

Week 6- Lab Exercise – IPv4

Objective

To learn about the details of IP (Internet Protocol). IP is the network layer protocol used throughout the Internet. We will examine IP version 4, since it is ubiquitously deployed, while the IP version 6 is partly deployed ! ☺.

traceroute / tracert: This lab uses “traceroute” to find the router level path from your computer to a remote Internet host. `traceroute` is a standard command-line utility for discovering the Internet paths that your computer uses. It is widely used for network troubleshooting. It comes pre-installed on Windows and Mac, and can be installed using your package manager on Linux. On Windows, it is called “tracert”. It has various options, but simply issuing the command “`traceroute www.uwa.edu.au`” will cause your computer to find and print the path to the remote computer (here `www.uwa.edu.au`).

Step 1: Capture a Trace (This part below is highlighted in yellow as we have seen the commands below in earlier labs; so, if you intend to use the provided trace file, you may skip it)

Proceed as follows to capture a trace assuming that your computer has IPv4 connectivity; alternatively, you may use a supplied trace. The trace we want to gather is a simple web fetch from a remote server, which will cause your computer to send and receive IP packets, followed by a `traceroute` to the remote server to find the path it uses over the Internet.

1. *Pick a URL at a remote server, e.g., <http://www.uwa.edu.au/> and check that you can fetch the contents with `wget` or `curl`, e.g., “`wget http://www.uwa.edu.au/`” or “`curl http://www.uwa.edu.au/`”. This will fetch the resource and either write it to a file (`wget`) or to the screen (`curl`). With `wget`, you want a single response with status code “200 OK”. If the fetch does not work then try a different URL; keep in mind that you may be referring to a URL by a shortcut for which browsers must do work to find the intended content, e.g., <http://mit.edu> may really be <http://web.mit.edu/index.html>. If no URLs seem to work then debug your use of `wget/curl` or your Internet connectivity.*
2. *Perform a `traceroute` to the same remote server to check that you can discover information about the network path.* On Windows, type, e.g., “`tracert www.uwa.edu.au`”. On Linux / Mac, type, e.g., “`traceroute www.uwa.edu.au`”. If you are on Linux / Mac and behind a NAT (as most home users or virtual machine users) then use the `-I` option (that was a capital i) to `traceroute`, e.g., “`traceroute -I www.uwa.edu.au`”. This will cause `traceroute` to send ICMP probes like `tracert` instead of its usual UDP probes; ICMP probes are better able to pass through NAT boxes. A successful example is shown below; save the output as you will need it for later steps. Note that `traceroute` may take up to a minute to run. Each line shows information about the next IP hop from the computer running `traceroute` towards the target destination. The lines with “*”s indicate that there was no response from the network to identify that segment of the Internet path. Some unidentified segments are to be expected.

However, if `tracert` is not working correctly then nearly all the path will be “*”s. In this case, try a different remote server, experiment with `tracert`, or use the supplied traces.

```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

Z:\>tracert www.uwa.edu.au

Tracing route to www.uwa.edu.au [130.95.128.140]
over a maximum of 30 hops:
  0  1 ms  5 ms  1 ms  acar-atg-02-vlan75.cac.washington.edu [128.208.2.102]
  1  <1 ms  <1 ms  <1 ms  vl3855.uwcr-atg-01.infra.washington.edu [205.175.109.21]
  2  <1 ms  <1 ms  <1 ms  vl1889.uwcr-atg-01.infra.washington.edu [205.175.102.157]
  3  <1 ms  <1 ms  <1 ms  vl1850.uwbr-kne-01.infra.washington.edu [205.175.102.2]
  4  1 ms  <1 ms  <1 ms  ae0--4011.iccr-sttlwa01-03.infra.pnw-gigapop.net [209.124.190.134]
  5  *  *  *  Request timed out.
  6  51 ms  52 ms  51 ms  so-1-0-0.bb1.a.hnl.aarnet.net.au [202.158.194.109]
  7  145 ms  145 ms  145 ms  so-2-1-0.bb1.a.syd.aarnet.net.au [202.158.194.105]
  8  157 ms  157 ms  157 ms  so-2-0-0.bb1.a.mel.aarnet.net.au [202.158.194.33]
  9  166 ms  166 ms  166 ms  so-2-0-0.bb1.a.adl.aarnet.net.au [202.158.194.17]
 10  194 ms  194 ms  194 ms  so-0-1-0.bb1.a.per.aarnet.net.au [202.158.194.5]
 11  194 ms  195 ms  194 ms  tengigabitethernet1-4.er2.uwa.cpe.aarnet.net.au [202.158.198.10]
 12  194 ms  195 ms  194 ms  gw1.er2.uwa.cpe.aarnet.net.au [113.197.9.118]
 13  *  *  *  Request timed out.
 14  *  *  *  Request timed out.
 15  *  *  *  Request timed out.
 16  195 ms  195 ms  195 ms  matrix.uwa.edu.au [130.95.128.140]

Trace complete.
Z:\>
```

