# Análisis y Desarrollo de Algoritmo de Optimización de Colonia de Hormigas para el Filtrado Eficiente de Paquetes en un Cortafuegos.

Camilo Vera Cortés
Seminario de computadores: Introducción a SoftComputing y Aplicaciones,
Departamento de Electronica,
Universidad Técnica Federico Santa María, Chile.
camilo.vera@alumnos.usm.cl

**Resumen.** Un cortafuegos es un componente de seguridad central dentro de una red, encargado de supervisar y administrar el flujo de paquetes a través de la misma. Un cortafuegos obedece a un conjunto de reglas, que se comparan con las cabeceras TCP e IP de un paquete de información, las que determinan la acción final a tomar. En la medida en que el ancho de banda disponible aumenta cada día, la eficiencia de los algoritmos de comparación en un cortafuegos se vuelve un criterio muy relevante en el desempeño global de una red. En este sentido, se presenta en este documento el análisis de un algoritmo de búsqueda de reglas basado en una colonia de hormigas propuesto en el trabajo "ANT COLONY OPTIMIZATION BASED APPROACH FOR EFFICIENT PACKET FILTERING IN FIREWALL[1]". Adicionalmente se presentan métodos para mejorar el desempeño del trabajo mencionado.

**Palabras Claves:** Algoritmo colonia de hormigas, cortafuegos, filtrado de paquetes, optimización de búsqueda.

## 1 Introducción

Los algoritmos de enjambres son un acercamiento a la capacidad de resolver problemas complejos que presentan algunos animales e insectos al trabajar en conjunto, capacidad que pierden cuando se analiza el individuo separado del grupo, la gracia de estos algoritmos es la simplicidad que presenta el individuo, y su potencial al hacerlo trabajar en conjunto para la búsqueda de soluciones optimas. Estos algoritmos usualmente son usados para la optimización de problemas complejos, toma de decisiones y robot colaborativos.

La optimización de Colonia de hormigas simula el funcionamiento real que presentan las hormigas en su búsqueda por alimentos, llegando a recorrer enormes distancias siguiendo rutas cercanas al optimo, básicamente cada hormiga tiene la capacidad de producir feromonas (sustancia química), la cual utiliza para marcar el camino que recorre, la deposición de esta sustancia atrae a otras hormigas a seguir ese

camino, las cuales a su ves también depositan su propia feromona en el camino, al final del proceso, los caminos mas eficientes para avanzar de un punto X a otro Y son los que prevalecen y los otros desaparecen con el tiempo al evaporarse la feromona. De es comportamiento se pueden sacar las ideas principales para la optimización de un variado espectro de problemas, en los que se incluye el que compete, que es el filtrado eficiente de paquetes en una red de TCP/IP.

Este documento tiene la finalidad de explicar y demostrar empíricamente el uso del algoritmo de colonia de hormigas propuesto por los autores del articulo "Ant Colony Optimization based approach for efficient packet filtering in firewall" para la búsqueda eficiente de reglas en un set de reglas dado un encabezado TCP/IP, con este propósito se desarrollo una aplicación en Java que implementa el algoritmo propuesto, una mejora a este mismo y un algoritmo de búsqueda secuencial, cabe destacar que el documento original también hace referencias a otros métodos de búsqueda de reglas que no se abordaran en este articulo.

En la siguiente sección se analiza el funcionamiento general de un cortafuegos y el algoritmo secuencial de búsqueda de paquetes, que representaría el acercamiento más básico y sencillo de implementar un filtrado de paquetes dado un conjunto de reglas. La sección 3 explica el funcionamiento de la búsqueda de reglas dentro de un conjunto de reglas utilizando el algoritmo de colonia de hormigas así como su implementación. En la sección 4 se visualizan y discuten los datos entregados por la simulación de los diferentes algoritmos, incluido una mejora al propuesto por el articulo en el que se basa este trabajo.

