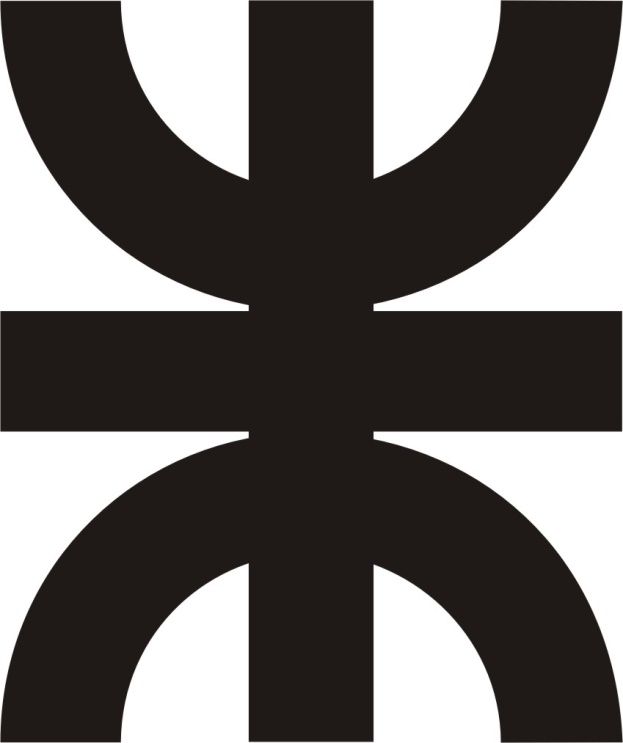
****

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

**De Pietro, Tomás**

**Anderson, Nicolás**

**Zapata, Mauro**

**Abstract**

Esta investigación intenta demostrar si es posible la introducción de un nuevo dispositivo de red que funcione como un firewall, siendo capaz de analizar los paquetes que circulen a través de él y detectar posibles ataques informáticos en la segunda y tercera capa del modelo OSI, siendo estas la capa de transporte de datos y la capa de red. El principal parámetro que debemos de tener en cuenta será el tiempo que tarda en procesar los paquetes de red, así que para tener valores aproximados se hará una simulación evento a evento.

**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 lo router, a su vez esto conlleva a que el tráfico de la red pase por el dispositivo que realiza este ataque, logrando espiar que acciones se realiza, pudiendo descartarlas o modificarlas.

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

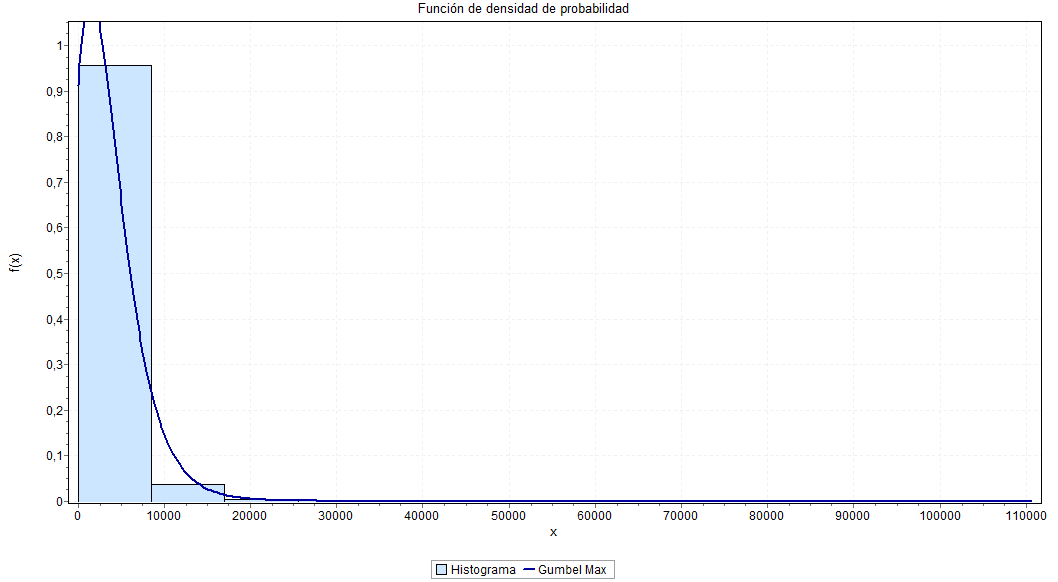
Obtención de las FDP:

