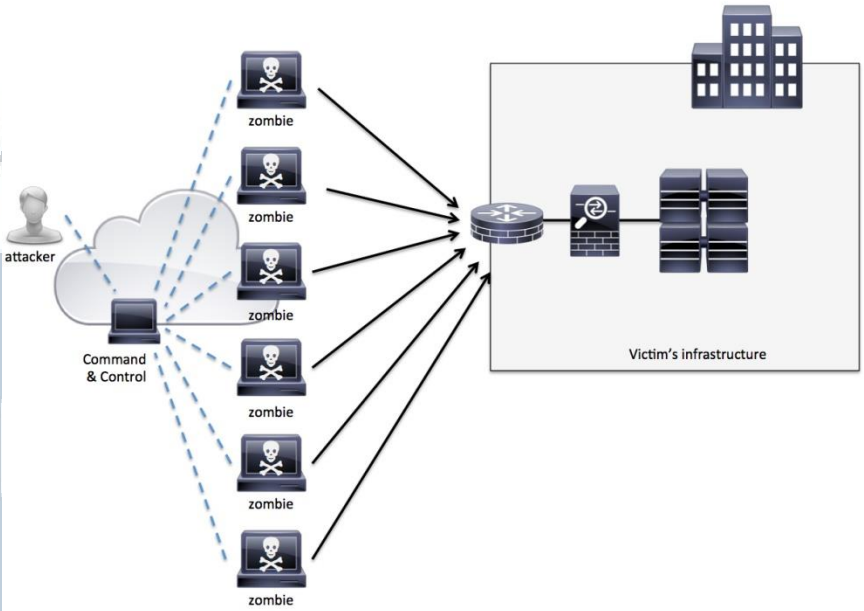
BAB II

TINJAUAN PUSTAKA

2.1. Denial of Service (DoS)

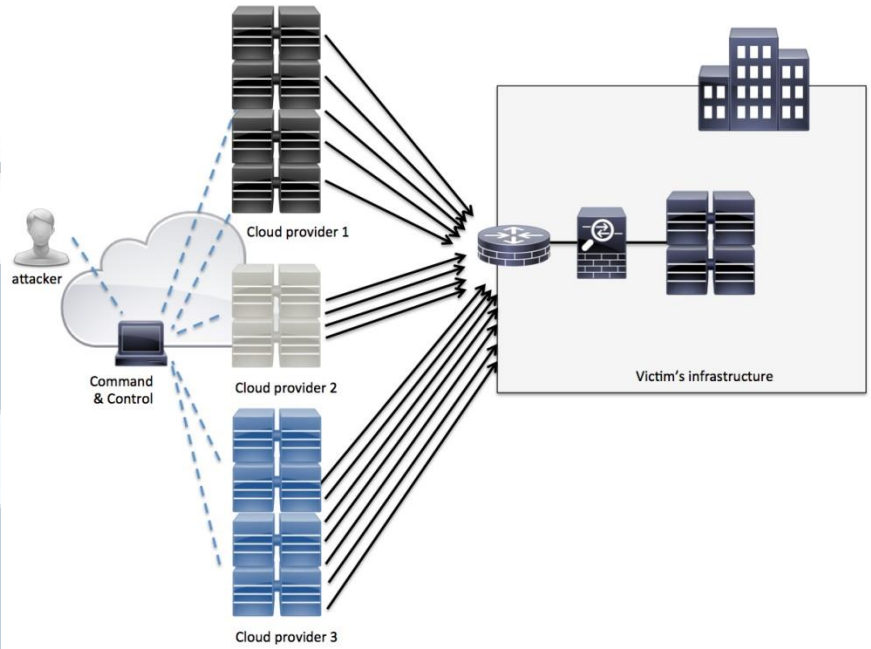
Serangan *DoS* adalah serangan yang menggunakan satu atau lebih mesin untuk menyerang sebuah target dengan tujuan mencegah target melakukan sesuatu yang berguna [7]. Tujuan dari serangan ini sendiri adalah untuk menghentikan layanan target, yang umumnya adalah *server* dalam jaringan. Salah satu teknik serangan yang paling terkenal adalah *Distributed Denial of Service* (*DDoS*). Serangan ini umumnya memanfaatkan *bot* atau *host* yang telah terinfeksi untuk melancarkan serangan *DoS* secara terorganisir untuk menghentikan layanan *server*. Pada *website*, Cisco mengelompokkan serangan *DDoS* secara umum ke dalam 3 kategori [8]:

1. *Volume-based DDoS attacks* : Penyerang membanjiri target dengan paket atau koneksi dalam volume yang besar sehingga menghabiskan *resource* atau *bandwidth* dari target. Serangan ini umumnya memanfaatkan beberapa mesin yang terinfeksi menjadi bagian dari *botnet* seperti pada Gambar 2.1.



Gambar 2.1 Skema *Volume-based DDoS attacks* dengan *botnet* **[8]**

Seiring berkembangnya waktu, serangan seperti ini juga dapat dilancarkan dari *data center* milik *cloud service provider*. Hal ini dimungkinkan karena penyerang dapat menyewa atau bahkan meretas sistem berbasis *cloud* yang memiliki *bandwidth* yang besar seperti pada Gambar 2.2.



Gambar 2.2 Skema *Volume-based DDoS attacks* dengan *cloud server* yang diretas **[8]**

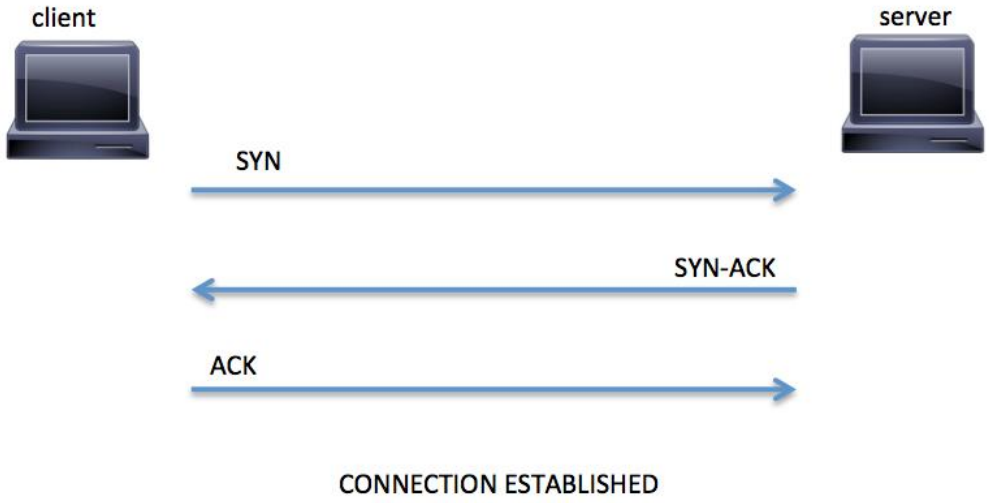
2. *Application DDoS attacks* : Sasaran serangan ini pada umumnya *web server* dan *service*. Serangan ini dilancarkan dengan melakukan request HTTP POST ataupun GET dalam jumlah banyak dan terus menerus sehingga *server* akan terus menerus melayani *request* yang diberikan penyerang sampai *resource* yang dimiliki tidak memadai untuk melayani *request* yang sebenarnya. Selain HTTP, ada juga serangan yang memanfaatkan Voice over IP (VoIP), DNS, dsb. Beberapa teknik serangan dalam kategori ini dianggap lebih efektif dari yang lain karena membutuhkan koneksi jaringan yang lebih kecil dari yang lain untuk mecapai tujuannya.
3. *Low-rate DoS (LDoS) attacks* :

   Serangan ini berbeda dari jenis yang lain karena memanfaatkan kelemahan dan celah desain yang diimplementasikan pada aplikasi. Hal ini menyebabkan serangan *LDoS* dapat mencapai tujuannya

tanpa melakukan serangan dalam jumlah yang besar secara terus menerus.

