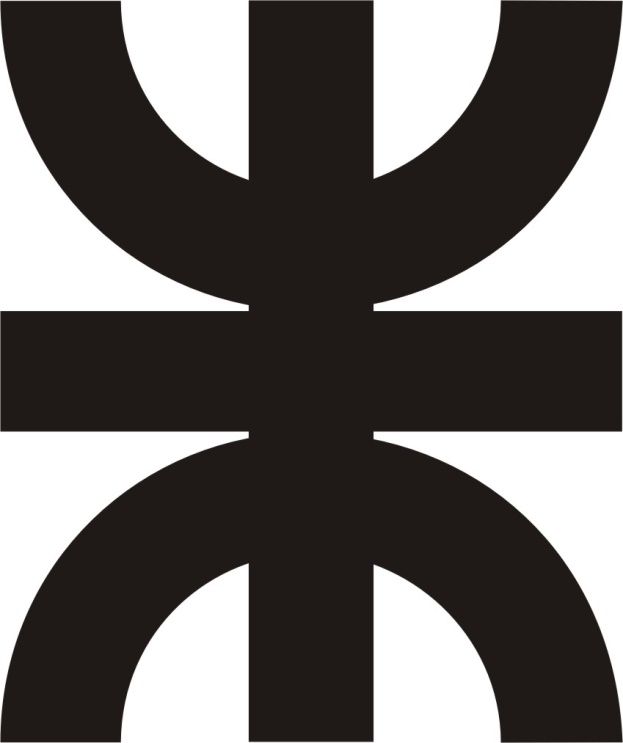
****

***Universidad Tecnológica Nacional, Facultad Regional Buenos Aires***

**De Pietro, Tomás**

**Anderson, Nicolás**

**Zapata, Mauro**

**Abstract**

This research tries to show if it is possible the introduction of a new network device that functions as a firewall, being able to analyze the packets that circulate through it and detect possible attacks in the second and third layers of the OSI model, such arp spoofing and dns spoofing for example. The main parameter that we must consider will be the time it takes to process the network packets, so in order to have approximate values, we are going to do an event-to-event simulation.

**Palabras clave**

Redes, ARP, DNS, Proxy, Ataques de Red

**Introducción**

Uno de los principales protocolos que utilizan las redes de internet es el protocolo ARP (Address Resolution Protocol). Este permite vincular o asociar direcciones lógicas con direcciones físicas, en otras palabras asociar una dirección IP con su respectiva MAC Address.

Cuando se realiza una petición ARP, se manda dicha petición a todos los dispositivos de la red (broadcast) preguntando quien tiene la MAC en cuestión, el dispositivo que efectivamente la tiene responde; luego el dispositivo que recibe la respuesta guarda en una tabla esta relación para evitar tener que hacer peticiones iguales en el futuro.

Sin embargo este protocolo tiene dos problemas: primero que nada, cuando un dispositivo recibe una respuesta no puede validar que esta sea legitima, y segundo se acepta una respuesta aunque nunca se haya enviado una petición. Esto conlleva a que un dispositivo de la red pueda hacerse pasar otro, en la mayoría de los casos por el router, a su vez esto conlleva a que el tráfico de la red pase por el dispositivo que realiza este ataque, logrando espiar que paquetes circulan, pudiendo descartarlos o modificarlos.

A partir de esto se logran realizar ataques más complejos, por ejemplo DNS spoofing que al modificar las respuestas de un servidor DNS permite conectar con equipos maliciosos.

Para solucionar esta problemática se propone diseñar un dispositivo de red que actué como proxy entre la computadora que se desea proteger y el router, analizando el tráfico que pase por él y detectando el ataque descripto anteriormente, entre otros. Es fundamental que el tráfico no se vea ralentizado demasiado debido al análisis y que el usuario lo siga notando fluido.

Con respecto al dispositivo físico la idea es utilizar una placa Raspbery Pi ya que es sistema embebido que corre sistemas operativos basados en unix y además es de fácil acceso en el mercado

**Elementos del trabajo y metodología**

Se realizó una simulación evento a evento para evaluar si los tiempos de respuesta son factibles y cuánto tiempo están los paquetes de red esperando a ser procesados.

