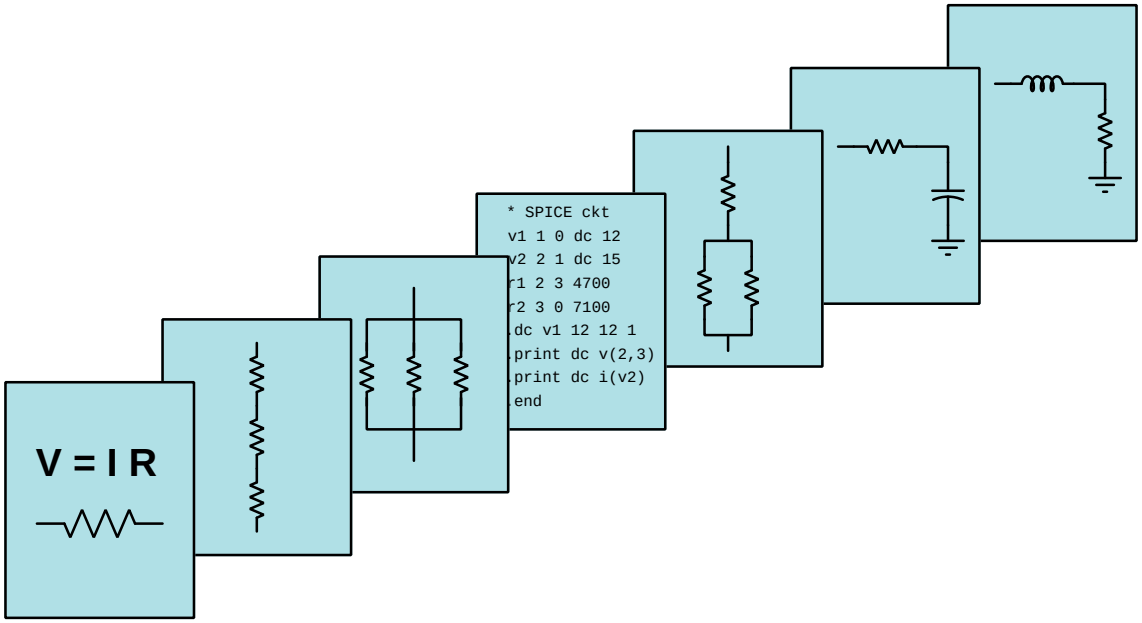
Modular Electronics Learning (ModEL) project



Internet Protocols

© 2020-2026 by Tony R. Kuphaldt – under the terms and conditions of the Creative Commons Attribution 4.0 International Public License

Last update = 24 April 2026

This is a copyrighted work, but licensed under the Creative Commons Attribution 4.0 International Public License. A copy of this license is found in the last Appendix of this document. Alternatively, you may visit `http://creativecommons.org/licenses/by/4.0/` or send a letter to Creative Commons: 171 Second Street, Suite 300, San Francisco, California, 94105, USA. The terms and conditions of this license allow for free copying, distribution, and/or modification of all licensed works by the general public.

Contents

1 Introduction **3**
- 1.1 Recommendations for students . . . 3
- 1.2 Challenging concepts related to internet protocols . . . 5
- 1.3 Recommendations for instructors . . . 6

2 Case Tutorial **7**
- 2.1 Example: bitwise logical operations . . . 8
  - 2.1.1 Bitwise-AND . . . 8
  - 2.1.2 Bitwise-OR . . . 8
  - 2.1.3 Bitwise-XOR . . . 9
  - 2.1.4 Bitwise-complement . . . 9
- 2.2 Example: simple TCP message analysis . . . 10

3 Tutorial **23**
- 3.1 Data encapsulation . . . 24
- 3.2 IPv4 addressing . . . 26
  - 3.2.1 The “ping” utility . . . 28
  - 3.2.2 IPv4 address ranges . . . 29
- 3.3 IP subnetworks and subnet masks . . . 30
- 3.4 Routing tables . . . 34
- 3.5 IP version 6 . . . 35
- 3.6 Address Resolution Protocol (ARP) . . . 36
- 3.7 Domain Name System (DNS) . . . 37
- 3.8 Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) . . . 38

4 Derivations and Technical References **41**
- 4.1 The OSI Reference Model . . . 42
- 4.2 Command-line diagnostic utilities . . . 44
- 4.3 ASCII character codes . . . 49

5 Questions **51**
- 5.1 Conceptual reasoning . . . 55
  - 5.1.1 Reading outline and reflections . . . 56
  - 5.1.2 Foundational concepts . . . 57

5.1.3 Industrial SCADA system . . . 59
5.1.4 Different protocols . . . 60
5.2 Quantitative reasoning . . . 61
5.2.1 Miscellaneous physical constants . . . 62
5.2.2 Introduction to spreadsheets . . . 63
5.2.3 IP address space . . . 66
5.2.4 PLC transmitting ASCII message via UDP . . . 67
5.2.5 PLC transmitting ASCII message to itself . . . 68
5.2.6 PLC transmitting integer data over network . . . 69
5.3 Diagnostic reasoning . . . 71
5.3.1 Tracing message routes . . . 72
5.3.2 Possible faults in a network . . . 73

6 Projects and Experiments 75
6.1 Recommended practices . . . 75
6.1.1 Safety first! . . . 76
6.1.2 Other helpful tips . . . 78
6.1.3 Terminal blocks for circuit construction . . . 79
6.1.4 Conducting experiments . . . 83
6.1.5 Constructing projects . . . 87
6.2 Experiment: PLC data transmission using IP . . . 88

A Problem-Solving Strategies 89

B Instructional philosophy 91
B.1 First principles of learning . . . 92
B.2 Proven strategies for instructors . . . 93
B.3 Proven strategies for students . . . 95
B.4 Design of these learning modules . . . 96

C Tools used 99

D Creative Commons License 103

E References 111

F Version history 113

Index 114

Chapter 1

Introduction

1.1 Recommendations for students

This module introduces the foundational protocols which make the *internet* possible as a world-wide network: Internet Protocol (IP) and Transmission Control Protocol (TCP). Together, these protocols provide an organizational framework within which many different lower-level communication protocols may exchange information, thus enabling a seamless network to be built from a patchwork of disparate data networks.

