Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías
Departamento de Ciencias Computacionales

Seminario de Solución de Problemas de Redes de Computadoras y Protocolos de Comunicación

Dra. Blanca Lorena Reynoso Gómez

Practica 3

Análisis de paquete ARP/RARP

Equipo 1 - Integrantes:

Castellanos Cerda Adriana Elizabeth

Guevara Del Real David Ernesto

Santana Hernández Leonardo Daniel

Vara Pérez Carlos Adonis

Objetivo:

El objetivo de esta práctica es; convertir las direcciones de protocolo de alto nivel (direcciones IP) a direcciones de red físicas, Intentar hallar la dirección en su caché. Si se encuentra el par buscado, devuelve la correspondiente dirección física de 48 bits al llamador (el manejador de dispositivo). Si no lo encuentra, descarta el paquete (se asume que al ser un protocolo de alto nivel volverá a transmitirlo) y genera un broadcast de red para una solicitud ARP.

Para facilitar este proceso, utilizaremos dos protocolos para la obtención de la dirección MAC de una dirección IP dada (***ARP***) y otro para la obtención de la dirección IP de una dirección MAC (***RARP***)-

• **ARP** Obtención de la dirección MAC de una dirección IP. Al recibir un paquete IP para X.Y.Z.T, el router pregunta cuál es la dirección MAC de esta dirección IP para enviarla por la redlocal.
• **RARP** Obtención de la dirección IP correspondiente a una dirección MAC dada.

El uso de los protocolos ARP y RARP propaga los mensajes a todos los ordenadores conectados en la red local, ya que el destinatario debe identificarse de entre todos los posibles candidatos. Los ordenadores que no responden ignoran estas peticiones.

