15-744 Computer Networks (Fall 2004)
Homework 0 - Solutions

1. **Ethernet**

   This is a recap on the basics of Ethernet. If your undergrad class did not cover this read the corresponding chapter in the Kurose/Ross book or the Peterson/Davie book.

   Question 1: Suppose Ethernet was the only existing LAN technology, so every host in the Internet was part of a local Ethernet and therefore had a globally unique Ethernet address. Would you now recommend to get rid of IP addresses by simply using Ethernet addresses instead of IP addresses? What about the other way around, why do we not simply assign network adapters IP addresses instead of Ethernet addresses so we don't have to deal with both, IP and Ethernet addresses?

   Answer: We still want to keep IP addresses because they are assigned in a hierarchical fashion which allows hierarchical routing. Ethernet addresses don't have a structure that can be used for hierarchical routing.

   We also don't want to replace Ethernet addresses by IP addresses, since that would mean hardcoding the IP address in a machine. Therefore, we couldn't do things like dynamically reassigning IP addresses (DHCP) and it would also create problems if a host moves to a new a new location.

   Question 2: Suppose you set up an 10 Mbps Ethernet with a maximum packet size of 1500 bytes and a diameter of 2.5 km. You measure the worst case RTT in your Ethernet, and find it to be 51 usec. Also, the resistance of the wire is 50 ohms.
   You have heard that when using Ethernet you should make sure that all packets have a certain minimum size. You want to avoid unnecessary overhead (stuffing packets to reach the minimum packet size) by choosing a small minimum packet size. How would you choose the minimum packet size?

   Answer: The idea here is that you want to transmit long enough so that you detect a collision. In the worst case, a host on the other end of the LAN start transmitting just when your first bit reached the other end of the network. In this case it will take 2 RTTs after starting the transmission to detect this collision, i.e. we need to be transmitting for at least 1 RTT = 51 usec to detect it. Hence, we have to choose the packet size such that
   packet_size/10 Mbps $\geq$ 51 usec.
   Therefore, we choose the packet size to be at least 510 bits or 64 bytes.

2. **Practical Tools**

   In this section, you will learn a couple of practical tools: **route**, **arp**, **ifconfig**, and **traceroute**. Below is a very brief description of what they do. For more detailed information, check the man pages.

   route The `route` command can be used to view and manipulate the IP routing table.

   arp Because there are both network-layer addresses (IP-addresses) and link-layer addresses (i.e.LAN addresses), there is a need to translate between them. For the Internet, this is the job of the address resolution protocol (ARP). The command `arp` allows you to view the *ARP table* which contains the mapping of IP addresses to LAN addresses on the local network.

ifconfig The `ifconfig` command is used to configure a network interface and to display the status of the currently active interfaces.

traceroute `traceroute` is a very useful tool that prints the route packets take to a given net-work host.

Question 3: Run the `route` command. What is the use of the entry with netmask 0.0.0.0?

Answer: The entry with netmask 0.0.0.0 gives the default entry in the routing table. This means that if none of the other entries match a particular packet, it is routed according to the default entry.

Question 4: Suppose you ran the command "`route -delete default`" on a cs.cmu.edu machine and then did the following:

ping www.cs.cmu.edu
ping www.apple.com
ping www.cmu.edu
ping www.psc.edu

Which pings do you expect to get through and which not? Why? What command would you have to type to create routing connectivity to only one of the hosts that are unreachable? What command do you need to use to restore connectivity to all hosts?

Answer: The pings to www.cs.cmu.edu and www.cmu.edu get through and the pings to www.psc.edu and www.apple.com do not.

The reason is that without the default entry we are only able to send data to hosts which are in the same network as we are. Since our netmask is 255.255.0.0, any host with and address of 128.2.x.y will be in the same network as we are and we should be able to communicate with them.