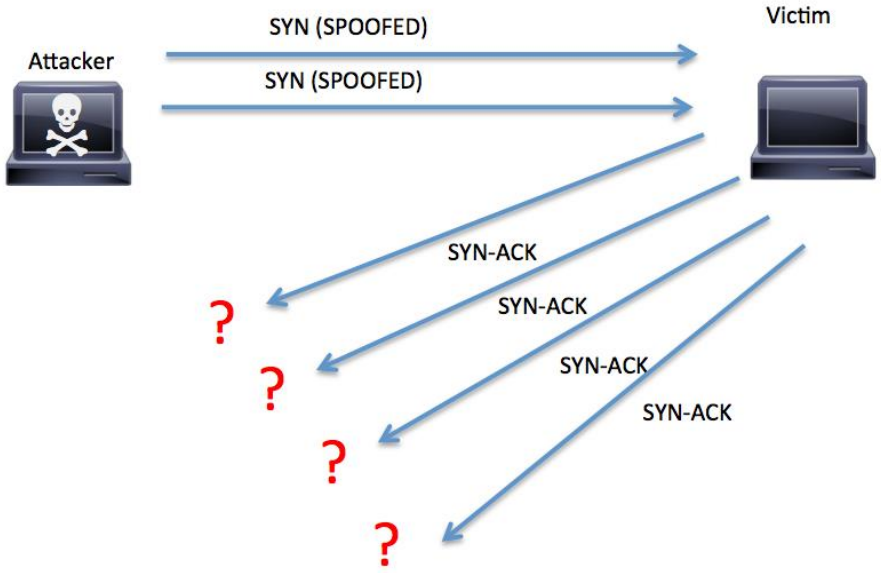
2.1.1. SYN Flood

SYN Flood adalah salah satu teknik serangan *DDoS* yang spesifik. Umumnya dalam protokol TCP, ketika *client* akan menginisiasi koneksi dengan *server*, dan kemudian *client* dan *server* akan saling bertukar paket untuk membentuk koneksi dengan urutan seperti pada Gambar 2.3. Pembentukan koneksi ini disebut *Three-way Handshake*.



Gambar 2.3 Skema TCP *Three-way Handshake* **[8]**

Dalam serangan *SYN Flood*, penyerang tidak akan menjawab *server* dengan paket ACK dengan menggunakan *source* IP palsu atau sebatas tidak menjawab SYN-ACK yang dikirimkan *server* seperti pada Gambar 2.4.



Gambar 2.4 Skema Serangan *SYN Flood* **[8]**

2.2. OpenFlow Protocol

OpenFlow adalah protokol yang masih dikembangkan dan merupakan salah satu dari sekian banyak protokol yang digunakan untuk melakukan pemrograman *behavior* switch. Protokol ini mampu memprogram *flow table* pada berbagai *router* dan *switch* yang mendukung protokol ini [9]. Protokol ini juga menghadirkan suatu standar kumpulan fungsi minimum yang harus didukung sebagai acuan berbagai vendor dalam menghadirkan produk dengan dukungan OpenFlow sehingga setiap *switch* yang mendukung dapat menghasilkan *behavior* yang sama ketika diberi instruksi yang sama melalui protokol OpenFlow. Hal ini mempermudah pengembang dan peneliti dalam mengembangkan fitur menggunakan OpenFlow karena dapat dipastikan berjalan disemua *switch* yang mendukung protokol ini.

