Jurnal Sistim Informasi dan Teknologi

https://jsisfotek.org/index.php

2023 Vol. 5 No. 3 Hal: 22-30 e-ISSN: 2686-3154

Sistem Pencegahan Serangan *Distributed Denial Of Service* Pada Jaringan SDN

Fidyatun Nisa[1]✉, Suci Ramadona[2]

[1]Universitas Malikussaleh
[2]Politeknik Caltex Riau

fidyatun.nisa@unimal.ac.id

Abstrak

Keberadaan teknologi *Software-Defined Networking* (SDN) menuntut adanya peningkatan pada kualitas keamanan jaringan. Dengan adanya pengontrol terpusat pada SDN, pengaturan jaringan lebih mudah untuk dilakukan, terutama dalam melakukan pertahanan terhadap serangan seperti *Distributed Denial of Service* (DDoS). Beberapa serangan DDoS menargetkan pada layanan yang lebih spesifik, sehingga perilaku serangan DDoS yang muncul tidak seperti biasanya. Penerapan skema *blocking* DDoS dengan menggunakan API *OpenFlow* dapat mendeteksi, mencegah serta mengantisipasi serangan DDoS. Disamping itu, implementasi SDN POX *Controller* sebagai sebuah platform pengembangan dari SDN yang dipadukan dengan dengan aplikasi *blocker* DDoS membuat jaringan / saluran komunikasi menjadi lebih aman.

Kata Kunci : Serangan DDoS, SDN, OpenFlow, SDN POX Controller.

JSISFOTEK is licensed under a Creative Commons 4.0 International License.

1. Pendahuluan

Deployment pada jaringan mengakibatkan interaksi antara banyak perangkat jaringan komputer dengan perangkat *node* menjadi cukup kompleks, sehingga dibutuhkan *effort* yang besar dalam membuat suatu rancangan. Selain itu, jaringan *multivendor* yang cukup besar untuk berbagai macam teknologi akan menimbulkan banyak permasalahan, baik dari aspek teknis maupun bisnis. Hal ini mengarahkan para penyedia layanan untuk dapat menyatukan *network management* dan *provisioning* melalui beragam domain. Untuk mendukung hal ini, dibutuhkan suatu model jaringan baru yang kemudian dikenal sebagai *Software-Defined Networking* (SDN).

Keamanan jaringan adalah bidang yang diharapkan dapat ditingkatkan kualitasnya dengan keberadaan teknologi SDN. Keamanan jaringan memerlukan koordinasi yang baik antara komponen jaringan untuk melakukan pertahanan terhadap serangan. *Router* tradisional sangat sulit untuk dimodifikasi prilakunya. SDN membuat prilaku *switch* lebih mudah dilakukan. Keberadaan pengontrol terpusat pada SDN juga membuat pengaturan seluruh jaringan secara terpusat menjadi lebih mudah dilakukan. Ketika serangan DDoS berkembang, pertahanan yang efektif menjadi tugas yang penting.

Serangan DDoS sekarang ini menargetkan layanan yang spesifik, sehingga aplikasi yang menjadi target akan menjadi *down*, sementara komponen jaringan yang lain seperti *link, switch, router* tidak mengalami masalah. Metode ini membuat serangan bisa menyembunyikan dirinya sebagai *traffic* normal, karena intensitasnya yang tidak seperti serangan DDoS yang biasanya membuat *traffic* besar-besaran. Contohnya adalah *flooding* HTTP GET yang memanfaatkan kerentanan dari suatu *web server*. Dalam jurnal ini akan dibahas beberapa aspek teknis, diantaranya :

a. Skema *blocking* DDoS yang menggunakan standar API OpenFlow.
b. Skema *blocking* DDoS diimplementasikan dengan menggunakan SDN POX *Controller*.
c. Implementasi kode dilakukan di atas emulator mininet.

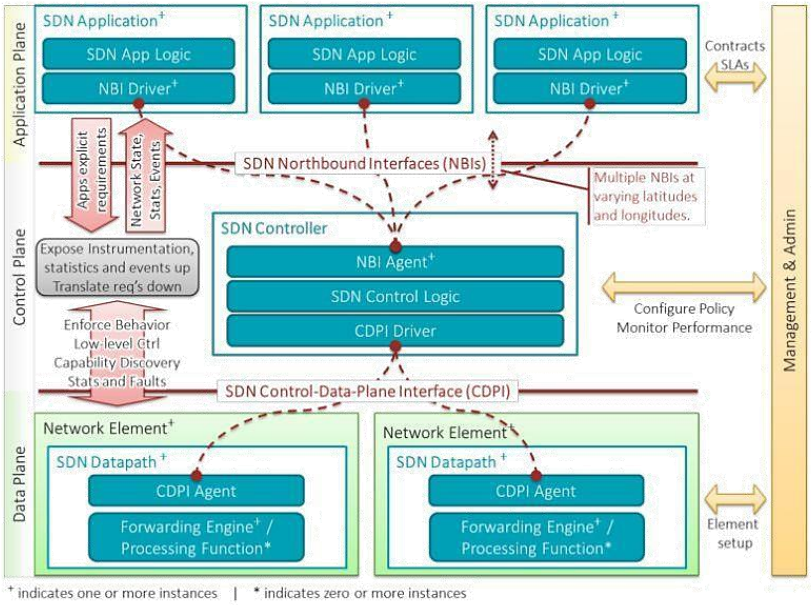
1.1 Software Defined-Networking (SDN), OpenFlow dan Cara Kerjanya [2]