To establish connection to just one of the two sites (say www.apple.com) we can add a specific route with
`route add -host www.apple.com gw 128.2.254.36`
To restore connectivity to all hosts we reenter the default entry into our routing table:
`route add default gw 128.2.254.36`

Question 5: Run `arp` on your machine. Is there an entry in the ARP table for every host on the same LAN? Find out how entries enter and leave the ARP table and give a brief description. How can you use this information to determine the Ethernet address of a host on the same LAN that is currently not present in the ARP table of your machine?

Answer: The arp table contains hardware addresses that are either preconfigured or hardware addresses of hosts that were recently accessed from our machine.

When we need to reach a local host (within our subnet), we need to have its hardware address. If there is no entry for it in the arp table we broadcast an ARP request, which says "who has IP address a.b.c.d"? All hosts examine the broadcast packet and the host that recognizes the address as its own will send an ARP reply back to us. The information is then cached in the ARP table. Entries that are created this way automatically expire after some time and are removed from the table.

To create an ARP table entry for a given host, we have to force our machine to send out an ARP request for this host. One way to do so is to ping this machine.

Please use a machine that is part of the CMU network to answer the following questions.

Update: Questions 6 and 7 would completely make sense only if you use a wired CS machine. Since some of you may not have access to machines on the CS network, feel free to use the following output from gs19.sp.cs.cmu.edu to answer questions 6 and 7.

```
[bovik@gs19 ~]$ /sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 00:02:B3:4C:71:3E
          inet addr:128.2.203.28  Bcast:128.2.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:108839056 errors:0 dropped:0 overruns:4182 frame:0
          TX packets:38475380 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:10 Base address:0xe000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:30990669 errors:0 dropped:0 overruns:0 frame:0
          TX packets:30990669 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

[bovik@gs19 ~]$ arp
Address HWtype HWaddress     Flags Mask   Iface
OLIVE.SRV.CS.CMU.EDU    ether   08:00:20:21:93:1C   C                     eth0
PEACH.SRV.CS.CMU.EDU    ether   08:00:20:A7:1A:88   C                     eth0
PAPAYA.SRV.CS.CMU.EDU   ether   08:00:20:0D:1C:7C   C                     eth0
TRETIAK.STAMPEDE.CS.CMU ether   00:30:48:23:83:C4   C                     eth0
GIGROUTER.NET.CS.CMU.ED ether   00:B0:8E:83:DF:50   C                     eth0
UNIX46.andrew.cmu.edu   ether   00:B0:8E:83:DF:50   C                     eth0
PLUM.SRV.CS.CMU.EDU     ether   08:00:20:21:77:33   C                     eth0
CUCUMBER.SRV.CS.CMU.EDU ether   08:00:20:0C:0F:4A   C                     eth0
MANGO.SRV.CS.CMU.EDU    ether   08:00:20:A6:BA:C3   C                     eth0
BLUEBERRY.SRV.CS.CMU.ED ether   08:00:20:A6:BA:2B   C                     eth0
FIG.SRV.CS.CMU.EDU      ether   08:00:20:7D:5F:BD   C                     eth0
BANANA.SRV.CS.CMU.EDU   ether   08:00:20:A7:19:73   C                     eth0
BLACK.ECE.CMU.EDU       ether   00:B0:8E:83:DF:50   C                     eth0
[bovik@gs19 ~]$ /usr/local/sbin/traceroute black.ece
traceroute to black.ece.cmu.edu (128.2.129.113), 30 hops max, 40 byte packets
 1  GIGROUTER.NET.CS.CMU.EDU (128.2.254.36)  2.174 ms  0.524 ms  0.443 ms
 2  ECE-VL1000.GW.CMU.NET (128.2.0.128)  0.751 ms  0.465 ms  0.533 ms
 3  BLACK.ECE.CMU.EDU (128.2.129.113)  0.429 ms  1.710 ms  1.580 ms
[bovik@gs19 ~]$
```

Question 6: Run ifconfig. What information does the field "Mask" give you? If you look at the value of this field you should see that 16 bits in the mask are set. Does this mean that all CMU machines are part of the same LAN (since CMU is a class B network)? (start with a guess and wait with your final answer until after the next question).

