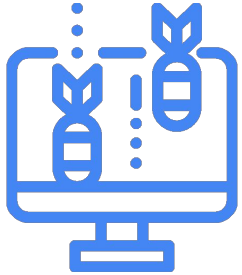
Smart and Lightweight DDoS Detection Using NFV

Autores: Talal Alharbi, Ahamed Aljuhani, Hang Liu, Chunqiang Hu

The Catholic University of America

Resumen del paper: Fernando Ostolaza, Milagros Oyague, Gabriel A. Spranger

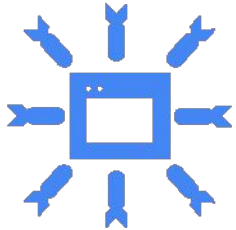
Índice:



1. Conceptos clave
    a. DDoS
    b. NFV
    c. Botnet
2. Objetivo del paper
3. Tendencias en ataques DDoS basados en Botnets
4. Detección de DDoS
    a. Métodos relacionados
    b. Método propuesto
        i. Mecanismos de detección
        ii. Resumen

Conceptos clave

DDoS



Es una amenaza a la infraestructura de red que llevará a uno de lo siguientes escenarios:

- Causará interrupción de los servidores y servicios de dicha red.

- Hará que no estén disponibles para usuarios legítimos.

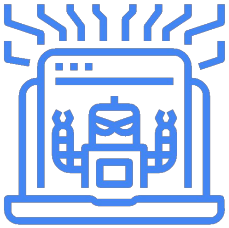
NFV

- Se utiliza para virtualizar servicios de red que tradicionalmente se hace en hardware específico con software específico.

- Se empaqueta como VM en hardware básico, pero no necesita uno exclusivo para cada función de la red.

- Mucho más flexible, fácil de configurar.

Botnet



- Red de dispositivos infectados
- Realizan tareas automatizadas
- Buscan infectar más dispositivos

Objetivo del paper

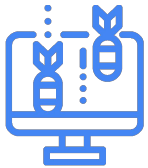
Smart and Lightweight DDoS Detection Using NFV

Talal Alharbi, Ahamed Aljuhani, Hang Liu, Chunqiang Hu

The Catholic University of America

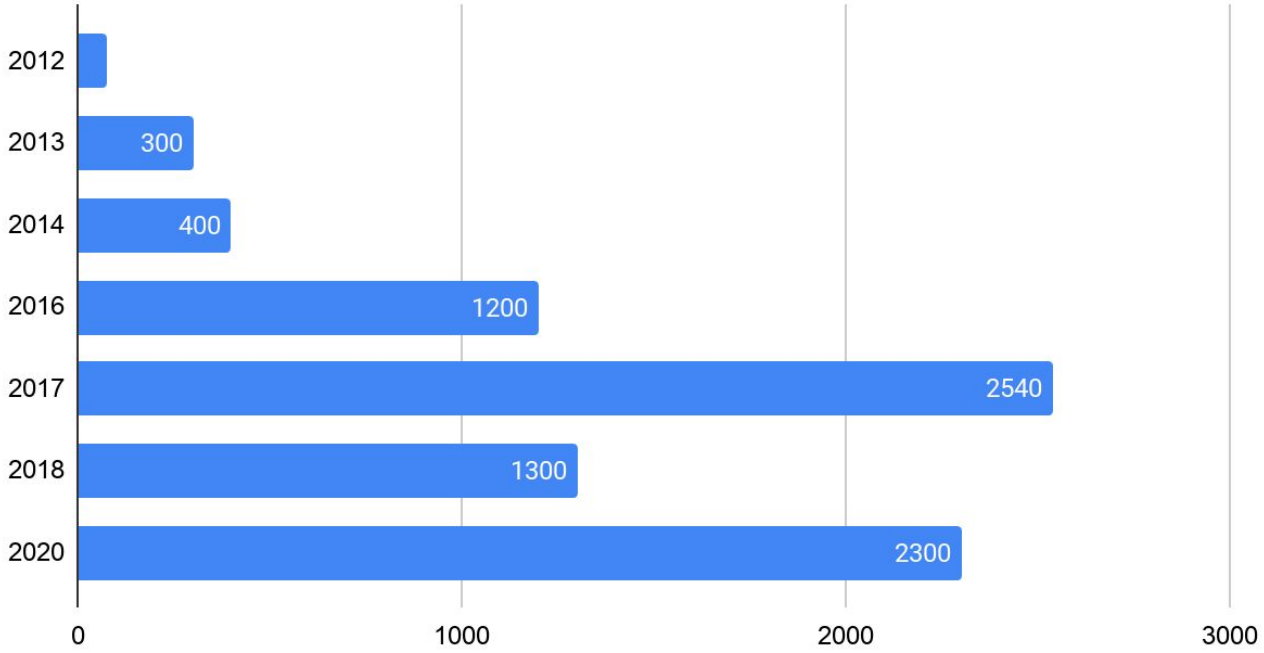
Proponer una solución para...

- Detectar ataques de DDoS mediante distintas técnicas
- Diferenciar tráfico real de tráfico de ataque



Tendencias en ataques DDoS basadas en Botnets

Tráfico Generado (GBPS)

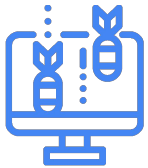


25% del tráfico de botnets viene de dispositivos IoT

Detección de DDoS

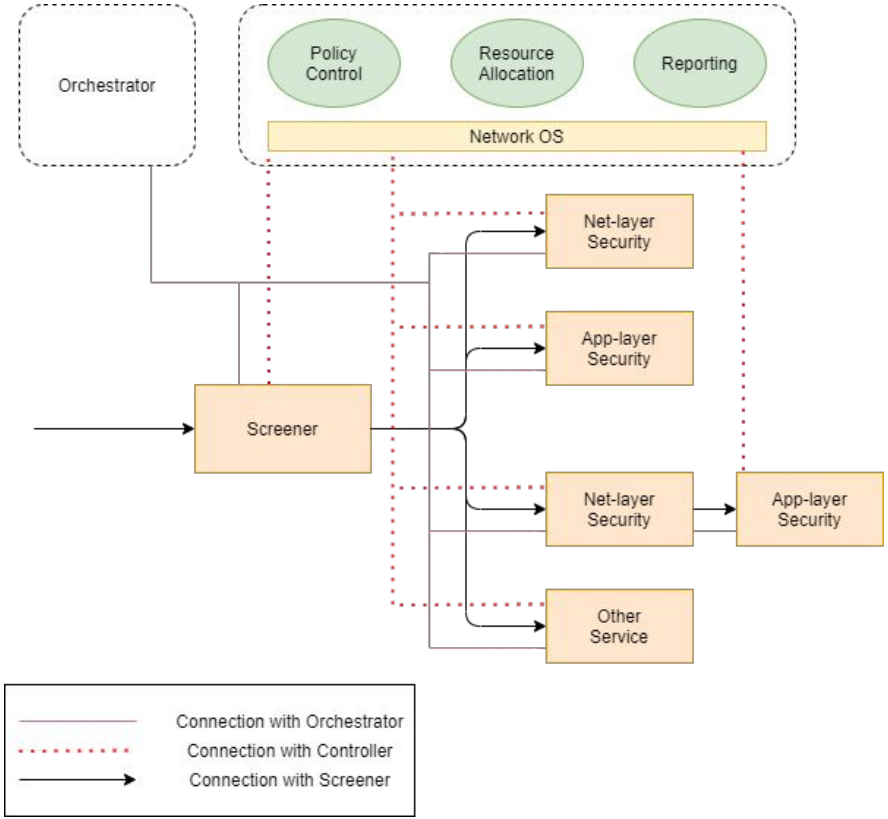
Métodos Relacionados

Método	¿En qué consiste?	Limitaciones
Estadístico	- PCA feature selector - Análisis de tasa, paquetes y protocolo de ataque	- Detecta en near real-time en lugar de real time - Puede causar latencia
Packet sampling	- Categoriza y mide el flujo de entropía de paquetes (uno de cada cinco) - Compara con umbrales (de puerto origen y de paquetes/seg)	- Falta de detección temprana - No menciona la precisión
Detección temprana	- Utiliza entropía para medir la aleatoriedad en los paquetes entrantes - Calcula con umbrales	- No proporciona cómo distinguir entre tráfico legítimo e ilegítimo
Mitigation	- Basado en intervención humana por una lista negra de source IP para filtrar el tráfico	- No puede proteger contra las spoofed source IPs