## 2 Cortafuegos y algoritmo secuencial

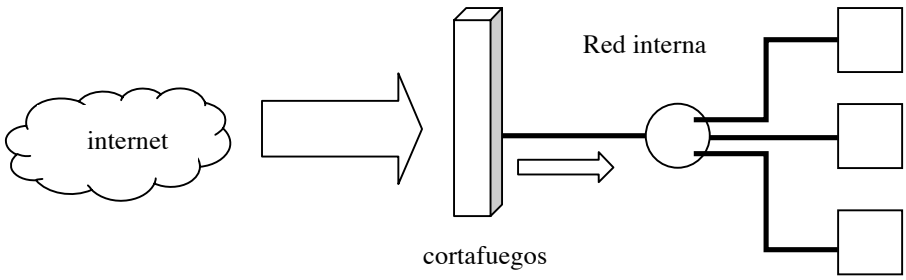



**Fig. 1.** Funcionamiento de un cortafuegos

Como se aprecia en la Figura 1, un cortafuegos es el componente encargado de decidir que paquete entra a la red interna o cual sale hacia la red externa, el funcionamiento básico de este dispositivo es bastante simple, analiza la cabecera TCP/IP del paquete entrante (o saliente) y busca en su tabla de reglas si existe alguna que coincida con el paquete en cuestión, de existir la regla aplica la acción

determinada por esta (permitir o bloquear), si la búsqueda no es exitosa, se debe especificar que acción seguir. (en este articulo se utiliza la acción de rechazar los paquetes que no tiene calzan con ninguna regla, aunque no tiene ninguna relevancia en el estudio).

La tabla 1 muestra los datos que contiene la cabecera de un paquete TCP/IP, a su vez, la tabla 2 despliega un set de reglas a modo de ejemplo.

**Tabla 1.** cabecera de paquete TCP/IP.

| Ip fuente | Puerto fuente | Ip destino | Puerto destino |
|---|---|---|---|
| **136.85.133.10** | 20 | 192.168.36.1 | 1998 |

**Tabla 2.** set de reglas de cortafuego.

| Ip fuente | Puerto fuente | Ip destino | Puerto destino | Acción |
|---|---|---|---|---|
| ***.*.*.*** | 20 | 192.168.*.* | Any | Drop |
| **180.92.101.7** | Any | 192.168.35.8 | 45970 | Drop |
| **Any** | Any | 192.168.10.* | 80 | Accept |

Una solución simple para el filtrado de paquetes es la búsqueda secuencial de reglas, la cual empieza recorriendo una columna previamente definida de manera secuencial y comparándola con el campo respectivo del paquete entrante, de encontrar un calce, comprueba los otros campos del paquete para ver si coinciden con la regla, si es así, ejecuta la acción, si no, continua su búsqueda.

Este algoritmo funciona bien para cortafuegos caseros o de pequeñas oficinas, en donde la tabla de reglas no es mayor a 10 ítems, sin embargo, dado un set de reglas de un tamaño considerable, la búsqueda secuencial presenta problemas dado la complejidad del algoritmo (O(n)), lo que la hace ineficiente y muchas veces impracticable en ciertas circunstancias.

## 3 Algoritmo de Colonia de Hormigas aplicado a la búsqueda de reglas en firewall

El algoritmo de Colonia de Hormigas es bastante más complejo de explicar y requiere una estructura de datos inicial distinta a la utilizada por un cortafuegos normal, en primera instancia, las columnas correspondientes a la ip de destino y fuente se deben representar como un numero, para lo cual se aplica la siguiente transformación ejemplificada: 123.*.92.1 -> 123000092001.

Un cortafuego que implemente este algoritmo debe dividir su tabla de reglas en 5 módulos distintos, donde cada uno representa un campo en particular, de este forma, todas las reglas que tengan una ip de fuente distinta de *.*.*.* se almacenaran en el modulo de ip fuente, los que contengan puerto fuente distinto de "Any" se almacenaran en el modulo de puerto fuente y así sucesivamente. Una regla que contenga más de un campo distinto de "Any" o *.*.*.* (según corresponda) debe almacenarse en todos los módulos que corresponda, una vez terminado este proceso, se debe ordenar cada modulo de menor a mayor teniendo como referencia la columna asociada a ese modulo. El articulo referenciado [1] contiene ejemplos de cómo debería quedar organizado cada modulo dentro del cortafuegos.