Important concepts related to these protocols include the **OSI Reference Model**, **data frames**, **encapsulation**, **IP addresses**, **masks**, **MAC addresses**, **subnetworks**, the **ping** utility, **routing** and routing **tables**, **IPv4** versus **IPv6**, **address resolution**, TCP **ports**, **flow control**, the **netstat** utility, and TCP **states**.

A useful reading strategy is to write your own summary page of important principles. This is useful with a topic such as network protocols, where you are faced with an array of different protocols handling different aspects of a complete communications link such as the case with IP and TCP/UDP. Your own summary describing what IP does, versus what TCP/UDP do, will help consolidate all the information you will read in the Tutorial.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to explore the concept of encapsulation for any given network protocol? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- What exactly is a *protocol*?
- How are large data files communicated over networks capable only of transmitting short bursts of data?
- How do IP and TCP work together to achieve a larger goal?
- What is *encapsulation*, and why is this important to the operation of data networks?

- How is the function of IP distinct from the function of TCP or UDP?
- How and why might you use `ping` to troubleshoot a network?
- What purpose do subnetwork *masks* serve in IP networks?
- How are IP addresses specified, and how do these differ from other types of addresses such as Ethernet MAC addresses?
- What is *routing* as it applies to the internet?
- How do IP version 4 and IP version 6 compare with one another?
- What are some of the other protocols besides IP and TCP/UDP used in internet communication?
- How are errors detected in a TCP/IP network?
- How do TCP and UDP compare with one another? Why might one be used instead of the other?
- What software tools are available to help configure, test, and diagnose these networks?
- What is the purpose of a *state* in TCP?

1.2 Challenging concepts related to internet protocols

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Encapsulation** – the notion of one protocol's data frame being contained as payload within another's is somewhat confusing, but is made clear by inspection of actual communications using packet-sniffing software (e.g. `Wireshark`).
- **Bitwise logical operations** – applying AND, OR, and XOR logical operations to respective bits of binary words is not a straight-forward concept, but is made more understandable by drawing out the binary word(s) in question and if necessary drawing small logic gates taking inputs from those bits to generate the respective bits of the output word. Another helpful perspective for understanding the purpose of bitwise operations is to view AND functions as *forcing* a 0 output if any input is 0, and OR functions as *forcing* a 1 output of any input is a 1. This is relevant to subnet masks in IP protocol.
- **Network arbitration** – the same is true for channel arbitration techniques such as CSMA/CD, master-slave, and others. It is helpful to imagine a set of devices all requiring access to a common channel of communication, and stepping through each of the protocols to see how they manage this access without having multiple devices "talk over" one another. This, like so many other things, simply takes time to digest and cannot be rushed.

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

  Assessment – Students present their "Conceptual Toolbox" Theory Journal entries listing important concepts applied in the Tutorial text(s). A "Conceptual Toolbox" serves as inspiration when solving problems, each concept suggesting a potential problem-solving action. Challenge students to collaboratively generate a Conceptual Toolbox for the day's topic during the class session and then use it when solving problems together, thus fostering the habit of reasoning from principles rather than recalling procedures!

- **Outcome** – Demonstrate the concept of encapsulation by experimenting with packet-sniffing software

  Assessment – Identify Ethernet, IP, and TCP/UDP message frames encapsulated within each other in packets intercepted using `Wireshark` software.

- **Outcome** – Diagnose a faulted network based on reported symptoms and testing

  Assessment – Determine the probability of various component faults in a LAN given symptoms and "ping" test results; e.g. pose problems in the form of the "Possible faults in a network" Diagnostic Reasoning question.

  Assessment – Determine the appropriateness of various diagnostic tests in identifying the location and nature of a fault in a LAN given symptoms and test results.

Chapter 2

Case Tutorial

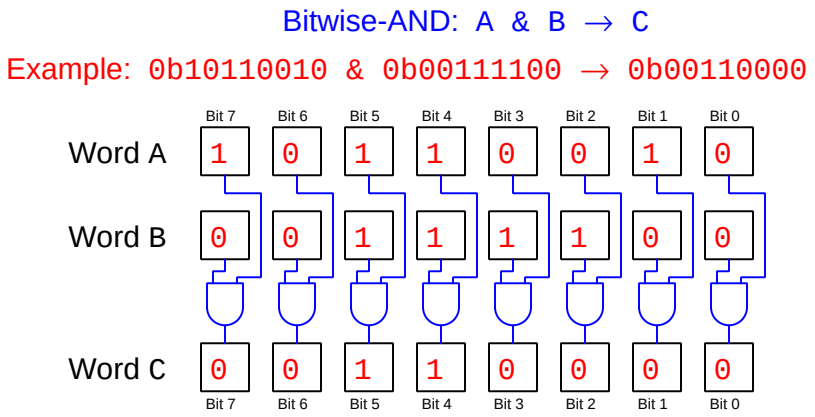
The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

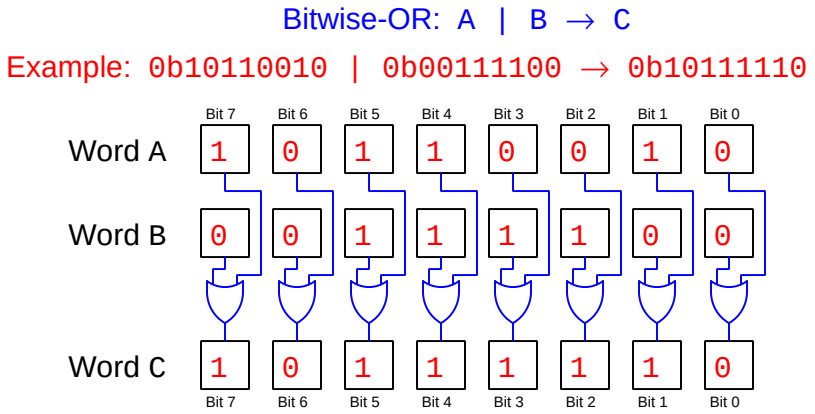
2.1 Example: bitwise logical operations

The following examples show bitwise operations being performed on 8-bit words (bytes).

