Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

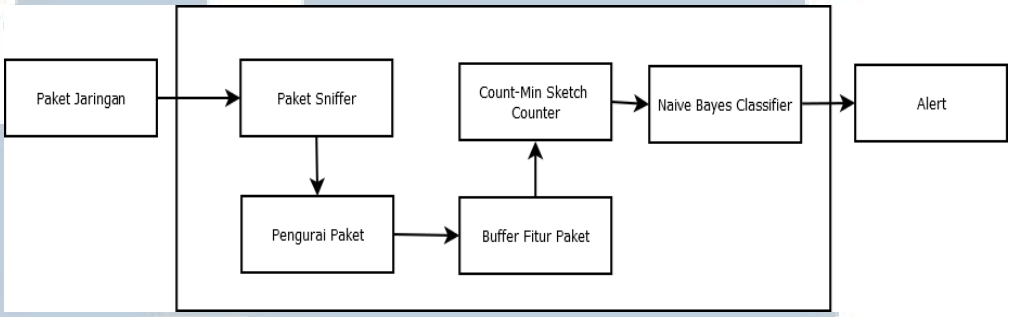
BAB III

METODE PENELITIAN

Diagram Blok Arsitektur Sistem

Berikut merupakan diagram blok arsitektur sistem secara keseluruhan.

3.1.

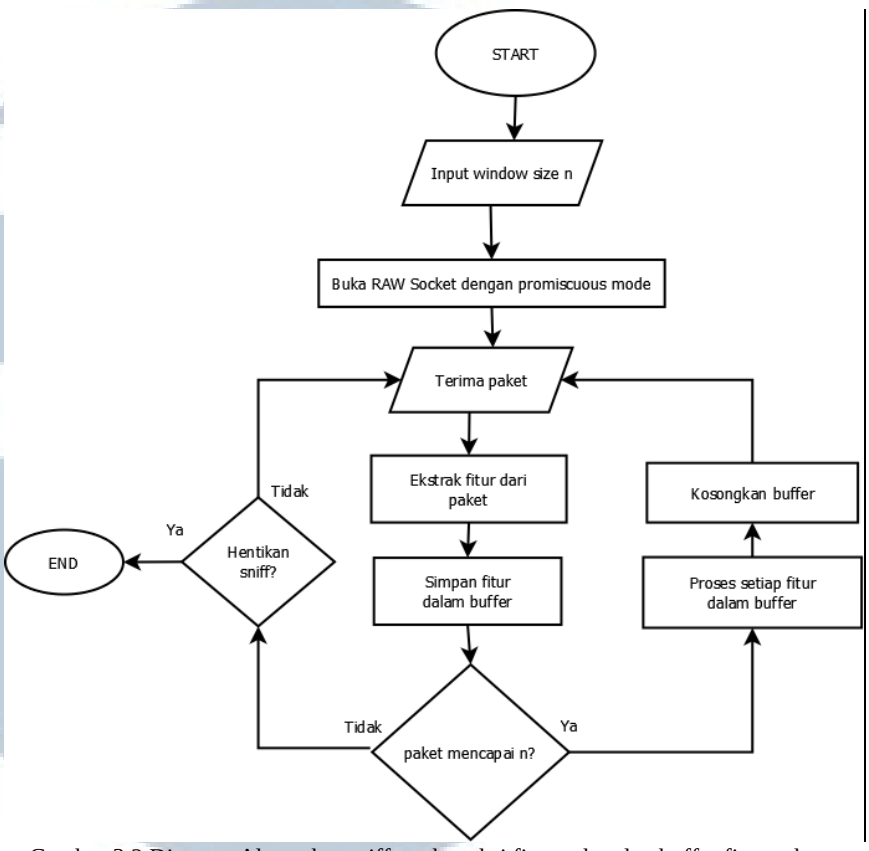


Gambar 3.1 Diagram Blok Sistem Anomaly Detection

Gambar 3.1 menunjukkan arsitektur *Anomaly Detection System* secara keseluruhan. Pada bagian awal sistem paket sniffer akan menerima paket jaringan sebagai *input* pada sistem, banyaknya paket yang dikumpulkan akan bergantung pada penentuan ukuran *window packet*. Setiap paket jaringan yang telah diterima akan disimpan dan diekstraksi fiturnya, fitur yang dimaksud adalah Alamat IP, Port, ukuran paket dan *TCP flag* paket.

