke datasetUdp

```

**Gambar 4.1 Pseudocode untuk Rekonstruksi paket data**

#### 4.2.3 Implementasi Proses Penggunaan Metode *N-Gram*

Implementasi proses penggunaan metode *n-gram* digunakan untuk mengetahui distribusi *byte* karakter pada sebuah paket. Pada tugas akhir ini metode *n-gram* yang digunakan adalah 1-gram, jadi menghitung kemunculan setiap *byte* karakter sampai *byte* paket habis. *Pseudocode* untuk menghitung *N-Gram* dapat dilihat pada Gambar 4.2.

```

1 Terima konten paket menggunakan fungsi Ngram()
2 Deklarasi double[] n = new double[256];
3 Baca konten paket
4 Konversi konten paket data menjadi unsigned
  integer
5 Tambahkan 1 ke n setiap konten paket yang sesuai

```

**Gambar 4.2 Pseudocode untuk menghitung N-Gram paket data**

#### 4.2.4 Implementasi Perancangan Model Data *Training*

Perancangan model data *training* merupakan proses pembuatan model data *training* untuk sistem sebelum digunakan untuk mendeteksi intrusi. Pada tugas akhir ini model data *training* yang dibuat adalah berdasarkan port tujuan dari sebuah paket data. Model data *training* ini yang nantinya digunakan untuk menghitung jarak mahalanobis setiap paket yang memiliki port tujuan yang sesuai dengan model. *Pseudocode* untuk perancangan mode data *training* dapat dilihat pada Gambar 4.3.

```

1  Deklarasi ArrayList<Double[]> dataTraining
2  Deklarasi ArrayList<DataPacket> datasetTcp
3  Deklarasi ArrayList<DataPacket> dasetUdp
4  Deklarasi ArrayList<DataModel> modelTcp
5  Deklarasi ArrayList<DataModel> modelUdp
6  Periksa tipe protocol dan port tujuan model
7      Jika tipe protocol adalah TCP
8          Baca datasetTcp
9          Jika port datasetTcp sama dengan port
            tujuan model
10             Tambahkan paket ke dataTraining
11             Hitung jumlah variable setiap model
12             Hitung rata-rata variable setiap model
13             Hitung standar variable deviasi setiap
                model
14             Tambahkan model ke modelTcp
15      Jika tipe protocol adalah UDP
16          Baca datasetUdp
17          Jika port datasetUdp sama dengan port
            tujuan model
18             Tambahkan paket ke dataTraining
19             Hitung jumlah variable setiap model
20             Hitung rata-rata variable setiap model
21             Hitung standar deviasi variable setiap
                model
22             Tambahkan model ke modelUdp

```

**Gambar 4.3** *Pseudocode* untuk membuat model data *training*

#### 4.2.5 Implementasi Sniffer

Fungsi sniffer digunakan untuk menangkap paket data yang melewati host tempat aplikasi ini dijalankan. Paket data yang ditangkap, yaitu paket yang menggunakan protokol TCP atau UDP. Paket yang menggunakan selain protokol TCP atau UDP tidak ditangkap atau dibiarkan lewat. Jumlah paket data yang ditangkap sesuai dengan keinginan pengguna yang dideklarasikan pada *file* konfigurasi. Jika jumlah paket data yang ditangkap belum sesuai dengan yang ditentukan oleh pengguna, maka proses penangkapan paket akan terus dilakukan sampai jumlah paket yang ditangkap sesuai dengan jumlah yang ditentukan sebelumnya. *Pseudocode* untuk *sniffer* dapat dilihat pada Gambar 4.4.

```

1 Deklarasi JpcapCaptor captor
2 Deklarasi NetworkInterface device
3 Deklarasi ukuranWindow
4 Dapatkan masukan device
5 Dapatkan masukan ukuranWindow
6 Deklarasi ArrayList<DataPacket> datasetTcp
7 Deklarasi ArrayList<DataPacket> dasetUdp
8 Tangkap paket menggunakan captor yang ada pada
  device
9   Jika jumlah paket sama dengan ukuranWindow
10     Berhenti tangkap paket
11   Jika tipe protocol paket adalah TCP
12     Rekonstruksi paket
13     Tambahkan paket ke datasetTcp
14   Jika tipe protocol paket adalah UDP
15     Rekonstruksi paket
16     Tambahkan paket ke datasetUdp

```

**Gambar 4.4** *Pseudocode* untuk *sniffer*

#### 4.2.6 Implementasi Proses Penggunaan Metode Mahalanobis Distance

Implementasi proses penggunaan metode mahalanobis *distance* digunakan untuk menghitung jarak mahalanobis antara paket baru dan model data *training*. *Pseudocode* untuk menghitung

jarak mahalanobis menggunakan Mahalanobis *Distance* dapat dilihat pada Gambar 4.5.

