Instituto Politécnico Nacional
Escuela Superior de Cómputo
Academia de Ingeniería de Software

Práctica 2

Integrantes del equipo:

Castro Flores Marcela
Sánchez Cruz Rosa María
Santiago Mancera Arturo Samuel

M. en C. Tanibet Pérez de los Santos Mondragón

México, Ciudad de México a 03 de noviembre de 2018

Índice general

1. Introducción **4**

2. Algoritmos implementados **5**
2.1. Algoritmo de Línea de Base 5
2.2. Algoritmo de Mínimos Cuadrados 5
2.3. Algoritmo de Holt Winters 5
2.3.1. Explicación de código 11

3. Conclusiones **16**

Índice de figuras

2.1. Nuevo agente añadido. 6
2.2. Pantalla principal de agentes. 6
2.3. Información del agente activo. 6
2.4. Ventana de gráficos disponibles. 6
2.5. Gráfica de OutNUCastPkts. 7
2.6. Gráfica de OutNUCastPkts. 7
2.7. Gráfica de InOctets. 7
2.8. Correo de inicio del error. 8
2.9. Correo de fin del error. 9
2.10. Correo de inicio del error. 10
2.11. Correo de fin del error. 10
2.12. Método actualizarHW. 11
2.13. Método crearHW. 11
2.14. Método actualizarHW. 12
2.15. Método graficar_HW. 13
2.16. Método check_aberration. 14
2.17. Método send_alert_attached. 15

CAPÍTULO 1

Introducción

Para la realización de esta práctica se utilizaró nuevamente el protocolo SNMP, sin embargo esta vez se implementaron 3 algoritmos que serán explicados posteriormente.
Dichos algoritmos son:

- Línea de base
- Mínimos cuadrados
- Holt Winters

Esta práctica se dividió en las tres partes siguientes:

1. EXPLIQUEN RAPIDO SU PARTE
2. EXPLIQUEN RAPIDO SU PARTE
3. Por último, la tercera parte correspondió al algoritmo de Holt Winters el cual se encargaba de identificar dentro de una gráfica no lineal, si los valores medidos salían de cierto rango de medición tanto superior como inferior y cuando esto sucedía, se notificaba por medio de un correo electrónico al administrador de la red.

En el capítulo mostrado a continuación se observa el desarrollo de la práctica.

CAPÍTULO 2

Algoritmos implementados

2.1. Algoritmo de Línea de Base

2.2. Algoritmo de Mínimos Cuadrados

2.3. Algoritmo de Holt Winters

El método Holt-Winters es un método de pronóstico de triple exponente suavizante y tiene la ventaja de ser fácil de adaptarse a medida que nueva información real está disponible. El método Holt- Winters es una extensión del método Holt que considera solo dos exponentes suavizantes. Holt-Winters considera nivel, tendencia y estacional de una determinada serie de tiempos.

El método de Holt-Winters es básicamente un procedimiento de suavizamiento exponencial. Este tipo de procedimientos facilitan los cálculos y reducen los requerimientos de almacenamiento en las bases de datos, lo cual cobra importancia cuando se están prediciendo muchas series de tiempo [1].

Existen tres fases de trabajo, con tres conjuntos de datos diferentes.

- Un primer grupo de datos es para inicializar el modelo, esto es determinar los indicadores de nivel, tendencia y estacionalidad.
- Un segundo conjunto de datos es necesario para probar o calibrar los índices de suavización Alfa, Beta y Gamma.
- Un tercer grupo de datos para pronosticar y evaluar el funcionamiento del modelo propuesto.

Ejecutar todas las fases en un solo grupo de datos puede conducir a tratar de encajar en exceso el modelo a los datos disponibles [2].

Una vez que ya se ha explicado a grandes rasgos el funcionamiento del algoritmo de Holt Winters, se explicará paso a paso el procedimiento de la aplicación de dicho algoritmo.