Setelah proses ekstraksi, fitur paket disimpan dalam sebuah *buffer* dan diberi sebuah identifikasi berupa tuple string yang terdiri dari Alamat IP Tujuan, Alamat IP Sumber dan Port Tujuan. Setiap fitur pada *buffer* diakumulasi berdasarkan dari *tuple string* oleh *Count-Min Sketch Counter*. Fitur yang terakumulasi dapat digunakan oleh *Naive Bayes Classifier* untuk fase training ataupun prediksi. Hasil dari prediksi(Anomali atau Normal) dapat menjadi sebuah *Alert* sebagai output dari sistem.

3.1.1. Diagram Alur Paket Sniffer



Gambar 3.2 Diagram Alur paket sniffer, ekstraksi fitur paket dan buffer fitur paket

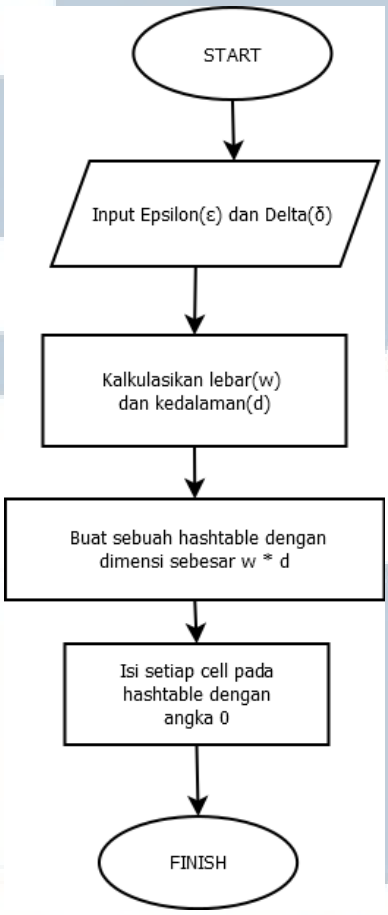
Gambar 3.2 merupakan diagram alur untur proses paket sniffing,ekstraksi fitur paket dan *buffer* untuk penyimpanan fitur paket. Paket sniffer pada sistem dimulai dengan mengambil paket sebanyak *n* dan setiap fitur paket diekstrasi dari paket tersebut. Fitur yang telah diekstrak disimpan dalam sebuah *buffer*, jika *buffer* telah mencapai batas *n* setiap fitur pada *buffer* akan diproses oleh *Count-Min Sketch Counter* yang akan dibahas pada sub bab selanjutnya. Setelah selesai diproses, *buffer* akan dikosongkan untuk diisi oleh paket selanjutnya. Pada sistem ini fitur yang diperlukan dari setiap paket merupakan :

1. *SYN Flag*
2. *ACK Flag*

3. Ukuran Paket
4. Port Tujuan
5. Alamat IP Sumber dan Tujuan

3.1.2. Diagram Alur *Count-Min Sketch*

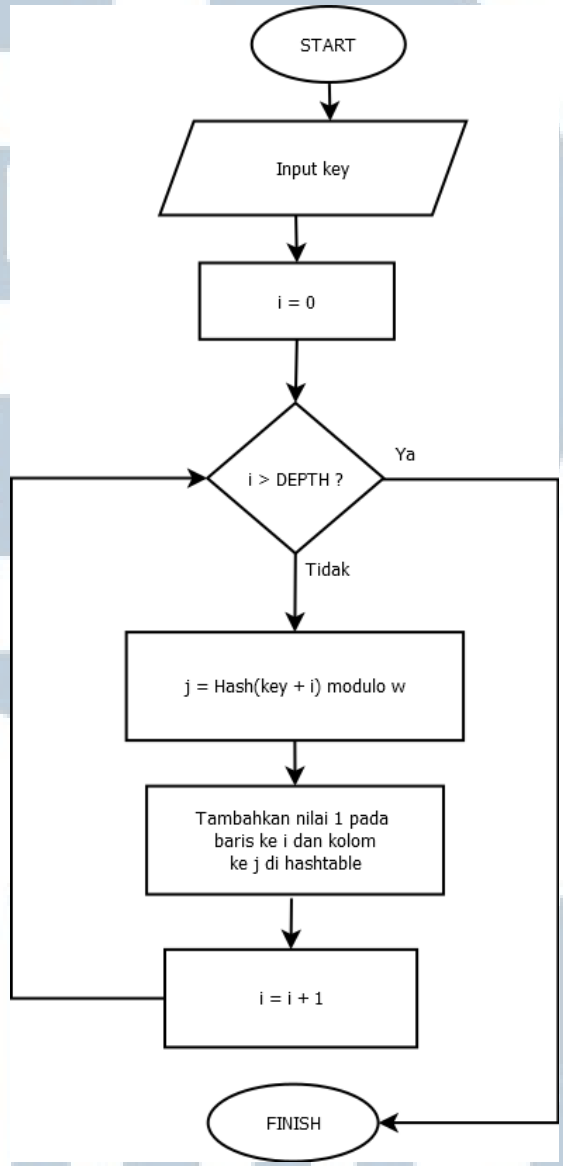
Program *Count-min sketch* akan mempunyai fungsi *update* dan *query*, namun sebelumnya *sketch* akan diinisiasikan bergantung pada *input* Epsilon(ε) dan Delta(δ) dari pengguna.