Software Defined Network (SDN) merupakan istilah yang mengacu pada suatu pemahaman atau konsep baru dalam melakukan desain, pengelolaan serta implementasi pada jaringan, khususnya untuk mendukung kebutuhan serta inovasi jaringan yang semakin berkembang. Paradigma dasar dari SDN yaitu melakukan pemisahan antara *forwarding plane* dan *control plane*, selanjutnya melakukan abstraksi terhadap sistem jaringan serta mengisolasi kompleksitas yang terdapat pada komponen maupun sub-sistem melalui pendefinisan *interface* yang sesuai standar.

Pada arsitektur SDN, terdapat *control plane* yang dipisahkan dari *data plane*. Selain itu, dilakukan pemusatan (sentralisasi) antara *network state* dengan *network intelligence*, serta infrastruktur jaringan dipisahkan dari aplikasi.

Diterima: 27-07-2023 | Revisi: 01-08-2023 | Diterbitkan: 02-08-2023 | doi: 10.60083/jsisfotek.v5i3.269

Dengan adanya pemisahan ini, perusahaan dan operator jaringan dapat memiliki kendali, memperoleh otomatisasi serta programabilitas terhadap jaringan sehingga memungkinkan untuk melakukan ekspansi pada jaringan tersebut secara lebih fleksibel agar siap beradaptasi dan diimplementasikan sesuai dengan keperluan bisnis.
SDN bertujuan untuk menyediakan *open interface* yang membantu pengembangan lebih lanjut pada sebuah *software*. Dengan ini, SDN dapat mengontrol konektivitas suatu jaringan serta aliran *traffic* dari jaringan yang melewatinya, seiring dengan melakukan modifikasi terhadap *traffic* yang diimplementasikan pada jaringan tersebut. *Data plane* yang terdapat pada layer paling bawah suatu jaringan terdiri dari elemen-elemen jaringan dimana kemampuan dari elemen jaringan ini digunakan oleh SDN *Datapath* melalui *Control Data Plane Interface* (CDPI). Selanjutnya, pada layer paling atas suatu jaringan terdapat *application plane* yang memiliki berbagai aplikasi SDN dimana aplikasi ini saling bertukar informasi terkait kebutuhan layer-layer tersebut melalui *North Bound Interface (NBI) Drivers.* Sedangkan SDN *Controller* yang terdapat pada *layer* tengah, mentranslasikan kebutuhan *layer* di atasnya serta memberikan *low-level control* melalui SDN *Datapath* dengan mengirimkan informasi kepada aplikasi SDN. Selain itu, terdapat *Management dan Admin Plane* yang berfungsi untuk mempersiapkan elemen jaringan, mendefinisikan SDN *Datapath,* dan mengkonfigurasi kebijakan terkait kendali atas SDN *Controller* atau aplikasi SDN. Arsitektur SDN dapat diimplementasikan bersamaan dengan jaringan non-SDN, terutama terkait dengan tujuan migrasi secara keseluruhan ke jaringan SDN. Gambar 1.a di bawah ini menunjukkan skema SDN :



Gambar 1.a Skema SDN [2]

1.2 Serangan DDoS, *Scope* dan Klasifikasinya [6]

Serangan DDoS yang merupakan serangan terdistribusi, menjadikan serangan ini sulit untuk dilawan ataupun ditelusuri. Penyerang biasanya menggunakan *IP Address* yang sudah dipalsukan yang digunakan untuk menyembunyikan identitas penyerang sesungguhnya. Ada banyak kerentanan keamanan pada beberapa *hosts* di internet yang bisa dieksploitasi oleh penyusup. Insiden penyerangan yang menargetkan *application layer* pada OSI meningkat jumlahnya secara signifikan. Satu langkah yang perlu dilakukan adalah membuat mekanisme pertahanan terhadap DDoS yang mencakup seluruh aspek dari serangan DDoS. Dalam jurnal ini akan dilakukan klasifikasi DDoS berdasarkan level protokol yang diserang :

1.2.1 Serangan terhadap Network Layer dan Transport Layer

Serangan dijalankan menggunakan protokol TCP, UDP, ICMP dan paket aplikasi DNS.