```

1 Deklarasi ngram
2 Deklarasi mean
3 Deklarasi standarDeviasi
4 Deklarasi smoothingFactor
5 Deklarasi jarak
6 jarak = nilai mutlak dari (ngram-mean) dibagi
  (standarDeviasi+smoothingFactor)

```

**Gambar 4.5 Pseudocode penggunaan metode Mahalanobis Distance**

#### 4.2.7 Implementasi Pendeteksian Intrusi

Implementasi pendeteksian intrusi merupakan proses menentukan apakah paket data yang diperiksa merupakan paket data normal atau paket data yang berupa intrusi. Proses ini dilakukan dengan membandingkan nilai jarak mahalanobis paket dengan nilai *threshold* yang sudah ditentukan sebelumnya. Jika nilai jarak mahalanobis paket lebih besar dari *threshold* maka paket tersebut dapat dikatakan sebagai paket tidak normal. Pseudocode untuk pendeteksian intrusi dapat dilihat pada Gambar 4.6

```

1 Deklarasi jarak
2 Deklarasi threshold
3 Jika nilai jarak lebih besar dari nilai
  threshold
4   Paket tersebut dianggap intrusi
5   Tulis informasi paket ke file log

```

**Gambar 4.6 Pseudocode untuk Pendeteksian Intrusi**

#### 4.2.8 Implementasi Proses *Incremental Learning*

Implementasi proses *incremental learning* digunakan untuk memperbaharui model data *training*. Nilai yang diperbaharui dari model, yaitu rata-rata dan standar deviasi dari model yang sesuai. Pseudocode untuk proses *incremental learning* dapat dilihat pada Gambar 4.7.

```

1 Deklarasi ngram
2 Deklarasi modelTcp
3 Deklarasi modelUdp
4 Deklarasi tipeProtoko
5 Deklarasi portPaket
6 Deklarasi mean
7 Deklarasi standarDeviasi
8 Dapatkan masukan tipeProtocol
9 Dapatkan masukan portPaket
10 Dapatkan masukan ngram
11 Jika tipe protocol adalah TCP
12     Ambil mean dari modelTcp
13     Ambil standarDeviasi dari modelTcp
14     Hitung mean yang baru dengan ngram
15     Hitung standarDeviasi yang baru dengan ngram
16     Perbaharui mean pada modelTcp sesuai
        portPaket
17     Perbaharui standarDeviasi modelTcp sesuai
        portPaket
18 Jika tipe protocol adalah TCP
19     Ambil mean dari modelTcp
20     Ambil standarDeviasi dari modelTcp
21     Hitung mean yang baru dengan ngram
22     Hitung standarDeviasi yang baru dengan ngram
23     Perbaharui mean pada modelTcp sesuai
        portPaket
24     Perbaharui standarDeviasi modelTcp sesuai
        portPaket

```

**Gambar 4.7 Pseudocode untuk Incremental Learning**



## LAMPIRAN

Bagian ini merupakan lampiran sebagai dokumen pelengkap dari buku Tugas Akhir, pada bagian ini diberikan informasi mengenai kode sumber dari sistem yang dibuat.

### A. Kode Sumber

#### A.1. Kode Sumber Proses Rekonstruksi Paket Data

```
import java.nio.charset.StandardCharsets;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.TimeZone;
import jpcap.JpcapCaptor;
import jpcap.packet.Packet;
import jpcap.packet.TCPPacket;
import jpcap.packet.UDPPacket;
/**
 *
 * @author agus
 */
```

```

public class PacketReader implements Runnable {
    private static int countPacket;
    private int input, files, type, counter, windowSize;
    private double[] numChars;
    private String tuples, timeFormat, startTime, regex = "\\r?\\n";
    private String[] header, time, line, split;
    private byte[] data;
    private Date date;
    private DateFormat format;
    private JpcapCaptor captor;
    private TCPPacket tcp;
    private UDPPacket udp;
    private ArrayList<DataPacket> datasetTcp;
    private ArrayList<DataPacket> datasetUdp;
    private ArrayList<DataPacket> dataTest;
    private Map<String, BodyPacket> packetBody = new HashMap<>();
    private Map<String, String> packetTime = new HashMap<>();
    private Map<String, String> dataPort;
    Ngram ng = new Ngram();
    BodyPacket bp;

    public PacketReader(){

    }

    public PacketReader(int files, JpcapCaptor captor, int input,
        ArrayList<DataPacket> datasetTcp, ArrayList<DataPacket> datasetUdp,

```