Gambar 3.3 Diagram alur inisiasi Count-Min Sketch

Diagram alur pada Gambar 3.3 mendeskripsikan proses insiasi *Count-Min Sketch* berdasarkan dari Epsilon dan Delta yang digunakan. Ukuran *hashtable Count-Min Sketch* dihitung menggunakan persamaan $w = \left\lceil \frac{e}{\varepsilon} \right\rceil$ untuk lebar dan $d = \left\lceil ln \frac{1}{\delta} \right\rceil$

untuk kedalamannya. Setelah dibuat *hashtable*nya, setiap cell pada *hashtable* diisi dengan angka 0.



Gambar 3.4 Diagram Alur untuk Prosedur Update

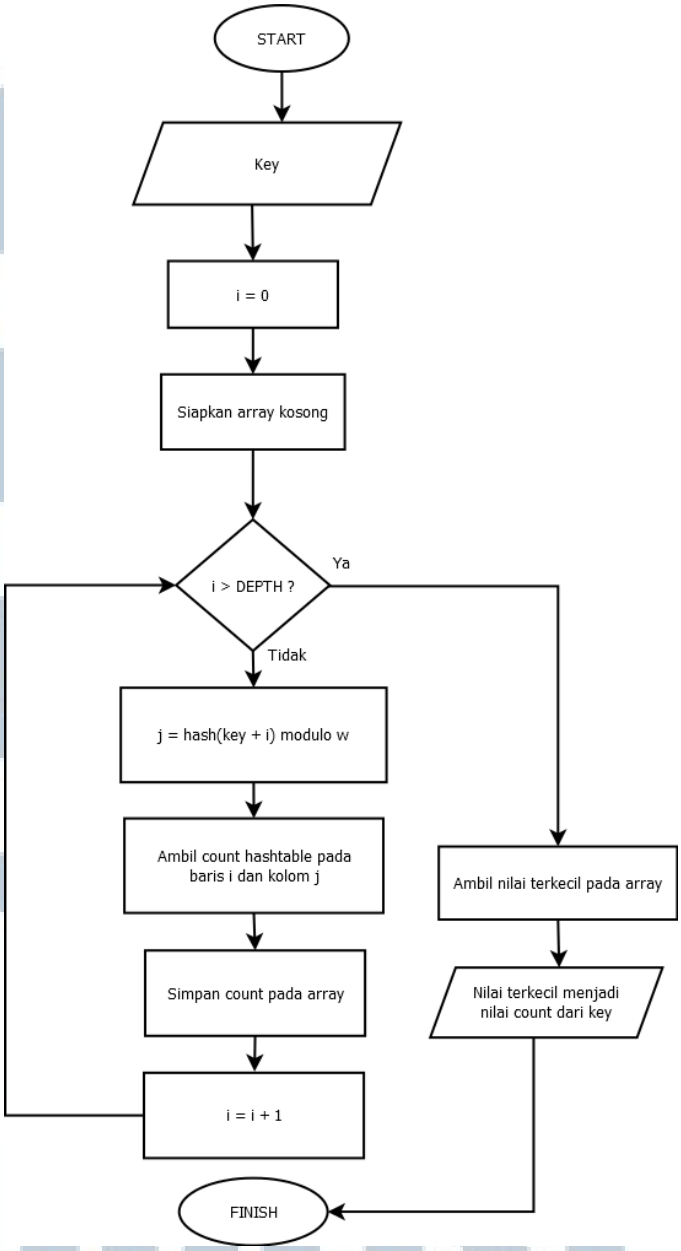
Gambar 3.4 merupakan diagram alur untuk prosedur *update*, berdasarkan dari Gambar 3.4 prosedur update dilakukan dengan mengambil *input key* berupa tuple string yang berisikan Alamat IP Sumber, Alamat IP sumber dan Port Tujuan, setiap elemen tersebut