El proceso de búsqueda se ejecuta por varios agentes hormiga ( desde ahora agente), el cual va buscando dentro de un modulo especificado una regla que calce con el paquete en cuestión, para la búsqueda va depositando su feromona dentro del set de reglas definido dentro de un modulo especifico, limitando el espacio de búsqueda para que el siguiente agente se acerque más a la solución optima.

### 3.1 Implementación

El articulo [1] en el cual este documento se basa explica detalladamente el funcionamiento de este algoritmo, sin embargo, se detectó un error en el seudo-código proporcionado por lo que se adjunta una versión corregida y más especifica del mismo.

```
Rule findRuleByPacket(Packet packet){
  posPos = 0, negPos = rulesUnit.size()+1, ceroPos=0;
  range = negPos – posPos;
  feromona = random(range);
  rule = rulesUnit.get(feromona);
  if(packet.comparedValue>rule.comparedValue)
    energy = 1; posPos = feromona;
  else if(packet.comparedValue<rule.comparedValue)
    energy = -1; negPos = feromona;
  else// packet compare value == rule compapare value
    energy = 0; ceroPos = feromona;
  while (energy != 0 && range != 1){
    range = negPos – posPos;
    feromona = random(range);
    rule = rulesUnit.get(feromona);
```

```
    //actualizar energy, posPos, negPos y ceroPos
  }
  if(energy == 0){
    gruop1CeroPos = ceroPos;
    gruop2CeroPos = ceroPos;
    while(group1CeroPos-posPos != 1){
      feromona = random(range);
      rule = rulesUnit.get(feromona);
      //actualizar energía y feromona
      if(packet.comparedValue == rule.comparedValue)
         tabuList.add(rule);
    }
    sort(tabuList);
    for(i = tabuList[0]; i<=tabuList[size];i++)
        if(packet.matches(rule)) return rule;
    while(negPos - gruop2CeroPos != 1){
       //similar a while anterior pero busca en lista
       //superior
    }
    sort(tabuList);
    for(i = tabuList[0]; i<=tabuList[size];i++)
        if(packet.matches(rule)) return rule;
  }if(negPos – posPos == 1)
     return null; //no se encontró regla
}
```

El algoritmo detallado en el articulo base comete un error en el primer bucle while, pues señala que el bucle se ejecutara hasta que la energía sea cero O la deferencia entre la posición negativa y positiva sea uno, es decir, con cualquiera de estas dos condiciones que se cumpla, el ciclo seguirá ejecutándose, para corregir esto, se debe cambiar el operador booleano Or (||) por And (&&) para que el algoritmo tenga sentido, además de eso, se debe tener cuidado con los márgenes de los arreglos, pues el documento original se basa en arreglos que inician en 1, cuando por lo general los arreglos y listas comienzan en cero.

La implementación completa del experimento contempla la lectura de las reglas y paquetes de ejemplo desde archivos Excel, así como un archivo de configuración para seleccionar el algoritmo y algunos variables que determinan el comportamiento de la simulación , el cortafuegos que recoge los paquetes entrantes y ejecuta el algoritmo y un receptor de paquetes que guarda en otro documento Excel los datos de salida del programa para después graficarlos utilizando herramientas estándar.

### 3.2 Mejoras

Una mejora propuesta en este articulo propone modificar la forma en la que se calcula la feromona, dejando de ser un valor aleatorio restringido al espacio de búsqueda actual (1) y remplazándolo por el valor ubicado en el medio entre la posición positiva y negativa (2).

$$positivePosition+random(range); \quad (\mathbf{1})$$

$$positivePosition+ceil((negativePosition-positivePosition)/2) \quad (\mathbf{2})$$

## 4 Resultados experimentales

### 4.1 Datos

Para los siguientes experimentos, se utilizaron los mismos datos de pruebas del articulo base, los cuales a su ves, en el caso de los paquetes de ejemplo, fueron obtenidos del sitio web del MIT Lincon Lab[2].

