Helps

Ethernet frames

Your code will need to parse Ethernet frames from the "wire" as `bytes` instances. The frame that you will be receiving looks like this:

Destination MAC Address	Source MAC Address	EtherType	Payload
(48 bit)	**(48 bit)**	**(16 bit)**	**(variable)**

Note that a complete Ethernet frame also has fields for preamble and Cyclic Redundancy Check (CRC) when it actually travels on the wire. However, these frames are read from a raw socket (i.e., of type `SOCK_RAW`, as opposed to `SOCK_STREAM` or `SOCK_DGRAM`), and those two fields are stripped before it is passed to the application.
In the case of 802.1Q, the frame will look like this:

Destination MAC Address	Source MAC Address	802.1Q Header	EtherType	Payload
(48 bit)	**(48 bit)**	**(32 bit)**	**(16 bit)**	**(variable)**

The most signficant (left-most) 16 bits of the 802.1Q header should have the value 0x8100 to indicate that it is an 802.1Q frame. The least significant (right-most) 12 bits of the 802.1Q header should contain the value of the VLAN ID. The 4 bits in between can be left as zero.

Note that there are libraries, including scapy, for parsing Ethernet frames and higher-level packets, but you may not use them for the lab.

Working with Bytes instances

A `bytes` object in Python is a sequence of arbitrary byte values. For example:

```
>>> bytes1 = b'\x01\x02\x03\x04\x0a\x0b\x0c\x0d'
>>> bytes2 = b'\x05\x06\x07\x08\x09'
```

The `b` prefix is always used with `bytes` objects to distinguish them from `str` (string) objects, which have no such prefix. In the above example, `bytes1` and `bytes2` are assigned values from `bytes` literals.
The `\x` notation indicates that the next two characters are hexadecimal values containing the actual value of the byte.
A "slice" of `bytes` objects produces a new `bytes` object with only the designated sequence. For example, the following gets only the first two bytes of `bytes1` (i.e., indexes 0 and 1):

```
>>> bytes1[:2]
b'\x01\x02'
```

Likewise, the following gets only the fifth and sixth bytes of `bytes1` (i.e., indexes 4 and 5):

```
>>> bytes1[4:6]
b'\n\x0b'
```

`bytes` objects can be concatenated together to yield a new `bytes` object:

```
>>> bytes1 + bytes2
b'\x01\x02\x03\x04\n\x0b\x0c\r\x05\x06\x07\x08\t'
```

Or even:

```
>>> bytes1[:2] + bytes1[4:6]
b'\x01\x02\n\x0b'
```

Printing out the representation of `bytes` objects can be a bit confusing. For example:

```
>>> bytes1[4:6]
b'\n\x0b'
```

In this case, `\n` is the ASCII equivalent of hexadecimal 0xa (i.e., `\x0a)` or decimal 10. To be "helpful", Python prints out the ASCII equivalent.
To print out everything as hexademical, use the `binascii` module:

```
>>> import binascii
>>> binascii.hexlify(bytes1[4:6])
b'0a0b'
```

Note that the result is still a `bytes` object. To convert to `str,` use the `decode` method:

```
>>> binascii.hexlify(bytes1[4:6]).decode('latin1')
'0a0b'
```

Finally, to convert `bytes` objects to integers, use the `struct` module. For example, to put the first two bytes from a `bytes` sequence (`bytes1`) into a short (two-byte) integer:

```
>>> import struct
>>> short1, = struct.unpack('!H', bytes1[:2])
>>> short1 #show the value of short1 as decimal
258
>>> '%04x' % short1 #show the value of short1 as hexadecimal
'0102'
```

Or, to put the first two bytes from a `bytes` sequence (`bytes1`) into a two one-byte integers (equivalent to `unsigned char` in C):

```
>>> byte1, byte2 = struct.unpack('!BB', bytes1[:2])
>>> byte1 #show the value of byte1 as decimal
1
>>> byte2 #show the value of byte2 as decimal
2
>>> '%02x' % byte1 #show the value of byte1 as hexadecimal
'01'
>>> '%02x' % byte2 #show the value of byte2 as hexadecimal
'02'
```

To convert a short (two-byte) integer to a `bytes` object:

```
>>> struct.pack('!H', 0x0102)
b'\x01\x02'
```

or:

```
>>> struct.pack('!H', 258)
b'\x01\x02'
```

To convert two one-byte integers to a `bytes` object:

```
>>> struct.pack('!BB', 0x1, 0x2)
b'\x01\x02'
```

or:

```
>>> struct.pack('!BB', 1, 2)
b'\x01\x02'
```

For more info see the following:

- bytes documentation
- binascii documentation
- struct documentation