akan dipisah dengan simbol titik koma (;). *Key* tersebut di*hash* bersama dengan variabel i sebagai seed pada fungsi *hash*.

Hash yang dihasilkan dimodulo dengan variabel w (Lebar *hashtable*), proses ini dilakukan untuk menghindari kesalahan alokasi memori saat fungsi *hash* menghasilkan angka yang lebih besar dari lebar *hashtable*.

Hasil modulo disimpan dalam variabel j dan digunakan sebagai indeks kolom *hashtable*, dan variabel i digunakan sebagai indeks baris *hashtable*. Berikut merupakan pseudocode prosedur update *count min sketch* pada sistem :

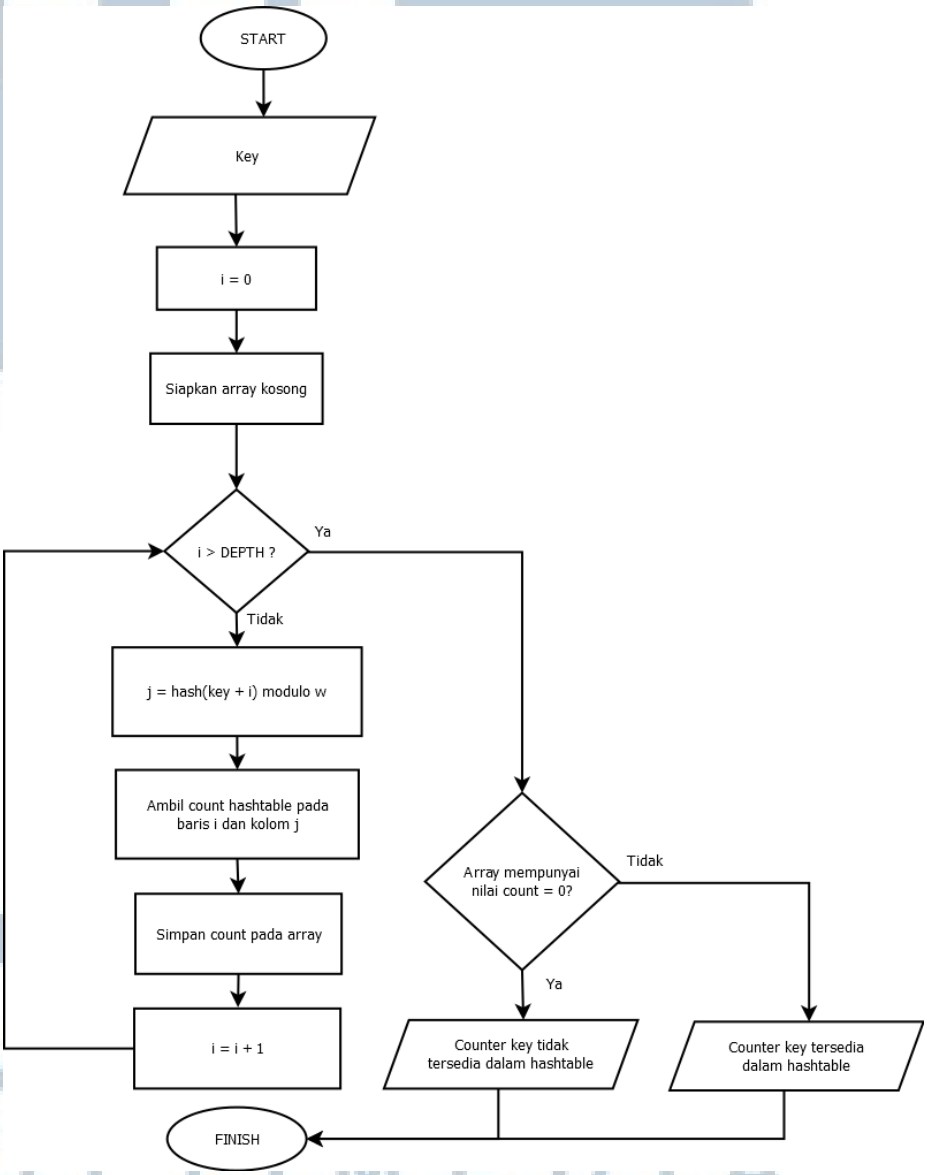
```
Input : Tuple String T, angka X
Output : Angka pada indeks counter untuk sebuah tuple T
         ditambahkan dengan X
Inisiasi hashtable dengan dimensi D x W
For i = 1 to D
         H ← Gunakan fungsi hash h_i(T)
         Indeks kolom j ← H mod W
         Tambahkan X pada counter pada indeks [i, j]
End
```