Figure 1: Running traceroute (as `tracert` on Windows)

3. Launch Wireshark and start a capture with a filter of “`tcp port 80`”. Make sure to check “enable network name resolution”. We use this filter to record only standard web traffic. Name resolution will translate the IP addresses of the computers sending and receiving packets into names. It will help you to recognize whether the packets are going to or from your computer. Your capture window should be similar to the one pictured below, other than our highlighting. Select the interface from which to capture as the main wired or wireless interface used by your computer to connect to the Internet. If unsure, guess and revisit this step later if your capture is not successful. Uncheck “capture packets in promiscuous mode”. This mode is useful to overhear packets sent to/from other computers on broadcast networks. We only want to record packets sent to/from your computer. Leave other options at their default values. The capture filter, if present, is used to prevent the capture of other traffic your computer may send or receive. On Wireshark 1.8, the capture filter box is present directly on the options screen, but on Wireshark 1.9, you set a capture filter by double-clicking on the interface.

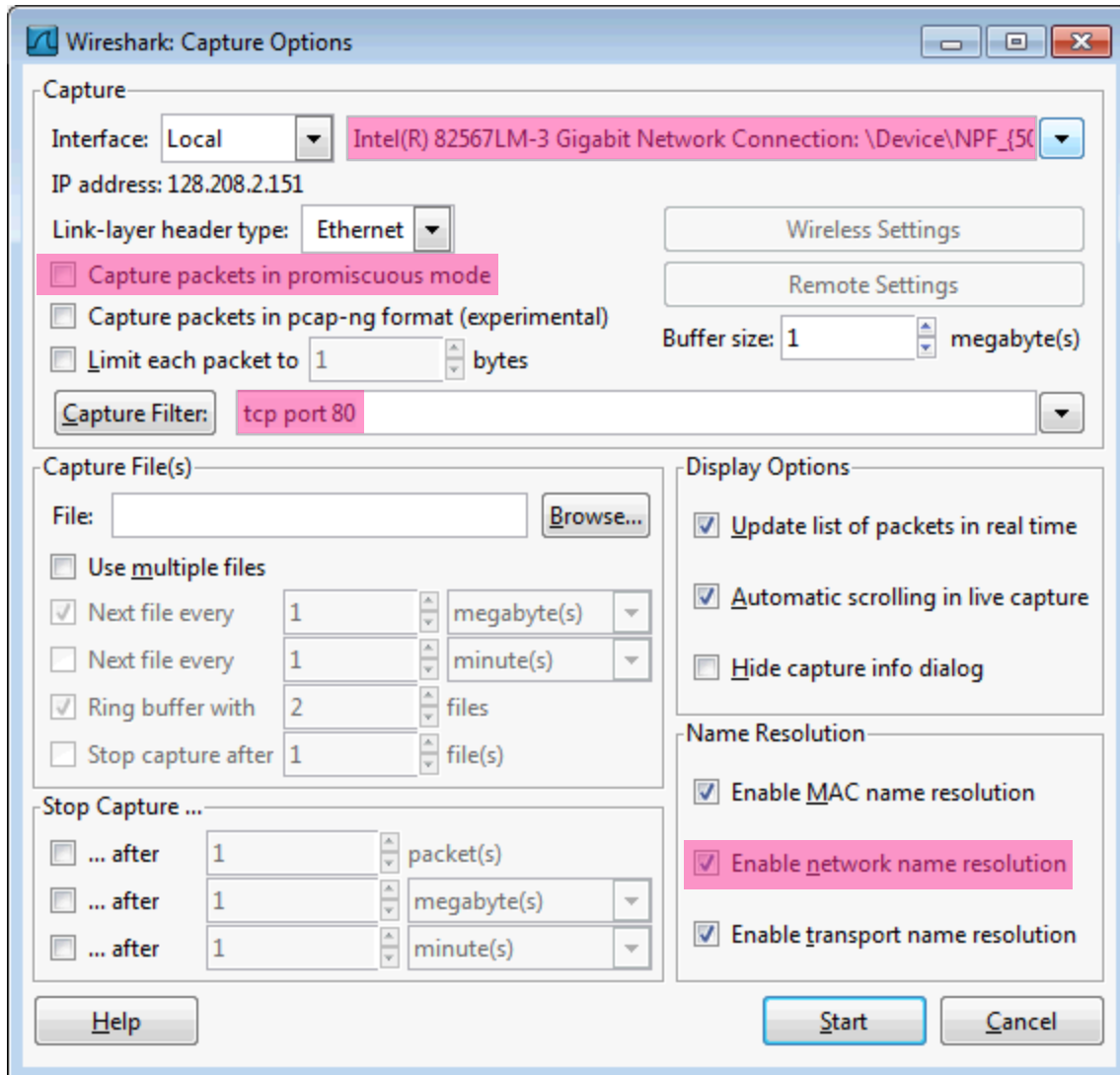


Figure 2: Setting up the capture options

4. After the capture is started, repeat the `wget/curl` command above. This time, the packets will also be recorded by Wireshark.
5. After the command is complete, return to Wireshark and stop the trace. You should now have a short trace similar to that shown in the figure below, along with the output of a traceroute you ran earlier to the corresponding server.