a. *Flooding attacks* : Penyerang berfokus untuk menghalangi konektivitas *users* ke jaringan dengan cara menghabiskan *bandwidth* jaringan target. Contoh: UDP *flood,* TCP *flood,* ICMP *flood,* DNS *flood* dan lain sebagainya.
b. *Protocol exploitation flooding attacks* : penyerang melakukan *exploit* terhadap fitur yang dimiliki oleh protokol dengan tujuan untuk mengkonsumsi sejumlah *resource* milik korban, contoh : ACK & PUSH ACK *flood,* TCP SYN-ACK *flood,* TCP SYN *flood,* dan lain sebagainya.
c. *Reflection-based flooding attack* : penyerang biasanya mengirimkan *request* yang sudah dipalsukan, contoh: ICMP *echo request,* saat mengirimkan respon, si reflector akan mengirimkan ICM *echo reply* yang akan

menghabiskan resource milik korban.

d. *Amplification-based flooding attacks* : penyerang melakukan *exploit* terhadap layanan untuk men-*generate message* berukuran besar atau beberapa *message* yang diterima untuk melakukan amplifikasi terhadap *traffic* ke korban. *Botnets* telah digunakan baik untuk tujuan refleksi ataupun amplifikasi. Teknik amplifikasi dan refleksi biasanya dgunakan secara bersamaan seperti pada *smurf attack.* Dalam hal ini, penyerang mengirimkan *request* dengan dengan source IP *address* yang di *spoof* (refleksi), ke sejumalah besar *reflector* dengan cara meng*exploit* fitur IP *broadcast.*

1.2.2 Application-Level DDoS Floding Attacks

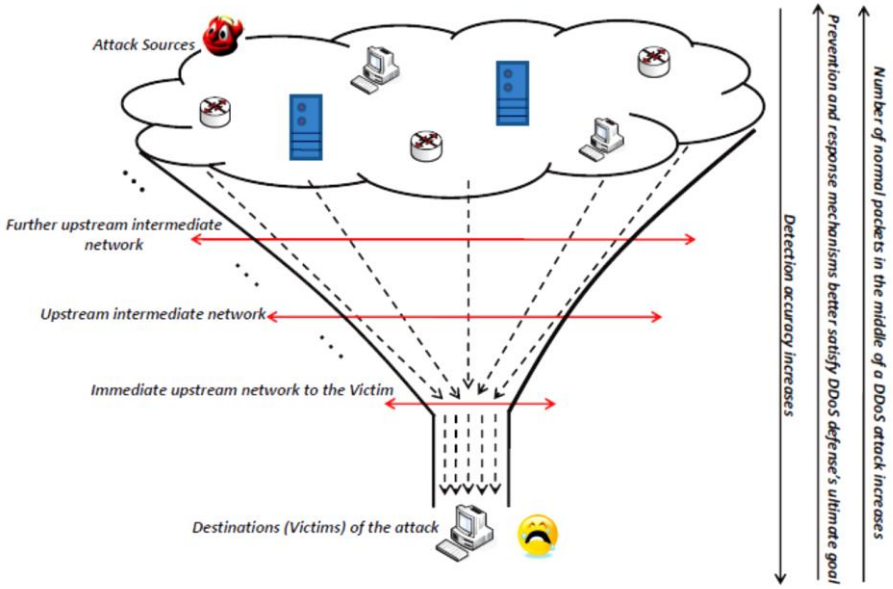
Serangan ini memfokuskan diri pada layanan jaringan dengan cara menghabiskan *resource server* seperti : soket, *memory, disk*, CPU, *bandwidth*, dan I/O *bandwidth*). *Application-level* DDoS *attack* secara umum mengkonsumsi *bandwidth* dalam jumlah yang tidak besar, namun *application-level* DDoS *flooding attack* biasanya mempunyai dampak yang sama kepada layanan, karenan menargetkan karakteristik yang spesifik dari suatu aplikasi seperti HTTP, DNS, ataupun SIP.