Gambar 3.5 Diagram Alur untuk Prosedur Point Query

Gambar Gambar 3.5 menunjukkan diagram alur prosedur *point query* yang merupakan sebuah prosedur untuk mengambil *count* sesuai dengan *key* yang di*input*. Setelah menerima *key* berupa *tuple string*, *key* tersebut di*hash* bersama dengan variabel *i* , lalu hasil dari *hash* dimodulo dengan lebar *hashtable* untuk menghindari kesalahan alokasi memori saat fungsi *hash* menghasilkan angka yang lebih besar dari lebar *hashtable*. Hasil modulo disimpan pada variabel *j* sebagai

indeks kolom *hashtable*. *Count* dari *key* yang di*input* didapatkan pada baris ke *i* dan kolom ke *j* di *hashtable*, lalu *count* dari setiap baris dikumpulkan dalam sebuah array. Hasil dari *count* ditentukan dengan mengambil nilai terkecil pada array yang berisikan *count* dari setiap baris.



Gambar 3.6 Diagram alur periksa ketersediaan counter

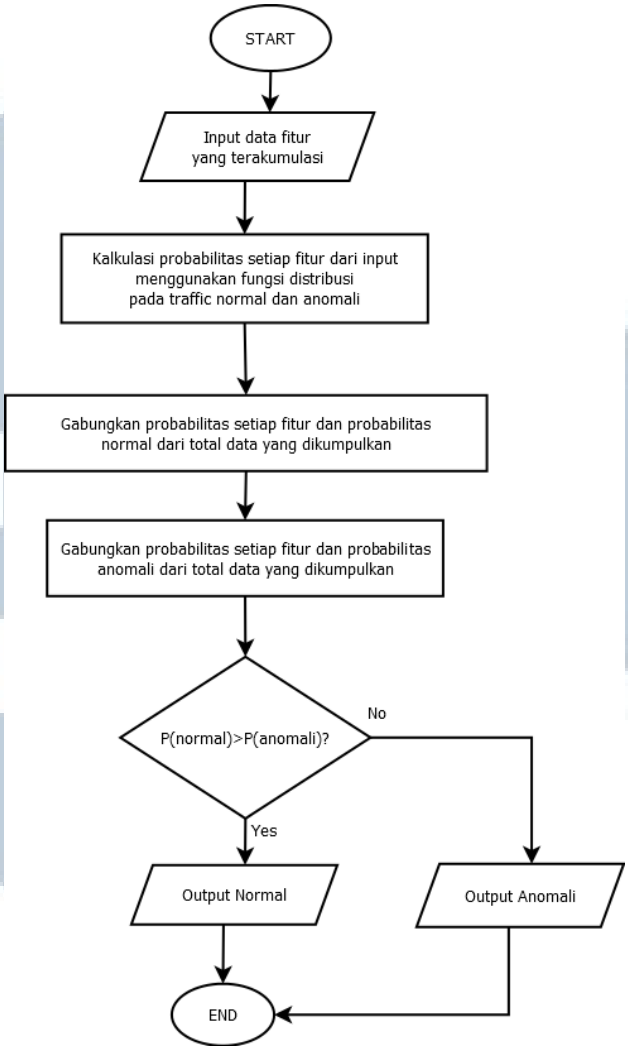
Gambar 3.6 menunjukkan diagram alur untuk memeriksa ketersediaan *counter* dari sebuah *key* yang di*input*. Prosedur dimulai