Código:

```
# @Author: schwarze_falke
# @Date:   2019-01-25T13:30:28-06:00
# @Last modified by:   schwarze_falke
# @Last modified time: 2019-02-18T17:12:33-06:00
from bitstring import BitArray
import codecs
import binascii
import json


# This function gives a string a defined format of the type "00:00:00"
# depending on the lenght [given by @top]. It returns a formatted string
def formatNetString(varString):
    finalString = ''
    for i in varString:
        finalString += i
        finalString += '.'
    return finalString[:-1]


def formatHexString(varString, top):
    finalString = ''
    a = 0
    b = 2
    while b != top:
        finalString += str(varString[a:b])
        a = b
        b += 2
        if(b != top):
            finalString += ':'
    return finalString
```

```
34  def dictionary():
35      JSONfile = open('ip_protocol_numbers.json')
36      JSONstr = JSONfile.read()
37      JSONdata = json.loads(JSONstr)
38      return JSONdata
39
40
41  # Main function
42  if __name__ == '__main__':
43
44      readInput = input('Ingrese el nombre del archivo a leer: ')
45      fileStr = "packages/" + readInput + ".bin"
46      with codecs.open(fileStr, 'rb+') as content_file:
47          file = content_file.read()
48      # All file's data is read an process as hex
49      hexString = binascii.hexlify(file).upper().decode('utf-8')
50
51      # End of testing part
52
53      # The origin address has a lenght of 6 bytes
54      # also the destination address; so, 6x2 = 12
55      # The type information has a lenght of 2 bytes
56      originAddress = formatHexString(hexString[0:12], 14)
57      destinationAddress = formatHexString(hexString[12:24], 14)
58      type = hexString[24:28]
59      print("Direccion MAC de origen: ", originAddress)
60      print("Direccion MAC de destino: ", destinationAddress)
61      if type == '0800':
62          print("Tipo: ", type, " (IPv4)")
63
```

```
63
64          # IP has a lenght of 20 bytes
65          ip = BitArray(hex=hexString[28:]).bin
66          version = ip[0:4]
67          header = ip[4:8]
68          header = int(header, 2)
69          service = ip[8:16]
70          long = int(ip[16:32], 2)
71          id = int(ip[32:48], 2)
72          flags = ip[48:51]
73          posFrag = int(ip[51:64], 2)
74          ttl = int(ip[64:72], 2)
75          protocol = int(ip[72:80], 2)
76          controlHeader = int(ip[80:96], 2)
77          originIP = int(ip[96:128], 2)
78          destinyIP = int(ip[128:160], 2)
79
80          if version == "0100":
81              print("Version: IPv4")
82
83              print("Cabecera: ", (header * 32), " bytes")
84
85              if service[0:3] == "000":
86                  print("Servicio: ", service[0:3], "de rutina")
87              elif service[0:3] == "001":
88                  print("Servicio: ", service[0:3], "Prioritario")
89              elif service[0:3] == "010":
90                  print("Servicio: ", service[0:3], "Inmediato")
91              elif service[0:3] == "011":
92                  print("Servicio: ", service[0:3], "Relampago")
93              elif service[0:3] == "100":
94                  print("Servicio: ", service[0:3], "Invalidacion Relampago")
95              elif service[0:3] == "101":
```

```
95              elif service[0:3] == "101":
96                  print("Servicio: ", service[0:3], "Procesando Llamada critica \
97                  y de emergencia")
98              elif service[0:3] == "110":
99                  print("Servicio: ", service[0:3], "Control de trabajo de \
100                 internet")
101             elif service[0:3] == "111":
102                 print("Servicio: ", service[0:3], "Control de red")
103
104             if service[4] == "0":
105                 print("Retardo: Normal (", service[4], ")")
106             elif service[4] == "1":
107                 print("Retardo: Bajo (", service[4], ")")
108
109             if service[5] == "0":
110                 print("Rendimiento: Normal (", service[5], ")")
111             elif service[5] == "1":
112                 print("Rendimiento: Bajo (", service[5], ")")
113
114             if service[6] == "0":
115                 print("Fiabilidad: Normal (", service[6], ")")
116             elif service[6] == "1":
117                 print("Fiabilidad: Alta (", service[6], ")")
118
119             print("Longitud: ", long)
120             print("Identificador: ", id)
121
122             print("Bandera 1: Reservado ({})".format(flags[0]))
123             if flags[1] == "0":
124                 print("Bandera 2: Divisible ({})".format(flags[1]))
125             elif flags[1] == "1":
126                 print("Bandera 2: No divisible DF ({})".format(flags[1]))
```

```
127
128             if flags[2] == "0":
129                 print("Bandera 3: Ultimo fragmentado ({})".format(flags[2]))
130             elif flags[2] == "1":
131                 print("Bandera 3: Fragmento intermedio ({})".format(flags[2]))
132
133             print("Posicion del fragmento: ", posFrag)
134
135             print("Tiempo de vida (TTL): ", ttl)
136
137             print("Protocolo: {} [{}] ({})".format(
138                   (dictionary()[protocol])['Protocol'],
139                   (dictionary()[protocol])['Keyword'], protocol))
140
141             print("Suma de control de cabecera: ", controlHeader)
142
143             print("Direccion IP de origen: ", formatNetString(str(originIP)))
144             print("Direccion IP de origen: ", formatNetString(str(destinyIP)))
145
146         elif version == "0100":
147             print("Version: IPv6")
148
149     if type == '0806':
150         # IP has a lenght of 20 bytes
151         # TCP's lenght is 23
152         ip = formatHexString(hexString[28:68], 42)
153         tcp = formatHexString(hexString[68:114], 48)
154         data = formatHexString(hexString[114:len(hexString)], (len(hexString)
155                                                                - 112))
156         print("IP: ", ip)
157         print("Tipo: ", type, " (ARP)")
158         print("TCP: ", tcp)
```

```
158             print("TCP: ", tcp)
159             print("Datos: ", data)
160         if type == '8035':
161             # IP has a lenght of 20 bytes
162             # TCP's lenght is 23
163             ip = formatHexString(hexString[28:68], 42)
164             tcp = formatHexString(hexString[68:114], 48)
165             data = formatHexString(hexString[114:len(hexString)], (len(hexString)
166                                                                    - 112))
167             print("Tipo: ", type, " (RARP)")
168             print("IP: ", ip)
169             print("TCP: ", tcp)
170             print("Datos: ", data)
171         if type == '08DD':
172             # IP has a lenght of 20 bytes
173             # TCP's lenght is 23
174             ip = formatHexString(hexString[28:68], 42)
175             tcp = formatHexString(hexString[68:114], 48)
176             data = formatHexString(hexString[114:len(hexString)], (len(hexString)
177                                                                    - 112))
178             print("Tipo: ", type, " (IPv6)")
179             print("IP: ", ip)
180             print("TCP: ", tcp)
181             print("Datos: ", data)
```