2.2.1. RYU Controller

Untuk menggunakan berbagai fitur dari OpenFlow, *switch* perlu terhubung ke *controller* yang akan memberikan instruksi melalui protokol OpenFlow. Salah satu *controller* yang ada saat pengembangan awal OpenFlow adalah Ryu. Ryu dibuat menggunakan bahasa pemrograman Python dan sampai saat ini telah mendukung OpenFlow versi 1.0, 1.2, 1.3, 1.4 dan 1.5. Dalam penelitian [10], Ryu dianggap cocok untuk digunakan dalam tahap *prototyping* serta memiliki hasil test keamanan yang cukup baik dibandingkan 6 *open-source controller* yang lain. Selain itu penelitian [11] juga menyimpulkan bahwa Ryu memiliki dokumentasi yang lengkap dan memenuhi 10 kriteria yang ditetapkan sebagai *controller* yang baik.

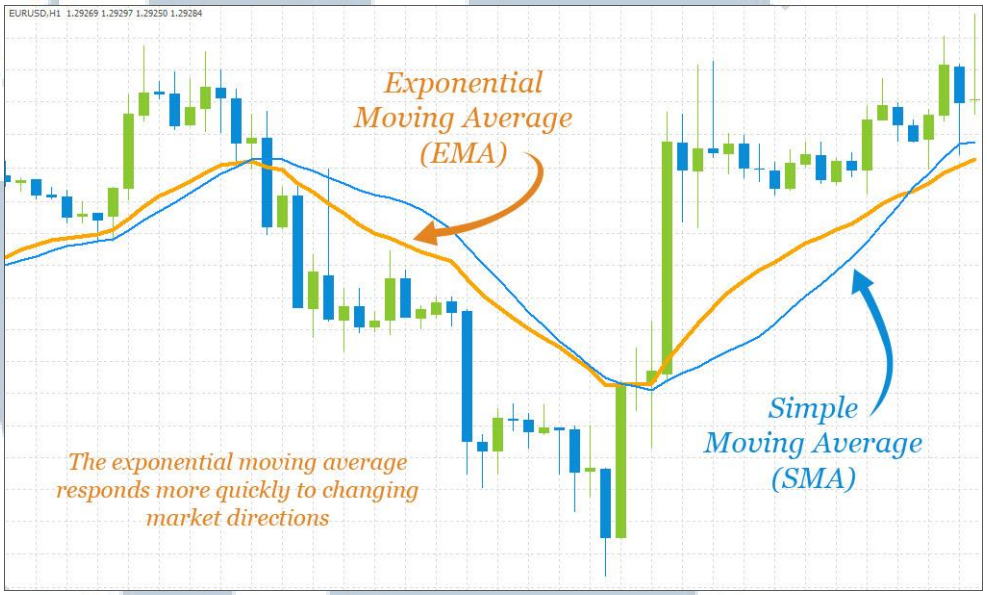
2.3. Moving Average

Dalam statistik, *moving average* (disebut juga *rolling average* atau *running average*) adalah cara penghitungan untuk menganalisa dan mengidentifikasi perubahan tren (kondisi) dalam statistik. Umumnya dimanfaatkan untuk mengurangi kesalahan dalam membaca statistik akibat fluktuasi jangka pendek sehingga sangat berguna pada sektor perdagangan saham sebagai indikator untuk membeli maupun menjual saham, namun penerapannya tidak terbatas hanya pada sektor bisnis, namun juga dalam bidang sains dan teknik.

2.3.1. Exponential Moving Average

Salah satu variasi dari *moving average* yang terkenal adalah *exponential moving average*, dimana data rata-rata terbaru diberi beban atau

persentase lebih besar daripada data rata-rata terdahulu sehingga memiliki respon yang lebih tanggap dan responsif terhadap perubahan yang terbaru seperti yang divisualisasikan pada Gambar 2.5.