dengan menerima *key* yang ingin diperiksa ketersediaan *counter*nya pada *hashtable*. Setelah itu program akan melakukan prosedur untuk mendapatkan nilai *count* dari *key*, sama seperti prosedur Point *Query* yang pada diagram alur Gambar 3.5. Perbedaannya terletak pada pengambilan nilai pada array, untuk memeriksa ketersediaannya *counter* sebuah *key*, setiap nilai *count* yang dikumpulkan pada array diperiksa jika mempunyai nilai *count* 0, jika terdapat nilai 0 maka *key* yang di*input* tidak mempunyai *counter* pada *hashtable*.

3.1.3. Diagram Alur Naïve Bayes Classifier

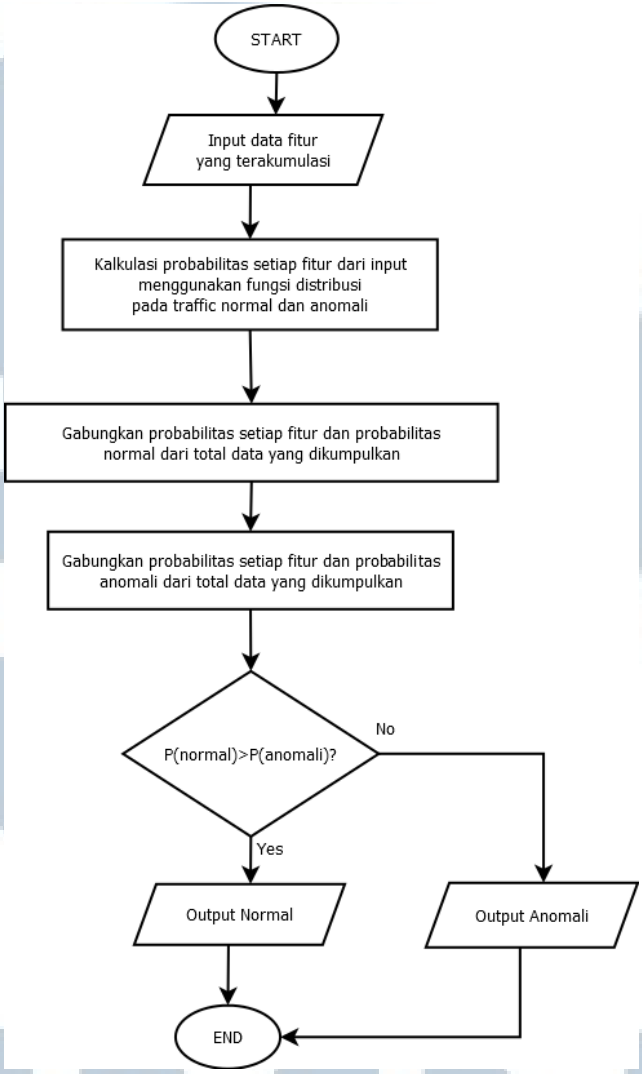
Alur untuk *Naïve Bayes Classifier* dibagi menjadi dua buah prosedur yaitu *training* dan prediksi.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.7 Diagram Alur Training Naive Bayes

Gambar 3.7 merupakan diagram alur prosedur *training* pada *Naive Bayes Classifier*. Prosedur *training* dimulai dengan mengambil *input* data fitur(ukuran paket, *SYN Flag count*, dll) yang telah terakumulasi dari struktur data yang digunakan (*Count-Min Sketch* atau *Linked-List*). Prosedur juga mengambil *input* berupa nama kelas(normal/anomali) data fitur yang dimasukan. Untuk setiap fitur pada setiap kelas data dihitung *mean* dan *variance*nya. Lalu *Mean* dan *variance* yang dihasilkan dari setiap fitur per kelas disimpan sebagai data yang diperlukan untuk memprediksi data baru.