**Tabla 3. set de reglas de filtrado para cortafuegos**

| Source Address | Source Port | Destination Address | Destination Port | Action |
|---|---|---|---|---|
| Any | 20 | Any | 1052 | Drop |
| Any | 20 | Any | 1104 | Drop |
| Any | 20 | Any | 1106 | Drop |
| Any | 20 | Any | 1118 | Drop |
| Any | 20 | Any | 1169 | Drop |
| Any | 20 | Any | 20 | Drop |
| Any | 1028 | Any | 80 | Drop |
| Any | 1050 | Any | 25 | Drop |
| Any | 1051 | Any | 21 | Drop |

| | | | | |
|---|---|---|---|---|
| Any | 1092 | Any | 139 | Drop |
| Any | 1098 | Any | 12345 | Drop |
| Any | 1107 | Any | 25 | Drop |
| Any | 1114 | Any | 23 | Drop |
| Any | 1115 | Any | 80 | Drop |
| Any | 1116 | Any | 21 | Drop |
| Any | 1169 | Any | 23 | Drop |
| Any | 1170 | Any | 23 | Drop |
| Any | 1173 | Any | 23 | Drop |
| Any | 1173 | Any | 25 | Drop |
| Any | 1301 | Any | 80 | Drop |
| Any | 1302 | Any | 80 | Drop |
| Any | 1303 | Any | 80 | Drop |
| Any | 1305 | Any | 80 | Drop |
| Any | 1321 | Any | 23 | Drop |
| Any | 1387 | Any | 21 | Drop |
| Any | 1389 | Any | 21 | Drop |
| Any | 1392 | Any | 21 | Drop |
| Any | 1394 | Any | 23 | Drop |
| Any | 1395 | Any | 80 | Drop |
| Any | 1396 | Any | 80 | Drop |
| Any | 1397 | Any | 80 | Drop |
| Any | 1398 | Any | 80 | Drop |
| Any | 1399 | Any | 80 | Drop |
| Any | 1400 | Any | 80 | Drop |
| Any | 1401 | Any | 80 | Drop |
| Any | 1427 | Any | 23 | Drop |
| Any | 4482 | Any | 80 | Drop |
| Any | 37433 | Any | 25 | Drop |
| Any | 49924 | Any | 80 | Drop |
| Any | 51173 | Any | 21 | Drop |
| Any | 55394 | Any | 22 | Drop |
| Any | 8080 | Any | Any | Accept |

**Tabla 4.** paquetes entrantes de ejemplo

| Source Address | Source Port | Destination Address | Destination Port |
|---|---|---|---|
| 194.027.251.021 | 1026 | 172.016.112.100 | 25 |
| 135.008.060.182 | 1107 | 172.016.112.100 | 25 |
| 206.048.044.018 | 1098 | 172.016.112.100 | 12345 |
| 172.016.113.204 | 1051 | 172.016.112.100 | 21 |
| 172.016.112.100 | 20 | 172.016.113.204 | 1052 |
| 172.016.112.100 | 20 | 172.016.113.204 | 1104 |
| 172.016.112.100 | 20 | 172.016.113.204 | 1106 |
| 153.107.252.061 | 37433 | 172.016.112.100 | 25 |
| 153.107.252.061 | 55394 | 172.016.112.100 | 22 |
| 153.107.252.061 | 49924 | 172.016.112.100 | 80 |
| 153.107.022.061 | 51173 | 172.016.112.100 | 21 |
| 172.016.118.070 | 1114 | 172.016.112.100 | 23 |
| 172.016.118.070 | 1115 | 172.016.112.100 | 80 |
| 205.160.208.190 | 1116 | 172.016.112.100 | 21 |
| 172.016.112.100 | 20 | 205.160.208.190 | 1118 |
| 172.016.112.100 | 20 | 205.160.208.190 | 1169 |
| 205.160.208.190 | 1170 | 172.016.112.100 | 23 |
| 172.016.118.070 | 1169 | 172.016.112.100 | 23 |
| 209.001.012.046 | 1028 | 172.016.112.100 | 80 |
| 207.230.054.203 | 1050 | 172.016.112.100 | 25 |
| 208.239.005.230 | 1173 | 172.016.112.100 | 25 |
| 172.016.112.100 | 4482 | 208.239.005.230 | 80 |
| 172.016.112.100 | 4482 | 208.239.005.230 | 80 |
| 194.027.251.021 | 1301 | 172.016.112.100 | 80 |
| 194.027.251.021 | 1302 | 172.016.112.100 | 80 |
| 194.027.251.021 | 1303 | 172.016.112.100 | 80 |
| 194.027.251.021 | 1305 | 172.016.112.100 | 80 |
| 172.016.113.204 | 1173 | 172.016.112.100 | 23 |
| 206.048.044.018 | 1387 | 172.016.112.100 | 21 |
| 172.016.112.100 | 20 | 206.048.044.018 | 20 |
| 206.048.044.018 | 1389 | 172.016.112.100 | 21 |
| 206.048.044.018 | 1392 | 172.016.112.100 | 21 |