Primero se añadió el agente desde el cual se obtuvieron los datos en tiempo real tal y como se muestra en la figura 2.1.

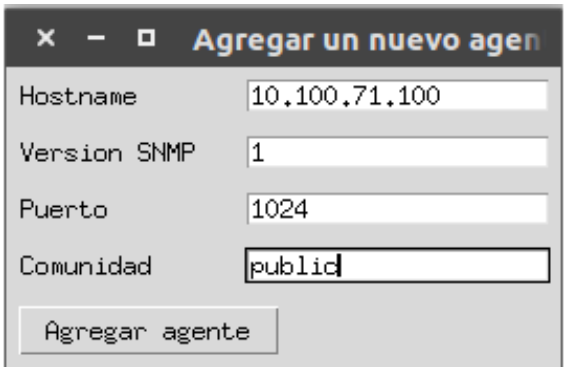

Figura 2.1: Nuevo agente añadido.

Y de los agentes añadidos, se mostró la información en la pantalla principal como se observa en la figura 2.2. En esta imagen no se observa que el agente se encuentre activo debido a que su ejecución se realizó directamente en el laboratorio, sin embargo, en la figura 2.3 se muestran los datos del host 10.100.71.100 cuando este se encontraba activo.

Figura 2.2: Pantalla principal de agentes.

Figura 2.3: Información del agente activo.

Una vez que el agente fue añadido, en los botones que se encuentran a lado derecho se pulsó sobre el botón con la leyenda **Graficos**, mismo que desplegaba una ventana con las opciones siguientes (figura 2.4):

Figura 2.4: Ventana de gráficos disponibles.

Al ser el equipo 10, nos fue asignado el monitoreo del OID de los **OutNUCastPkts**, por tal motivo se

presionó sobre el primer botón y al realizar dicha acción se comenzó el monitoreo de los paquetes mostrando una gráfica similar a la de la figura 2.5.

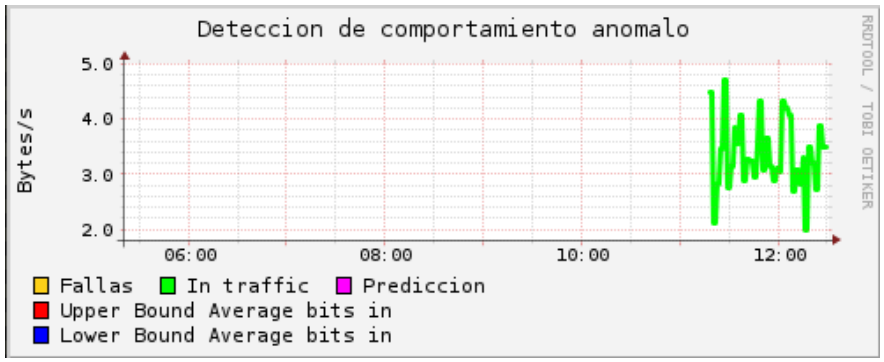

Figura 2.5: Gráfica de OutNUCastPkts.

Misma que posteriormente y conforme fueron surgiendo los distintos fallos se mostró como las gráficas de a continuación:

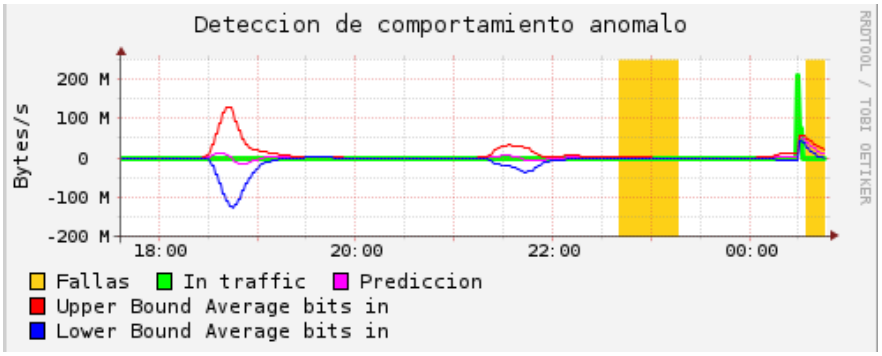

Figura 2.6: Gráfica de OutNUCastPkts.