Gambar 2.5 Visualisasi *Exponential Moving Average* **[12]**

Formula penghitungan EMA menggunakan multiplier adalah seperti berikut:

$$Multiplier = \frac{2}{TimePeriods + 1}$$

$$EMA = (CurrentValue - PreviousEMA) * Multiplier + PreviousEMA$$

2.4. Naïve Bayes Classifier

Dalam *Machine Learning, naïve bayes classifier* adalah teknik untuk membuat sebuah *classifier* yang digunakan dalam menentukan label *class* suatu kumpulan data berdasarkan nilai dari fitur data tersebut, dimana label dari *class* yang ada merupakan hasil pengelompokkan berdasarkan *training* yang dilakukan

sebelumnya. Prinsip utama dari *classifier* ini adalah sebuah asumsi bahwa nilai dari sebuah fitur tidak terikat atau terpengaruh dengan nilai dari fitur yang lain atau dapat disebut independen, mesikpun dalam penerapannya seringkali prinsip tersebut dilanggar karena adanya korelasi antar fitur yang dipilih, namun Harry Zhang telah membuktikan dalam penelitiannya bahwa *naïve bayes* tetap dapat bekerja dengan optimal meskipun fitur-fiturnya saling berkaitan [13]. Keuntungan dalam penggunaan *naïve bayes* adalah jumlah data yang diperlukan dalam *training* untuk menentukan nilai probabilistik suatu fitur terhadap *class* sedikit.

2.4.1. Gaussian Naïve Bayes

Dalam *gaussian naïve bayes,* distribusi normal digunakan untuk merepresentasikan nilai probabilitas suatu fitur terhadap suatu *class* dalam *naïve bayes classifier*. Langkah pertama dalam melakukan estimasi menggunakan *Gaussian Naïve Bayes* adalah mencari persebaran normal dari data *training* dengan mengkalkulasi rata-rata(μ) dan *variance*(σ^2) setiap fitur serta menghitung *prior probability*($p(C_n)$) setiap *class*.

$$p(C_n) = \frac{N_c}{\Sigma(N_C)} \quad (1)$$

Persamaan (1) merupakan persamaan *prior probability* suatu class berdasarkan data training, dimana C adalah *class* dan N adalah jumlah kejadian suatu *class*.

$$p(X_i|C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} exp(-\frac{(X_i-\mu_C)^2}{2\sigma_C^2}) \quad (2)$$

Persamaan (2) merupakan persamaan probabilitas suatu fitur terhadap sebuah class, dimana X_i adalah salah satu fitur suatu data dan C adalah *class*.

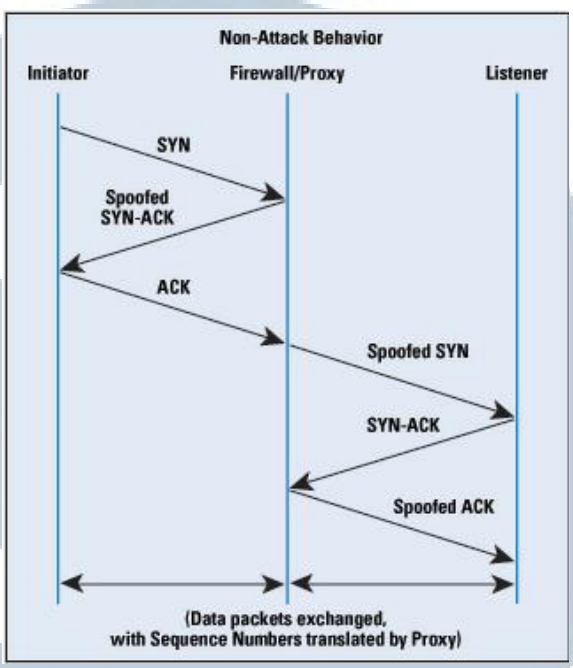
$$p(X|C) = p(X_1|C) * p(X_2|C) * \ldots * p(X_n|C) \quad (3)$$

Persamaan (3) menunjukkan persamaan probabilitas suatu data terhadap sebuah class, dimana semua nilai probabilitas fitur dari data tersebut yang didapatkan dari persamaan (2) dikalikan karena asumsi semua fitur independen dan tidak memiliki korelasi satu sama lain.

$$P(C_n|X) = \frac{p(X|C_n) * p(C_n)}{p(X|C_1) * p(C_1) + p(X|C_2) * p(C_2) + \cdots + p(X|C_n) * p(C_n)} \quad (4)$$