| | | | |
|---|---|---|---|
| 206.048.044.018 | 1394 | 172.016.112.100 | 23 |
| 206.048.044.018 | 1395 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1396 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1397 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1398 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1399 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1400 | 172.016.112.100 | 80 |
| 206.048.044.018 | 1401 | 172.016.112.100 | 80 |

### 4.2 Resultados

los valores obtenidos de la simulación tienen como primer objetivo comprobar empíricamente los datos obtenidos por los autores del articulo de referencia [1], es por eso que los primeros gráficos son similares a los mostrados en dicho documento para los algoritmos correspondientes.

**Fig. 2.**

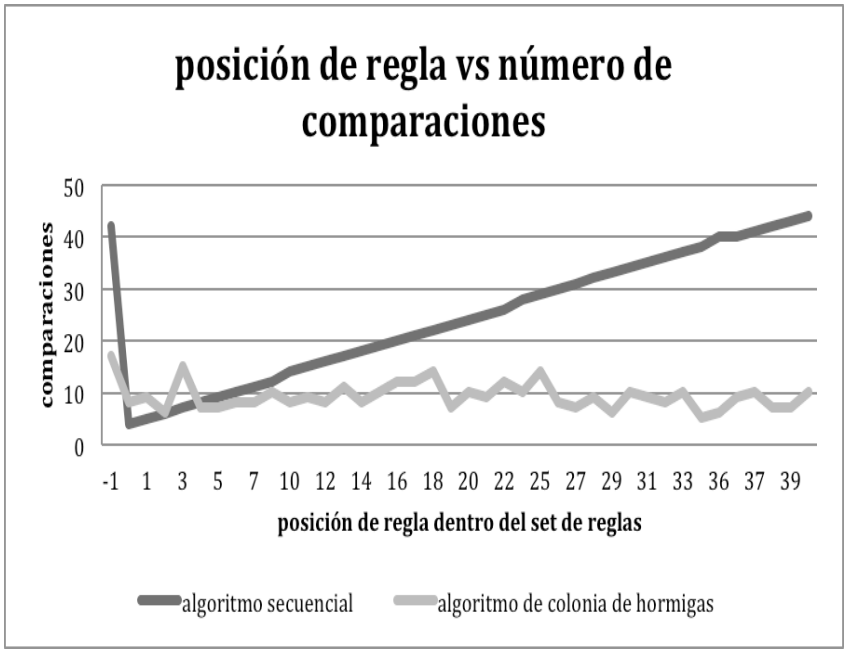
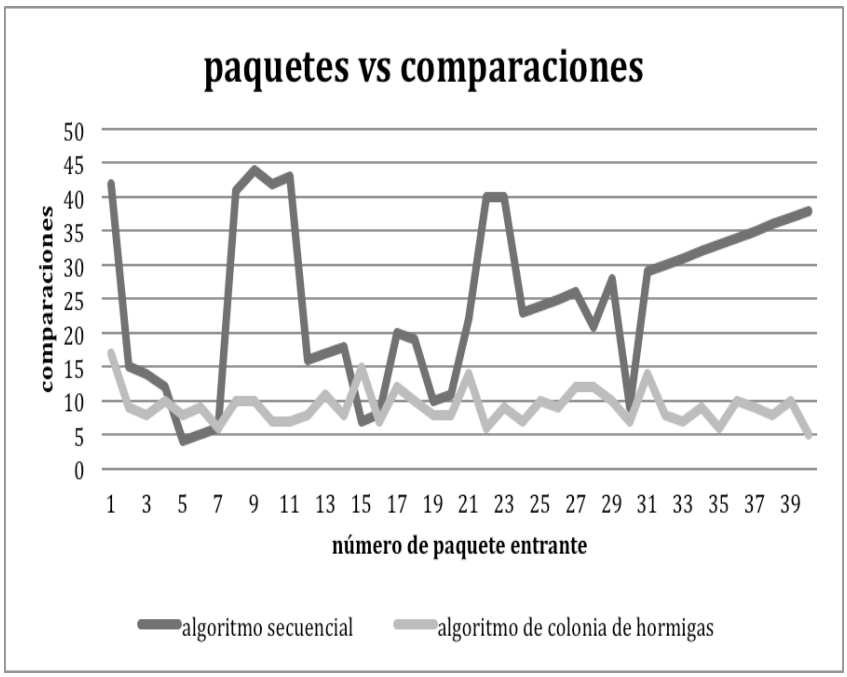