Sin embargo, también se realizaron mediciones con otros paquetes tal y como la gráfica de la figura 2.7 en la cual se plasman los datos obtenidos con el OID de InOctets.

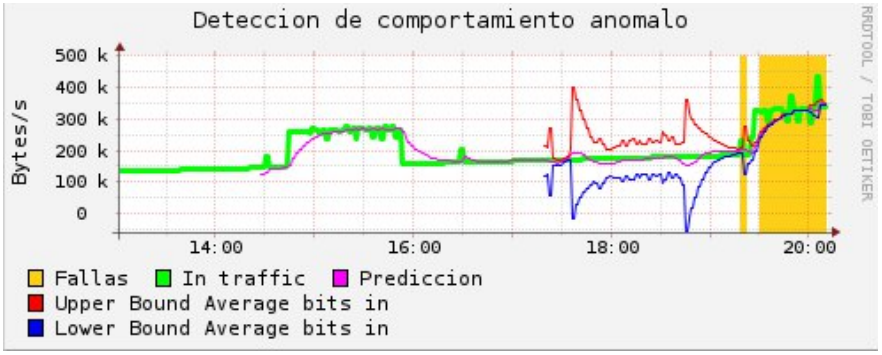

Figura 2.7: Gráfica de InOctets.

Y como se observa en las gráficas mostradas anteriormente, una línea amarilla vertical marca las secciones en las cuales se ha sobrepasado el límite inferior o superior. Cuando dicho error sucede, se envían

dos correos electrónicos al administrador indicando tanto el inicio del error como el final del mismo y adjuntando en dicho correo las gráficas como se puede observar en las figuras a continuación:

Respecto a los datos de entrada de OutNUCastPkts entre los correos recibidos se observan los de las figuras 2.8 y 2.9.

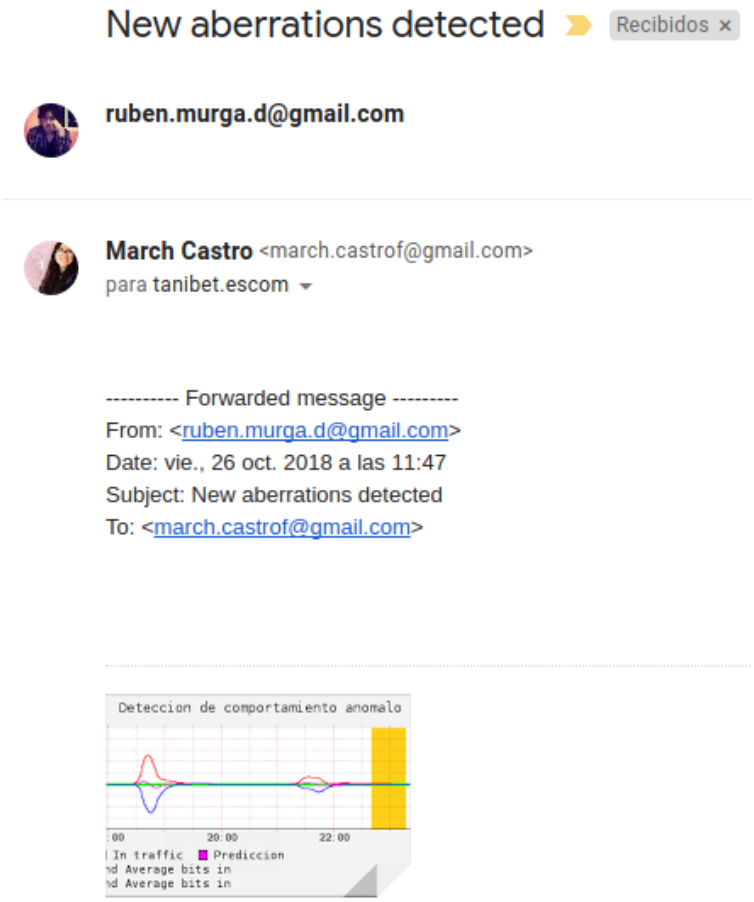

Figura 2.8: Correo de inicio del error.