Gambar 3.8 Diagram Alur Prediksi Naive Bayes Classifier

Gambar 3.8 menunjukkan diagram alur untuk memprediksi kelas data baru. Proses dimulai dengan menghitung probabilitas kedua kelas (normal atau anomali) dan probabilitas fitur per kelas dengan Distribusi Normal menggunakan *Mean* dan *Variance* yang didapatkan saat *training*. Setelah itu probabilitas bahwa data baru dari kedua kelas dibandingkan dan ditentukan kelasnya jika salah satu probabilitas dari kelas lebih besar dari yang lainnya.

Uji Banding dan Performa

3.1.4.1 Pengujian 1 : Uji Kecepatan Menggunakan Banyak *Distinct flow*

3.1.4.

Sistem akan disimulasikan untuk menerima *Distinct flow* yang berisikan kombinasi dari Alamat IP Sumber dan Alamat IP Tujuan yang bervariasi dalam jumlah yang banyak. Simulasi ini bertujuan untuk membandingkan performa *Linked-List* dan *Count-Min Sketch* saat lalu lintas jaringan mempunyai *Distinct flow* yang banyak.

Pengujian ini juga akan membandingkan kecepatan prediksi menggunakan 2 buah fungsi *hash* yang berbeda pada *Count-Min Sketch*, yaitu *MurmurHash3* dan *Universal Hash*. Kedua fungsi *hash* banyak digunakan pada berbagai penelitian karena kecepatan *hash* yang cepat dan menghasilkan *hash collision* yang rendah.

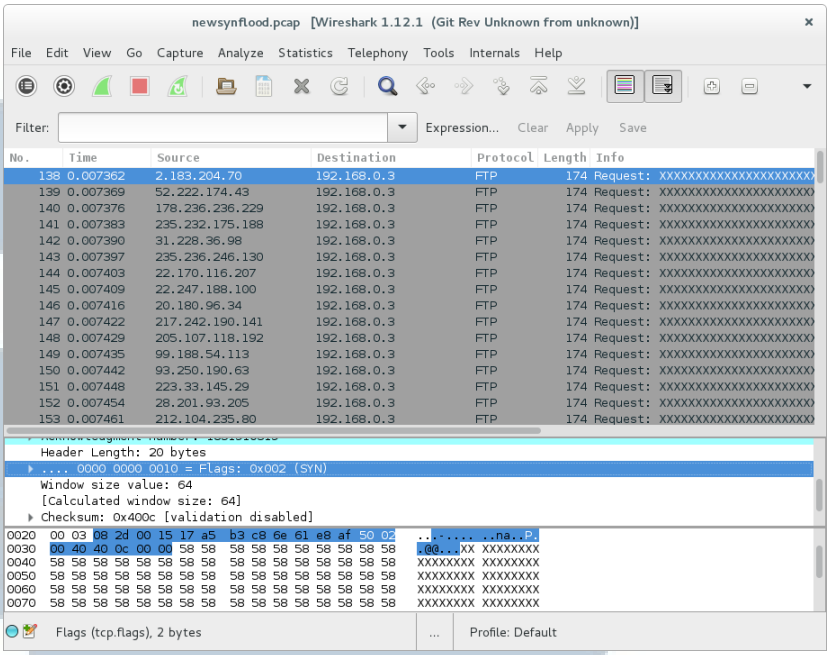
Universal Hash seringkali diimplementasikan sebagai sebuah *general hash function* dalam penelitian berbasis *hardware* salah satunya dari "*Implementing an online sketch based change detection on a NetFPGA Platform*" [8]. *Universal Hash* mempunyai keunggulan dimana proses *hashing* tidak melibatkan operasi yang banyak. Sedangkan *MurmurHash* banyak digunakan pada penelitian yang menggunakan fungsi *hash* tersebut pada *Bloom Filter* seperti pada "*Bloom Filter Based Intrustion Detection For Smart Grid*" [17].

Performa diukur dalam segi kecepatan proses penambahan data pada *Naïve Bayes* dan total pengalokasian memori yang digunakan oleh kedua struktur data. Kecepatan proses dihitung dengan memodifikasi pada program untuk memberikan *output* berupa waktu dalam mikrosekon, sementara jumlah alokasi memori diukur menggunakan *Valgrind Memory Checker Tool v3.10.0* [18].