Pantallas de Resultado:

```
schwarze_falke@SUPREME:~/dev/WireSniff[master]$ python3 script.py
Ingrese el nombre del archivo a leer: ethernet_ipv4_icmp
Direccion MAC de origen:  1C:BD:B9:B7:44:BE
Direccion MAC de destino:  5C:51:4F:77:C2:7A
Tipo:  0800  (IPv4)
Version: IPv4
Cabecera:  160  bytes
Servicio:  000 de rutina
Retardo: Normal ( 0 )
Rendimiento: Normal ( 0 )
Fiabilidad: Normal ( 0 )
Longitud:  84
Identificador:  21169
Bandera 1: Reservado (0)
Bandera 2: No divisible DF (1)
Bandera 3: Ultimo fragmentado (0)
Posicion del fragmento:  0
Tiempo de vida (TTL):  64
Protocolo: Internet Control Message [ICMP] (1)
Tipo: Echo (solicitud de eco)
Codigo: No se puede llegar a la red
Checksum: 55368
Suma de control de cabecera:  44231
Direccion IP de origen:  2.8.8.7.1.7.4.1.4.6
Direccion IP de origen:  3.4.9.5.8.8.4.4.7.2
```

```
schwarze_falke@SUPREME:~/dev/WireSniff[master]$ python3 script.py
Ingrese el nombre del archivo a leer: ethernet_ipv4_icmp_redirect
Direccion MAC de origen:  5C:61:4E:66:C2:7A
Direccion MAC de destino:  08:00:27:8D:A0:38
Tipo:  0800  (IPv4)
Version: IPv4
Cabecera:  160  bytes
Servicio:  110 Control de trabajo de                internet
Retardo: Normal ( 0 )
Rendimiento: Normal ( 0 )
Fiabilidad: Normal ( 0 )
Longitud:  112
Identificador:  39306
Bandera 1: Reservado (0)
Bandera 2: Divisible (0)
Bandera 3: Ultimo fragmentado (0)
Posicion del fragmento:  0
Tiempo de vida (TTL):  64
Protocolo: Internet Control Message [ICMP] (1)
Tipo: Redirect (redireccionar - cambio de ruta)
Codigo: No se puede llegar al host o aplicación de destino
Checksum: 34537
Suma de control de cabecera:  63406
Direccion IP de origen:  2.8.8.6.9.7.7.5.4.2
Direccion IP de origen:  2.8.8.6.9.7.7.6.3.8
```

Conclusión:

Al concluir con la tercera parte, aprendimos que, si se desea enviar datos a una determinado dirección IP de destino, el mecanismo de encaminamiento IP determina primero la dirección IP del siguiente salto del paquete y el dispositivo hardware al que se debería enviar.

Que existen diferntes procesos para facilitar la obtención de la dirección MAC de una dirección IP dada (***ARP***) y otro para la obtención de la dirección IP de una dirección MAC (***RARP***)

Repositorio del Programa: SchwarzeFalke/WireSniff