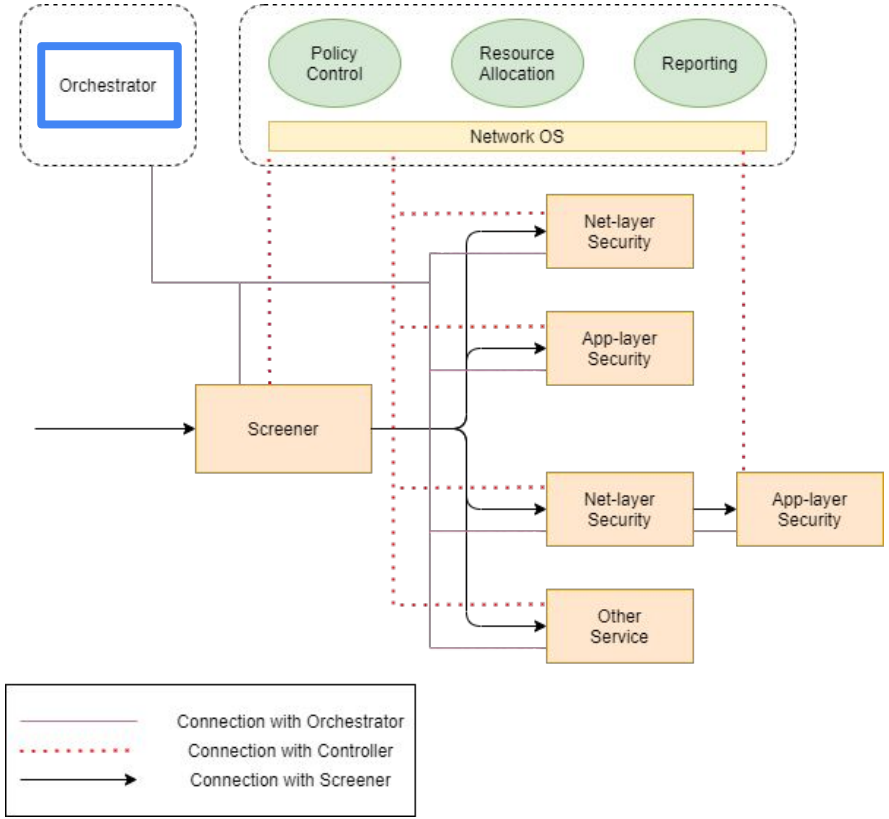


Método Propuesto

Componentes

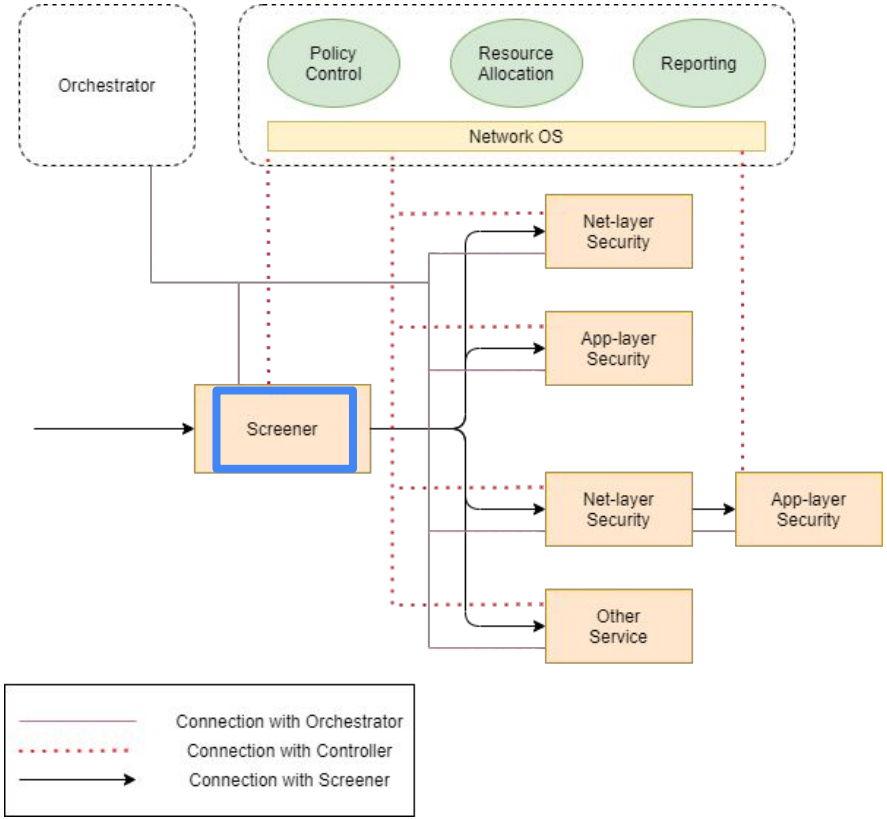


Orchestrator



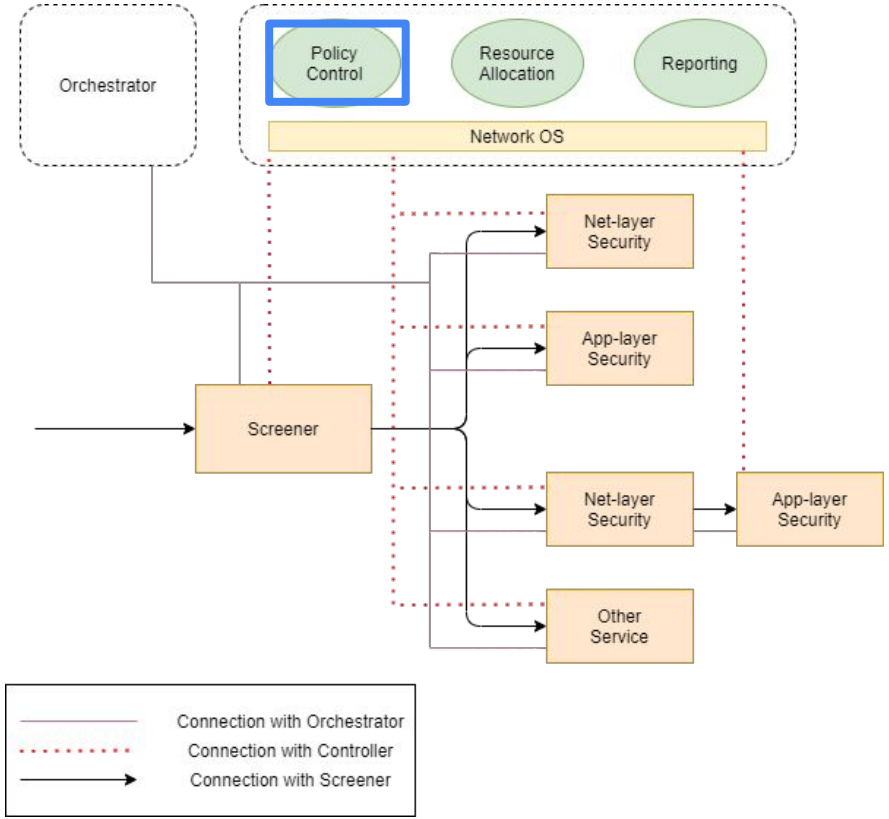
- Se encarga de la escalabilidad y asignación de recursos en tiempo real.
- Maneja todos los componentes (network functions).

Screener



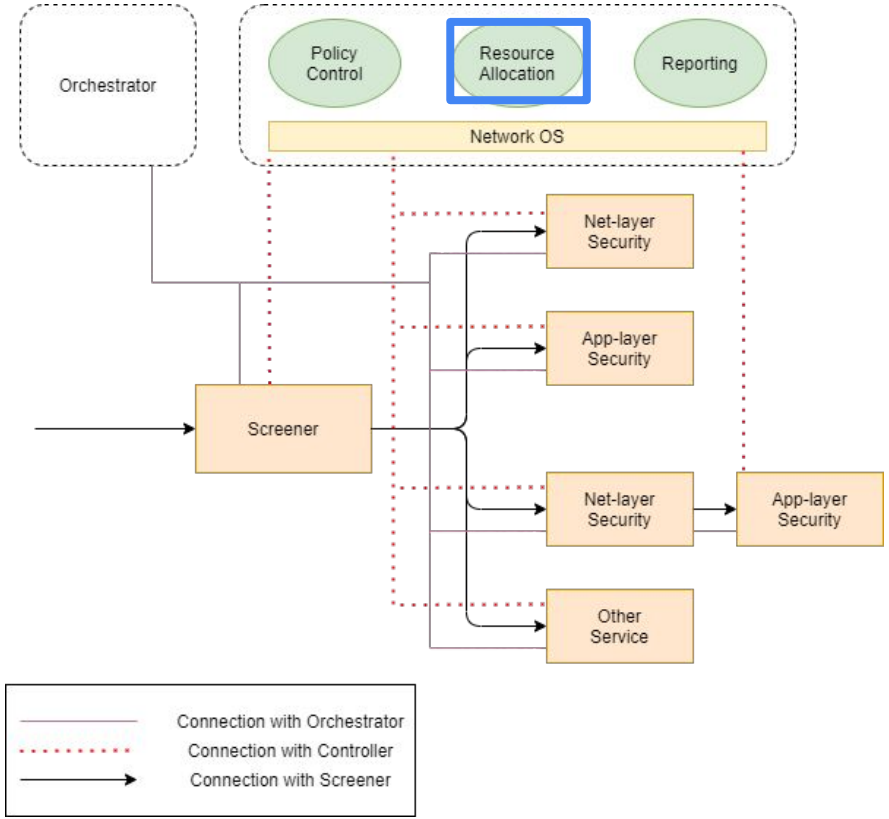
- Analiza el tráfico.
- Usa algoritmos para clasificar el tráfico.
- Toma ciertas acciones dependiendo de la clasificación.