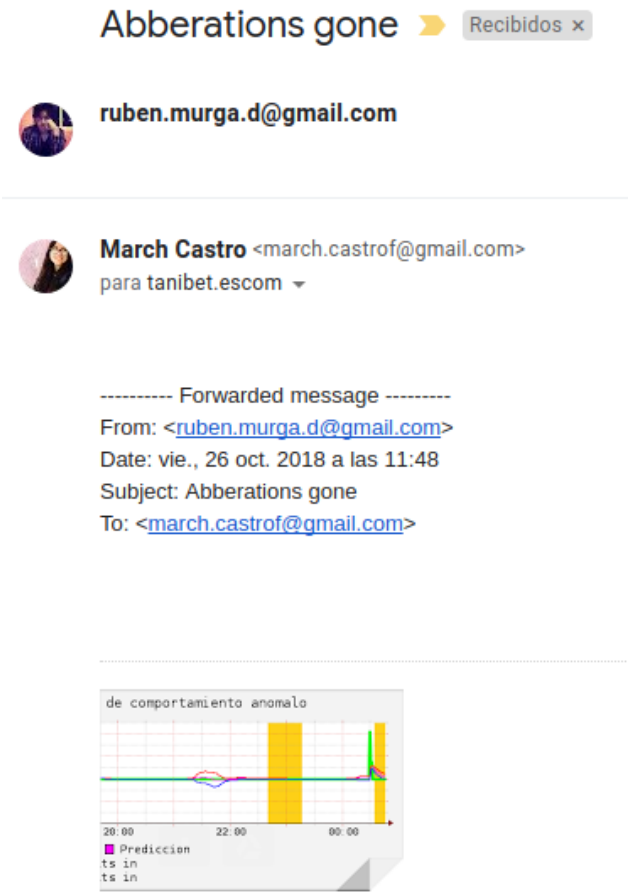

Figura 2.9: Correo de fin del error.

Por otro lado, respecto a los datos de entrada de InOctets entre los correos recibidos se observan los de las figuras 2.10 y 2.11.

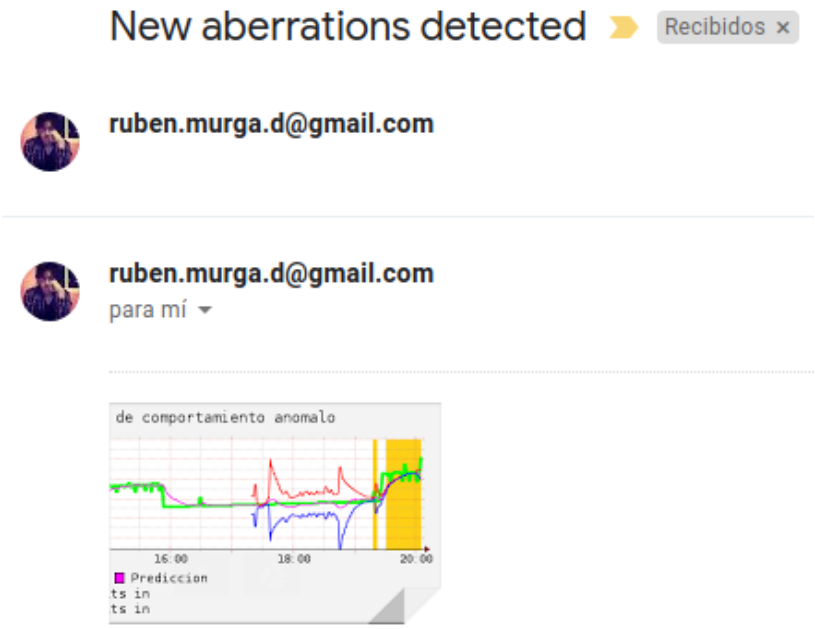

Figura 2.10: Correo de inicio del error.