```

ArrayList<DataPacket> dataTest, Map<String, String> dataPort, int type, int
windowSize){
    this.files = files;
    this.captor = captor;
    this.input = input;
    this.datasetTcp = datasetTcp;
    this.datasetUdp = datasetUdp;
    this.dataTest = dataTest;
    this.dataPort = dataPort;
    this.type = type;
    this.windowSize = windowSize;
}

@Override
public void run(){
    while (true) {
        Packet packet = captor.getPacket();

        synchronized(packetBody){
            if (packet == null || packet == Packet.EOF || (input == 3 &&
counter == windowSize)) break;

            if (packet instanceof TCPPacket && packet.data.length != 0){
                tcp = (TCPPacket) packet;
                if (dataPort.containsKey("TCP"+tcp.dst_port)) {
                    if (input == 3) {
                        time = new String(tcp.toString()).split(":");
                        date = new Date(Long.parseLong(time[0])*1000L);

```

```

        format = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");

format.setTimeZone(TimeZone.getTimeZone("America/New_York"));
        timeFormat = format.format(date);
    }
    tuples = input+"-TCP-
"+tcp.src_ip.toString().substring(1)+"-"+tcp.src_port+"-
"+tcp.dst_ip.toString().substring(1)+"-"+tcp.dst_port;
    if (packetBody.containsKey(tuples)) {
        bp = packetBody.get(tuples);
        bp.addBytes(tcp.data);
    } else {
        bp = new BodyPacket(tcp.data);
        packetBody.put(tuples, bp);
        packetTime.put("TCP-
"+tcp.src_ip.toString().substring(1)+"-"+tcp.src_port+"-
"+tcp.dst_ip.toString().substring(1)+"-"+tcp.dst_port, timeFormat);
    }
    countPacket++;
    counter++;
}
}
else if(packet instanceof UDPPacket && packet.data.length != 0){
    udp = (UDPPacket) packet;
    if (dataPort.containsKey("UDP"+udp.dst_port)) {
        if (input == 3) {
            time = new String(udp.toString()).split(":");
            date = new Date(Long.parseLong(time[0])*1000L);

```

```

        format = new SimpleDateFormat("MM/dd/yyyy HH:mm:ss");

format.setTimeZone(TimeZone.getTimeZone("America/New_York"));
        timeFormat = format.format(date);
    }
    tuples = input+"-UDP-
"+udp.src_ip.toString().substring(1)+"-"+udp.src_port+"-
"+udp.dst_ip.toString().substring(1)+"-"+udp.dst_port;
    if (packetBody.containsKey(tuples)) {
        bp = packetBody.get(tuples);
        bp.addBytes(udp.data);
    } else {
        bp = new BodyPacket(udp.data);
        packetBody.put(tuples, bp);
        packetTime.put("UDP-
"+udp.src_ip.toString().substring(1)+"-"+udp.src_port+"-
"+udp.dst_ip.toString().substring(1)+"-"+udp.dst_port, timeFormat);
    }
    countPacket++;
    counter++;
}

}

}

for (Map.Entry<String, BodyPacket> entry : packetBody.entrySet()) {
    String key = entry.getKey();
    header = key.split("-", 0);
    BodyPacket value = entry.getValue();

```

```

        numChars = ng.Ngram(value.getBytes());
        if (header[0].equals("1") && header[1].equals("TCP")) {
            datasetTcp.add(new DataPacket(startTime, header[1], header[2],
Integer.parseInt(header[3]), header[4], Integer.parseInt(header[5]), null,
numChars, type));
        }
        else if (header[0].equals("1") && header[1].equals("UDP")) {
            datasetUdp.add(new DataPacket(startTime, header[1], header[2],
Integer.parseInt(header[3]), header[4], Integer.parseInt(header[5]), null,
numChars, type));
        }
        else {
            startTime = packetTime.get(header[1]+"-"+header[2]+"-
"+header[3]+"-"+header[4]+"-"+header[5]);
            dataTest.add(new DataPacket(startTime, header[1], header[2],
Integer.parseInt(header[3]), header[4], Integer.parseInt(header[5]),
value.getBytes(), numChars, type));
        }
    }
    packetBody.clear();
    packetTime.clear();
}
}

```

## A.2. Kode Sumber Proses Penggunaan Metodel N-Gram