Controller: Policy Control



- Se definen reglas para filtrar el tráfico.
- Se definen umbrales que el screener usa.

Controller: Resource Allocation Protocol



- Protocolo para que el screener se comunique con distintos network functions.

Algorithm 1 Resource Allocation Protocol (RAP)

1: **input:** Service_ID and M, // M is maximum capacity

2: calculate $U = \frac{C}{M}$ // U: used resource, C: current resources

3: **if** $U \geq 70\%$

4: {send alert_message to the screener,

5: send alert_message to the reporting module}

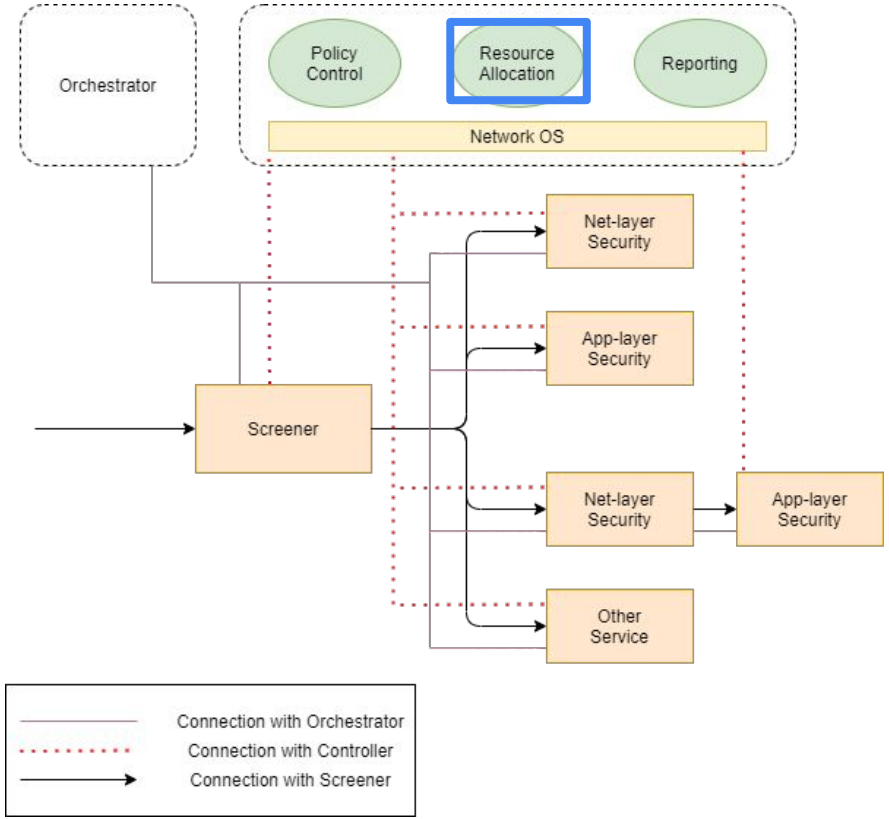
6: **else if** $U \leq 30\%$

8: {send alert_message to the screener,

9: send alert_message to the reporting module}

8: **end if**

Controller: Resource Allocation Protocol



- Protocolo para que el screener se comunique con distintos network functions.

Algorithm 1 Resource Allocation Protocol (RAP)

1: **input:** Service ID and M, // M is maximum capacity

2: calculate $U = \frac{C}{M}$ // U: used resource, C: current resources

3: **if** $U \geq 70\%$

4: {send alert_message to the screener,

5: send alert_message to the reporting module}

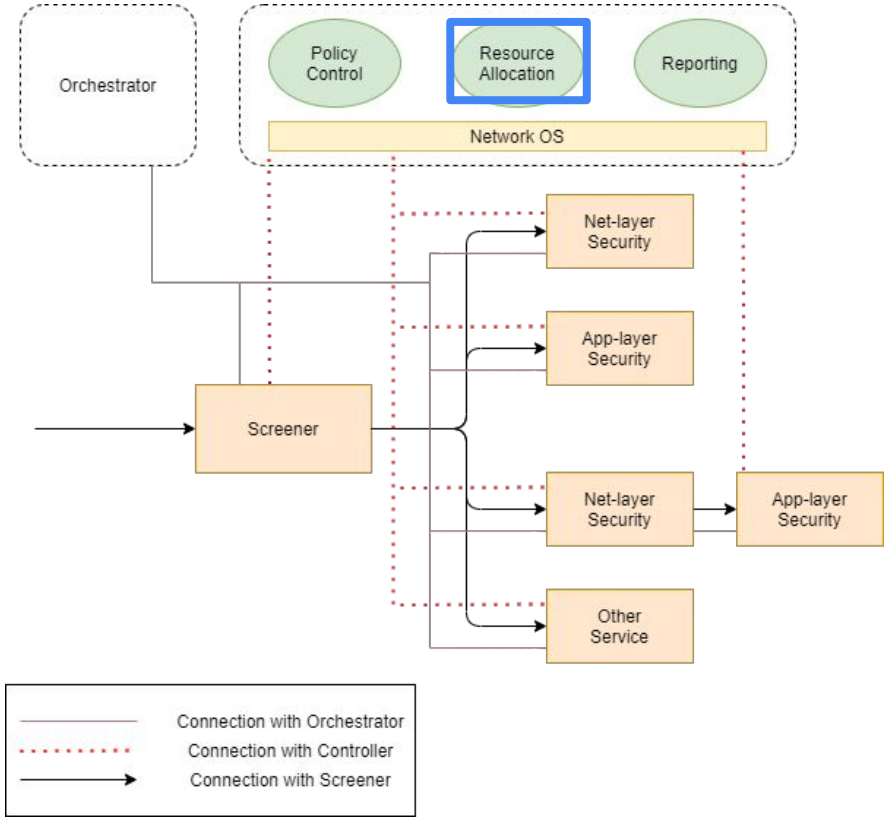
6: **else if** $U \leq 30\%$

8: {send alert_message to the screener,

9: send alert_message to the reporting module}

8: **end if**

Controller: Resource Allocation Protocol



- Protocolo para que el screener se comunique con distintos network functions.

Algorithm 1 Resource Allocation Protocol (RAP)

1: **input:** Service_ID and M, // M is maximum capacity

2: calculate $U = \frac{C}{M}$ // U: used resource, C: current resources

3: **if** $U \geq 70\%$

4: {send alert_message to the screener,

5: send alert_message to the reporting module}

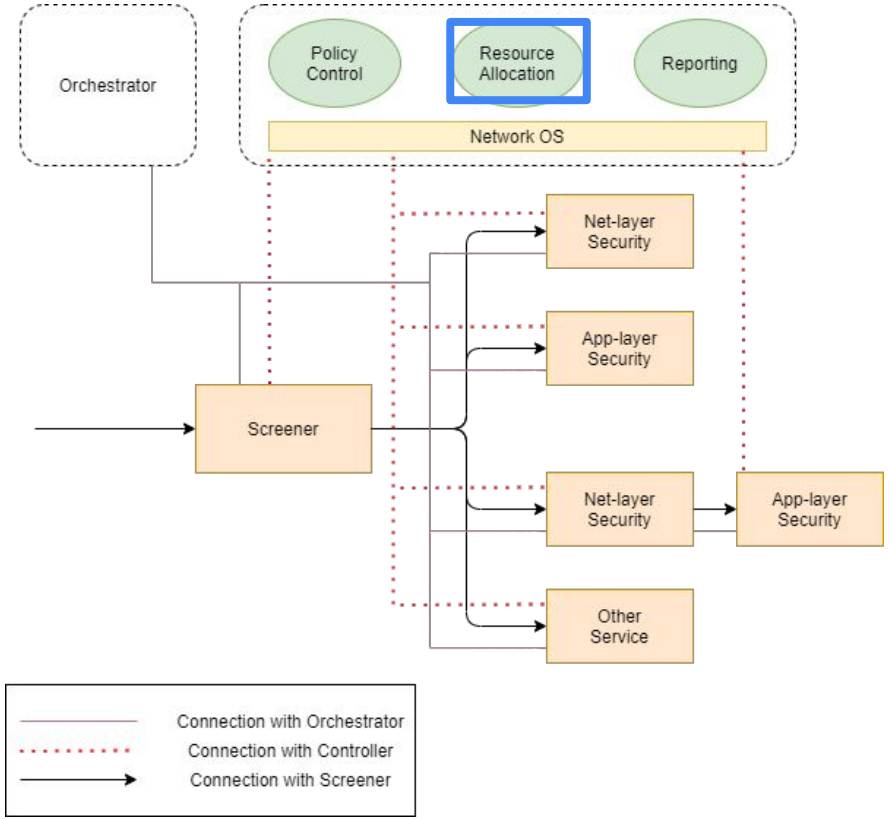
6: **else if** $U \leq 30\%$

8: {send alert_message to the screener,

9: send alert_message to the reporting module}

8: **end if**

Controller: Resource Allocation Protocol



- Protocolo para que el screener se comunique con distintos network functions.