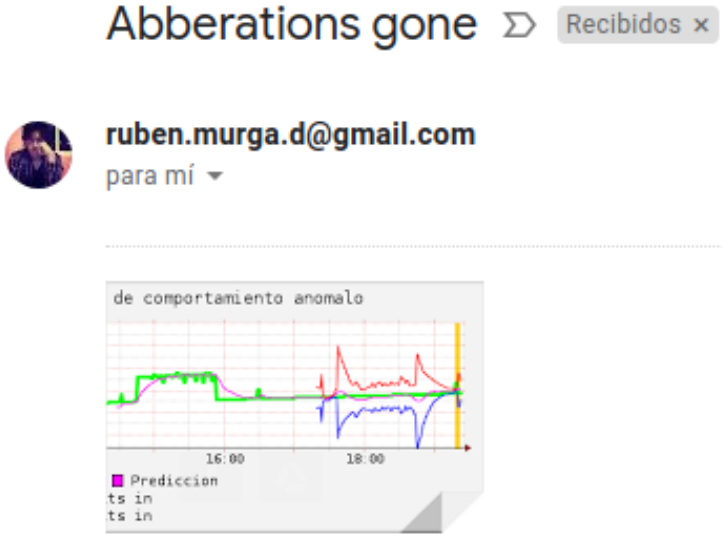

Figura 2.11: Correo de fin del error.

2.3.1. Explicación de código

En esta sección se explicarán las partes del código más importante. En el método **actualizarHW** mostrado en la figura 2.12, lo primero que se realiza es al momento de obtener por parámetros los datos del host al cual se desea gráficar, se verifica si existe ya un archivo .rrd en el cual ya se almacenen los datos, en caso contrario se manda a llamar al método **crearHW**.

```
def actualizarHW(cadena,comunidad,host,puerto,rrd):
    print 'Entro actualizarHW ',cadena,comunidad,host,puerto,rrd

    total_input_traffic = 0
    total_output_traffic = 0
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    fname=rrd+".rrd"
    pngfname=rrd+".png"
    #Verifica que exista una rrd asociada al host, en caso de no e
    archivo_rrd = Path(rrdpath+fname)
    if archivo_rrd.is_file() == False:
        crearRRD.crearHW(rrdpath+fname)
        print "rrd HW Creada... en",rrdpath,fname
    else:
        print "Abriendo rrd HW..."
```

Figura 2.12: Método actualizarHW.

El método **crearHW**, mostrado en la figura 2.13 se encarga de crear el archivo rrd en el cual se almacenarán los datos indicados que en este caso se refieren a los datos de entrada de OutNUCastPkts.

```
def crearHW(nombre):
    print 'NOMBREEE '+nombre

    ret = rrdtool.create(nombre,
                         "--start",'N',
                         "--step",'60',
                         "DS:outucastpkts:COUNTER:600:U:U",
                         "RRA:AVERAGE:0.5:1:1209",
                         "RRA:HWPREDICT:600:0.9:0.0035:172:3",
                         "RRA:SEASONAL:172:0.9:2",
                         "RRA:DEVSEASONAL:172:0.9:2",
                         "RRA:DEVPREDICT:600:4",
                         "RRA:FAILURES:172:7:9:4")
    if ret:
        print rrdtool.error()
```

Figura 2.13: Método crearHW.