a. *Reflection/amplification based flooding attack*
   Serangan ini menggunakan teknik yang sama seperti *network/transport-level peers.* Contohnya, serangan *amplifikasi* DNS melibatkan teknik refleksi dan amplifikasi. Penyerang/*zombie* men-*generate* DNS *query* dengan ukuran kecil dengan *source* IP *address* yang *forged* , karena DNS merespon *query message* dan *response message* yang ukurannya lebih besar. Maka selanjutnya *network traffic* dalam ukuran besar diatur untuk menuju sistem target untuk mematikannya. Penggunaan *application-level attack* adalah VOIP *flooding*, serangan ini adalah variasi spesifik dari UDP *flooding.*
b. HTTP *flooding attack*
   Ada empat tipe serangan dalam kategori ini yaitu:
   1. *Session flooding attack* : Dalam serangan dengan tipe ini, *session connection request rate* dari penyerang lebih tinggi dari *legitimate users*, hal ini membuat *resource server* menjadi habis. Salah satu serangan yang terkenal dalam kategori ini adalah HTTP *get/post flooding attack.* Dalam hal ini, penyerang men-generate sejumlah besar HTTP *request* ke *web server* korban. Penyerang biasanya menggunakan *Botnet* untuk melancarkan serangan ini. Karena setiap *bots* bisa men-*generate* sejumlah besar *request*, maka tidak dibutuhkan jumlah *boots* yang terlalu besar.
   2. *Requests flooding attack* : Dalam tipe serangan ini, penyerang mengirimkan *sessions* yang mengandung jumlah *request* yang memiliki jumlah yang lebih besar dari pada *request* yang wajar. Salah satu dari serangan yang popular dalam kategori ini adalah *single-session* HTTP get/post *flooding.* Serangan ini adalah variasi dari HTTP get/post *flooding attack* yang mempergunakan fitur HTTP 1.1 untuk menjalankan *multiple request* dalam satu buah HTTP *session.* Penyerang bisa membatasai *rate* serangan HTTP dan melewati *rate session* perangkat keamanan.
   3. *Asymmetric attacks* : Dalam serangan tipe ini, penyerang mengirim *session* yang memiliki *high-workload request.* Disini ada beberapa serangan terkenal dalam kategori ini, yaitu :
      a. *Multiple* HTTP get/posts *flood* : Dalam serangan ini, penyerang membuat *multiple* HTTP request dengan cara membuat satu buah *packet* yang di-*embed* dengan *multiple requests.* Dengan cara ini seorang penyerang bisa menjalankan *load* tinggi di *server* korban dengan *rate packet* yang rendah, hal ini membuat penyerang menjadi tidak bisa dideteksi oleh perangkat *packet inspection.*
      b. *Faulty application* : Dalam serangan ini, penyerang mengambil keuntungan dari *website* yang didesain dengan buruk ataupun integrasi yang buruk dengan *database*. Sebagai contoh, penyerang bisa menjalankan SQL *injection* untuk men-*generate request* untuk mengunci *query database*, serangan ini sangat spesifik dan efektif karena mengkonsumsi *resource server* (*memory*, CPU dll)
   4. *Slow request/response attack* : Dalam tipe serangan ini, penyerang mengirim *sessions* yang mengandung *high-workload requests*. Ada beberapa serangan terkenal dari kelompok ini :
      a. *Slowloris attack* : yaitu serangan berbasis HTTP *get-based attack* yang bisa membuat *web server* menjadi *down* dengan menggunakan sejumlah host penyerang. Penyerang mengirimkan *partial* HTTP *request* yang berkelanjutan dan tumbuh secara cepat, secara perlahan di *update* dan tidak pernah ditutup. Serangan ini akan terus berlanjut sampai seluruh *socket server* habis oleh *request* si penyerang, IP *address* penyerang biasanya di-*spofed.*
      b. HTTP *fragmentation attack* : Mirip dengan *Slowloris*, tujuan dari serangan ini adalah untuk membuat *down web server* dengan memegang koneksi HTTP dalam jangka waktu yang lama. Penyerang/*boots/non-spoofed* menjalankan koneksi HTTP yang *valid* ke *web server* lalu penyerang melakukan fragmentasi HTTP *packet* kef ragmen-fragmen yang kecil lalu mengirimkan *fragment* selambat *server timeout* yang diperbolehkan. Dengan membuka *multiple sessions* untuk setiap *boot*, seorang penyerang bisa membuat *down web server* secara diam-diam.

c. HTTP *slowposts attack* : Mirip dengan slowloris *attack* yang mengirim HTTP posts yang secara perlahan membuat *web server* menjadi *down*. Penyerang mengirim HTTP *header* yang mendefinisikan *field* „*content-length'* lalu penyerang mengirim data untuk mengisi *message body* dengan *rate* satu *byte* setiap dua menit. Dengan cara ini maka *server* akan menunggu setiap *message body* untuk komplit sementara *slowpost attack* akan tumbuh yang menyebabkan *server* menjadi *down.*

1.3 Pertahanan Terhadap Serangan DDoS [7]

Biasanya ketika serangan DDoS terdeteksi, tidak ada hal lain yang bisa kita lakukan kecuali dengan cara melakukan pemutusan (*disconnect*) server korban dari jaringan dan secara manual memperbaiki masalahnya. DDoS menghamburkan banyak *resource* di jalur antara penyerang dan target, tujuan utama dari mekanisme pertahanan terhadap DDoS adalah untuk mendeteksi serangan secepat mungkin dan menghentikannya sedekat mungkin dari serangan. Gambar 1.b menunjukkan lokasi pertahanan terhadap serangan DDoS.



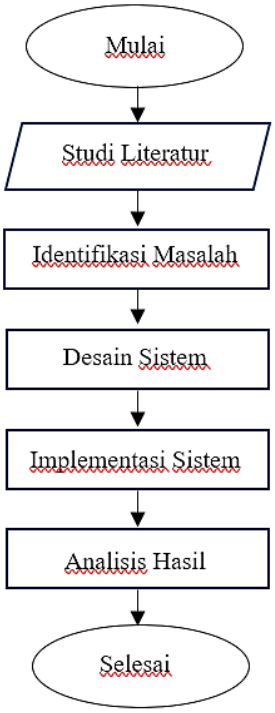
Gambar 1.b Lokasi pertahanan DDoS [7]

Pada jurnal ini akan dibahas secara khusus penerapan teknologi SDN untuk memerangi serangan DDoS yang dilakukan oleh *Botnet* dengan cara mempelajari prilaku *traffic*.