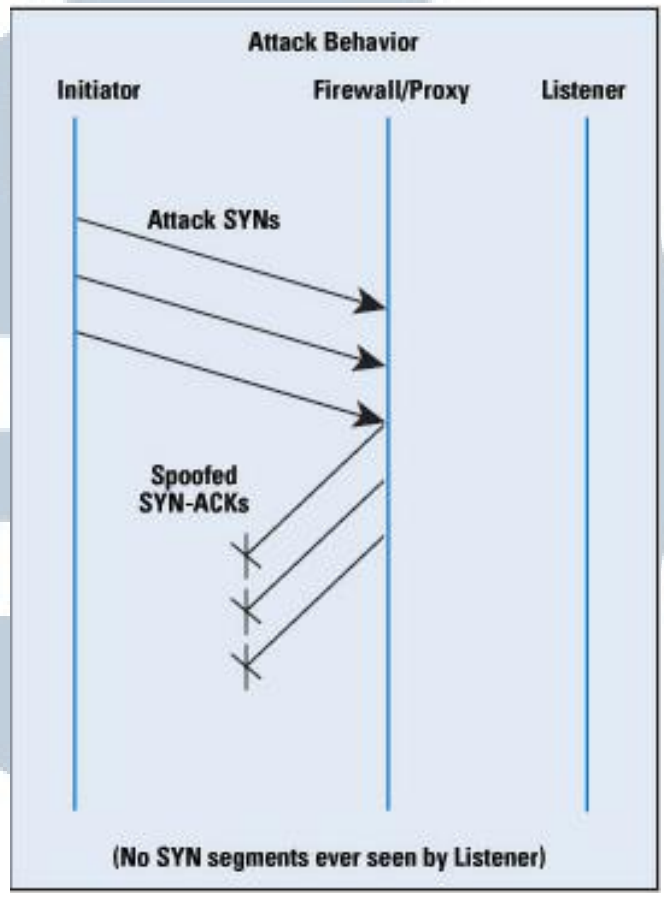
Persamaan (4) adalah persamaan yang digunakan untuk menentukan suatu data termasuk *class* tertentu, dimana untuk setiap *class* hasil dari Persamaan (3) dikalikan dengan nilai dari Persamaan (1) dan dibagi dengan jumlah dari perkalian hasil Persamaan (3) dengan Persamaan (1) untuk semua *class*. *Class* dengan nilai Persamaan (4) tertinggi merupakan *class* dari data yang diprediksi.

2.5. SYN Proxy



Gambar 2.6 Alur Komunikasi *SYN Proxy* Dalam Kondisi Normal

SYN Proxy merupakan salah satu jenis *firewall* yang digunakan memitigasi serangan *DDoS* jenis *SYN Flood* dan telah diterapkan oleh beberapa vendor jaringan pada produk *firewallnya*. *SYN Proxy* juga tersedia secara *open-source* untuk diimplementasikan dalam *firewall* Linux seperti firehol. Cara kerja *SYN Proxy* adalah menjadi perantara antara *client* dan *server*. Untuk setiap paket TCP SYN yang masuk, *proxy* akan mengembalikan SYN+ACK dan menunggu balasan ACK dari *client*. Dalam kondisi normal, maka *client* akan membalas dengan ACK sehingga *three-way handshake* terpenuhi dan koneksi akan terjalin antara *client* dengan *proxy*, kemudian *proxy* akan mewakili *client* untuk melakukan *three-way handshake* ke *server* dan setelah *connection established*, maka setiap paket yang dikomunikasikan akan langsung diteruskan sampai selesai seperti pada Gambar 2.6.



Gambar 2.7 Alur Komunikasi *SYN Proxy* Dalam Kondisi Diserang

Sedangkan pada kondisi dalam serangan, maka *proxy* akan terus mengirimkan SYN+ACK kembali untuk membangun koneksi dengan *client* tanpa membangun koneksi ke *server*, sehingga serangan yang datang akan terhenti di *proxy* dan tidak masuk ke *server* seperti yang ditampilkan pada Gambar 2.7.