Answer: The mask field gives you the netmask of the subnet that your machine is on. This mask is used to determine if a given destination is on the same network as your machine or whether it can be reached only through a gateway.

If you are on a CS machine the netmask is 255.255.0.0 which means that your machine considers all

CS machines to be on the same Ethernet. However, you will see in the next question that this is just the view of your machine and not necessarily what the network really looks like.

Question 7: Next try to verify your guess of whether all CMU machines are on the same LAN or not in the following two ways:
1. If all CMU machines are on the same LAN you should be able to use the arp trick from above to find out the Ethernet address of any CMU machine. Try to do this for black.ece.cmu.edu. What do you find?
2. If all CMU machines are on the same LAN they should be reachable from your machine in one hop (without passing through intermediate hops). Use a traceroute to black.ece.cmu.edu to verify this. What do you see?
How do you explain what you observe in 1. and 2.? (Hint: Look at the Ethernet addresses of all the machines involved.)

Note: If your machine is an ece.cmu.edu machine, replace black.ece.cmu.edu by gs19.sp.cs.cmu.edu in the above question.

Answer: If you are on a CS machine you will find that you can create an ARP table entry for black.ece.cmu.edu. This indicates that black.ece.cmu.edu is on the same Ethernet as your machine. However, if you to a traceroute to you find that there is more than one hop on the path to black.ece.cmu.edu which contradicts the initial guess.

A closer look at the ARP table reveals that the hardware address of black.ece.cmu.edu is the same as that of the gateway (the first hop on the path to black.ece.cmu.edu). What is happening is that the gateway is doing a proxy-arp for black.ece.cmu.edu: whenever there is an ARP request for black.ece.cmu.edu the gateway will actually reply with its own hardware address. When it finally receives packets destined to black.ece.cmu.edu it will forward them.

Question 8: Instead of traceroute, you can also use ping to find the route that packets take to a given network host. It does so by using the IP Record Route option. Try the following two commands:

ping -R www.cmu.edu
traceroute www.cmu.edu

What is the difference in their output? Now try the following:

ping -R www.yahoo.com

Do you get the desired output? Explain your observation.

Answer: The difference in the output is that ping -R includes the reverse path while traceroute only includes the path from our system to the other system. If you look at the output carefully you will see that the traceroute output is similar to the output of ping -R on the reverse path (i.e. the routers ping -R records when returning from www.cmu.edu to your machine). This is because ping -R records the outgoing interface of all routers it traverses whereas traceroute shows the incoming interface of all routers.

There are two reasons why you might not get the desired output when using ping -R.

First, ping -R uses the IP record route option and is thus limited by the maximum size of the IP header. Since the maximum size of the IP header (with options) is 60 bytes, ping -R can at most be used to show 9 hops (as the record route option takes 3 bytes, leaving 37 bytes in the options filed for storing IP addresses; each IP address is 4 bytes). As such ping -R is useful only for tracing routes to destinations close by. Another reason why ping -R might not be successful is that some routers might not support the record route option.

3. **Link-layer flags and bit-stuffing**

Question 11 (and its solution) and the description of `bit-stuffing` that follows are due to Hari Balakrishnan.

Consider the problem of using a special sequence of bits, or a *flag* , to indicate the start (or equivalently, end) of a frame at the link-layer. A typical link-layer flag is the string 01^60 (where 1^6 indicates a sequence of six 1's). Simply appending this string to the end of each frame, however, is not enough to indicate frame boundaries, since the original frame may actually contain the sequence 01^60 in its payload. To guard against this problem when transmitting the frame, the sender will insert a 0 in to the data string after the appearance of five successive 1's; this is called *bit-stuffing.* On the receive side, the receiving link-layer will delete the first 0 after each string of five consecutive 1's and will only detect that the frame is finished if it sees the string the string 01^60.

Question 9: What is the worst case overhead of this scheme? (The answer must be in terms of n, the number of bits sent).