2. Metodologi Penelitian

Pada jurnal ini, penelitian akan berfokus pada desain / skema *blocking* DDoS pada SDN dengan menggunakan standar API *OpenFlow*. Selanjutnya skema *blocking* DDoS ini diimplementasikan menggunakan SDN POX *Controller*, yang disimulasikan menggunakan emulator mininet. Penelitian dimulai dengan mengidentifikasi berbagai masalah mengenai DDoS *attacks*. Setelah serangan DDoS teridentifikasi, maka dilakukan desain sistem yang sesuai jenis serangan DDoS tersebut. Desain sistem yang dilakukan adalah menggunakan OpenFlow. Kemudian sistem *blocking* serangan DDoS ini diimplementasikan pada SDN POX *Controller*.

Adapun diagram alir penelitian yang dilakukan seperti pada gambar 2 di bawah ini :



Gambar 2. Diagram Alir Penelitian

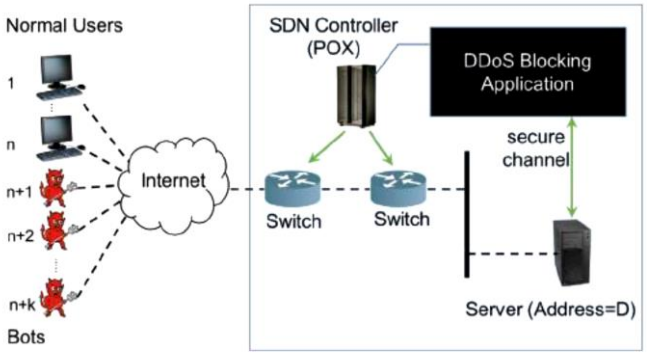
3. Hasil dan Pembahasan

3.1 Arsitektur Sistem Pencegahan Serangan DDoS berbasis SDN [1], [3], [5]

SDN *controller* dan aplikasi yang dijalankan pada *controller*, *OpenFlow switch* yang dikontrol oleh *controller* melalui *interface OpenFlow*. Ada sejumlah *n user* asli dan sejumlah *k* bots yang mengakses layanan. Untuk selanjutnya dalam membuat sistem ini terdapat beberapa asumsi sebagai berikut :

a. *Server* yang dilindungi di dalam jaringan berbasis SDN membuat *secure communication channel* dengan aplikasi *blocker* DDoS yang berjalan pada SDN POX *controller*. Untuk selanjutnya kita akan menyebut aplikasi ini sebagai DDoS *Blocking Application* (DBA). *Secure channel* digunakan oleh *server* untuk memberitahu DBA adanya serangan DDoS dan melakukan tindak lanjut.
b. DBA mengelola *pool* IP *address public* yang bisa digunakan untuk melakukan *redirection server* yang dilindungi. IP *address* merupakan replika layanan di lokasi yang lain. Tapi IP *address* lain yang ada di *subnet* yang sama dengan *server* yang diproteksi bisa digunakan. Selanjutnya *server* yang dalam kondisi diserang bisa memindahkan layanannya secara *logic* dari IP *address* yang diserang ke alamat yang dialihkan (*redirected address*). *User* asli diminta oleh *server* untuk melakukan pengalihan (*redirect*) access ke *service* pada IP *address* yang sudah dialihkan tadi.
c. Serangan DDoS dijalankan oleh Botnet dan Botnet tidak melakukan IP *address spoofing.*

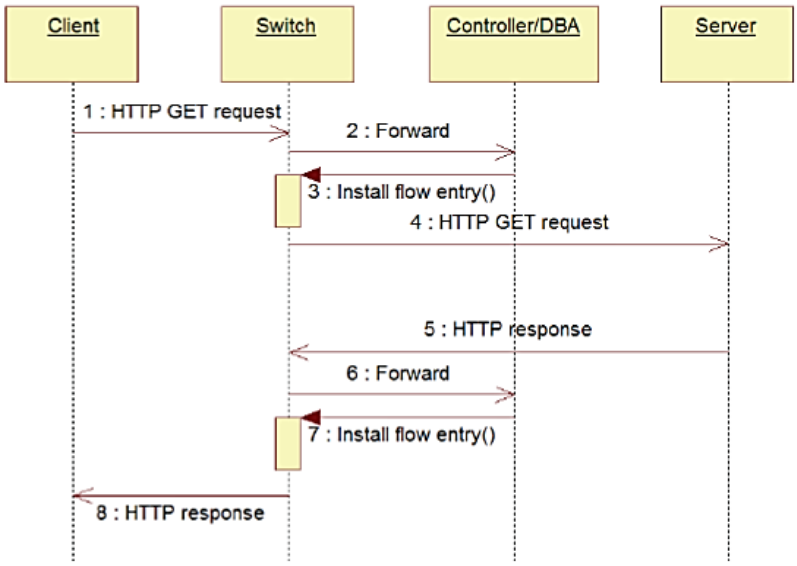
Pada gambar 3 memerlihatkan arsitektur sistem SDN dengan aplikasi *blocking* DDoS. Sistem ini dibuat untuk melindungi *service.*