**Fig. 3.**




El primer grafico (fig. 2) muestra como a medida que el cortafuegos tiene más reglas definidas peor es su comportamiento utilizando algoritmo secuencial, necesitando bastantes más comparaciones para encontrar una regla que su contraparte de colonia de hormigas. El segundo grafico (fig. 3) muestra como evolucionaron los algoritmos a medida que iban recibiendo los paquetes, se aprecia claramente la superioridad del algoritmo de colonia de hormigas, para casi todos los escenarios posibles (menos en el mejor caso, donde el algoritmo secuencial tiene una pequeña ventaja). Se aprecia también un salto inesperado al principio de los dos gráficos, para los dos algoritmos, este salto se debe al que se considero con identificador -1 a la acción que se ejecuta cuando el paquete no calza con ninguna regla definida.

Los datos obtenidos de la simulación coinciden perfectamente con los datos obtenidos por los autores del articulo de referencia[1], lo que demuestra la rigurosidad de sus mediciones.

**Fig. 4.**

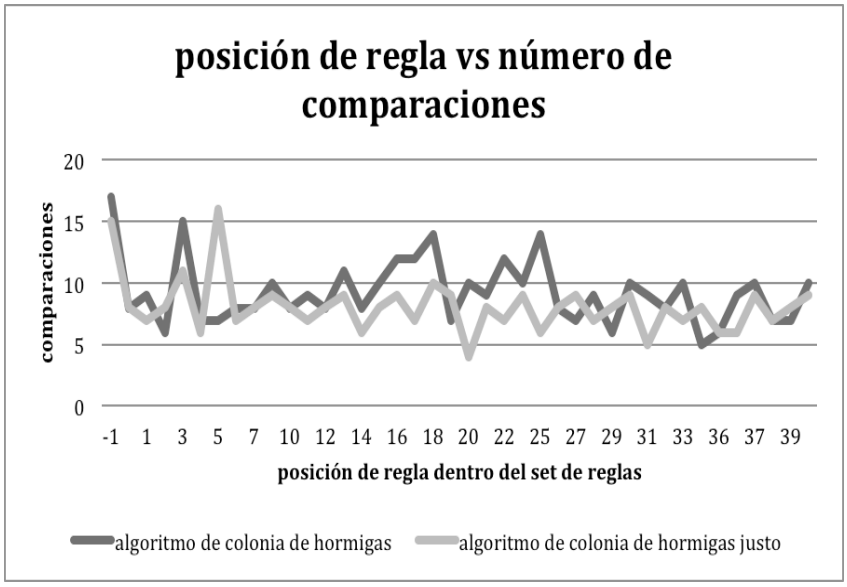
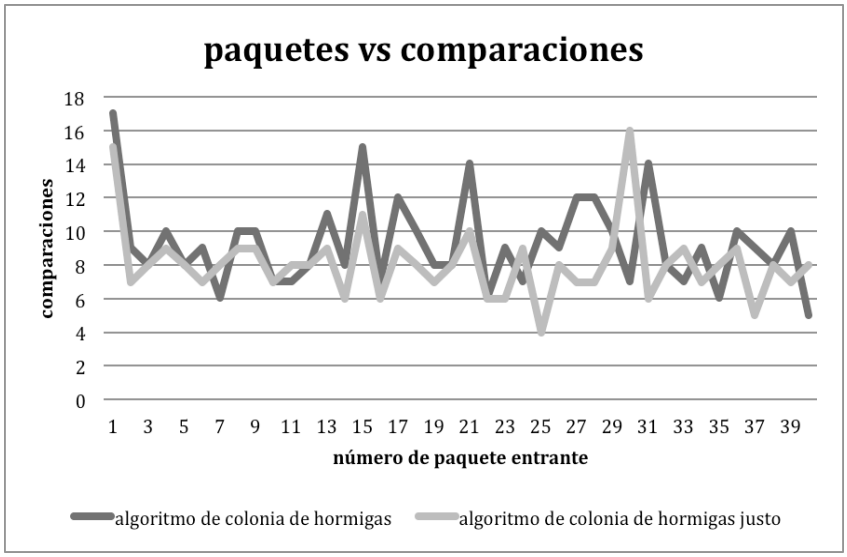