Una vez que se ha creado la rrd, volvemos al códgo completo del método actualizarHW mostrado en la figura 2.14, en el cual se obtienen los datos de entrada utilizando el protocolo SNMP con el cual se indica el OID del cual se desea obtener dichos datos y con los cuales se actualiza el archivo rrd.

```
def actualizarHW(cadena,comunidad,host,puerto,rrd):
    print 'Entro actualizarHW ',cadena,comunidad,host,puerto,rrd

    total_input_traffic = 0
    total_output_traffic = 0
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    fname=rrd+".rrd"
    pngfname=rrd+".png"
    #Verifica que exista una rrd asociada al host, en caso de no existir crea una rrd nueva
    archivo_rrd = Path(rrdpath+fname)
    if archivo_rrd.is_file() == False:
        crearRRD.crearHW(rrdpath+fname)
        print "rrd HW Creada... en",rrdpath,fname
    else:
        print "Abriendo rrd HW..."

    endDate = rrdtool.last(rrdpath+fname) #ultimo valor del XML
    begDate = endDate - 3600
    #Inicia proceso de adquisicion de datos HW
    while 1:
        total_input_traffic = int(consultaSNMP('public','10.100.71.100',1024,'1.3.6.1.2.1.2.2.1.18.1'))
        total_output_traffic = int(consultaSNMP(comunidad,host,puerto,'1.3.6.1.2.1.2.2.1.16.3'))
        valor = str(rrdtool.last(rrdpath+fname)+30)+":" + str(total_input_traffic)
        print 'Valor: ',valor
        rrdtool.update(rrdpath+fname,valor)
        rrdtool.dump(rrdpath+fname,rrd+'.xml')
        #print "actualizar: ",rrdpath,fname, rrd
        rrdtool.tune(rrdpath+fname,'--alpha','0.1')
        #time.sleep(1)
    if ret:
        print rrdtool.error()
        time.sleep(300)
```

Figura 2.14: Método actualizarHW.

Posteriormente, la figura 2.15 muestra el código utilizado para realizar la graficación de los valores que se estan leyendo desde el archivo rrd. En este método se indican que valores se desean gráficar, los colores y lo que significará cada una de las etiquetas mostradas en la parte inferior de la gráfica.

```
def graficar_HW(cadena,rrd,image_name,id_grafica):
    print "GRAFICAR ",rrd,image_name
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    title="Deteccion de comportamiento anomalo"
    var = 0
    if id_grafica == 1:
        while 1:
            ret = rrdtool.graph(pngpath+image_name,
                                '--start', str(rrdtool.last(rrdpath+rrd)-25800),
                                #'--start', str(rrdtool.last(fname)-51600),
                                '--end', str(rrdtool.last(rrdpath+rrd)),
                                '--title=' + title,
                                "--vertical-label=OutUCastPkts/s",
                                '--slope-mode',
                                "DEF:obs="      + rrdpath+rrd + ":outucastpkts:AVERAGE",
                                "DEF:pred="     + rrdpath+rrd + ":outucastpkts:HWPREDICT",
                                "DEF:dev="      + rrdpath+rrd + ":outucastpkts:DEVPREDICT",
                                "DEF:fail="     + rrdpath+rrd + ":outucastpkts:FAILURES",

                                "CDEF:scaledobs=obs,8,*",
                                "CDEF:upper=pred,dev,2,*,+",
                                "CDEF:lower=pred,dev,2,*,-",
                                "CDEF:scaledupper=upper,8,*",
                                "CDEF:scaledlower=lower,8,*",
                                "CDEF:scaledpred=pred,8,*",

                                "TICK:fail#FDD017:1.0:  Fallas",
                                "LINE3:scaledobs#00FF00:In traffic",
                                "LINE1:scaledpred#FF00FF:Prediccion\\n",
                                #"LINE1:outoctets#0000FF:Out traffic",
                                "LINE1:scaledupper#ff0000:Upper Bound Average bits in\\n",
                                "LINE1:scaledlower#0000FF:Lower Bound Average bits in")

            time.sleep(1)
            returned_value = check_aberration(rrdpath,rrd)
            print returned_value

            if var == returned_value:
                pass
            else:
                if var == 0 and returned_value == 1 or var == 2 and returned_value == 1:
                    send_alert_attached('New aberrations detected',pngpath+image_name)
                    var = returned_value
                elif var == 1 and returned_value == 2:
                    send_alert_attached('Abberations gone',pngpath+image_name)
                    var = returned_value
```