Gambar 3. Arsitektur SDN dengan DDoS *Blocking Application*

3.2 *Workflow* DDoS *Blocking* berbasis SDN [4]

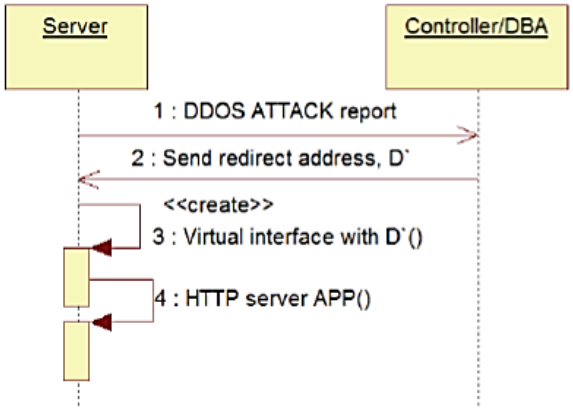
Berdasarkan asumsi ini maka arsitektur yang ada akan dimodifikasi dengan skema *blocking* berbasis SDN sesuai Gambar 4 sebagai berikut :



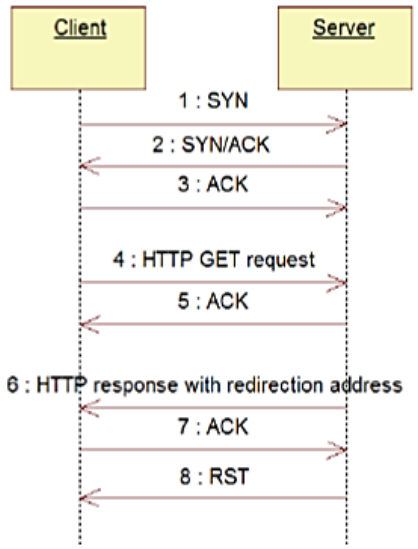
Gambar 4. *Workflow* Skema DDoS *Blocking* Berbasis SDN

Ketika *request* datang dari *client* menuju *switch* OpenFlow, *request* tersebut tidak akan cocok dengan *existing flow*. Kemudian *controller* akan diberi laporan, yang akan memerintahkan *switch* untuk membuat *entry flow table* bagi *packet* tersebut. Ketika *server* merespon *request* dari *client, flow entry* yang lain dibuat pada *switch* namun dengan arah berlawanan.

Dengan menggunakan *flow report* yang baru, DBA memonitor jumlah dari *flow* pada setiap *flow switch.* Sementara *server* memonitor metrik yang mengindikasikan serangan DDoS. Ketika *server* mengindikasikan bahwa serangan DDoS telah ada, *server* menjadi *collapse* sementara. Kemudian DBA memberikan *server* sebuah IP *address* baru yaitu D (Gambar 3.3). Dengan IP *address* baru tersebut layanan bisa berjalan lagi. Lalu selanjutnya koneksi yang lama akan dimatikan (Gambar 3.3) untuk mencegah *bots* mengerti kode *redirect* dan kemudian akan menyasar ke *address* D. Dalam jurnal ini, digunakan *CAPTCHA* yang akan mempersulit *bots* mengikuti alur baru. Namun, skema apapun membuat *bots* akan menjadi kesulitan untuk mengerti *redirect messages* yang bisa digunakan. Dalam jurnal ini, kita mengasumsikan bahwa D‘‘, D‘‘‘‘ dan D‘‘‘‘‘‘ adalah alamat yang memiliki *prefix* yang sama dengan D, yang akan digunakan untuk melakukan *redirect requests* dari *client*. Perlu diketahui bahwa *server* tidak perlu memiliki replika fisik. Dapat diasumsikan bahwa alamat ini ditetapkan / ditugaskan ke *server* fisik yang sama oleh *controller* SDN.