Análisis previo:

Metodología: Evento a Evento

Variables exógenas:

Datos: IA, TE

Control: CT

Variables endógenas:

Estado: PARP, PT(i)

Resultado: PPS(i), PEC(i), PTO(i)

Explicación de las variables:

* IA: intervalo de arribo de paquetes en microsegundos
* TE: tiempo de evaluación de paquetes en microsegundos
* CT: cantidad de threads
* PARP: cantidad de paquetes con prioridad que están encolados en la cola virtual
* PT(i): cantidad de paquetes que tiene cada thread
* PPS: promedio de permanencia en el sistema
* PEC(i): promedio de espera en cola de cada thread
* PTO(i): porcentaje de tiempo oscioso de cada thread

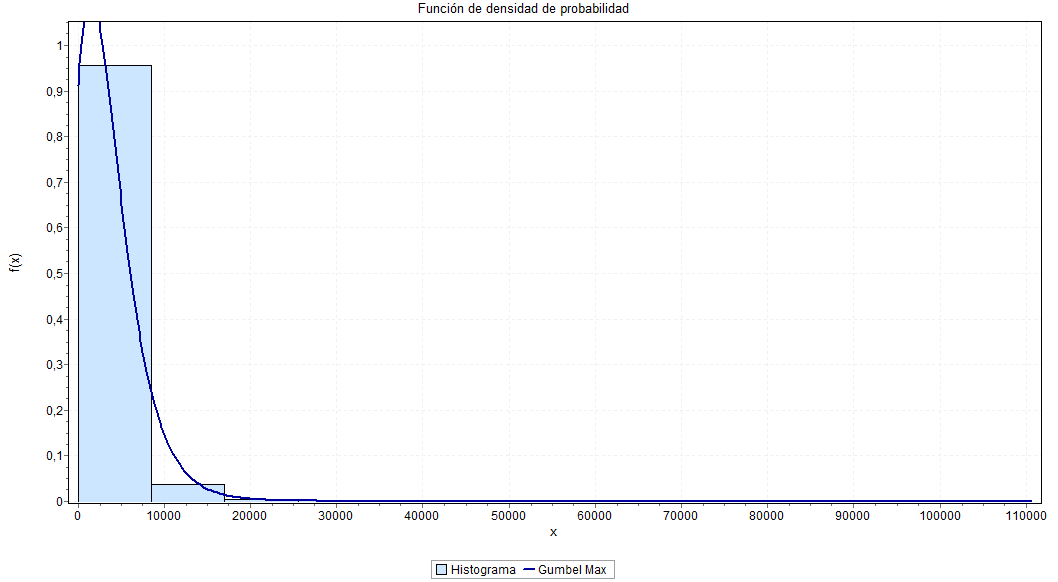
Obtención de las FDP:

Para esta simulación tuvimos que conseguimos dos FDP: IA y TE

Para la obtención de IA lo que hicimos fue correr un programa llamado Wireshark para capturar los paquetes de red que llegaban. El programa nos proporcionó el tiempo de llegada relativo según el primer paquete (siendo para este t=0) entonces exportamos esto a formato csv y mediante un script hicimos los respectivos cálculos para sacar el intervalo de arribo, luego cargamos los valores en el Easy Fit y la distribución que nos sugería fue la de Gumbel Max, cuya función de densidad de probabilidad acumulada es:

Donde y

Grafico obtenido:



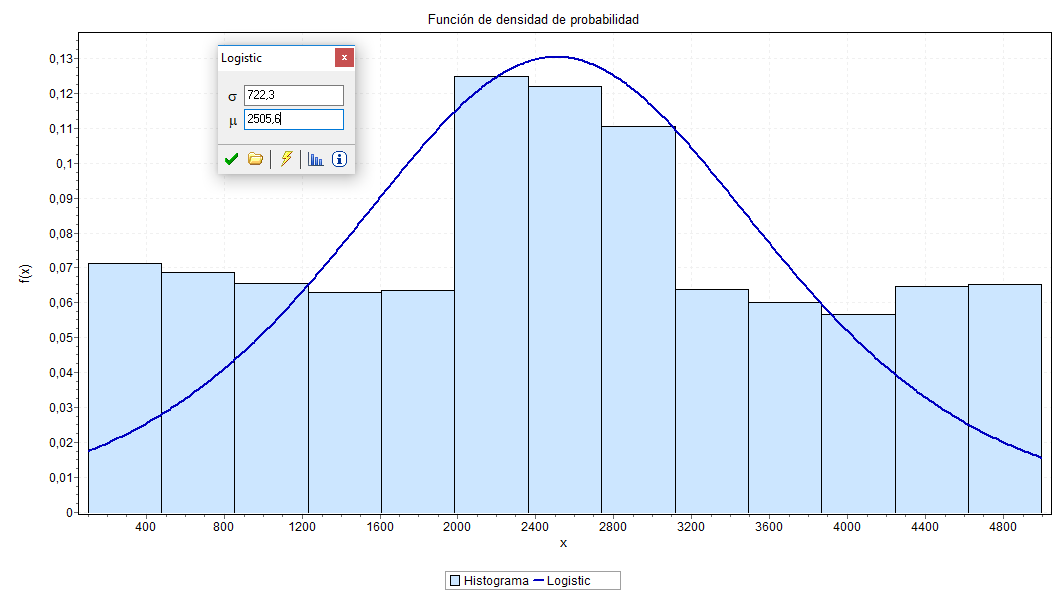
*Figura 1 – Función de densidad de probabilidad Gumbel Max para los tiempos de arribo obtenidos*