```
package ids;

/**
 *
 * @author agus
 */
public class Ngram {
    private int ascii;
    private double[] n;

    public double[] Ngram(byte[] data){
        if (data != null) {
            n = new double[256];

            for (byte b : data) {
                ascii = b & 0xFF;
                n[ascii] += 1;
            }
        }
        return n;
    }
}
```

### A.3. Kode Sumber Proses Perancangan Model Data Training

```
package ids;

import java.util.ArrayList;
import org.apache.commons.lang3.ArrayUtils;

/**
 *
 * @author agus
 */
public class DataTraining implements Runnable {
    private int port, ascii = 256;
    private String proto;
    private double[] sumData = new double[ascii], meanData = new double[ascii],
    deviasiData = new double[ascii], quadraticData = new double[ascii], standardData
    = new double[ascii];
    private ArrayList<DataPacket> datasetTcp;
    private ArrayList<DataPacket> datasetUdp;
    private ArrayList<DataModel> modelTcp;
    private ArrayList<DataModel> modelUdp;
    private ArrayList<Double[]> dataTraining = new ArrayList<>();

    public DataTraining(String proto, ArrayList<DataPacket> datasetTcp,
    ArrayList<DataPacket> datasetUdp, ArrayList<DataModel> modelTcp,
    ArrayList<DataModel> modelUdp, int port){
        this.proto = proto;
    }
}
```



```

        this.datasetTcp = datasetTcp;
        this.datasetUdp = datasetUdp;
        this.modelTcp = modelTcp;
        this.modelUdp = modelUdp;
        this.port = port;
    }

    public void run() {
        synchronized(dataTraining) {
            if (proto.equals("TCP")) {
                for (DataPacket dataSetTcp : datasetTcp) {
                    if (dataSetTcp.getDstPort() == port) {
dataTraining.add(ArrayUtils.toObject(dataSetTcp.getNgram()));
                        }
                    }

                for (int i = 0; i < dataTraining.size(); i++) {
                    for (int j = 0; j < ascii; j++) {
                        sumData[j] += dataTraining.get(i)[j];
                        quadraticData[j] += Math.pow(dataTraining.get(i)[j], 2);
                    }
                }

                for (int i = 0; i < meanData.length; i++) {
                    meanData[i] = sumData[i]/dataTraining.size();
                }
            }
        }
    }

```

```

        for (int i = 0; i < deviasiData.length; i++) {
            deviasiData[i] =
Math.sqrt((dataTraining.size()*quadraticData[i]-Math.pow(sumData[i],
2))/(dataTraining.size()*(dataTraining.size()-1)));
        }

        modelTcp.add(new DataModel(port, sumData, meanData, deviasiData,
quadraticData, dataTraining.size()));
    }

    else if (proto.equals("UDP")) {
        for (DataPacket dataSetUdp : datasetUdp) {
            if (dataSetUdp.getDstPort() == port) {

dataTraining.add(ArrayUtils.toObject(dataSetUdp.getNgram()));
            }
        }

        for (int i = 0; i < dataTraining.size(); i++) {
            for (int j = 0; j < ascii; j++) {
                sumData[j] += dataTraining.get(i)[j];
                quadraticData[j] += Math.pow(dataTraining.get(i)[j], 2);
            }
        }

        for (int i = 0; i < meanData.length; i++) {
            meanData[i] = sumData[i]/dataTraining.size();
        }
    }

```

```
        for (int i = 0; i < deviasiData.length; i++) {  
            deviasiData[i] =  
Math.sqrt((dataTraining.size()*quadraticData[i]-Math.pow(sumData[i],  
2))/(dataTraining.size()*(dataTraining.size()-1)));  
        }  
  
        modelUdp.add(new DataModel(port, sumData, meanData, deviasiData,  
quadraticData, dataTraining.size()));  
    }  
}  
  