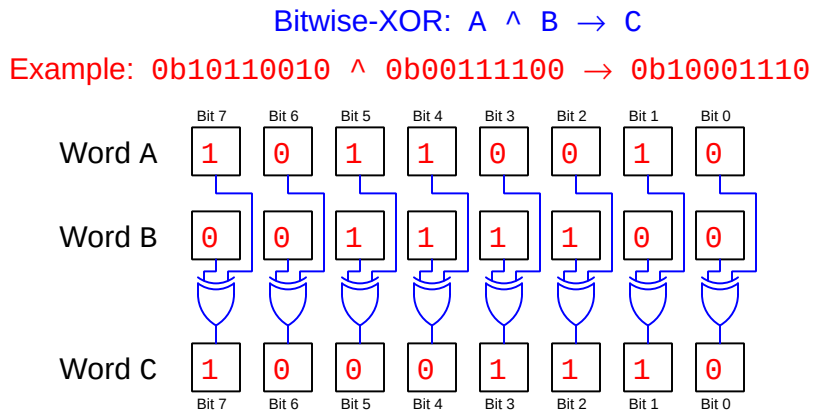
2.1.1 Bitwise-AND



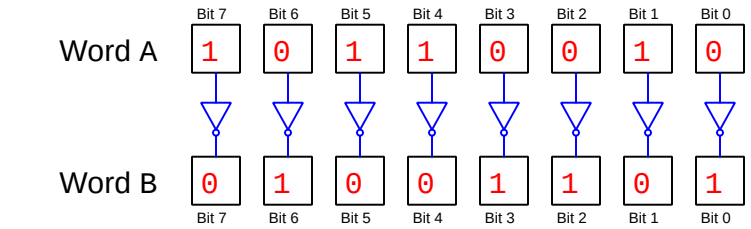
2.1.2 Bitwise-OR



2.1.3 Bitwise-XOR



2.1.4 Bitwise-complement



2.2 Example: simple TCP message analysis

Here we will use the Unix utility `netcat` to send simple ASCII text messages back and forth between two computers connected to a common Ethernet network, hereafter referred to as computer A and computer B. Our task will be to send a text message from computer A to computer B, and then to send a reply message from computer B back to computer A. Computer A's IPv4 address is `192.168.254.24`, while computer B's IPv4 address is `192.168.254.18`.

First we must invoke `netcat` on computer B to operate in "listen" mode. This opens up a specified TCP port on computer B, which computer A may then communicate to. For this exercise we will arbitrarily specify a TCP port number of `55555`, using the following command-line instruction[1]:

```
nc -l -vv -p 55555
```

After doing this, we invoke `netcat` on computer A to prepare for sending the first message. Here we instruct `netcat` as to the IPv4 address and TCP port number of computer B, so the message we type on computer A will arrive at its intended destination. To do this, we will use the following command-line instruction[2]:

```
nc 192.168.254.18 55555
```

After invoking `netcat` on both computers, in this order, anything typed at the console of computer A (after pressing the Enter button) will be displayed on the console of computer B, and vice-versa.

Furthermore, we may use a "packet-sniffing" utility such as `Wireshark` to record this exchange for close analysis.

[1] In this mode, we use the `-l` argument to instruct `netcat` to *listen*, using the `-vv` option to specify "very verbose" operation, and lastly using the `-p` option to specify the TCP port number `55555` to open.

[2] In this mode, the two arguments to `netcat` are the IPv4 address and the TCP port number, in that order.

Using the filter string `tcp.port == 55555` in `Wireshark` to show only those network packets related to this exchange, we get the following summary of four packets:

```
Source          Destination     Pro.  Information
192.168.254.24  192.168.254.18  TCP   54171 -> 55555 [PSH, ACK] Seq=1 Ack=1 Win=1026 Len=38
192.168.254.18  192.168.254.24  TCP   55555 -> 54171 [ACK] Seq=1 Ack=39 Win=1026 Len=0
192.168.254.18  192.168.254.24  TCP   55555 -> 54171 [PSH, ACK] Seq=1 Ack=39 Win=1026 Len=36
192.168.254.24  192.168.254.18  TCP   54171 -> 55555 [ACK] Seq=39 Ack=37 Win=1026 Len=0
```

Note how computer B's TCP port number is 55555 as specified in our invocation of `netcat`, while computer A's TCP port is a value chosen by the computer (54171).

Let us begin with the first packet in this filtered capture of network traffic, where we typed a text message at computer A destined for computer B:

```
Source          Destination     Pro.  Information
192.168.254.24  192.168.254.18  TCP   54171 -> 55555 [PSH, ACK] Seq=1 Ack=1 Win=1026 Len=38
```

Here is `Wireshark`'s analysis of that first packet:

```
92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface
    Interface id: 0 (\Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592})
        Interface name: \Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592}
        Interface description: Ethernet
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  5, 2022 19:57:51.418052000 Pacific Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1654484271.418052000 seconds
    [Time delta from previous captured frame: 3.488957000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 20.408713000 seconds]
    Frame Number: 40
    Frame Length: 92 bytes (736 bits)
    Capture Length: 92 bytes (736 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:data]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
Ethernet II, Src: EliteGro_16:ec:cb, Dst: EliteGro_f8:16:06
    Destination: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        Address: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        Address: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.254.24, Dst: 192.168.254.18
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 78
    Identification: 0x54f3 (21747)
    Flags: 0x40, Don't fragment
```

```
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.254.24
    Destination Address: 192.168.254.18
Transmission Control Protocol, Src Port: 54171, Dst Port: 55555, Seq: 1, Ack: 1, Len: 38
    Source Port: 54171
    Destination Port: 55555
    [Stream index: 5]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 38]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 2579253006
    [Next Sequence Number: 39    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 3511748816
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····AP···]
    Window: 1026
    [Calculated window size: 1026]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x7dbd [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    [Timestamps]
        [Time since first frame in this TCP stream: 0.000000000 seconds]
        [Time since previous frame in this TCP stream: 0.000000000 seconds]
    [SEQ/ACK analysis]
        [Bytes in flight: 38]
        [Bytes sent since last PSH flag: 38]
```

```
    TCP payload (38 bytes)
Data (38 bytes)
    Data: 546869732069732061206d6573736167652074797065642061742 0636f6d707574657220...
    [Length: 38]
```