Figura 2.15: Método graficar_HW.

Por último, las figuras 2.16 y 2.17 son las encargadas de revisar los datos en búsqueda de un cambio de valores en los fallos y en caso de encontrarlo se realiza el envío del correo electrónico. El método check_aberration se encarga de verificar los datos que se generan en los fallos mismos que solo toman un

valor de 0 o 1, en caso de que el valor haya cambiado de 0 a 1, significa que un fallo ha comenzado, si el valor cambia de 1 a 0, significará que el fallo ha finalizado.

```
def check_aberration(rrdpath, fname):
    """ This will check for begin and end of aberration
        in file. Will return:
        0 if aberration not found.
        1 if aberration begins
        2 if aberration ends
    """
    ab_status = 0
    rrdfilename = rrdpath + fname
    print rrdfilename
    info = rrdtool.info(rrdfilename)

    rrdstep = int(info['step'])
    print 'STEP',rrdstep
    lastupdate = info['last_update']
    print 'LASTUPD ', int(lastupdate)
    previosupdate = str(lastupdate - rrdstep - 1)
    graphtmpfile = tempfile.NamedTemporaryFile()

    try:
        values = rrdtool.graph(graphtmpfile.name+'F',
                               '--start', str(previosupdate),
                               '--end', str(lastupdate),
                           'DEF:f0=' + rrdfilename + ':outucastpkts:FAILURES',
                           'PRINT:f0:LAST:%1.0lf')
        print values

        if str(values[2][0]) == '-nan':
            print 'ERROR'
        else:
            flast = int(values[2][0])
            if (flast == 1):
                ab_status = 1
            else:
                ab_status = 2
        return ab_status
    except:
        pass
```

Figura 2.16: Método check_aberration.

Por otro lado, el método se encarga de enviar el correo correspondiente con el titular necesario para indicar si el fallo comenzó o finalizó y anexando la imagen de la gráfica en el punto en el cual comenzó o termino dicho fallo.

```
def send_alert_attached(subject,file):
    """ Will send e-mail, attaching png
    files in the flist.
    """
    msg = MIMEMultipart()
    msg['Subject'] = subject
    msg['From'] = mailsender
    msg['To'] = mailreceip
    #for file in flist:
    png_file = file
    print png_file
    fp = open(png_file, 'rb')
    img = MIMEImage(fp.read())
    fp.close()
    msg.attach(img)

    mserver = smtplib.SMTP('smtp.gmail.com',587)
    mserver.ehlo()
    mserver.starttls()
    mserver.ehlo()
    mserver.login(mailsender, gmail_password)
    mserver.sendmail(mailsender, mailreceip, msg.as_string())
    mserver.close()
```

Figura 2.17: Método send_alert_attached.

CAPÍTULO 3

Conclusiones

aaa

Referencias y bibliografías

[1] Juan C. Hernández M., *Método Holt Winters (2017). Disponible en: http://rstudio-pubs-static.s3.amazonaws.com/283175_1d0898ed1b704812a4eeb29b1fdcb213.html [Consultado el 01 Nov. 2018].*

[2] Omar Maguiña G., *El Método de Pronóstico de Holt Winters (2016). Disponible en: https://administration21.files.wordpress.com/2017/01/pronc3b3sticos-holt-winters-omr-nov2016.pdf[Consultado el 01 Nov. 2018].*