Algorithm 1 Resource Allocation Protocol (RAP)

1: **input:** Service_ID and M, // M is maximum capacity

2: calculate $U = \frac{C}{M}$ // U: used resource, C: current resources

3: **if** $U \geq 70\%$

4: {send alert_message to the screener,

5: send alert_message to the reporting module}

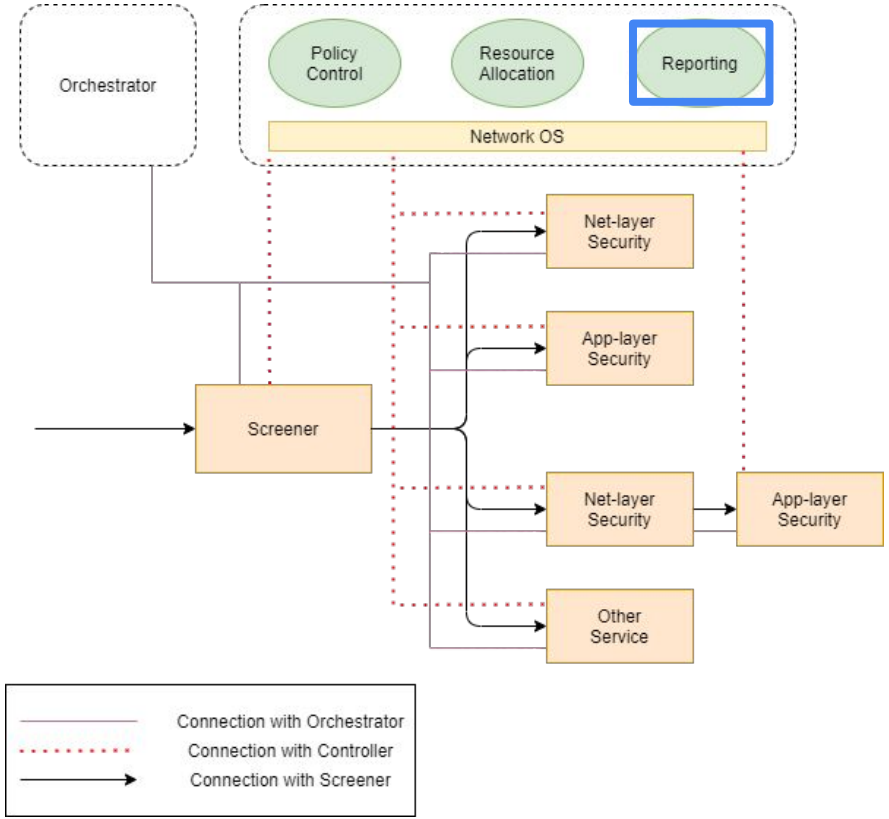
6: **else if** $U \leq 30\%$

8: {send alert_message to the screener,

9: send alert_message to the reporting module}

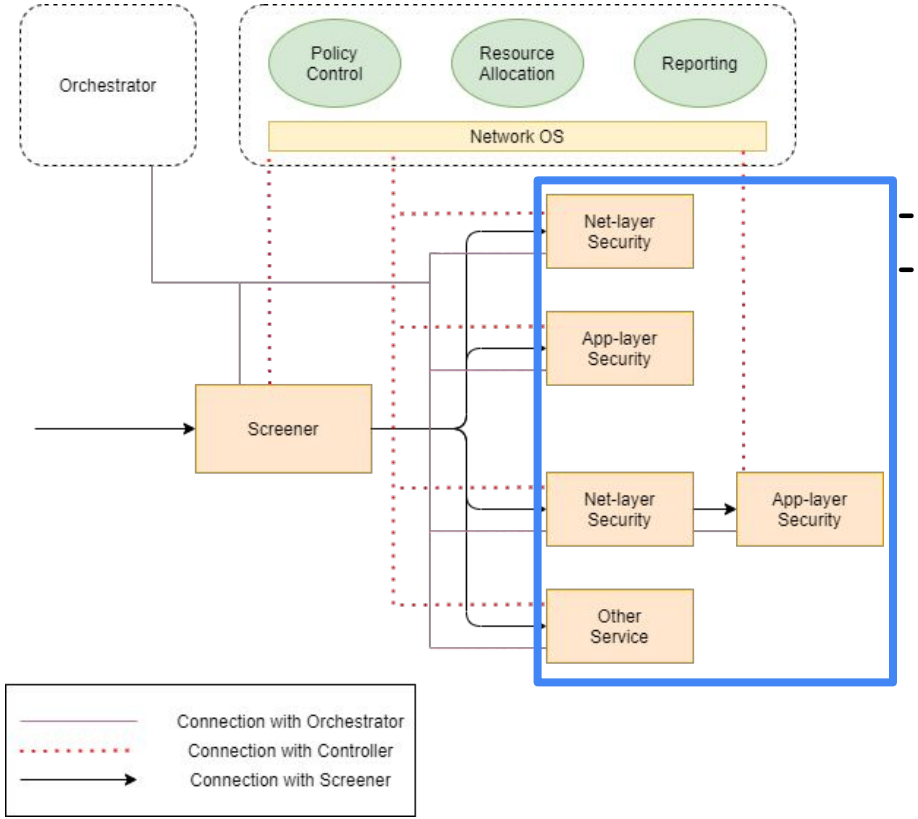
8: **end if**

Controller: Reporting



- Guarda todas las acciones y todos los mensajes hechos dentro del sistema.
- Todo lo pone en un log file.

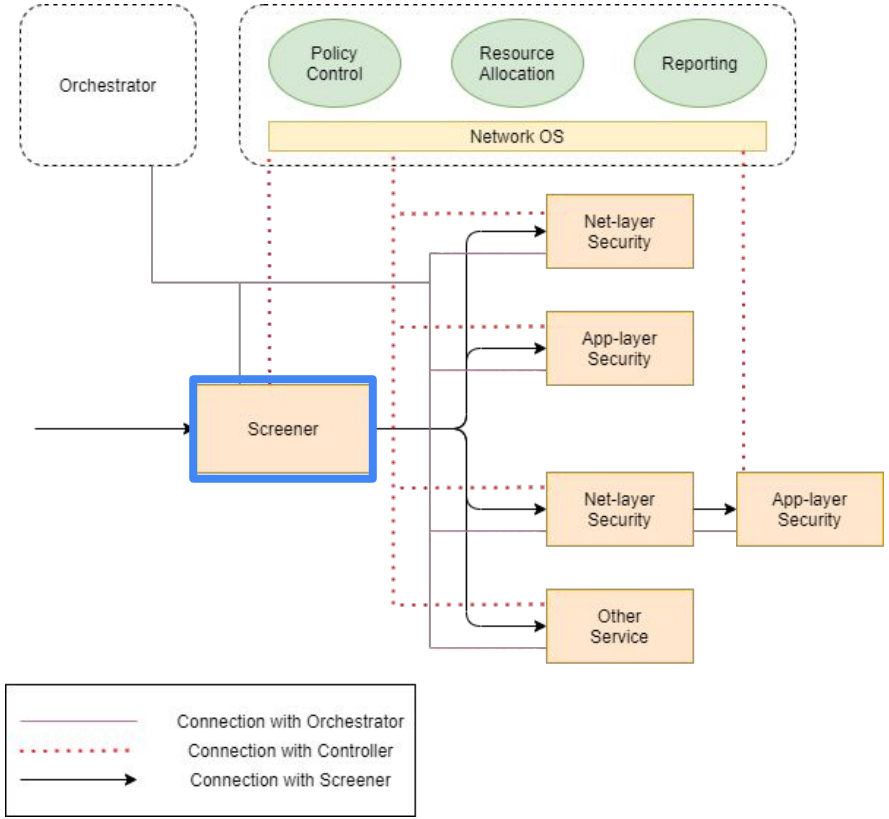
Layers



- Ofrecen los servicios de seguridad.
- Actúan de acuerdo a lo que el screener detecta con sus algoritmos.

Mecanismos de detección

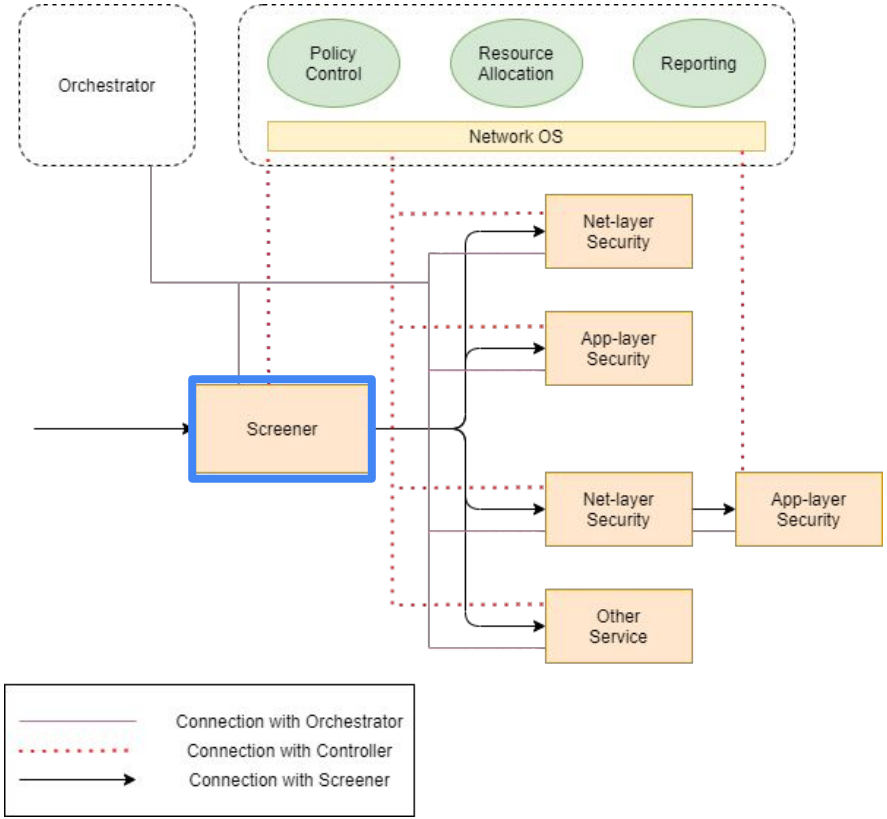
Screening



- Fast Screening
- Deep Screening

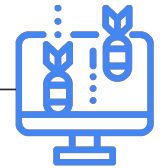
Fast Screening

$$traffic\ rate = \frac{No.\ of\ packets}{No.\ of\ seconds}$$



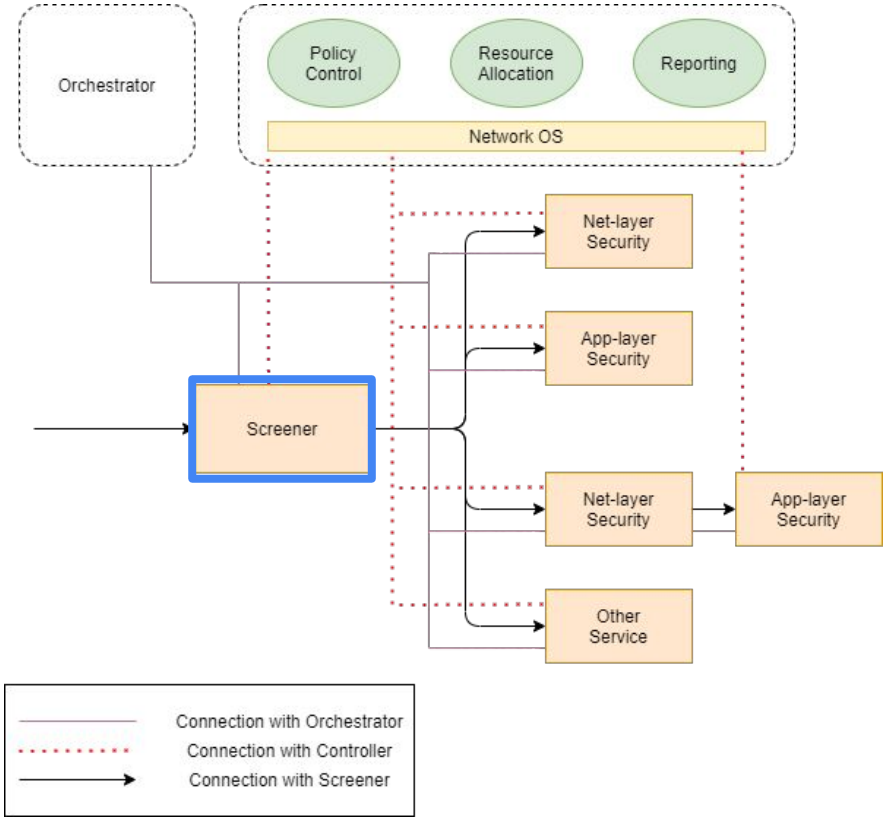
Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:             send a message to the reporting module}
11:    else if Service_ID ==VSF or Service_ID == Screener
12:                    {send instantiation or resource
                  modification request to orchestration,
13:               send a message to the reporting module}
14:    end if
15: else if alert_message with flag=S
16:           {send a resource modification request to
              Orchestration,
17:            send a message to the reporting module}
18:  else
19:    {return false
20:    capture next packet window}
21: end if
```



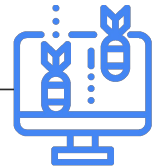
Fast Screening

$$traffic\ rate = \frac{No.\ of\ packets}{No.\ of\ seconds}$$

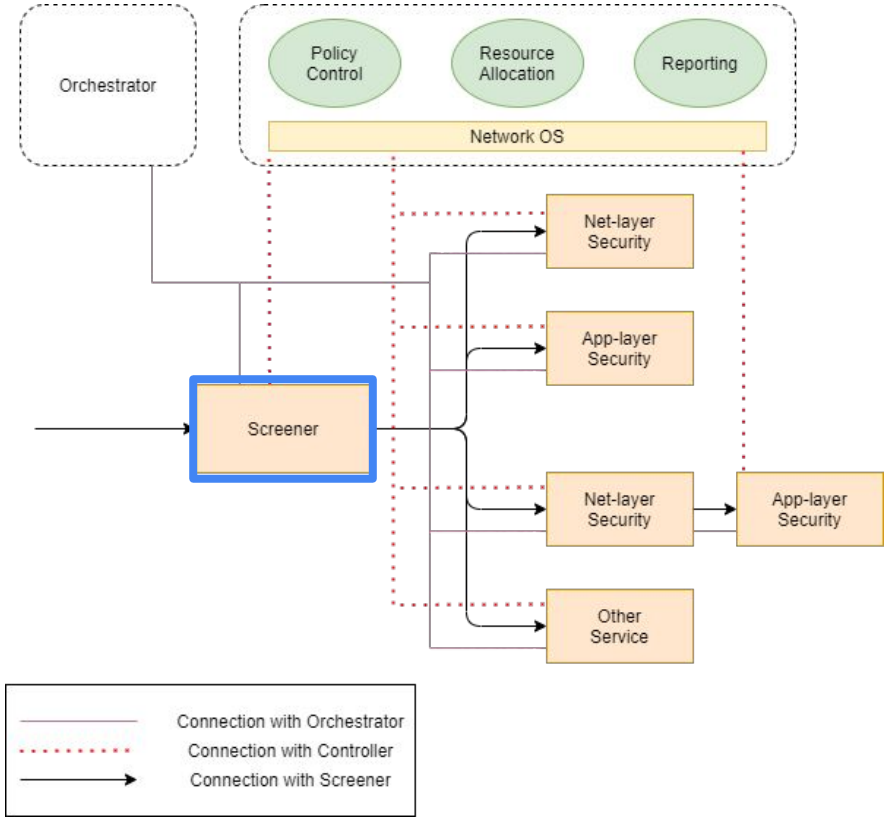


Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:             send a message to the reporting module}
11:    else if Service_ID ==VSF or Service_ID == Screener
12:                    {send instantiation or resource
                  modification request to orchestration,
13:               send a message to the reporting module}
14:    end if
15: else if alert_message with flag=S
16:           {send a resource modification request to
              Orchestration,
17:            send a message to the reporting module}
18:  else
19:    {return false
20:    capture next packet window}
21: end if
```



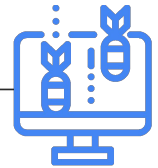
Fast Screening



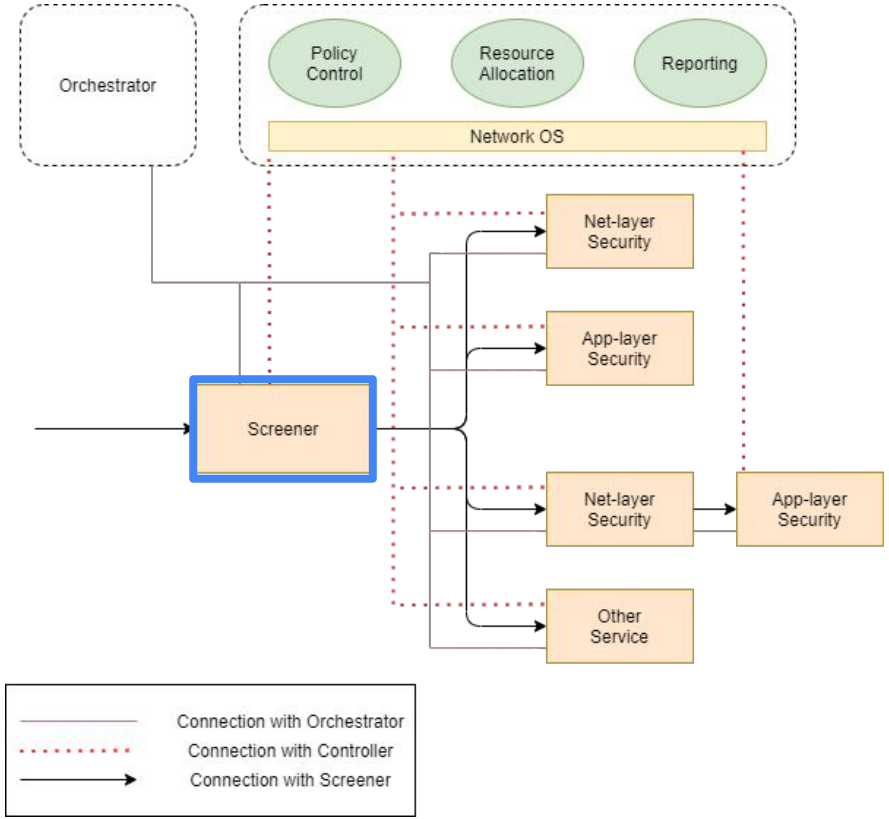
Service_ID: ID de la VM
VNF: Virtual Network Function

Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:            send a message to the reporting module}
11:    else if Service_ID ==VSF or Service_ID == Screener
12:                          {send instantiation or resource
                          modification request to orchestration,
13:                       send a message to the reporting module}
14:    end if
15: else if alert_message with flag=S
16:           {send a resource modification request to
              Orchestration,
17:           send a message to the reporting module}
18:  else
19:    {return false
20:    capture next packet window}
21:  end if
```



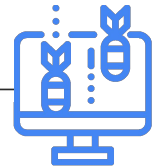
Fast Screening



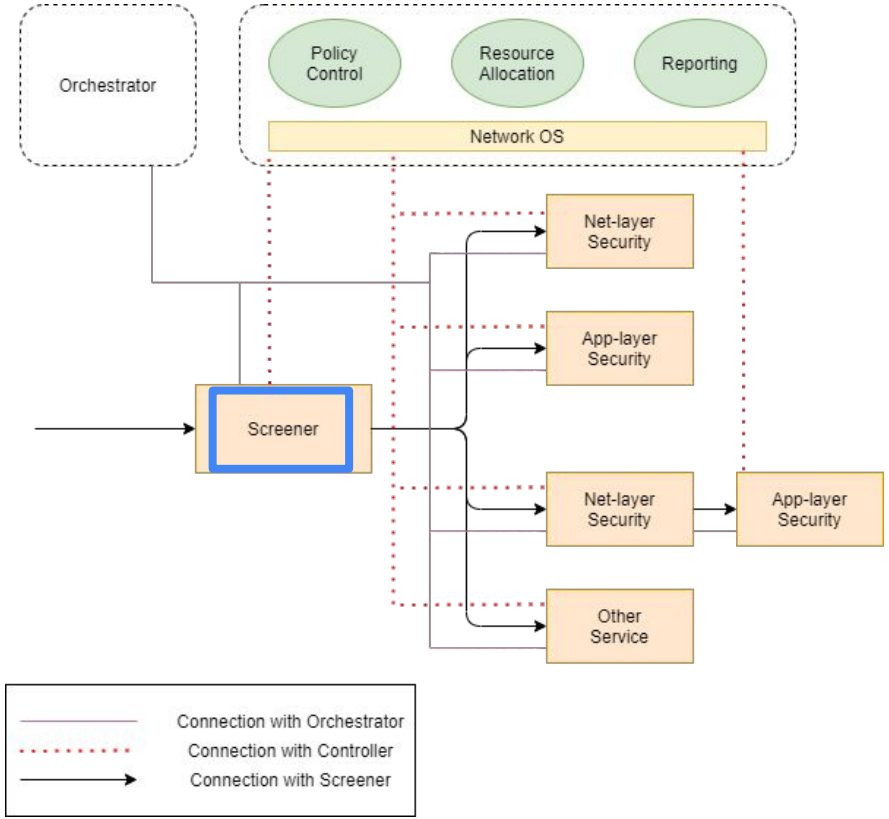
VSF: Virtual Security Function

Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:          send a message to the reporting module}
11:   else if Service_ID ==VSF or Service_ID == Screener
12:                  {send instantiation or resource
                   modification request to orchestration,
13:                send a message to the reporting module}
14:    end if
15:  else if alert_message with flag=S
16:           {send a resource modification request to
               Orchestration,
17:           send a message to the reporting module}
18:   else
19:    {return false
20:    capture next packet window}
21:  end if
```

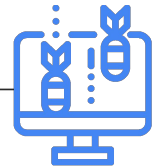


Fast Screening

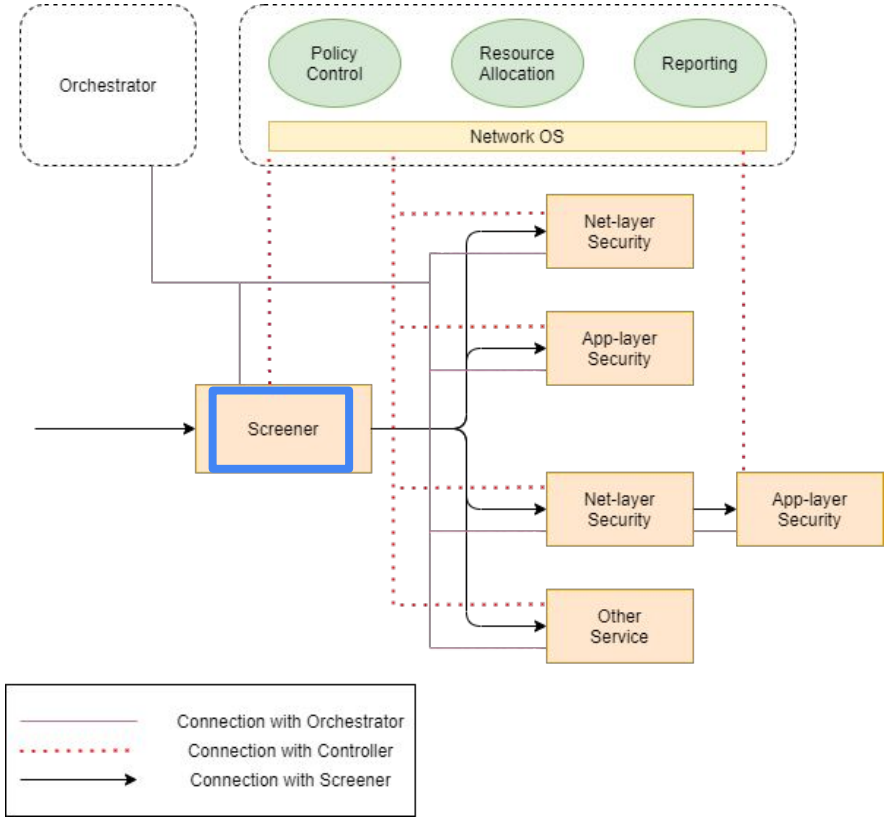


Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:             send a message to the reporting module}
11:    else if Service_ID ==VSF or Service_ID == Screener
12:                    {send instantiation or resource
                  modification request to orchestration,
13:               send a message to the reporting module}
14:    end if
15: else if alert_message with flag=S
16:           {send a resource modification request to
              Orchestration,
17:            send a message to the reporting module}
18:  else
19:    {return false
20:    capture next packet window}
21: end if
```

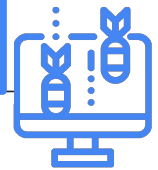


Fast Screening

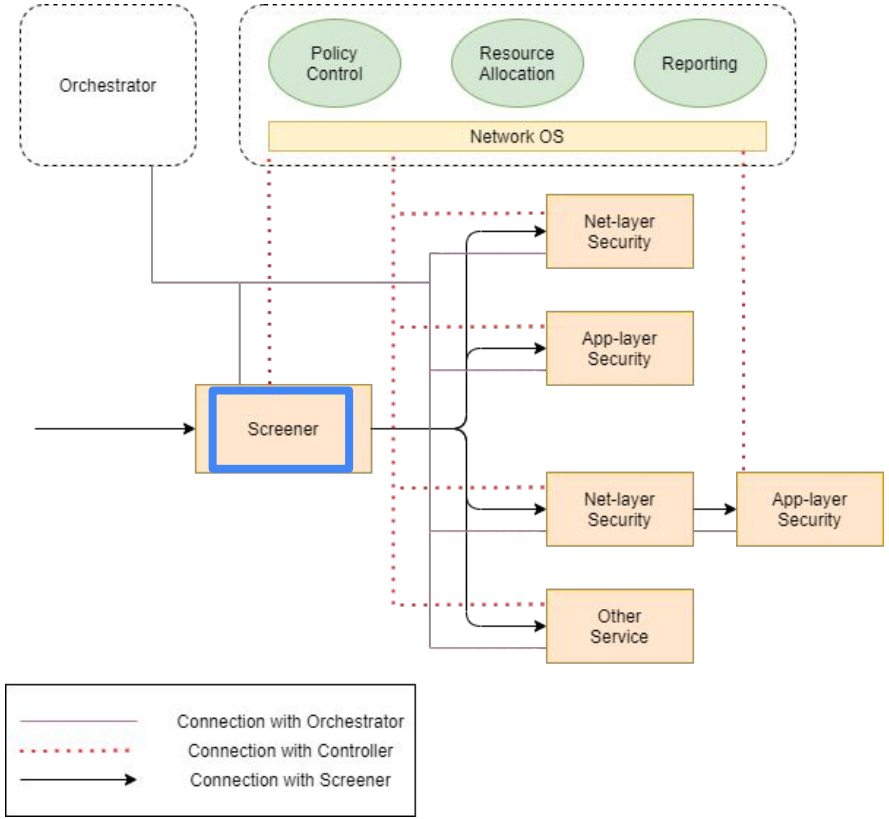


Algorithm 2 Fast Screening algorithm

```
1:  input: n: normal rate range
2:  calculate tr // tr: incoming traffic rate
3:  if tr > n
4:     {return  true,
5:      send a message to the reporting module}
6:  else if alert_message with flag=W
7:     if  Service_ID ==VNF
8:           {return  true,
9:           send a resource modification request to Orchestration,
10:             send a message to the reporting module}
11:    else if Service_ID ==VSF or Service_ID == Screener
12:                    {send instantiation or resource
                    modification request to orchestration,
13:             send a message to the reporting module}
14:    end if
15: else if alert_message with flag=S
16:           {send a resource modification request to
              Orchestration,
17:           send a message to the reporting module}
18: else
19:    {return false
20:    capture next packet window}
21: end if
```



Deep Screening



- Solo se ejecuta si el **tr > promedio** o si se detecta un mensaje con el **flag=W.**
- Distinguir entre tráfico elevado benigno o maligno.
- Un tráfico elevado tiene varias posibles causas.
- Uso de algoritmos para detectar distintos ataques.

Algoritmos de Deep Screening

Algoritmo: Amplification Attack

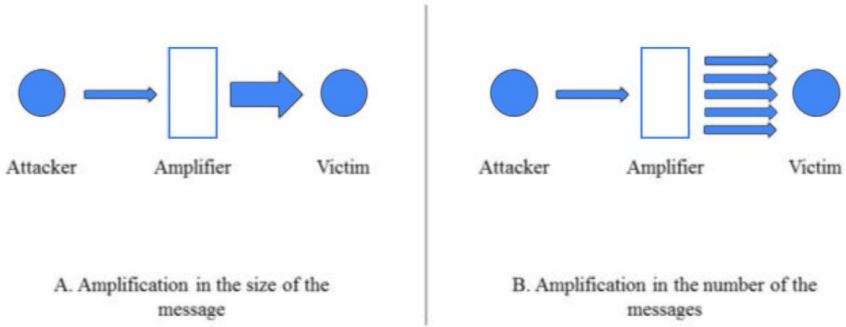


Figure 2. Types of DDoS amplification attacks.

Amplifica un mensaje inicial.
Dos tipos: cantidad y tamaño.
Uso de servidores como los DNS o NTP.

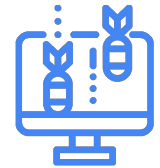
Algoritmo: Amplification Attack

Algorithm 3 Amplification in message size attack check

```
1: input: Server_IP // Server that reached the warning limit
2: while VNF is still drowning do
3:   for every incoming UDP packet
4:   if packet IP destination == Server_IP && packet_size >
     threshold
5:       if source_port==53 //DNS amplification
6:         DNS++
7:       elseif source_port ==123 //NTP amplification
8:         NTP++
9:       elseif source_port ==161 // SNMP amplification
10:        SNMP++
11:      elseif source_port ==1900 //SSDP amplification
12:        SSDP++
13:      end if
14:   end if
15: end for
16: for Amp_set
17:   if any element> threshold
18:     call mitigation
19:   else
20:     benign packets
21:   end if
22: end for
```

Amp_set {DNS,NTP,SNMP,SSDP}.

- Server_IP sería la IP de la VM que mandó el mensaje con flag=W mediante RAP.



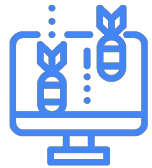
Algoritmo: Amplification Attack

Algorithm 3 Amplification in message size attack check

```
1: input: Server_IP // Server that reached the warning limit.
2: while VNF is still drowning do
3:   for every incoming UDP packet
4:    if packet IP destination == Server_IP && packet_size >
      threshold
5:       if source_port==53 //DNS amplification
6:         DNS++
7:       elseif source_port ==123 //NTP amplification
8:         NTP++
9:       elseif source_port ==161 // SNMP amplification
10:       SNMP++
11:      elseif source_port ==1900 //SSDP amplification
12:        SSDP++
13:      end if
14:   end if
15:  end for
16: for Amp_set
17:   if any element> threshold
18:     call mitigation
19:   else
20:     benign packets
21:   end if
22:  end for
```

Amp_set {DNS,NTP,SNMP,SSDP}.

- Mientras el VNF siga con una alta utilización de recursos, se analizan los paquetes UDP.



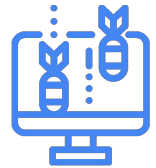
Algoritmo: Amplification Attack

Algorithm 3 Amplification in message size attack check

```
1: input: Server_IP // Server that reached the warning limit.
2: while VNF is still drowning do
3:   for every incoming UDP packet
4:    if packet IP destination == Server_IP && packet_size >
      threshold
5:        if source_port==53 //DNS amplification
6:          DNS++
7:        elseif source_port ==123 //NTP amplification
8:          NTP++
9:        elseif source_port ==161 // SNMP amplification
10:         SNMP++
11:       elseif source_port ==1900 //SSDP amplification
12:         SSDP++
13:       end if
14:    end if
15:  end for
16: for Amp_set
17:   if any element> threshold
18:     call mitigation
19:   else
20:     benign packets
21:    end if
22:  end for
```

Amp_set {DNS,NTP,SNMP,SSDP}.

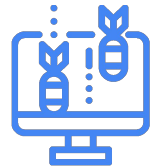
- Para cada tipo de servidor, contar paquetes que cumplieron criterio.
- Mayor a umbral, significa que hay un DDoS.



Algoritmo: Reflection Attack

Algorithm 4 Reflection attack check

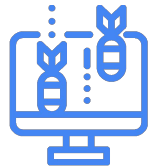
```
1:  input: Server_IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:      for every incoming packet
4:          detect source_IP range (subnet)
5:              if packet IP destination == Server IP &&
                source_IP_address within same subnet
6:              call mitigation
7:          else
8:              legitimate packet
9:          end if
10:     end for
```



Algoritmo: Reflection Attack

Algorithm 4 Reflection attack check

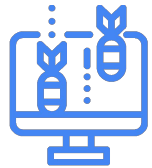
```
1:  input: Server_IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:    for every incoming packet
4:      detect source_IP range (subnet)
5:          if packet IP destination == Server IP &&
            source_IP_address within same subnet
6:          call mitigation
7:        else
8:          legitimate packet
9:        end if
10: end for
```



Algoritmo: Volumetric Attack

Algorithm 5 Volumetric attack check

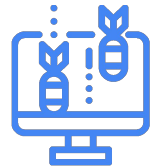
1: **input:** Server_IP // Server that reached the warning limit.
2: **while** VNF is still drowning **do**
3: **for** every incoming packet
4: **if** packet = = TCP
5: **for** every TCP connection
6: TCP_set{ SYN++ , ACK ++}
7: **end for**
8: SYN_rate = $\frac{No.of\ SYN\_packets}{No.of\ seconds}$
9: ACK_rate = $\frac{No.of\ ACK_{packets}}{No.of\ seconds}$
10: **for** TCP_set
11: **if** any element is in linear or exponential increase
12: call mitigation
13: **else**
14: benign packets
15: **end if**
16: **end for**
17: **end if**



Algoritmo: Volumetric Attack

Algorithm 5 Volumetric attack check

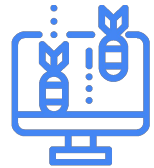
1: **input:** Server_IP // Server that reached the warning limit.
2: **while** VNF is still drowning **do**
3: **for** every incoming packet
4: **if** packet = = TCP
5: **for** every TCP connection
6: TCP_set{ SYN++ , ACK ++}
7: **end for**
8: SYN_rate = $\frac{No.of\ SYN\_packets}{No.of\ seconds}$
9: ACK_rate = $\frac{No.of\ ACK_{packets}}{No.of\ seconds}$
10: **for** TCP_set
11: **if** any element is in linear or exponential increase
12: call mitigation
13: **else**
14: benign packets
15: **end if**
16: **end for**
17: **end if**



Algoritmo: Volumetric Attack

Algorithm 5 Volumetric attack check

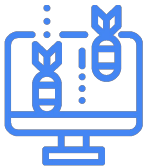
1: **input:** Server_IP // Server that reached the warning limit.
2: **while** VNF is still drowning **do**
3: **for** every incoming packet
4: **if** packet = = TCP
5: **for** every TCP connection
6: TCP_set{ SYN++ , ACK ++}
7: **end for**
8: $\text{SYN\_rate} = \frac{No.of\ SYN\_packets}{No.of\ seconds}$
9: $\text{ACK\_rate} = \frac{No.of\ ACK_{packets}}{No.of\ seconds}$
10: **for** TCP_set
11: **if** any element is in linear or exponential increase
12: call mitigation
13: **else**
14: benign packets
15: **end if**
16: **end for**
17: **end if**



Algoritmo: Volumetric Attack

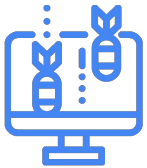
Algorithm 5 Volumetric attack check

```
1: input: Server_IP // Server that reached the warning limit.
2: while VNF is still drowning do
3:  for every incoming packet
4:   if packet = = TCP
5:     for every TCP connection
6:       TCP_set{ SYN++ , ACK ++}
7:     end for
8:     SYN_rate = No.of SYN_packets / No.of seconds
9:     ACK_rate = No.of ACK_packets / No.of seconds
10:  for TCP_set
11:      if any element is in linear or exponential increase
12:        call mitigation
13:      else
14:        benign packets
15:      end if
16:   end for
17:  end if
```



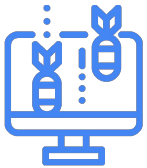
Algoritmo: Volumetric Attack

```
18:   if packet==UDP
19:       if packet_size > threshold // check for UDP Garbage
          flood
20:       Garbage++
21:       elseif destination_port==53 //DNS amplification
22:         DNS++
23:       elseif destination_port==123 //NTP amplification
24:         NTP++
25:       elseif destination_port==161 // SNMP amplification
26:         SNMP++
27:       elseif destination_port==1900 //SSDP amplification
28:         SSDP++
29:       end if
30:   end if
31:       for UDP_set
32:         if any element in is in linear or exponential increase
33:           call mitigation
34:         else
35:          benign packets
36:         end if
37:       end for
```



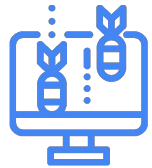
Algoritmo: Volumetric Attack

```
18:   if packet==UDP
19:     if packet_size > threshold // check for UDP Garbage
        flood
20:     Garbage++
21:     elseif destination_port==53 //DNS amplification
22:       DNS++
23:     elseif destination_port==123 //NTP amplification
24:       NTP++
25:     elseif destination_port==161 // SNMP amplification
26:       SNMP++
27:     elseif destination_port==1900 //SSDP amplification
28:       SSDP++
29:     end if
30:   end if
31:     for UDP_set
32:       if any element in is in linear or exponential increase
33:         call mitigation
34:       else
35:        benign packets
36:       end if
37:     end for
```



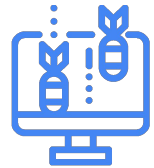
Algoritmo: Volumetric Attack

```
18:   if packet==UDP
19:     if packet_size > threshold // check for UDP Garbage
        flood
20:     Garbage++
21:     elseif destination_port==53 //DNS amplification
22:       DNS++
23:     elseif destination_port==123 //NTP amplification
24:       NTP++
25:     elseif destination_port==161 // SNMP amplification
26:       SNMP++
27:     elseif destination_port==1900 //SSDP amplification
28:       SSDP++
29:     end if
30:   end if
31:     for UDP_set
32:       if any element in is in linear or exponential increase
33:         call mitigation
34:       else
35:        benign packets
36:       end if
37:     end for
```



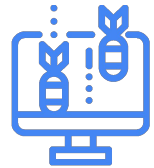
Algoritmo: Volumetric Attack

38: **if** packet==ICMP
39: ICMP++
40: **end if**
41: ICMP_rate = $\frac{No.of\ ICMP_{packets}}{No.of\ seconds}$
42: **if** ICMP_rate is in linear or exponential increases
43: call mitigation
44: **end if**
45: **end for**



Algoritmo: Volumetric Attack

38: **if** packet==ICMP
39: ICMP++
40: **end if**
41: ICMP_rate = $\frac{No.of\ ICMP_{packets}}{No.of\ seconds}$
42: **if** ICMP_rate is in linear or exponential increases
43: call mitigation
44: **end if**
45: **end for**

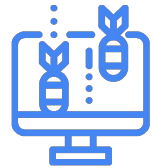


Algoritmo: High rate application attacks

Algorithm 6 High-rate attack check

```
1:  input: Server IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:  for every TCP incoming packet
4:    for every source_ IP
5:      Http_flood{http_get++, http_post++}
6:    end for
7:  end for
8:  for Http_flood
9:     if any element> threshold
10:     return true //call mitigation
11:   else
12:     return false //benign packets
13:   end if
14:  end for
```

Http_flood{(source_IP, http_get++), (source_IP, http_post++)} contains the total number of packet_requests per IP_source address.

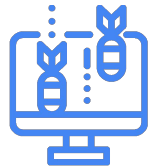


Algoritmo: High rate application attacks

Algorithm 6 High-rate attack check

```
1:  input: Server IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:  for every TCP incoming packet
4:    for every source_ IP
5:      Http_flood{http_get++, http_post++}
6:    end for
7:  end for
8:  for Http_flood
9:      if any element> threshold
10:       return true //call mitigation
11:     else
12:       return false //benign packets
13:     end if
14:  end for
```

Http_flood{(source_IP, http_get++), (source_IP, http_post++)} contains the total number of packet_requests per IP_source address.

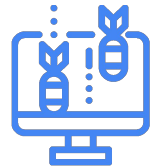


Algoritmo: High rate application attacks

Algorithm 6 High-rate attack check

```
1:  input: Server IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:  for every TCP incoming packet
4:    for every source_ IP
5:      Http_flood{http_get++, http_post++}
6:    end for
7:  end for
8:  for Http_flood
9:     if any element> threshold
10:     return true //call mitigation
11:   else
12:     return false //benign packets
13:   end if
14:  end for
```

Http_flood{(source_IP, http_get++), (source_IP, http_post++)} contains the total number of packet_requests per IP_source address.

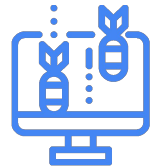


Algoritmo: High rate application attacks

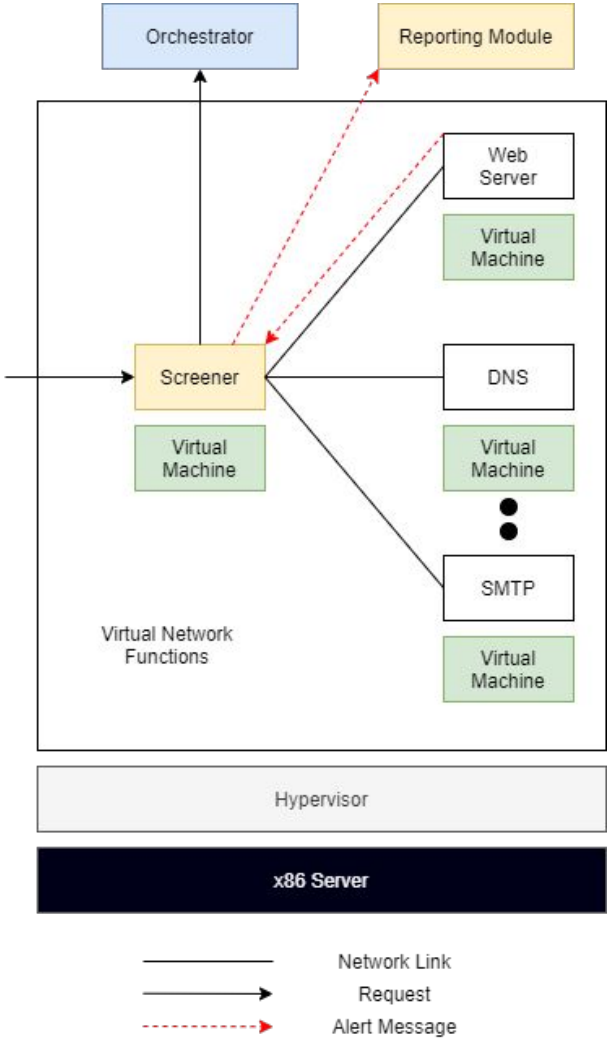
Algorithm 6 High-rate attack check

```
1:  input: Server IP // Server that reached the warning limit.
2:  while VNF is still drowning do
3:  for every TCP incoming packet
4:    for every source_ IP
5:      Http_flood{http_get++, http_post++}
6:    end for
7:  end for
8:  for Http_flood
9:    if any element> threshold
10:     return true //call mitigation
11:   else
12:     return false //benign packets
13:   end if
14: end for
```

Http_flood{(source_IP, http_get++), (source_IP, http_post++)} contains the total number of packet_requests per IP_source address.



Conclusiones



Conclusiones