posición de regla vs número de comparaciones
comparaciones
20
15
10
5
0
-1 1 3 5 7 10 12 14 16 18 20 22 25 27 29 31 33 36 37 39
posición de regla dentro del set de reglas
algoritmo de colonia de hormigas
algoritmo de colonia de hormigas justo

**Fig. 5.**


paquetes vs comparaciones
comparaciones
18
16
14
12
10
8
6
4
2
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
número de paquete entrante
algoritmo de colonia de hormigas
algoritmo de colonia de hormigas justo

Los gráficos 3 y 4 (fig. 4, fig. 5) hacen las mismas comparaciones que los gráficos anteriores pero en este caso utilizando el algoritmo de hormiga propuesto por los autores del documento de referencia [1] y el algoritmo de hormiga justo propuesto en este trabajo. Se aprecia una leve mejora en promedio en el numero de comparaciones realizadas por el algoritmo de colonia de hormigas justo, y si bien, puede no ser significativa, este algoritmo presenta otras ventajas que no se aprecian en los gráficos, la mas importante de ellas presenta al implementar realmente el algoritmo dentro de un firewall, en donde la complejidad de calcular un numero aleatorio es mucho mayor a tomar la media de dos valores, haciendo el tiempo de ejecución total aun menor que el algoritmo de hormiga basado en números aleatorios para el calculo de la feromona.

Finalmente, en la tabla 5 se muestran el numero de comparaciones totales y promedio que realizó cada algoritmo después de procesar todos los datos de ejemplo entregados.

**Tabla 5.**

| **Algoritmo** | **Comparaciones totales** | **Promedio de comparaciones** |
|---|---|---|
| Secuencial | 987 | 24,675 |
| Colonia de hormigas | 370 | 9,25 |
| Colonia de hormigas justo | 324 | 8,1 |

## 5 Conclusiones

En este articulo se implementó una optimización utilizando un algoritmo de colonia de hormigas para la búsqueda de reglas de un cortafuegos, comparándola con una búsqueda secuencial de las mismas reglas, los resultados son conclusivos y muestran una clara mejoría en cuanto al numero de comparaciones necesarias para encontrar una regla se refiere, el único caso en el cual el algoritmo secuencial se comporta mejor es en el de mejor caso, que se refiere a que los paquetes que entran se encuentran siempre entre las primeras reglas del cortafuegos.

Se aprecia el comportamiento estable del algoritmo de hormiga en comparación con el secuencial, lo que le permite funcionar y ser competitivo incluso cuando el set de reglas es considerablemente grande (100 ~ 100000 reglas).

La modificación realizada al algoritmo de colonia de hormiga tuvo el

efecto esperado de disminuir las comparaciones en comparación con el propuesto por los autores del documento de referencia [1], esto se debe a que maximiza las comparaciones buscando en la mitad de dos reglas conocidas, y eliminando un poco la aleatoriedad del sistema, el cual de por si ya tiene suficiente considerando que los paquetes que ingresan son bastante aleatorios.

## Referencias

1. N. K. Sreelaja, G. A. Vijayalakshmi Pai, Ant Colony Optimization based approach for efficent packet filtering in firewall, PSG College of Technology, Coimbatore, India, 22 agosto 2009
2. MIT Lincoln Laboratory
http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/