A hexdump of this packet reveals the ASCII text message typed at computer A, as the data encapsulated by the TCP, IP, and Ethernet protocols:

```
0000   94 c6 91 f8 16 06 1c 69 7a 16 ec cb 08 00 45 00   .......iz.....E.
0010   00 4e 54 f3 40 00 80 06 00 00 c0 a8 fe 18 c0 a8   .NT.@...........
0020   fe 12 d3 9b d9 03 99 bc 47 0e d1 51 08 d0 50 18   ........G..Q..P.
0030   04 02 7d bd 00 00 54 68 69 73 20 69 73 20 61 20   ..}...This is a
0040   6d 65 73 73 61 67 65 20 74 79 70 65 64 20 61 74   message typed at
0050   20 63 6f 6d 70 75 74 65 72 20 41 0a                computer A.
```

Next we will examine the other three packets in order of their capture. The second of the filtered packets is this acknowledgement from computer B as having received the transmission from computer A:

```
Source          Destination     Pro.  Information
192.168.254.18  192.168.254.24  TCP   55555 -> 54171 [ACK] Seq=1 Ack=39 Win=1026 Len=0
```

And here is `Wireshark`'s analysis of the second packet:

```
60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface
    Interface id: 0 (\Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592})
        Interface name: \Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592}
        Interface description: Ethernet
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  5, 2022 19:57:51.462605000 Pacific Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1654484271.462605000 seconds
    [Time delta from previous captured frame: 0.044553000 seconds]
    [Time delta from previous displayed frame: 0.044553000 seconds]
    [Time since reference or first frame: 20.453266000 seconds]
    Frame Number: 41
    Frame Length: 60 bytes (480 bits)
    Capture Length: 60 bytes (480 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
Ethernet II, Src: EliteGro_f8:16:06, Dst: EliteGro_16:ec:cb
    Destination: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
```

```
        Address: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        Address: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
    Padding: 000000000000
Internet Protocol Version 4, Src: 192.168.254.18, Dst: 192.168.254.24
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 40
    Identification: 0x561d (22045)
    Flags: 0x40, Don't fragment
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x2736 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.254.18
    Destination Address: 192.168.254.24
Transmission Control Protocol, Src Port: 55555, Dst Port: 54171, Seq: 1, Ack: 39, Len: 0
    Source Port: 55555
    Destination Port: 54171
    [Stream index: 5]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 0]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3511748816
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 39    (relative ack number)
    Acknowledgment number (raw): 2579253044
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
```

```
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····A····]
    Window: 1026
    [Calculated window size: 1026]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xc6a3 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    [Timestamps]
        [Time since first frame in this TCP stream: 0.044553000 seconds]
        [Time since previous frame in this TCP stream: 0.044553000 seconds]
    [SEQ/ACK analysis]
        [This is an ACK to the segment in frame: 40]
        [The RTT to ACK the segment was: 0.044553000 seconds]
```

Here is the hexdump of this second packet:

```
0000   1c 69 7a 16 ec cb 94 c6 91 f8 16 06 08 00 45 00   .iz...........E.
0010   00 28 56 1d 40 00 80 06 27 36 c0 a8 fe 12 c0 a8   .(V.@...'6......
0020   fe 18 d9 03 d3 9b d1 51 08 d0 99 bc 47 34 50 10   .......Q....G4P.
0030   04 02 c6 a3 00 00 00 00 00 00 00 00               ............
```

Next is the third packet, this one containing the text message typed at computer B in reply to the original message from computer A:

```
Source          Destination     Pro.  Information
192.168.254.18  192.168.254.24  TCP   55555 -> 54171 [PSH, ACK] Seq=1 Ack=39 Win=1026 Len=36
```

And here is `Wireshark`'s analysis of the third packet:

```
90 bytes on wire (720 bits), 90 bytes captured (720 bits) on interface
    Interface id: 0 (\Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592})
        Interface name: \Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592}
        Interface description: Ethernet
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  5, 2022 19:58:33.805408000 Pacific Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1654484313.805408000 seconds
    [Time delta from previous captured frame: 31.549835000 seconds]
    [Time delta from previous displayed frame: 42.342803000 seconds]
    [Time since reference or first frame: 62.796069000 seconds]
    Frame Number: 63
    Frame Length: 90 bytes (720 bits)
    Capture Length: 90 bytes (720 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp:data]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
Ethernet II, Src: EliteGro_f8:16:06, Dst: EliteGro_16:ec:cb
    Destination: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        Address: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        Address: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.254.18, Dst: 192.168.254.24
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 76
    Identification: 0x561e (22046)
    Flags: 0x40, Don't fragment
```

```
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x2711 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.254.18
    Destination Address: 192.168.254.24
Transmission Control Protocol, Src Port: 55555, Dst Port: 54171, Seq: 1, Ack: 39, Len: 36
    Source Port: 55555
    Destination Port: 54171
    [Stream index: 5]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 36]
    Sequence Number: 1    (relative sequence number)
    Sequence Number (raw): 3511748816
    [Next Sequence Number: 37    (relative sequence number)]
    Acknowledgment Number: 39    (relative ack number)
    Acknowledgment number (raw): 2579253044
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x018 (PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····AP···]
    Window: 1026
    [Calculated window size: 1026]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0xe4b5 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    [Timestamps]
        [Time since first frame in this TCP stream: 42.387356000 seconds]
        [Time since previous frame in this TCP stream: 42.342803000 seconds]
    [SEQ/ACK analysis]
        [Bytes in flight: 36]
        [Bytes sent since last PSH flag: 36]
```

```
    TCP payload (36 bytes)
Data (36 bytes)
    Data: 546869732069732061207265706c7920747970656420617420636f6d707574657220420a
    [Length: 36]
```