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:



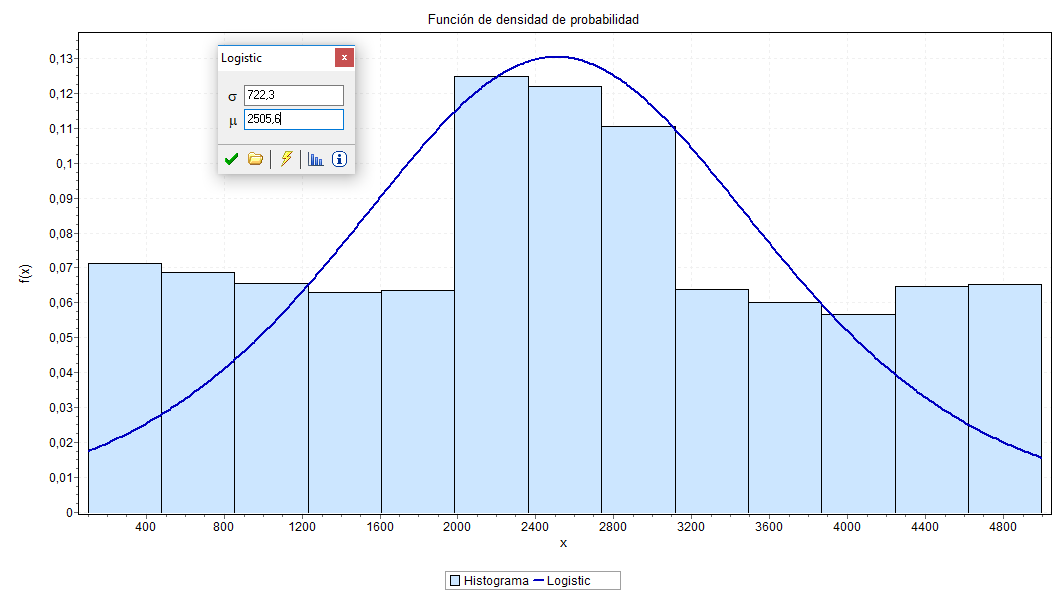
*Figura 1 – Función de densidad de probabilidad Gumbel Max para los tiempos de arribo obtenidos*

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

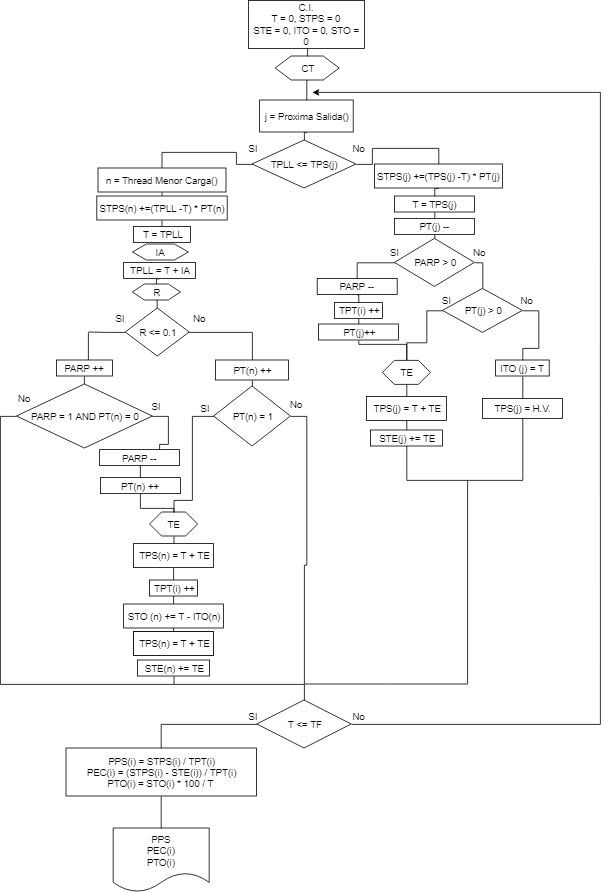
Grafico obtenido:



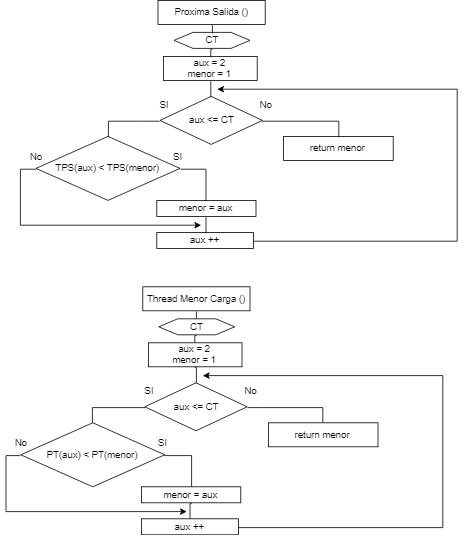
*Figura 2 – Función de densidad de probabilidad logística para el tiempo de evaluación de paquetes*

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 = 10000000000

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)

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

def getTiempoEvaluacion():

rand = random.uniform(0,1)

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

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)

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

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

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

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, los resultados fueron los siguientes:

1 Thread:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 488897.31 | 488534.87 | 0.0 |

*Tabla 1 – Resultado simulación para 1 Thread*

2 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 209041.57 | 208341.72 | 0.0 |
| 2 | 202997.22 | 202289.25 | 0.014 |

*Tabla 2 – Resultado simulación para 2 Thread*

3 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 115817.66 | 114755.88 | 0.0 |
| 2 | 118435.36 | 117332.66 | 0.039 |
| 3 | 114728.78 | 113638.31 | 0.072 |

*Tabla 3 – Resultado simulación para 3 Thread*

4 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 69790.81 | 68379.47 | 0.0 |
| 2 | 77712.06 | 76308.00 | 0.044 |
| 3 | 73567.04 | 72155.43 | 0.14 |
| 4 | 71819.71 | 70390.34 | 0.15 |

*Tabla 4 – Resultado simulación para 4 Thread*

5 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 38840.07 | 37018.69 | 0.015 |
| 2 | 39006.62 | 37149.24 | 0.04 |
| 3 | 37929.33 | 36076.78 | 0.13 |
| 4 | 41003.20 | 39153.75 | 0.13 |
| 5 | 42930.95 | 41175.06 | 0.072 |

*Tabla 5 – Resultado simulación para 5 Thread*

6 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 20010.85 | 17879.58 | 0.0 |
| 2 | 21046.65 | 18875.34 | 0.49 |
| 3 | 23192.82 | 21059.79 | 0.15 |
| 4 | 19075.89 | 17010.78 | 0.25 |
| 5 | 20258.00 | 18229.27 | 0.42 |
| 6 | 19829.23 | 17623.10 | 0.84 |

*Tabla 6 – Resultado simulación para 6 Thread*

7 Threads:

|  |  |  |  |
| --- | --- | --- | --- |
| Thread | PPS | PEC | PTO |
| 1 | 10317.11 | 7881.05 | 0.0 |
| 2 | 10418.89 | 8039.53 | 0.099 |
| 3 | 10882.49 | 8426.57 | 0.11 |
| 4 | 9892.63 | 7484.39 | 0.12 |
| 5 | 11470.40 | 9042.02 | 0.26 |
| 6 | 11062.40 | 8666.04 | 0.28 |
| 7 | 11085.65 | 8641.66 | 0.34 |

*Tabla 7 – Resultado simulación para 7 Thread*

**Discusión**

**Conclusión**

**Agradecimientos:**

**Referencias**

**Datos de contacto**