Usamos la metodología de función inversa, quedando:

Para la obtención de la TE corrimos un programa prototipo, similar al que correría en la placa Rasperry Pi. El input de este programa fue la misma captura de paquetes que para la obtención de la IA, por cada paquete analizado se guardó en un archivo la diferencia de tiempo en microsegundos respecto al paquete anterior, luego ese archivo se cargó en el Easy Fit. La distribución que sugirió fue Logistica, cuya función de densidad acumulada es:

Donde y

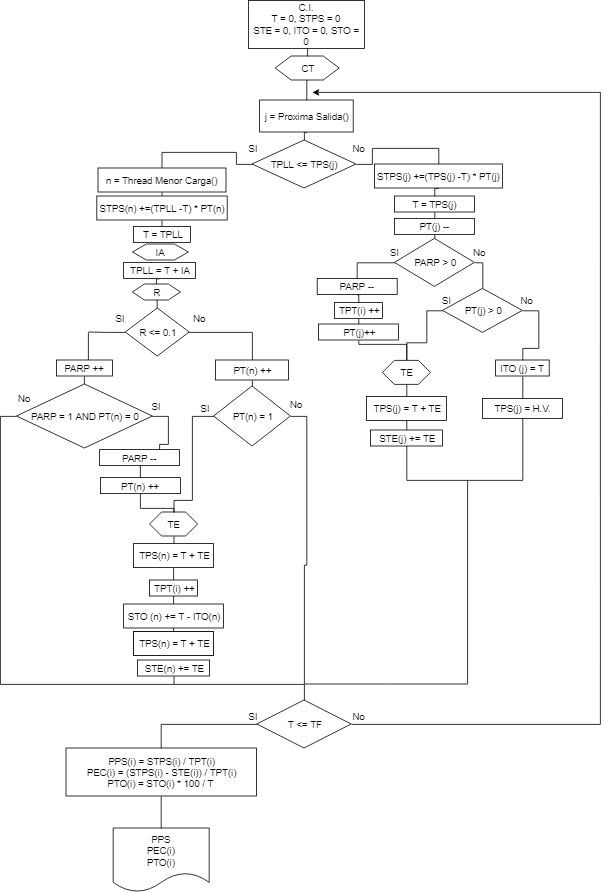
Grafico obtenido:



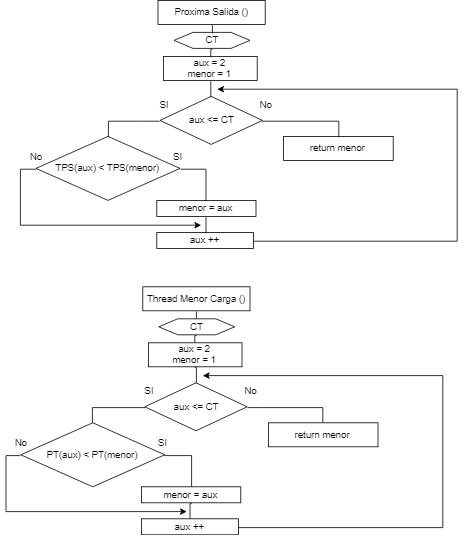
*Figura 2 – Función de densidad de probabilidad logística para el tiempo de evaluación de paquetes*