Answer: [5 pts] The worst case occurs when the payload contains a stream of 1s. In this case, the sender needs to stuff a 0 after every 5 1s. Thus, the overhead is $\lfloor \frac{n}{5} \rfloor$ bits.

Question 10: Assume for now that the marker is 0^8, and the payload is such that if a byte-by-byte scanning is done, there would be at least 1 zero byte (0^8) in any 255 consecutive bytes of the payload. Also, the payload starts and ends with a zero byte. Given this, is it possible to come up with a encoding/decoding scheme which has no overhead (the encoded payload has the same length as the original payload)?

Answer: It needs to be ensured that zero byte (0^8) does not appear in the payload. The solution is to divide the bits into chunks such that each chunk begins with a zero byte, but does not contain a zero byte thereafter. We know that such a division is possible, and the length of each chunk, n, excluding the zero byte would be such that $0 \leq n \leq 254$. So, instead of the default zero byte at the beginning of each chunk, we can put $(n+1)$ in its binary representation (Note, n would not work, because we cannot have zero byte in the encoded payload).

For decoding, we read the first byte (typically called the code byte) of the chunk, replace it with a zero byte, and skip the next $(n-1)$ bytes. Now, we are guarantteed to be at the beginning of the next chunk, or just before the zero byte (end marker).

Extra credit Can you think of some other clever scheme which has a lower overhead than the bit-stuffing scheme? (Hint: think of how the solution to the previous question can be extended to the general case).

Answer: The above encoding scheme needs to be extended for the case in which there are no zero bytes in a sequence of 255 bytes. So, we ned an escape route. For this, we can decrease the chunk size by 1 byte, and use the value $0xFF$ to indicate that there is no zero byte following after this chunk of 254 bytes. Thus, the overhead of this scheme is at most $\frac{1}{254}$. This scheme was presented as COBS, in SIGCOMM '97.

Small description of COBS: A scheme which requires fewer bits to be stuffed is **Consistent Overhead Byte Stuffing** (Cheshire and Baker, `SIGCOMM '97`) which has a worst case overhead of $\frac{1}{254}$. Let the marker is a zero byte (0^8). It needs to be ensured that 0^8 does not appear in the payload. COBS takes its input data and encodes it as a series of variable length code blocks. Each code block begins with a single code byte, followed by zero or more data bytes. A code byte value of n indicates that the next $n-1$ bytes are non-zeroes, followed by an implicit byte that is 0^8. When there is no zero byte in the next 254 bytes, the code value is 0xFF. Thus, the worst case overhead in this case is $\frac{1}{254}$ (since there is no overhead when there is a zero byte - the implicit zero byte saves a byte).

Question 11: Bit stuffing must avoid the appearance of 01^6 within the transmitted frame. Assume that we will always convert this string to 01^501. Consider the string 01^50, however, which the receiver would never mistake for the flag 01^6, even before stuffing. We might be tempted to avoid the extra stuffing overhead in this case leaving this string as is and using a more sophisticated destuffing rule at the receiver. Show that, contrary to this intuition, the sender must always stuff a 0 after 01^5 in order for the receiver to be able to correctly destuff the string.

Answer: There are two cases to consider:

(a) 01^50^x. If the original string is 01^50^x, then it *is* possible to avoid stuffing a bit, because the unstuffed version of this string could never be mistaken for the string 01^60. Because our flag contains a 1, however, every string of 0s in the frame will always be terminated by a 1. This case could **never** arise.

(b) 01^50^x1. If the original string is 01^50^x1, then we must stuff a bit. This follows from a bit of formal inductive reasoning. We start with the string 01^501. If we don't stuff a bit, then it will be impossible for the receiver to disambiguate this string from the string 01^601, which we know must be stuffed. Therefore, we must stuff the string 01^501. Assume now, that the string $01^50^{x-1}01$ must be stuffed. Then the string 01^50^x01 must also be stuffed, otherwise we would not be able to disambiguate it, form the string $01^50^{x-1}01$. It follows from induction then that all strings of the form 01^50^x1 must therefore be stuffed.