Here is the hexdump of this third packet, where you can see the ASCII-encoded message:

```
0000   1c 69 7a 16 ec cb 94 c6 91 f8 16 06 08 00 45 00   .iz...........E.
0010   00 4c 56 1e 40 00 80 06 27 11 c0 a8 fe 12 c0 a8   .LV.@...'.......
0020   fe 18 d9 03 d3 9b d1 51 08 d0 99 bc 47 34 50 18   .......Q....G4P.
0030   04 02 e4 b5 00 00 54 68 69 73 20 69 73 20 61 20   ......This is a
0040   72 65 70 6c 79 20 74 79 70 65 64 20 61 74 20 63   reply typed at c
0050   6f 6d 70 75 74 65 72 20 42 0a                     omputer B.
```

Lastly, we have the TCP acknowledgement from computer A upon receiving the response typed at computer B:

```
Source          Destination     Pro.  Information
192.168.254.24  192.168.254.18  TCP   54171 -> 55555 [ACK] Seq=39 Ack=37 Win=1026 Len=0
```

And here is `Wireshark`'s analysis of this fourth packet:

```
54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface
    Interface id: 0 (\Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592})
        Interface name: \Device\NPF_{257A10A2-544E-4CD5-B1CB-CCEB5313B592}
        Interface description: Ethernet
    Encapsulation type: Ethernet (1)
    Arrival Time: Jun  5, 2022 19:58:33.849491000 Pacific Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1654484313.849491000 seconds
    [Time delta from previous captured frame: 0.044083000 seconds]
    [Time delta from previous displayed frame: 0.044083000 seconds]
    [Time since reference or first frame: 62.840152000 seconds]
    Frame Number: 64
    Frame Length: 54 bytes (432 bits)
    Capture Length: 54 bytes (432 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
Ethernet II, Src: EliteGro_16:ec:cb, Dst: EliteGro_f8:16:06
    Destination: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        Address: EliteGro_f8:16:06 (94:c6:91:f8:16:06)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        Address: EliteGro_16:ec:cb (1c:69:7a:16:ec:cb)
        .... ..0. .... .... .... .... = LG bit: Globally unique address (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 192.168.254.24, Dst: 192.168.254.18
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        0000 00.. = Differentiated Services Codepoint: Default (0)
        .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
    Total Length: 40
    Identification: 0x54f4 (21748)
    Flags: 0x40, Don't fragment
```

```
        0... .... = Reserved bit: Not set
        .1.. .... = Don't fragment: Set
        ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: TCP (6)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.254.24
    Destination Address: 192.168.254.18
Transmission Control Protocol, Src Port: 54171, Dst Port: 55555, Seq: 39, Ack: 37, Len: 0
    Source Port: 54171
    Destination Port: 55555
    [Stream index: 5]
    [Conversation completeness: Incomplete (12)]
    [TCP Segment Len: 0]
    Sequence Number: 39    (relative sequence number)
    Sequence Number (raw): 2579253044
    [Next Sequence Number: 39    (relative sequence number)]
    Acknowledgment Number: 37    (relative ack number)
    Acknowledgment number (raw): 3511748852
    0101 .... = Header Length: 20 bytes (5)
    Flags: 0x010 (ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 0... = Push: Not set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...0 = Fin: Not set
        [TCP Flags: ·····A····]
    Window: 1026
    [Calculated window size: 1026]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x7d97 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
    [Timestamps]
        [Time since first frame in this TCP stream: 42.431439000 seconds]
        [Time since previous frame in this TCP stream: 0.044083000 seconds]
    [SEQ/ACK analysis]
        [This is an ACK to the segment in frame: 63]
        [The RTT to ACK the segment was: 0.044083000 seconds]
```

Here is the hexdump of this fourth packet:

```
0000   94 c6 91 f8 16 06 1c 69 7a 16 ec cb 08 00 45 00   .......iz.....E.
0010   00 28 54 f4 40 00 80 06 00 00 c0 a8 fe 18 c0 a8   .(T.@...........
0020   fe 12 d3 9b d9 03 99 bc 47 34 d1 51 08 f4 50 10   ........G4.Q..P.
0030   04 02 7d 97 00 00                                 ..}...
```

Chapter 3

Tutorial

3.1 Data encapsulation

The very first time I recall learning about the existence of a world-wide *Internet*, my first question was "What does the network *look* like?" The first vision entering my mind when people told me about a computer network spanning nearly all of the United States and many other parts of the world was that of a thick cable strung along telephone poles and buried underground, with a big sign on it saying "Internet." I also remember well the shock of learning that although the Internet made use of several high-capacity networks (called *backbones*) connecting large data centers in different cities, the real "magic" of the Internet did not reside in any particular piece of hardware. Instead, what made the Internet so widespread and accessible was actually a series of *protocols* describing ways to exchange data amongst disparate network types. These "protocols" allowed digital data to be packaged in such a way that it could be sent along nearly any kind of communications link from copper wires to fiber-optic to radio waves – and indeed along multiple pathways between the same two points – while arriving at the destination intact. Thus, the Internet was akin to a random patchwork of existing communications pathways pressed into coordinated service by the sharing of a common "language."

The OSI Reference model helps to describe different aspects of data communication, from the physical hardware involved (lowest layer) all the way up to the end-purpose of the data (highest layer). My technical knowledge of data networks at the time I first learned of the internet was limited to lower-level standards such as EIA/TIA-232 and Ethernet which are mostly defined by their physical (electronic) construction. EIA/TIA-232 excels at point-to-point communication, and Ethernet with communication amongst a relatively small groups of computers, hence my instinct to think of the internet in physical terms. However, these networks alone lack the capability to manage details of communication necessary for anything as large and as complex as the internet.

At the heart of the internet is a concept known as *packet switching*, whereby communicated data is first broken into smaller packages called "packets" and then these packets are sent as separate communication events over a network with a multitude of possible pathways from source to destination. Each packet is "routed" along the network by dedicated computer devices deciding the best route(s) to take, and then when the packets are received at the destination they are reassembled into the original form. Two of the most common protocols enabling packet switching are IP (Internet Protocol) responsible for defining unique addresses to points along the network, and TCP (Terminal Control Protocol) responsible for defining how large blocks of data are to be divided up into smaller sections. Every computer along the way in a complex network must be programmed to properly interpret and manipulate data according to these protocols.

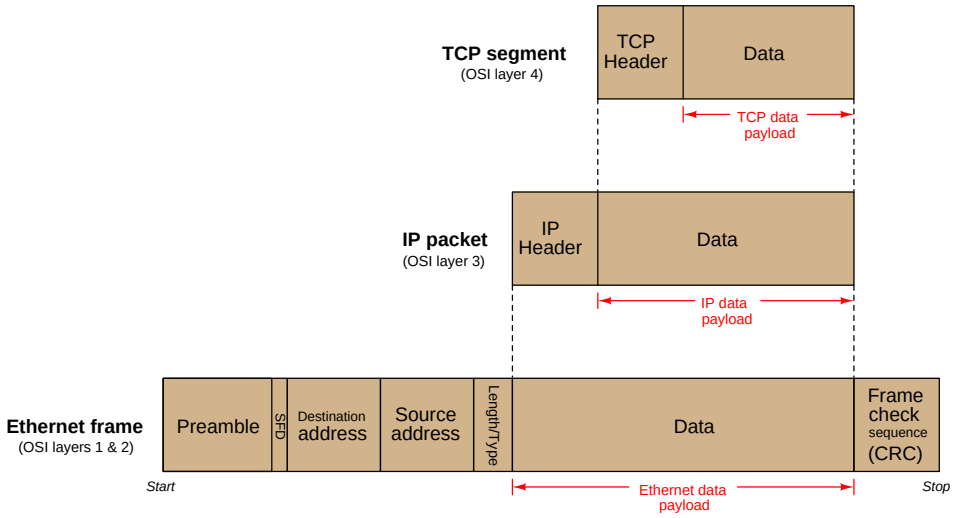
Data packets need not all take the same route from source to destination, nor do they even need to travel along the same *kinds* of networks. In a web-shaped network such as the internet where multiple pathways exist between any two points, the ability to direct packets of data along alternate routes increases the reliability of that network: failure of any one routing node or communications pathway does not necessarily prevent data from reaching its final destination because any lost packets may simply be re-sent along different paths. This fault tolerance was one of the design criteria for what came to be the Internet when it was first developed by the United States' Department of Defense.

Suppose a computer is tasked with sending a 500 kilobyte data file to another computer located some distance away and connected via Ethernet. A single Ethernet frame is only able to support a data message 1500 bytes *maximum*, and so sending this file within a single Ethernet frame is out of the question. Using TCP and IP packetize and *encapsulate* this data file for transport over Ethernet involved the following steps:

1. The source computer executes the TCP protocol to subdivide the data file into smaller pieces, prepending a TCP "header" containing identifying information to each of those pieces, thus creating a TCP *segment*

2. The source computer then executes the IP protocol to properly label each of those TCP segments for transport over the next lower-level protocol, prepending another "header" with identifying information, so that the TCP segment becomes the data payload for an IP *packet*

3. The source computer finally embeds each IP packet as the data payload for a separate Ethernet transmission



The receiving computer then "de-encapsulates" each Ethernet frame as it arrives, stripping the header information from each layer until the raw data is left to be reassembled into the original 500 kilobyte file.

In this example, TCP and IP are the "magic" that makes it possible to send the data file over the Internet to other computers. Any protocols above TCP (e.g. at the Application layer) merely send their messages to TCP for handling, and any protocols below IP (e.g. at the Physical and Data Link layers) handle all the details of message encoding and decoding, hardware-level addressing, logic level interpretation, etc. From the perspective of TCP and IP, both the *purpose* of the data and the *means of physical communication* are irrelevant, for their sole purpose is to form a reliable message-delivery service for *any* high-level application, working over *any* low-level physical network.

3.2 IPv4 addressing

Internet Protocol, or *IP*, is analogous to a postal system with its standards for labeling device addresses, specifying data package sizes, etc. It provides a structure necessary for organizing and routing data in volumes that would be too large and too complex for lower-level communication protocols to handle. The first widely-deployed version of Internet Protocol was version 4, popularly known as *IPv4*. We will discuss IPv4's addressing scheme as well as practical diagnostic tools useful for troubleshooting problems in IPv4-based communication systems.

IP is a "layer 3" technology according to the OSI Reference Model, being concerned with network-wide addresses for routing information between two different locations. IP is not concerned with the details of communication along any particular wire or fiber-optic cable. It is not "aware" of how bits are represented electrically, or what kind of connectors are used to couple cables together. IP is only concerned with "networks" in the broad sense of the word, as abstract collections of computers that are *somehow* (it doesn't care exactly how) connected to each other.

Networking equipment (DCE) designed to pay attention to IP addresses for routing purposes are called, not surprisingly, *routers*. Their purpose is to direct packets to their appropriate destinations in the shortest amount of time.

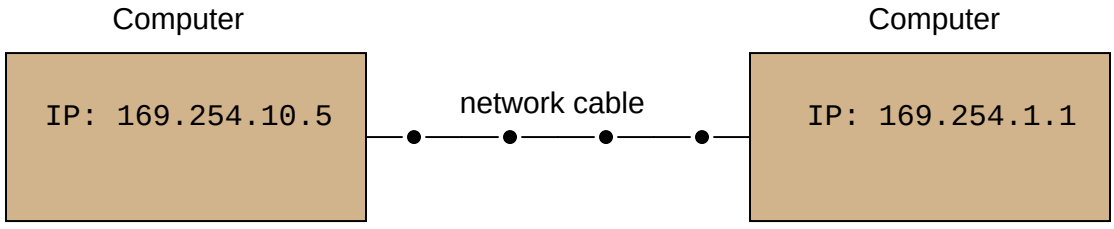
In order for the Internet Protocol to specify where packets are coming from and where they are going to, each source and destination must be marked with its own *IP address*. IP version 4 uses 32-bit addresses, usually expressed as four octets (four bytes) written using decimal numbers. For example:

IPv4 address `00000000 00000000 00000000 00000000` is written as `0.0.0.0`

IPv4 address `11111111 11111111 11111111 11111111` is written as `255.255.255.255`

IPv4 address `10101001 11111010 00101101 00000011` is written as `169.250.45.3`

In order for two inter-connected computers to exchange data using Internet Protocol, each one must have a unique IP address:



At first, this may seem redundant. Doesn't each and every Ethernet device already have its own unique "MAC address" 48 bits in length to distinguish it from every other Ethernet device in existence? If so, why add *another* set of identifying addresses to the system?

This is true – Ethernet devices are already uniquely addressed – but those MAC addresses serve different purposes than IP addresses. Recall that Ethernet is a standard only at layers 1 and 2, and is not "aware" of any higher-level concerns. Ethernet MAC addresses are useful to switching hubs and other Ethernet DCE devices tasked with management of Ethernet data frames, but those

MAC addresses – unique as they may be – have little relevance in the greater picture of IP where we must fragment and reassemble messages over very large-scale networks. More importantly, the reason we need IP addresses is to be able to use interconnecting networks other than Ethernet. For example, two computers may be connected to each other with a simple EIA/TIA-232 cable (or even using radio transceiver units for a "wireless" connection) instead of Ethernet, but still use Internet Protocol to route packets to their destinations[1]. By having its own dedicated addressing scheme, IP ensures computers can send and receive data packets with no regard to *physical interconnection details, channel arbitration methods, or anything else in between.* In a sense, IP is the "glue" that holds disparate networks together, and makes something like a world-wide Internet possible when so many different network types exist to connect digital devices together. If we attempted to use Ethernet MAC addresses for the same purpose, *the entire Internet would have to consist solely of Ethernet networks!*

A helpful analogy is to think of Ethernet MAC addresses like Social Security numbers for United States citizens, while IP addresses are like street addresses used to route mail. Each US citizen should have their own unique Social Security number, shared by no one else. This number is used for many purposes, including identification on Federal tax documents, to help route specific information (such as income records and Social Security payments) to the proper people. Despite the uniqueness of these numbers, though, people still need separate mailing addresses in order to receive mail through the postal service and other package distribution agencies. The mailing address serves a different purpose than the Social Security "address" each US citizen possesses. Furthermore, the existence of separate mailing addresses ensures even non-citizens living in the United States (e.g. foreign students, ambassadors, etc.) who have no Social Security numbers still have a way to send and receive mail. The mapping of device MAC addresses to IP addresses is handled by a protocol called *ARP* (Address Resolution Protocol) discussed later in this chapter.

[1] In fact, this is precisely the state of affairs if you use a *dial-up* telephone connection to link your personal computer with the Internet. If you use dial-up, your PC may not use Ethernet at all to make the connection to your telephone provider's network, but rather it might uses EIA/TIA-232 or USB to a modem (modulator/demodulator) device, which turns those bits into modulated waveforms transmittable over a voice-quality analog telephone line.

3.2.1 The "ping" utility

Computers enabled to communicate using Internet Protocol (IP) are equipped with a utility program named `ping` useful for detecting the presence of other IP-enabled computers connected to the same network. The classic format of this program is execution by typing the word "ping" at the computer's command-line interface followed by the IP address of the other computer you wish to detect the presence of. For example, if I wished to check for the presence of a computer on the network with an IP address of 133.82.201.5, I would type this command at my computer's command line and press the "Enter" key:

```
ping 133.82.201.5
```

The `ping` utility works by sending a very short digital message[2] to the specified IP address, requesting a reply from that computer (usually with multiple attempts). The `ping` command as implemented on the Microsoft Windows operating system typically makes four attempts before quitting. Some other operating systems' implementation of `ping` continue indefinitely until halted by the user with the "Control-C" keystroke interrupt combination.

When diagnosing problems with IP-enabled network devices, few utilities are as immediately useful as `ping`. Networking professionals commonly use the word "ping" as a verb, as in "I tried to *ping* that computer, but it gave no response." There are many reasons why a computer might fail to respond to a `ping` query, but a successful `ping` attempt proves several things:

- The destination device is powered up and its IP functionality is working
- All network devices (DCE) between your computer and the destination device are communicating
- All cables necessary for the communication of data between your computer and the destination are functional
- Both your computer and the destination device are on the same *subnet* (this topic covered in more detail later)

Since `ping` requires the first three layers of the OSI model to properly function (Physical, Data Link, and Network layers), using this as a diagnostic test neatly identifies where in the OSI model a problem exists. If two computers are not communicating with each other as they should but the `ping` utility works between them, the communication fault must lie within one of the upper OSI layers (e.g. Transport, Session, Presentation, or Application). Thus, we see the `ping` utility as a tool for "divide-and-conquer" style troubleshooting, where we may prove good connections between certain devices and thereby narrow the scope of the problem by elimination.

[2] The "ping" command is technically defined as an "Echo Request" command, which is part of the *Internet Control Message Protocol* (ICMP) suite.

3.2.2 IPv4 address ranges

Given the addressing purpose of Internet Protocol (to designate addresses over an extremely large collection of digital communication devices), addresses must be chosen with care. IP version 4 uses a 32-bit field to designate addresses, limiting its address capacity to 2^{32} unique addresses. As large as this number is, it is not enough to uniquely identify all Internet-capable devices worldwide. The inventors of IP did not dream their Internet would grow to the proportions it has today. Let this be a lesson to all those involved with computers: the future is usually bigger than you think! A variety of clever techniques has been developed to deal with this shortage of IP addresses. One of them is to dynamically assign addresses to Internet-connected computers *only when they are turned on*. This is how most personal Internet connections work: when you power up your personal computer to connect to the Internet, your service provider assigns you a temporary IP address through a protocol called DHCP (Dynamic Host Configuration Protocol). Your provider then forces you to relinquish this temporary IP address when you shut down your computer, so someone else may use it for theirs.