}
```

### A.4. Kode Sumber Sniffing

```

NetworkInterface[] device = JpcapCaptor.getDeviceList();
for (int i = 0; i < device.length; i++) {
    System.out.println(i+": "+device[i].name + "(" +
device[i].description+")");
    //print out its datalink name and description
    System.out.println("    Datalink: "+device[i].datalink_name + "(" +
device[i].datalink_description+")");
    //print out its MAC address
    System.out.print("    MAC address: ");
    for (byte b : device[i].mac_address)
        System.out.print(Integer.toHexString(b&0xff) + ":");
    System.out.println();
    //print out its IP address, subnet mask and broadcast address
    for (NetworkInterfaceAddress a : device[i].addresses)
        System.out.println("    Address: "+a.address + " " + a.subnet + " "+
a.broadcast);
    System.out.println("\n");
}
System.out.println("Choose active network interface...(0,1,2)!");
sc = new Scanner(System.in);
input = sc.nextInt();
System.out.println("Selected network interface name : "+device[input].name);
fwRunning.append("Selected network interface name : "+device[input].name+"\n");
snifferStatus = true;
while (snifferStatus) {

```

```

System.out.println("Start Sniffing...");
time = ids.getTime();
windowSize = Integer.parseInt(ids.getData("Window Size "));
thresholdAll = ids.getData("Threshold ");
portTh = ids.getThreshold();
sFactor = Double.parseDouble(ids.getData("Smoothing Factor "));
dateTime = time.split("_");
fileLog = new File(dateTime[0]+"/"+dateTime[1]+"/"+dateTime[2]);
if (!fileLog.exists()) {
    fileLog.mkdirs();
}
fileLog = new
File(dateTime[0]+"/"+dateTime[1]+"/"+dateTime[2]+"/Sniff_Result_log_"+dateTime[3]
);
    fileRecord = new
File(dateTime[0]+"/"+dateTime[1]+"/"+dateTime[2]+"/Sniff_Record_log_"+dateTime[3]
);
    fileRunning = new
File(dateTime[0]+"/"+dateTime[1]+"/"+dateTime[2]+"/Running_Test_log");
    if (!fileLog.exists() | !fileRecord.exists()) {
        fileLog.createNewFile();
        fileRecord.createNewFile();
        fileRunning.createNewFile();
    }
    fwLog = new FileWriter(fileLog,true);
    fwRecord = new FileWriter(fileRecord, true);
    freePacket = new ArrayList<>();
    attackPacket = new ArrayList<>();

```

```
System.out.println("Window Size : "+windowSize);
fwRunning.append("Window Size : "+windowSize+"\n");
PacketSniffer ps = new PacketSniffer(device[input], input, dataTest,
dataPort, windowSize);
Thread threadPs = new Thread(ps);
start = System.currentTimeMillis();
threadPs.start();
try {
    threadPs.join();
} catch (InterruptedException ex) {
    Logger.getLogger(IDS.class.getName()).log(Level.SEVERE, null, ex);
}
}
```

### A.5. Kode Sumber Proses Penggunaan Metode Mahalanobis Distance

```
package ids;

/**
 *
 * @author agus
 */
public class Mahalanobis {

    public double distance(double[] x, double[] y, double[] sd, double sf){
        double sum = 0;

        for (int i = 0; i < x.length; i++) {
            sum += Math.abs(x[i] - y[i]) / (sd[i] + sf);
        }
        return sum;
    }

}
```

## A.6. Kode Sumber Proses Pendeteksian Serangan

```

for (DataPacket dataPacketTes : dataTest) {
    if ("TCP".equals(dataPacketTes.getProtokol())) {
        for (DataModel dataTcp : modelTcp) {
            if (dataTcp.getDstPort() == dataPacketTes.getDstPort()) {
                mahalanobis = new Mahalanobis();
                mDist = mahalanobis.distance(dataPacketTes.getNgram(),
dataTcp.getMeanData(), dataTcp.getDeviiasiData(), sFactor);
                if (mDist > portTh[dataPacketTes.getDstPort()]) {
                    System.out.println("TCP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" |
"+Math.round(mDist*100.0)/100.0+"\n");
                    fwLog.append("TCP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" |
"+Math.round(mDist*100.0)/100.0+"\n");

                    fwLog.append("+++++START+++++
+++++\n");
                    fwLog.append(new
String(dataPacketTes.getPacketData(), StandardCharsets.US_ASCII)+"\n");

```



```

        fwLog.append("++++++END++++++\n");
        fwRecord.append("TCP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" | Attack\n");
        attackPacket.add(Math.round(mDist*100.0)/100.0);
    } else {
        fwRecord.append("TCP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" | Normal\n");
        ids.incremental("TCP", dataPacketTes.getNgram(),
dataPacketTes.getDstPort());
        freePacket.add(Math.round(mDist*100.0)/100.0);
    }
}