Gambar 5. *Redirection* dilakukan setelah serangan DDoS



Gambar 6. *Redirection* dilakukan setelah serangan DDoS

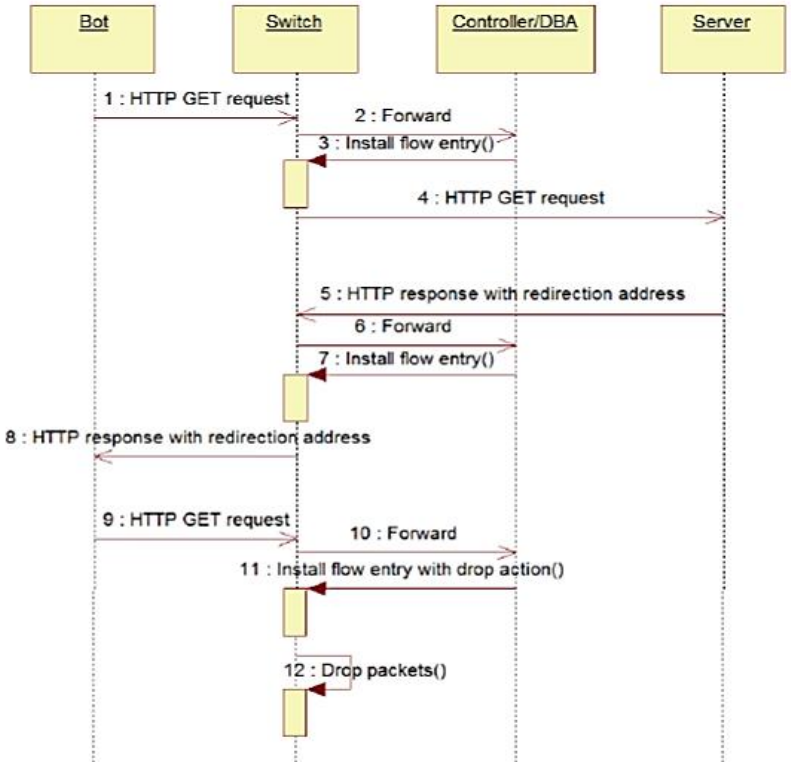
Ketika *server* mulai mengeluarkan *redirect messages* untuk *requests* yang datang, DBA menginstrusikan *switch* untuk memperbolehkan *flow* yang ditujukan kepada D‟, yang merupakan *redirect* IP *address* dari *server* yang diserang. *Counter* juga dijalankan untuk *flow* yang ditujukan kepada D, dan jika *counter* untuk *client* tertentu masuk ke *treshold* θ maka akan diklasifikasikan sebagai *bots* dan melakukan *drop* paket *bots.*

Pada gambar selanjutnya (Gambar 3.4), langkah 1 sampai dengan 8 diulangi untuk setiap koneksi baru yang datang ke *server*. Respon *redirection* dengan alamat D‟ yang dikodekan dengan *CAPTCHA* diberikan kepada *client. CAPTCHA* tidak memiliki HTTP *redirect response,* namun respon normal membawa data, yang merupakan informasi *redirection*. *Bots* diasumsikan tidak bisa mengerti hal tersebut dan mengacuhkan *request* tersebut.

Setelah mengulangi pelanggaran sampai mencapai nilai *θ* maka *client* diklasifikasikan sebagai *bot* dan kemudian langkah 9 sampai dengan 12 (Gambar 3.5) dijalankan untuk paket data yang datang dari *client.* Walaupun tidak ditunjukan pada Gambar 3.5, namun *flow entry* yang dibuat oleh paket yang datang dari *client* dan yang dibuat oleh *server* sebagai respon, dihapus dari *flow switch* untuk mencegah *flow table* untuk terisi penuh oleh *entry* yang tidak perlu.

Pada langkah 3 dan 7 (Gambar 7) dibuat *flow entry* pada *flow table.* Perbedaannya adalah pada langkah 3, *flow entry* telah mempunyi tiga *matching filed*, yaitu : *source* IP *address*, *destination* IP *address*, dan *source port number.* Dalam situasi lain, *flow entries* untuk langkah ke 7 hanya mempunyai dua *matching entries*, yaitu : *source* IP *address* dan *destination* IP *address*. Alasannya adalah *flow entry* yang dibuat untuk paket yang berasal dari *client* mempunyai *source port number* akan membedakan *request* yang berulang dari *client* yang sama setelah *redirection* dijalankan. *Server* akan melakukan *reset* setelah respon *redirection* dilakukan oleh TCP *reset.* Lalu *request* selanjutnya dari *client* tidak bisa menggunakan koneksi yang sama, maka untuk *request* selanjutnya, koneksi baru akan dibuat, yang akan menciptakan *flow entry* baru karena koneksi yang baru menggukan *source port* yang berbeda.

Pembuatan *flow entry* hanya dilakukan setelah kedatangan suatu paket yang tidak cocok dengan *flow entry* yang ada. Kemudian *controller* diberikan *notifikasi*. DBA bisa menyimpan jumlah *flow entries* yang memiliki pasangan *source* IP *address* dan *destination* IP *address* berdasarkan fakta *client* mana yang didefinisikan sebagai *bot.*



Gambar 7. Prilaku sistem setelah *request* berulang setelah *redirection*

Ketika client dikategorikan sebagai sebuah bot, *flow entry* dengan "drop" di-*install* pada *flow table*, seperti pada gambar 7 langkah ke 11, untuk selanjutnya akan packet tersebut akan dikeluarkan dari DBA.