2.6. Penelitian Terdahulu

Penelitian tentang cara mendeteksi maupun memitigasi serangan *DDoS* telah dilakukan. Beberapa diantaranya adalah sebagai berikut:

2.6.1. Implementasi Count-Min Sketch pada Anomaly Detection System Berbasis Naïve Bayes

Nelson Wijaya dalam penelitiannya mengimplementasikan struktur data *count-min sketch* pada *Anomaly Detection System* berbasis *Naïve Bayes* untuk mendeteksi adanya serangan *SYN Flood* secara *offline*. Hasil dari penelitiannya menunjukkan bahwa struktur data *count-min sketch* dapat memberikan performa yang lebih baik dari *linked-list* pada kondisi-kondisi tertentu seperti saat *window size* dan *distinct flow* lebih banyak dengan mengorbankan akurasi, namun tanpa penurunan ketepatan secara signifikan. Namun dalam kondisi yang sebaliknya, struktur data tersebut tidak dapat mengimbangi performa dari struktur data *linked-list*. Hal ini dapat menjadi masalah karena hanya akan memiliki performa yang baik saat melakukan deteksi pada *traffic* jaringan yang sedang mendapat serangan dari banyak *source* IP, namun dapat memberikan performa yang buruk dan keputusan yang kurang tepat ketika mendapatkan serangan dari *source* IP yang sedikit [4].

2.6.2. Detection of DoS/DDoS attack against HTTP Servers using Naïve Bayesian

Vijay Katkar, Amol Zinjade, Suyed Dalvi, Tejal Bafna, Rashmi Mahajan dalam penelitiannya melakukan pencatatan *traffic* secara independen pada masing-masing *cluster web server* dan kemudian dikirim ke *Intrusion Detection System* sentral untuk di agregat dan di analisa menggunakan *Naïve Bayes Classifier*. Salah satu poin dari hasil penelitian

ini adalah Naïve Bayes Classifier dapat mendeteksi *SYN Flood* dengan tingkat akurasi diatas 90% [6].

2.6.3. An Efficient DDoS Detection with Bloom Filter in SDN

Peng Xiao, Zhiyang Li, Heng Qi, Wenyu Qu, dan Haisheng Yu dalam penelitiannya menggunakan struktur data *Bloom Filter* untuk melakukan pencatatan informasi *flow* dan alamat IP yang dicurigai sebagai *link flooding attack* secara efisien. Sistem yang dirancang dibagi menjadi 2 modul, *collector* dan *detector* dimana modul *collector* akan melakukan *query* ke *controller* OpenFlow setiap jangka waktu tertentu untuk mendapatkan fitur statistik mengenai *flow table* yang ada dan mencatatnya pada *Bloom Filter*, sedangkan *detector* akan melakukan *sniffing* semua paket yang ada di jaringan secara *real-time* dan mengekstrak fitur IP paket untuk disimpan di *Bloom Filter*. Hasil dari penelitian ini menunjukkan *Bloom Filter* dapat mendeteksi *link flooding attack* dengan baik [5].

2.6.4. Real-time detection of changes in network with OpenFlow based on NetFPGA implementation

Yu-Kuen Lai, Chun-Chieh Lee, Bo-Hsun Huang, Theophilus Wellem, Nan-Cheng Wang, Tze-Yu Chou, dan Hargyo Tri Nugroho dalam penelitiannya memanfaatkan kemampuan c*ontroller* OpenFlow dalam mengalihkan dan menentukan tindakan terhadap paket yang sesuai. Dengan memanipulasi paket yang masuk untuk dikirimkan ke tujuan dan sistem

pendeteksi, setiap paket yang masuk dapat dianalisa dan jika didapati melebihi *treshold* yang telah dikalkulasi, maka dapat disimpulkan bahwa *flow* tersebut merupakan serangan [14].