Usamos la metodología de función inversa, quedando:

Diagrama:



*Figura 3 – Rutina Principal de la simulacion*



*Figura 4 – Rutinas auxiliares*

Código:

import random

import sys

import math

# CONDICIONES INICIALES

CT = int(sys.argv[1])

T = 0 # tiempo

HighValue = 10000000000000000

SS = [0] \* CT # sumatoria tiempo de salida de cada thread

SLL = [0] \* CT # sumatoria tiempo de llegada de cada thread

STE = [0] \* CT # sumatoria tiempo de evaliacion de cada thread

ITO = [0] \* CT # inicio tiempo oscioso de cada thread

STO = [0] \* CT # sumatoria tiempo oscioso de cada thread

TPLL = 0 # tiempo de proxima llegada

TPS = [HighValue] \* CT # tiempo de proxima salia de cada thread

PARP = 0 # cantidad de paquetes ARP encolados

PT = [0] \* CT # cantidad de paquetes en cada thread

TF = int(sys.argv[2]) # tiempo final

TPT = [0] \* CT # total paquetes que pasaron por el thread

STPS = [0] \* CT

outputFile = sys.argv[3]

print ("Cantidad de Threads " + str(CT))

print ("Simulacion para " + str(TF) + " Microsegundos")

def getIndexMinValueOfList(list):

return list.index(min(list))

def getIntervaloArribo():

rand = random.uniform(0,1)

resultado = 157.33 - 285.38 \* math.log(-1\*math.log(rand))

if resultado < 0:

return getIntervaloArribo()

else:

return resultado

def getTiempoEvaluacion():

rand = random.uniform(0,1)

resultado = 2505.6 - math.log((1/rand) - 1)\*722.3

if resultado < 0:

return getTiempoEvaluacion()

else:

return resultado

def atenderPaquete(threadMenorCarga):

global TPS

global ITO

global STO

global STE

global T

TE = getTiempoEvaluacion()

TPS[threadMenorCarga] = T + TE

STO[threadMenorCarga] += T - ITO[threadMenorCarga]

STE[threadMenorCarga] += TE

while T <= TF:

threadProximaSalida = getIndexMinValueOfList(TPS)

if TPLL <= TPS[threadProximaSalida]: # llegada

threadMenorCarga = getIndexMinValueOfList(PT)

for thread in range(0, CT):

STPS[thread] += (TPLL - T)\*PT[thread]

T = TPLL

IA = getIntervaloArribo()

TPLL = T + IA

R = random.randint(0,99)

if(R <= 9): # paquete de prioridad

PARP +=1

if(PARP == 1 and PT[threadMenorCarga] == 0):

PARP -= 1

PT[threadMenorCarga] += 1

TPT[threadMenorCarga] += 1

atenderPaquete(threadMenorCarga)

else: # paquete baja prioridad

PT[threadMenorCarga] += 1

TPT[threadMenorCarga] += 1

if(PT[threadMenorCarga] == 1):

atenderPaquete(threadMenorCarga)

else: # salida

for thread in range(0, CT):

STPS[thread] += (TPS[threadProximaSalida] - T)\*PT[thread]

T = TPS[threadProximaSalida]

PT[threadProximaSalida] -= 1

if PARP > 0 or (PARP == 0 and PT[threadProximaSalida] > 0):

if(PARP > 0):

PARP -= 1

PT[threadProximaSalida] += 1

TPT[threadProximaSalida] += 1

TE = getTiempoEvaluacion()

TPS[threadProximaSalida] = T + TE

STE[threadProximaSalida] += TE

else: # el thread pasa a estar oscioso

ITO[threadProximaSalida] = T

TPS[threadProximaSalida] = HighValue

file = open(outputFile, "w")

for thread in range(0, CT):

print("Informacion Thread " + str(thread+1))

file.write("Informacion Thread " + str(thread+1)+"\n")

print("PPS " + str(STPS[thread] / TPT[thread]))

file.write("PPS " + str(STPS[thread] / TPT[thread])+"\n")

print("PEC " + str((STPS[thread] - STE[thread]) / TPT[thread]))

file.write("PEC " + str((STPS[thread] - STE[thread]) / TPT[thread])+"\n")

print("PTO " + str((STO[thread] \* 100 / T)))

file.write("PTO " + str((STO[thread] \* 100 / T))+"\n")