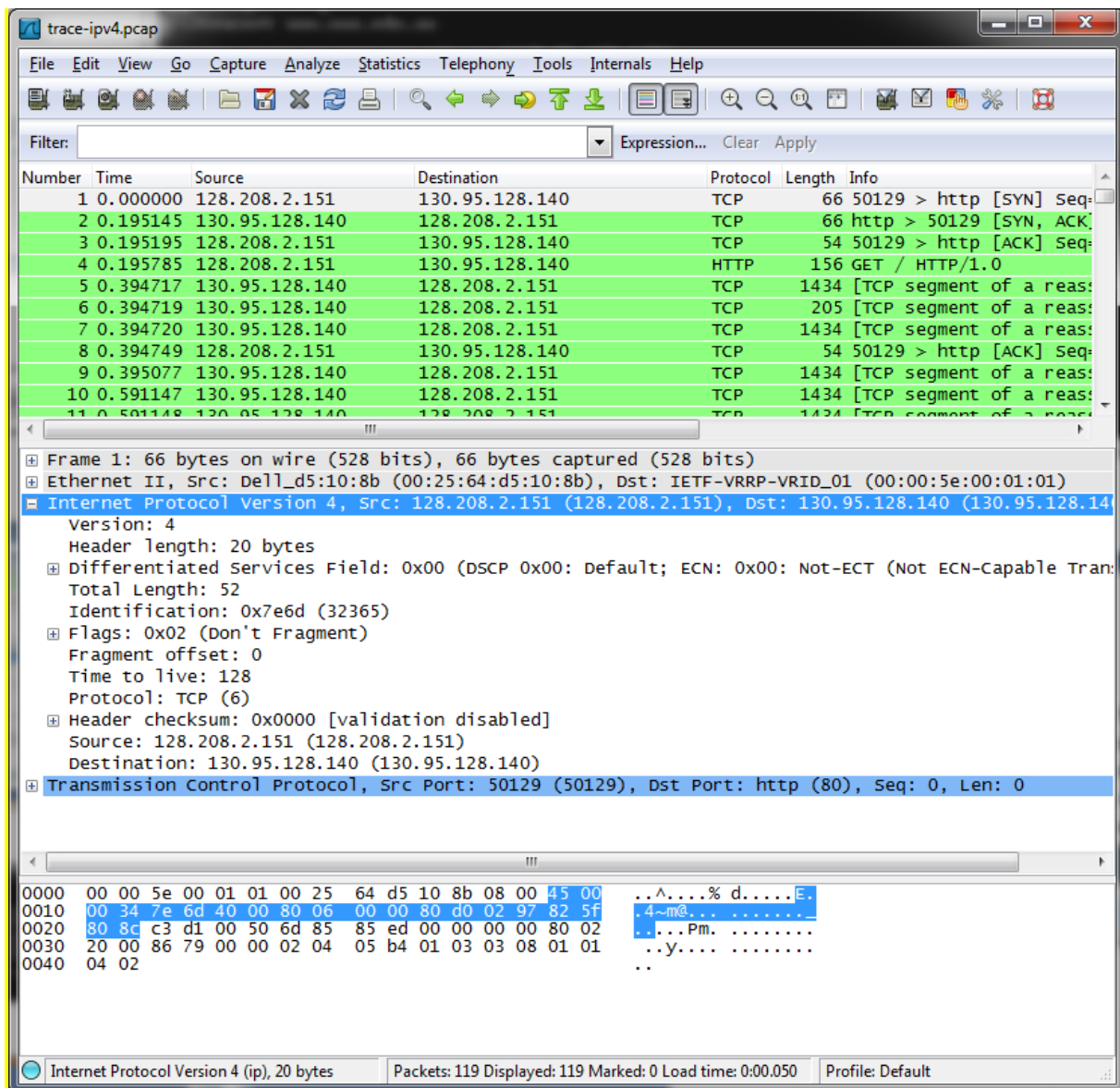


Figure 3: Trace of wget/curl traffic showing the details of the IP header

Step 2: Inspect the Trace

Select any packet in the trace and expand the IP header fields (using the “+” expander or icon) to see the details. You can simply click on a packet to select it (in the top panel). You will see details of its structure (in the middle panel) and the bytes that make up the packet (in the bottom panel). Our interest is the IP header, and you may ignore the other higher and lower layer protocols. When you click on parts of the IP header, you will see the bytes that correspond to the part highlighted in the bottom panel. We have expanded the IP header and clicked on all the IP header fields in the figure above.

Let us go over the fields in turn:

- The version field is set to 4. This is “IPv4” after all.

- Then there is the header length field. Observe by looking at the bytes selected in the packet data that version and header length are both packed into a single byte.
- The Differentiated Services field contains bit flags to indicate whether the packet should be handled with quality of service and congestion indications at routers.
- Then there is the Total Length field.
- Next is the Identification field, which is used for grouping fragments, when a large IP packet is sent as multiple smaller pieces called fragments. It is followed by the Flags and the Fragment offset fields, which also relate to fragmentation. Observe they share bytes.
- Then there is the Time to live or TTL field, followed by the Protocol field.
- Next comes the header checksum. Is your header checksum carrying 0 and flagged as incorrect for IP packets sent from your computer to the remote server? On some computers, the operating system software leaves the header checksum blank (zero) for the NIC to compute and fill in as the packet is sent. This is called protocol offloading. It happens after Wireshark sees the packet, which causes Wireshark to believe that the checksum is wrong and flag it with a different color to signal a problem. A similar issue may happen for the TCP checksum. You can remove these false errors if they are occurring by telling Wireshark not to validate the checksums. Select “Preferences” from the Wireshark menus and expand the “Protocols” area. Look under the list until you come to IPv4. Uncheck “Validate checksum if possible”. Similarly, you may uncheck checksum validation for TCP if applicable to your case.
- The last fields in the header are the normally the source and destination address. It is possible for there to be IP options, but these are unlikely in standard web traffic.
- The IP header is followed by the IP payload. This makes up the rest of the packet, starting with the next higher layer header, TCP in our case, but not including any link layer trailer (e.g., Ethernet padding).