3.1.4.2 Pengujian 2 : Uji Akurasi dan Kecepatan Prediksi Pada *Naive Bayes Classifier*

Pengujian ini dilakukan untuk membandingkan akurasi prediksi antara *Count-Min Sketch* dan *Linked-List*. Pengujian dilakukan dengan menggunakan dua buah *trace file*. *Trace* pertama berisikan lalu lintas jaringan anomali berupa serangan *SYN Flood* yang melakukan penyerangan menggunakan *Alamat IP* palsu, lalu lintas jaringan ini dihasilkan menggunakan *Hping v3.0.0* [19]. Hasil lalu lintas yang didapatkan dapat dilihat pada Gambar 3.9.

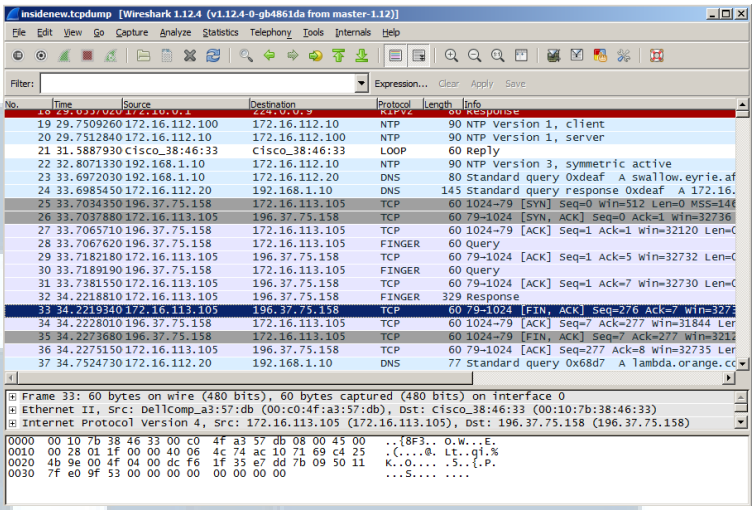
UNIVERSITAS
MULTIMEDIA
NUSANTARA



Gambar 3.9 Lalu lintas Anomali yang dihasilkan

Trace kedua pada Gambar 3.10 berisikan lalu lintas jaringan normal dari *1999 DARPA Intrusion Detection Evaluation Data Set* [20]. Kedua *trace file* digunakan sebagai data *training Naïve Bayes Classifier* untuk mengklasifikasikan lalu lintas jaringan normal dan anomali. Kedua *trace file* juga digunakan ulang sebagai data untuk diprediksi, sehingga dapat menjadi referensi untuk menilai keakuratan prediksi. Jumlah paket dari kedua *trace* dipotong hingga menjadi 60000 paket untuk mendapatkan hasil yang adil.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.10 Lalu lintas Normal yang digunakan

Pengujian juga dilakukan untuk menguji kecepatan prediksi dari *Count-Min Sketch* dan *Linked-List*. Pengujian dimulai dengan memvariasikan *packet count window* sebesar 100 , 1000, 2000, 10000, 15000, 20000 dan 30000 untuk melihat pengaruh jumlah paket yang diproses dengan kecepatan prediksi.

3.1.4.3 Pengujian 3 : Pengaruh Ukuran *Hashtable* terhadap Akurasi Prediksi

Pengujian ini dilakukan untuk mengetahui pengaruh ukuran *hashtable* terhadap akurasi prediksi pada *Naive Bayes Classifier*. Ukuran *hashtable* dapat diatur dengan mengubah konstanta ε(Epsilon) dan σ(Delta) pada inisiasi *hashtable*. Pengujian dimulai dengan menggunakan *packet count window size* 100 untuk *training* dan memprediksi sebuah *trace* yang mempunyai lalu lintas jaringan yang normal dan anomali menggunakan ukuran *hashtable* yang berbeda-beda.

Akurasi prediksi diukur dengan membandingkan jumlah *Distinct flow* yang terdeteksi sebagai normal dan anomali.

Instrumen Penelitian

Penelitian dilakukan menggunakan instrumen perangkat keras dan perangkat lunak berikut :

3.2.

1. PC berspesifikasi prosesor Intel core i3, RAM 8GB dan Windows 10 sebagai sistem operasinya.
2. Virtual Box v5.1.26 dengan Debian OS v9.1.0 32-Bit
3. NMAP v6.47
4. Libpcap v1.6.2
5. Wireshark v1.12.1
6. GNU G++ Compiler v4.9.2
7. Visual Studio Code Text Editor
8. Valgrind Memory Checker V3.10.0

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA