Escaneo de IEEE 802.11: análisis de datos experimentales

Prof. Laudin Alessandro Molina

31 de octubre de 2014

1. Objetivos

1. Describir herramientas que fáciliten la gestión y análisis de datos experimentales
2. Analizar gráficamente los datos registrados

2. Introducción

El escaneo activo de redes IEEE 802.11 involucra el intercambio de tramas P_{req} y P_{resp}. La estación (MS) que ejecuta el escaneo transmite en *broadcast* un P_{req} y los puntos de acceso que lo reciben (AP) responden usando tramas P_{resp}, que son transmitidas en *unicast*. La Figura 1 presenta una representación simplificada del escaneo activo.

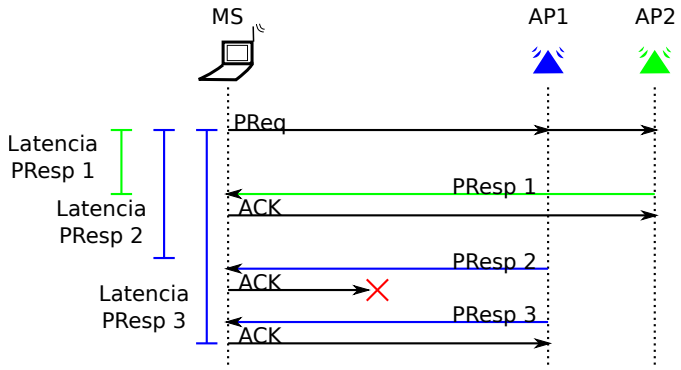

Figura 1: Intercambio P_{req} — P_{resp}

Observe que en el estándar IEEE 802.11 se definen 11 canales para Venezuela, permitiendo que en la misma ubicación se tengan múltiples AP operando en el mismo canal y/o en canales diferentes. En la actualidad las redes IEEE 802.11 son muy populares y existen despliegues masivos de AP, por lo que es común encontrar múltiples AP compartiendo el mismo canal, destacándose los canales 1, 6 y 11 como los canales con mayor cantidad de AP. Esto, sumado con las características propias del espectro de los $2{,}4\,GHz$, genera aumento en la ocupación del medio, incrementando la contención en el acceso y la probabilidad de las colisiones y variaciones en el intercambio de tramas, de manera que el escaneo activo resulta en un proceso no determinista, que además se debe realizar en cada uno de los canales disponibles.

En el estudio de los despliegues de redes IEEE 802.11 resulta interesante conocer las siguientes características:

- Ocupación de los canales
- Tiempo de respuesta de los AP durante el escaneo

- Potencia con que se perciben los AP

En la Figura 2 se presenta un esquema de trabajo para recolectar datos, organizarlos y analizarlos. Este tiene como objetivo un proceso reproducible, razón por la que los datos deben son unificados y almacenados en una base de datos.

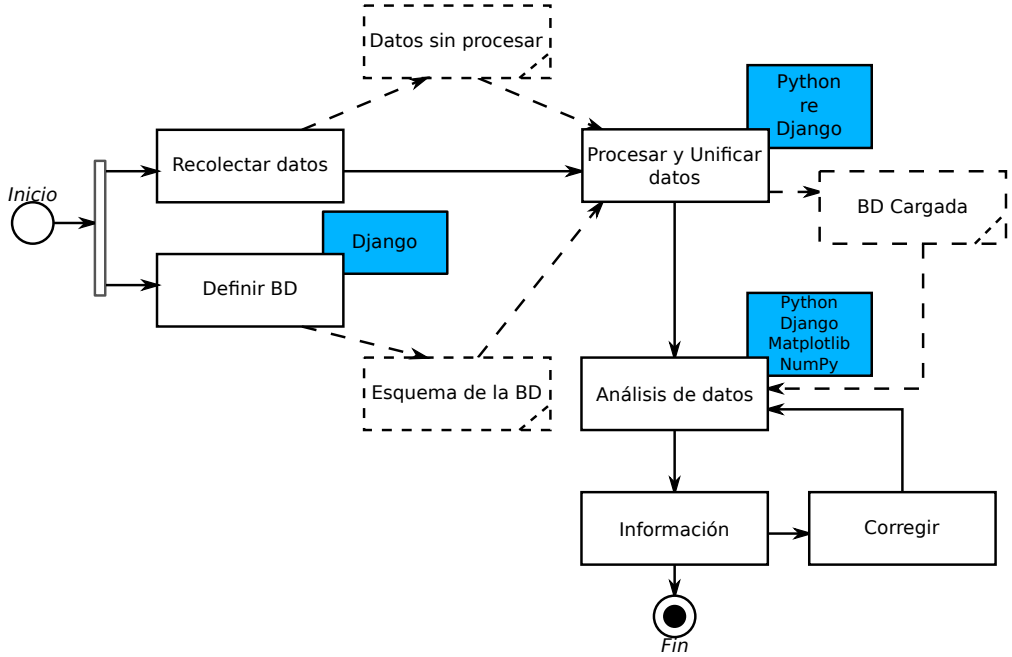

Figura 2: Marco de trabajo

3. Análisis de datos

Para el análisis de datos se utilizaran las siguientes herramientas:

- Python[1] como lenguaje de programación.
- Django[2] para el almacenamiento de los datos. Django es un *framework* diseñado para el desarrollo de sistemas Web, en nuestro caso solo se utilizará el componente que permite tratar como objetos los datos almacenados en la base de datos.
- Numpy[3] para realizar cálculos estadísticos y operaciones de cálculo científico.
- Matplotlib[4] para representar información gráficamente.
- Statmodels[5].

[1] https://www.python.org
[2] http://www.djangoproject.com
[3] http://www.numpy.org
[4] http://matplotlib.org
[5] http://statsmodels.sourceforge.net/

3.1. Instalación y configuración del software

El seminario requiere la instalación de distintos componentes de software en el computador, se supone que se cuenta con computadores instalados con una versión reciente de GNU/Linux Debian[6]. Sin embargo, los ejercicios propuestos hacen uso de herramientas multiplataforma.

```
$ sudo apt-get install python-django python-matplotlib python-numpy ipython python-statsmodels
```

Descargar y copiar el proyecto de Django que a utilizar. Una base de datos, junto con la configuración del proyecto ha sido preparada y está disponible en `www.com`. Descargar y descomprimir.

```
$ wget -c http://webdelprofesor.ula.ve/nucleovigia/laudin/sistemas/wifidb.tgz
$ tar xzf wifidb.zip
```

3.2. Calentamiento

Los datos se encuentran almacenados en el archivo `wifidb.db`. Para acceder a los datos se puede utilizar el shell de Python de la siguiente manera¿

```
$ cd wifidb
$ python manage.py shell

In [1]: from scanning.models import *
In [2]: ProbeResponse.objects.all()
Out[2]: [<ProbeResponse: 40:f4:ec:b2:4c:30 9814.037514>, <ProbeResponse:
30:7e:cb:e7:ad:14 9814.045978>, '...(remaining elements truncated)...']
```

El comando 2 muestra los P_{resp} registrados, la consulta retorna una lista de P_{resp}. Para conocer el número de P_{resp} registrados consultar el tamaño de la lista:

```
In [3]: len(ProbeResponse.objects.all())
Out[3]: 3855
```

También puede contar el número de elementos en la base de datos:

```
In [4]: ProbeResponse.objects.count()
Out[4]: 3855
```

Los P_{resp} son modelados como objetos, de manera que tienen un conjunto de atributos y de métodos, algunos relevantes son:

addr2 BSSID de la red que generó el P_{resp}.

op_channel canal en el que opera el AP que lo originó

nic_channel canal en el que operaba la MS cuando se recibió

signal potencia percibida de la señal en dBm

jiffies marca temporal en jiffies[7], este campo es relativo al `uptime` de la MS al momento en que se recibió el P_{resp}.

ProbeRequest el P_{req} al que pertenece.

[6] http://www.debian.org/

[7] Para el experimento mostrado 1 jiffie = 1 ms

delayj() retardo en jiffies. El retardo es calculado como la diferencia entre la marca de tiempo del P_{resp}y la marca de tiempo del P_{req}.

Mostrar el BSSID de la red que originó el P_{resp} almacenado en la posición cero de la lista:

```
In [5]: presp=ProbeResponse.objects.all()[0]

In [6]: presp.addr2
Out[6]: u'40:f4:ec:b2:4c:30'
```

Mostrar el retardo asociado al P_{resp} almacenado en la posición cero de la lista:

```
In [7]: presp.delayj()
Out[7]: 3
```

Recorrer la lista, mostrando el retardo de cada P_{resp}:

```
In [9]: for presp in ProbeResponse.objects.all():
   ...:     print presp.delayj()
   ...:
```

Filtrar el contenido de la lista para mostrar únicamente los P_{resp} generados en el canal 6:

```
In [10]: ProbeResponse.objects.filter(op_channel=6)
Out[10]: [<ProbeResponse: 40:f4:ec:b2:4c:30 9814.037514>,
<ProbeResponse: 30:7e:cb:e7:ad:14 9814.045978>, '...(remaining elements truncated)...']
```

Contar el número de P_{resp} registrados que fueron generados en los canales 6, 7, 8, 9:

```
In [12]: ProbeResponse.objects.filter(op_channel__gte=6, op_channel__lte=9).count()
Out[12]: 1395
```

Los P_{req} también están modelados como objetos, los atributos y métodos relevantes son:

channel que indica el canal en el que fue transmitido el P_{req}.

jiffies que indica el instante de tiempo en que se transmitió el P_{req}. Al igual que con los P_{resp}, los jiffies son relativos al *uptime* del computador

proberesponse_set que permite el acceso al conjunto de P_{resp} asociados al P_{req}.

first_response() retorna el primer P_{resp} recibido y asociado al P_{req}.

last_response() retorna el último P_{resp} recibido y asociado al P_{req}.

Mostrar el número de P_{resp} asociados al P_{req} almacenado en la posición 1 de la lista de P_{req}:

```
In [15]: ProbeRequest.objects.all()[1].proberesponse_set.all()
Out[15]: [<ProbeResponse: 38:46:08:c8:99:b8 9814.299134>, <ProbeResponse:
40:16:9f:4b:f0:58 9814.30655>, <ProbeResponse: 00:80:48:75:51:30 9814.311064>,
<ProbeResponse: 06:80:48:75:51:30 9814.31274>, <ProbeResponse: 78:92:9c:06:1a:d6
9814.314218>, <ProbeResponse: 34:a8:4e:70:80:d5 9814.341447>]
```

Mostrar el retardo, potencia y canal de operación del primero de los P_{resp} asociados al primero de los P_{req} de la lista:

```
In [16]: presp=ProbeRequest.first_response().proberesponse_set.all()[0]

In [17]: presp.delayj()
Out[17]: 3

In [18]: presp.signal
Out[18]: -69

In [19]: presp.op_channel
Out[19]: 6
```

Mostrar el tiempo de respuesta del primer P_{resp} recibido luego de cada P_{req}:

```
In [24]: for preq in ProbeRequest.objects.all():
   ....:     presp=preq.first_response()
   ....:     if presp != None:
   ....:         print presp.delayj()
```

3.3. Guardar los guiones de análisis

Para reutilizar el trabajo guarde el guión en un archivo ubicado en el directorio "`<WIFIDB/>scanning/management/commands/`". El archivo puede llevar el nombre de su conveniencia. Por ejemplo,

Mostrar el mínimo, el máximo, la media, la mediana y la desviación estándar del retardo del primer P_{resp} recibido luego de cada P_{req}:

```python
# -*- coding: utf-8 -*-

from __future__ import print_function

import numpy as np

from optparse import make_option

from django.core.management.base import BaseCommand, CommandError

from scanning.models import ProbeRequest
from scanning.models import ProbeResponse

def first_delay():
    delays = []
    for preq in ProbeRequest.objects.all():
        presp = preq.first_response()
        if presp != None:
            delays.append(presp.delayj())

    print("Min:", np.min(delays))
    print("Max:", np.max(delays))
    print("Mean:", np.mean(delays))
    print("Mean:", np.median(delays))

class Command(BaseCommand):
    def handle(self, *args, **options):
```

```
        first_delay()
```

Guardar el código en `<WIFIDB>/scanning/management/commands/first_delay.py`, para ejecutarlo:

```
$ python manage.py first_delay
```

Mostrar la función de distribución acumulada del primer P_{resp} recibido luego de cada P_{req}:

```
# -*- coding: utf-8 -*-

from __future__ import print_function

import numpy as np
import statsmodels.api as sm
import pylab as plt

from optparse import make_option

from django.core.management.base import BaseCommand, CommandError

from scanning.models import ProbeRequest
from scanning.models import ProbeResponse

def first_delay():
    delays = []
    for preq in ProbeRequest.objects.all():
        presp = preq.first_response()
        if presp != None:
            delays.append(presp.delayj())

    plt.figure()
    ecdf = sm.distributions.ECDF(delays)
    x = np.linspace(min(delays), max(delays))
    y = ecdf(x)
    plt.step(x, y, label='Primera respuesta')
    plt.legend(loc='lower right')
    plt.ylabel('CDF')
    plt.xlabel('Tiempo de respuesta (ms)')
    plt.grid(True)
    fig_name = 'cdf_first_delays.pdf'
    plt.savefig(fig_name, format='pdf')
    plt.close()


class Command(BaseCommand):
    def handle(self, *args, **options):
        first_delay()
```

Guardar el código en `<WIFIDB>/scanning/management/commands/cdf_first_delay.py`, para ejecutarlo:

```
$ python manage.py cdf_first_delay
```

Se genera una figura en el archivo `cdf_first_delays.pdf`, que debe ser similar al mostrado en la Figura 3.

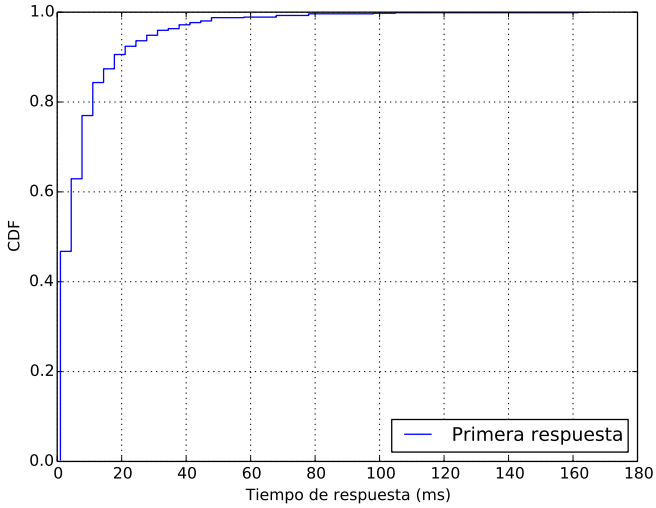

Figura 3: CDF empírica del retardo del primer P_{resp} asociado a cada P_{req}

3.4. Ejercicios

1. Preparar un guión que premita graficar la CDF correspondiente al retardo del último P_{resp} asociado a cada P_{req}.

2. Indique qué porcentaje correspondiente a cada un de los tiempos de retardo indicados en la tabla

	Porcentaje	
Rango del retardo	**Primero**	**Último**
$retardo < 10\,ms$		
$retardo < 40\,ms$		
$retardo > 100\,ms$		

Cuadro 1: Porcetaje de la muestra correspondiente a cada rango de retardos

3. Preparar un guión que muestre la cantidad de AP operando en cada canal. Utilice un diagrama de barras para mostrar el resultado. El diagrama de barras es creado con la siguiente instrucción:

   ```
   plt.bar(x, freq)
   ```

   Donde `x` es una lista con los valores que toma la variable. En este caso será una lista con 11 valores. El parámetro `freq` es una lista que almacena la frecuencia de cada uno de los valores de `x`.

4. Indicar cuales son los 3 canales más poblados, completar la Tabla 2, indique qué porcentaje de los AP opera en los 3 canales más poblados.

4. Más información

Para mayor información en relación al escaneo de redes IEEE 802.11, así como el estudio de despliegues espontáneos consulte los siguientes trabajos:

Canal	Porcentaje

Cuadro 2: Población de AP en los 3 canales más poblados

- **Caracterización de redes 802.11: caso de estudio del área metropolitana de la ciudad de Mérida**. Disponible en: `http://www.saber.ula.ve/dspace/handle/123456789/39101`
- **Estudio del descubrimiento de topologías espontáneas en redes 802.11**. Disponible en: `http://www.saber.ula.ve/dspace/handle/123456789/39045`