Step 3: IP Packet Structure

To show your understanding of IP, sketch a figure of an IP packet you studied. It should show the position and size in bytes of the IP header fields as you can observe using Wireshark. Since you cannot easily determine sub-byte sizes, group any IP fields that are packed into the same bytes. Your figure can simply show the frame as a long, thin rectangle. Try not to look at the figure of an IPv4 packet in your text; check it afterwards to note and investigate any differences.

To work out sizes, observe that when you click on a protocol block in the middle panel (the block itself, not the “+” expander) Wireshark will highlight the corresponding bytes in the packet in the lower panel, and display the length at the bottom of the window. You may also use the overall packet size shown in the Length column or Frame detail block. Note that this method will not tell you sub-byte positions.

By looking at the IP packets in your trace, answer these questions:

1. *What are the IP addresses of your computer and the remote server?*
2. *Does the Total Length field include the IP header plus IP payload, or just the IP payload?*
3. *How does the value of the Identification field change or stay the same for different packets? For instance, does it hold the same value for all packets in a TCP connection or does it differ for each packet? Is it the same in both directions? Can you see any pattern if the value does change?*

4. *What is the initial value of the TTL field for packets sent from your computer? Is it the maximum possible value, or some lower value?*
5. *How can you tell from looking at a packet that it has not been fragmented?* Most often IP packets in normal operation are not fragmented. But the receiver must have a way to be sure.
6. *What is the length of the IP Header and how is this encoded in the header length field?* Hint: notice that only 4 bits are used for this field, as the version takes up the other 4 bits of the byte.

Step 4: Internet Paths

The source and destination IP addresses in an IP packet denote the endpoints of an Internet path, not the IP routers on the network path the packet travels from the source to the destination. `tracert` is a utility for discovering this path. It works by eliciting responses (ICMP TTL Exceeded messages) from the router 1 hop away from the source towards the destination, then 2 hops away from the source, then 3 hops, and so forth until the destination is reached. The responses will identify the IP address of the router. The output from `tracert` normally prints the information for one hop per line, including the measured round trip times and IP address and DNS names of the router. The DNS name is handy for working out the organization to which the router belongs. Since `tracert` takes advantage of common router implementations, there is no guarantee that it will work for all routers along the path, and it is usual to see “*” responses when it fails for some portions of the path.

Using the `tracert` output, sketch a drawing of the network path. If you are using the supplied trace, note that we have provided the corresponding `tracert` output as follows:

```

Z:\>tracert www.uwa.edu.au
Tracing route to www.uwa.edu.au [130.95.128.140]
over a maximum of 30 hops:
  1      1 ms      1 ms      1 ms  acar-atg-02-
vlan75.cac.washington.edu [128.208.2.102]
  2     <1 ms     <1 ms     <1 ms  vl3855.uwcr-atg-
01.infra.washington.edu [205.175.109.21]
  3     <1 ms     <1 ms     <1 ms  vl1889.uwcr-atg-
01.infra.washington.edu [205.175.102.157]
  4     <1 ms     <1 ms     <1 ms  vl1850.uwbr-kne-
01.infra.washington.edu [205.175.102.2]
  5      1 ms     <1 ms     <1 ms  ae0--4011.iccr-
sttlwa01-03.infra.pnw-gigapop.net [209.124.190.134]
  6      *        *        *      Request timed out.
  7     51 ms     51 ms     51 ms  so-1-0-
0.bb1.a.hnl.aarnet.net.au [202.158.194.109]
  8    145 ms    145 ms    145 ms  so-2-1-
0.bb1.a.syd.aarnet.net.au [202.158.194.105]
  9    157 ms    157 ms    157 ms  so-2-0-
0.bb1.a.mel.aarnet.net.au [202.158.194.33]
 10    166 ms    166 ms    166 ms  so-2-0-
0.bb1.a.adl.aarnet.net.au [202.158.194.17]
 11    193 ms    193 ms    194 ms  so-0-1-
0.bb1.a.per.aarnet.net.au [202.158.194.5]
 12    194 ms    194 ms    195 ms  tengigabitethernet1-
4.er2.uwa.cpe.aarnet.net.au [202.158.198.10]
 13    194 ms    194 ms    194 ms
gw1.er2.uwa.cpe.aarnet.net.au [113.197.9.118]
 14      *        *        *      Request timed out.
 15      *        *        *      Request timed out.
 16      *        *        *      Request timed out.
 17    195 ms    195 ms    195 ms  matrix.uwa.edu.au
[130.95.128.140]
Trace complete.