**Resultados**

Corrimos la simulación variando la cantidad de threads y siendo la duración de 2 segundos, los resultados fueron los siguientes (recordar que los valores están en microsegundos):

1 Thread:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO (%) |
| 1 | 830142.51 | 829726.51 | 0.0 |

*Tabla 1 – Resultado simulación para 1 Thread*

3 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO (%) |
| 1 | 511014.84 | 509750.67 | 0.0 |
| 2 | 508667.44 | 507409.24 | 0.029 |
| 3 | 508263.09 | 507004.87 | 0.034 |

*Tabla 2 – Resultado simulación para 3 Threads*

5 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO (%) |
| 1 | 218132.16 | 216078.74 | 0.020 |
| 2 | 218605.81 | 216547.13 | 0.09 |
| 3 | 222123.87 | 220029.87 | 0.17 |
| 4 | 220983.83 | 218897.01 | 0.16 |
| 5 | 222435.62 | 220331.56 | 0.36 |

*Tabla 3 – Resultado simulación para 5 Threads*

8 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO (%) |
| 1 | 3637.66 | 1065.89 | 5.40 |
| 2 | 3387.26 | 800.36 | 7.57 |
| 3 | 3237.09 | 594.49 | 11.58 |
| 4 | 3113.10 | 443.95 | 15.17 |
| 5 | 2933.81 | 256.39 | 20.77 |
| 6 | 2761.78 | 161.26 | 25.91 |
| 7 | 2704.52 | 113.39 | 33.89 |
| 8 | 2700.93 | 72.77 | 43.43 |

*Tabla 4 – Resultado simulación para 8 Thread*

10 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO (%) |
| 1 | 2866.38 | 289.98 | 9.91 |
| 2 | 2718.27 | 137.95 | 13.36 |
| 3 | 2725.69 | 66.48 | 15.91 |
| 4 | 2737.52 | 31.73 | 19.41 |
| 5 | 2756.96 | 6.35 | 24.76 |
| 6 | 2669.83 | 0 | 35.10 |
| 7 | 2630.18 | 0 | 43.14 |
| 8 | 2732.40 | 8.63 | 53.48 |
| 9 | 2665.58 | 17.27 | 66.37 |
| 10 | 2506.90 | 9.71 | 75.88 |

*Tabla 5 – Resultado simulación para 10 Threads*

**Discusión**

Por experiencia los programas que por cuyo objetivo necesitan una gran capacidad de computo (por ejemplo algoritmos de fuerza bruta) utilizan por default diez threads, así que se puede tomar como referencia este número. Al realizar las simulaciones fuimos aumentando la cantidad de threads, cuando llegamos a diez vimos que el tiempo ocioso de algunos era muy elevado, por lo tanto decimos detenernos en ese punto. Algunos threads superaban el 50 % de tiempo ocioso, por lo tanto la cantidad óptima debe ser menor.

Nos parece correcto usar ocho threads ya que el tiempo de respuesta es rápido, aproximadamente tres milisegundos de permanencia en el sistema. Si tenemos en cuenta que la idea original es que el dispositivo se pueda integrar en una red hogareña o a lo sumo en una red de una empresa que cumpla funciones administrativas, es decir, redes que no participen en tareas de gran importancia o que no necesiten tiempos de respuesta inmediatos, este valor nos parece acertado.

**Conclusión**

Se lo logro a partir de la simulación ver los tiempos de respuesta en varios escenarios, vimos cómo estos se reducían al aumentar la variable de control, mientras más threads haya, mejor se distribuyen las tareas. Además vimos que con una gran cantidad de threads, la mayoría tiene un gran tiempo ocioso por lo tanto es poco performante. Finalmente los tiempos obtenidos para ocho threads nos parecieron acordes, a la vez que esa cantidad no va a generar mucha carga sobre el sistema operativo.

**Referencias:**

[1] <https://es.wikipedia.org/wiki/Protocolo_de_resoluci%C3%B3n_de_direcciones>

[2] <https://es.wikipedia.org/wiki/Suplantaci%C3%B3n_de_ARP>

[3] https://hipertextual.com/archivo/2014/06/ataque-man-in-the-middle/

**Datos de contacto:**

Nombre: Tomás De Pietro:

Email: [tomas94depi@gmail.com](mailto:tomas94depi@gmail.com)

Institución: UTN FRBA