Berikut adalah implementasi kode DBA yang dijalankan diatas SDN POX *controller* :

Algoritma : Kode DDoS Blocking Application

```
function handle_PacketIn(under_ddos_attack, , p)
1.  if under_ddos_attack /* only executed under attack */
2.  if p.srcIP flowCount
3.  flowCount[p.srcIP] = p.tcpSrcPort
4.  else if p.tcpSrcPort flowCount[p.srcIP]
5.  flowCount[p.srcIP].add(p.tcpSrcPort)
6.  if length(flowCount[p.srcIP])> /* bot */
7.  match.srcIP = p.srcIP
8.  match.dstIP = p.dstIP
9.  action = drop
10. else /* flow count is still small or server is not under attack */
11. match.srcIP = p.srcIP
12. match.dstIP = p.dstIP
13. match.tcpSrcPort = p.tcpSrcPort
14. action = forward
15. if packet.dstIP == D‟ /* a redirected flow arrives */
16. match.srcIP = p.srcIP
17. match.dstIP = D
18. sendFlowDeletionCommandToSwitch(match)
19. match.srcIP = D
20. match.dstIP = p.srcIP
21. sendFlowDeletionCommandToSwitch(match)
22. sendFlowAdditionCommandToSwitch(match,action)
```

4. Kesimpulan

Sifat dasar dari keamanan suatu jaringan komunikasi adalah : kerahasiaan (*confidentiality*), (keaslian) *integrity*, ketersediaan informasi, *non-repudiation* dan otentikasi *(authentication)*. Ide dari implementasi SDN adalah memisahkan antara *data plane* dengan *control plane* dan melakukan sentralisasi pada *control plane* ke dalam suatu *central controller*. *Central controller* memegang peran utama pada sistem ini. Analisis *traffic* atau metode deteksi

anomali digunakan di dalam jaringan untuk menghasilkan data berisi informasi yang berhubungan dengan keamanan dan dapat ditransfer secara berkala ke *controller* SDN terpusat. Aplikasi bisa dijalankan pada *controller* untuk menganalisis dan menghubungkan *feedback* ini dari jaringan. Dengan demikian, arsitektur SDN dapat digunakan untuk meningkatkan keamanan jaringan dengan bekal *monitoring* keamanan, analisis, dan respon sistem yang sangat reaktif

Dalam sebuah sistem pendeteksi dan pencegah serangan DDoS membutuhkan komunikasi antara aplikasi pencegah DDoS yang berjalan di SDN *controller* dan server yang di lindungi. Namun interaksi lainnya dilakukan dalam komunikasi *OpenFlow*. Implementasi yang dilakukan terbukti bisa melakukan pencegahan terhadap serangan DDoS.

Daftar Rujukan

[1] Joshua A. Alcorn, C. Edward Chow. (2014) A Framework for Large-Scale Modeling and Simulation of Attacks on an OpenFlow Network. *23rd International Conference on Computer Communications amd Networks (ICCCN)*. 4-7 August 2014. Shanghai : China. http://dx.doi.org/10.1109/ICCCN.2014.6911848

[2] ONF. (2013, December) SDN Architecture Overview. White Paper. [Online]. Diakses pada 2 Juli 2023 https://www.opennetworking.org/images/stories/downloads/sdnresources/technicalreports/SDN-architecture-overview-1.0.pdf

[3] Scott-Hayward, S., O'Callaghan, G., and Sezer, S. (2013). SDN Security: A Survey. 2*013* IEEE SDN for Future Networks and Services (SDN4FNS) pp. 1-7.11-13 Nov 2013. Trento : Italy. http://dx.doi.org/ 10.1109/SDN4FNS.2013.6702553

[4] Sezer, S et al., (2013). Are We Ready for SDN? Implementation Challenges for Software Defined Networks. *IEEE Communications Magazine*, Vol. 51, No. 7, pp. 36-43, July 2013. http://dx.doi.org/10.1109/MCOM.2013.6553676

[5] S. Lim, J. Ha, H. Kim, Y. Kim, S. Yang, (2014). A SDN-Oriented DDoS Blocking Scheme for Botnet-Based Attacks. 2014 *Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, 8-11 July 2014. Shanghai : China. http://dx.doi.org/10.1109/ICUFN.2014.6876752

[6] Zargar, Saman Taghavi., Joshi, James., Tipper, David. (2013). A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials Volume: Issue: 4*, 2013. http://dx.doi.org/ 10.1109/SURV.2013.031413.00127

[7] Zhang, Yi., Liu, Qiang. (2010) A Real-Time DDoS Attack Detection and Prevention System Based on per-IP Traffic Behavioral Analysis. *2010 3rd IEEE International Conference Computer Science and Information Technology (ICCSIT)*. 9-11 July 2010. Chengdu : China. http://dx.doi.org/ 10.1109/ICCSIT.2010.5563549