```

Figure 4: Output of `tracert` showing the router-level path

Show your computer (lefthand side) and the remote server (righthand side), both with IP addresses, as well as the routers along the path between them numbered by their distance on hops from the start of the path. You can find the IP address of your computer and the remote server on the packets in the trace that you captured. The output of `traceroute` will tell you the hop number for each router.

To finish your drawing, label the routers along the path with the name of the real-world organization to which they belong. To do this, you will need to interpret the domain names of the routers given by `traceroute`. If you are unsure, label the routers with the domain name of what you take to be the organization. Ignore or leave blank any routers for which there is no domain name (or no IP address).

This is not an exact science, so we will give some examples. Suppose that `traceroute` identifies a router along the path by the domain name `arouter.cac.washington.edu`. Normally, we can ignore at least the first part of the name, since it identifies different computers in the same organization and not different organizations. Thus we can ignore at least “`arouter`” in the domain name. For generic top-level domains, like “.com” and “.edu”, the last two domains give the domain name of the organization. So for our example, it is “`washington.edu`”. To translate this domain name into the real-world name of an organization, we might search for it on the web. You will quickly find that `washington.edu` is the University of Washington. This means that “`cac`” portion is an internal structure in the University of Washington, and not important for the organization name. You would write “University of Washington” on your figure for any routers with domain names of the form `*.washington.edu`.

Alternatively, consider a router with a domain name like `arouter.syd.aarnet.net.au`. Again, we ignore at least the “`arouter`” part as indicating a computer within a specific organization. For country-code top-level domains like “.au” (for Australia) the last three domains in the name will normally give the organization. In this case the organization’s domain name is `aarnet.net.au`. Using a web search, we find this domain represents AARNET, Australia’s research and education network. The “`syd`” portion is internal structure, and a good guess is that it means the router is located in the Sydney part of AARNET. So for all routers with domain names of the form `*.aarnet.net.au`, you would write “AARNET” on your figure. While there are no guarantees, you should be able to reason similarly and at least give the domain name of the organizations near the ends of the path.

Step 5: IP Header Checksum

We will now look at the IP header checksum calculation by validating a packet. The checksum algorithm adds the header bytes 16 bits at a time. It is computed so that re-computing the sum across the entire IP header (including the checksum value) will produce the result of zero. A complicating factor for us is that this is done using 1s complement arithmetic, rather than 2s complement arithmetic that is normally used for computing. The steps below explain how to perform the necessary computation.

From the trace, pick a packet sent from the remote server to your computer and check that you have a non-zero value in the checksum field. The checksum value sent over the network will be non-zero, so if you have a zero value it is because of the capture setup. Try a packet that has an IP header of 20 bytes, the minimum header size when there are no options, to make this exercise easier.

Follow these steps to check that the checksum value is correct:

1. Divide the header into 10 two byte (16 bit) words. Each word will be 4 hexadecimal digits shown in the packet data panel in the bottom of the Wireshark window, e.g., 05 8c
2. Add these 10 words using regular addition. You may add them with a hexadecimal calculator (Google to find one), or convert them to decimal, add them, and convert them back to hexadecimal. Do whatever is easiest.
3. To compute the 1s complement sum from your addition so far, take any leading digits (beyond the 4 digits of the word size) and add them back to the remainder. For example: 5a432 will become a432 + 5 = a437.
4. The end result should be 0xffff. This is actually zero in 1s complement form, or more precisely 0xffff is -0 (negative zero) while 0x0000 is +0 (positive zero).

If you cannot get your sum to come out and are sure that the checksum must be wrong, you can get Wireshark to check it. See whether it says “[correct]” already. If it does not then use the menus to go to Preferences, expand Protocols, choose IPv4 from the list, and check “validate header checksum”. Now Wireshark will check the checksum and tell you if it is correct.

[END]