}

else if ("UDP".equals(dataPacketTes.getProtokol())) {
    for (DataModel dataUdp : modelUdp) {
        if (dataUdp.getDstPort() == dataPacketTes.getDstPort()) {
            mahalanobis = new Mahalanobis();
            mDist = mahalanobis.distance(dataPacketTes.getNgram(),
dataUdp.getMeanData(), dataUdp.getDevisasiData(), sFactor);
            if (mDist > portTh[dataPacketTes.getDstPort()]) {

```

```

        System.out.println("UDP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" |
"+Math.round(mDist*100.0)/100.0+"\n");
        fwLog.append("UDP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" |
"+Math.round(mDist*100.0)/100.0+"\n");

        fwLog.append("+++++START+++++
+++++\n");
        fwLog.append(new
String(dataPacketTes.getPacketData(), StandardCharsets.US_ASCII)+"\n");

        fwLog.append("+++++END+++++
+++++\n");
        fwRecord.append("UDP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" | Attack\n");
        attackPacket.add(Math.round(mDist*100.0)/100.0);
    } else {
        fwRecord.append("UDP |
"+dataPacketTes.getTimePacket()+" |
"+dataPacketTes.getSrcIP()+":"+dataPacketTes.getSrcPort()+" |
"+dataPacketTes.getDstIP()+":"+dataPacketTes.getDstPort()+" | Normal\n");
    }
}

```



## A.7. Kode Sumber Proses Incremental Learning

```
private void incremental(String proto, double[] ngram, int port){
    if (proto.equals("TCP")) {
        for (DataModel dataTcp : modelTcp) {
            if (dataTcp.getDstPort() == port) {
                int numNew = dataTcp.getTotalModel();
                double[] sumNew = dataTcp.getSumData();
                for (int i = 0; i < sumNew.length; i++) {
                    sumNew[i] = sumNew[i]+ngram[i];
                }
                double[] quadraticNew = dataTcp.getQuadraticData();
                for (int i = 0; i < quadraticNew.length; i++) {
                    quadraticNew[i] =
quadraticNew[i]+Math.pow(ngram[i], 2);
                }
                double[] meanNew = dataTcp.getMeanData();
                for (int i = 0; i < meanNew.length; i++) {
                    meanNew[i] =
(meanNew[i]*numNew+ngram[i])/(numNew+1);
                }
                numNew = numNew + 1;
                double[] deviasiNew = dataTcp.getDeviasiData();
                for (int i = 0; i < deviasiNew.length; i++) {
                    deviasiNew[i] =
Math.sqrt((numNew*quadraticNew[i]-Math.pow(sumNew[i], 2))/(numNew*(numNew-1)));
                }
            }
        }
    }
}
```

```

        dataTcp.setSumData(sumNew);
        dataTcp.setDeviassiData(deviasiNew);
        dataTcp.setMeanData(meanNew);
        dataTcp.setQuadraticData(quadraticNew);
        dataTcp.setTotalModel(numNew);
    }
}

else if (proto.equals("UDP")) {
    for (DataModel dataUdp : modelUdp) {
        if (dataUdp.getDstPort() == port) {
            int numNew = dataUdp.getTotalModel();
            double[] sumNew = dataUdp.getSumData();
            for (int i = 0; i < sumNew.length; i++) {
                sumNew[i] = sumNew[i]+ngram[i];
            }
            double[] quadraticNew = dataUdp.getQuadraticData();
            for (int i = 0; i < quadraticNew.length; i++) {
                quadraticNew[i] =
quadraticNew[i]+Math.pow(ngram[i], 2);
            }
            double[] meanNew = dataUdp.getMeanData();
            for (int i = 0; i < meanNew.length; i++) {
                meanNew[i] =
(meanNew[i]*numNew+ngram[i])/(numNew+1);
            }
            numNew = numNew + 1;
        }
    }
}

```

```

double[] deviasiNew = dataUdp.getDeviasiData();
for (int i = 0; i < deviasiNew.length; i++) {
    deviasiNew[i] =
Math.sqrt((numNew*quadraticNew[i]-Math.pow(sumNew[i], 2))/(numNew*(numNew-1)));
}
dataUdp.setSumData(sumNew);
dataUdp.setDeviasiData(deviasiNew);
dataUdp.setMeanData(meanNew);
dataUdp.setQuadraticData(quadraticNew);
dataUdp.setTotalModel(numNew);
}
}
}

```