The *Internet Corporation for Assigned Names and Numbers*, or *ICANN*, is the organization responsible[3] for assigning IP addresses to Internet users worldwide (among other tasks). This group has designated certain IP address ranges specific to internal (i.e. *Local Area Network*, or *LAN*) network devices, which shall never be used "publicly" to address devices on the world-wide Internet. These specially-designated "private" LAN address ranges are as follows:

`10.0.0.0` to `10.255.255.255`

`172.16.0.0` to `172.31.255.255`

`192.168.0.0` to `192.168.255.255`

Additionally, all computers have their own special *loopback* IP address, used to send IP message packets to itself for certain purposes (including diagnostics): `127.0.0.1`. This IP address is completely *virtual*, not associated with any network hardware at all[4]. Therefore, the `ping` command executed on any computer should *always* be able to detect address `127.0.0.1`, regardless of the status or even existence of actual network hardware (cards or interfaces) on that computer. Failure of the `ping` command to detect the loopback address is a sign that the computer's operating system is not configured to use Internet Protocol.

A computer's loopback address may have uses other than diagnostic. Some computer applications are network-oriented by nature, and rely on IP addresses even if the application is performing some local function rather than a function between computers on an actual network. The *X-windows* graphic-user interface (GUI) system popularly used on UNIX operating systems is an example of this, referencing the loopback address to form a connection between client and server applications running on the same computer.

[3] Prior to ICANN's formation in 1999, the *Internet Assigned Numbers Authority*, or *IANA* was responsible for these functions. This effort was headed by a man named Jon Postel, who died in 1998.

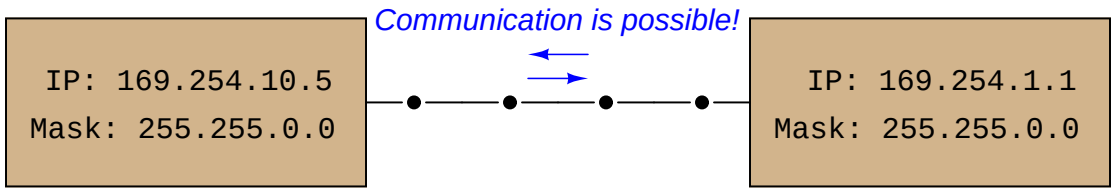
[4] The term "loopback" refers to an old trick used by network technicians to diagnose suspect serial port connections on a computer. Using a short piece of copper wire (or even a paperclip) to "jumper" pins 2 and 3 on an EIA/TIA-232 serial port, any serial data transmitted (out of pin 3) would be immediately received (in pin 2), allowing the serial data to "loop back" to the computer where it could be read. This simple test, if passed, would prove the computer's low-level communication software and hardware was working properly and that any networking problems must lie elsewhere.

3.3 IP subnetworks and subnet masks

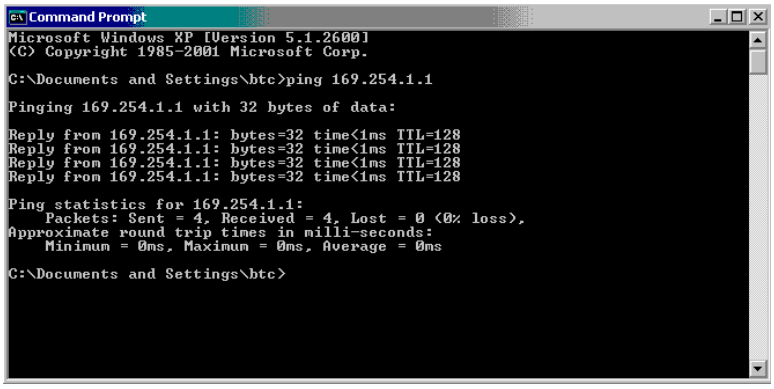
IP (version 4) addresses are used in conjunction with something called *subnet masks*[5] to divide IP networks into "subnetworks." A "subnetwork" is a range of IP-addressed devices allowed to communicate with each other. You may think of the subnet mask to be a sort of "filter" used to identify IP addresses belonging to the proper range.

The subnet mask works as a bitwise-AND filter, identifying those bits in the IP address defining the subnetwork. For example, if the subnet mask on a computer is set to `255.0.0.0` (binary `11111111 00000000 00000000 00000000`), it means the first 8 bits of the IP address define the subnetwork, and thus the computer is only allowed to communicate with another computer belonging to the same subnetwork (i.e. having the same first octet in its IP address).

A set of examples showing two interconnected computers with differing IP addresses (and in some cases, different masks) illustrates how this works[6]. In the first example, two computers with IP addresses differing in the last two octets are able to communicate because their subnets are the same (`169.254`):



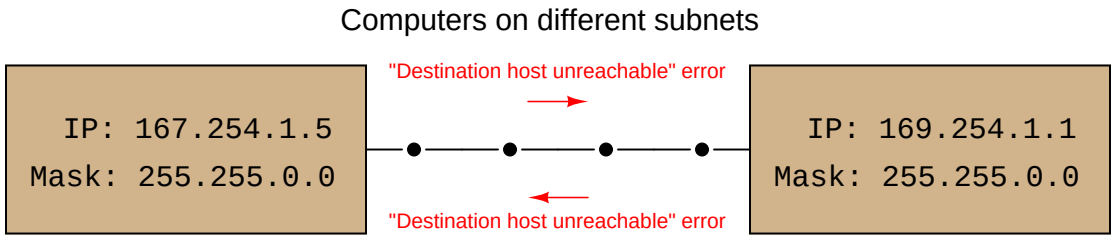
We may check to see the IP addresses and subnet masks are correct by using the `ping` command. A screenshot of `ping` being used on a personal computer running the Microsoft Windows XP operating system is shown here:



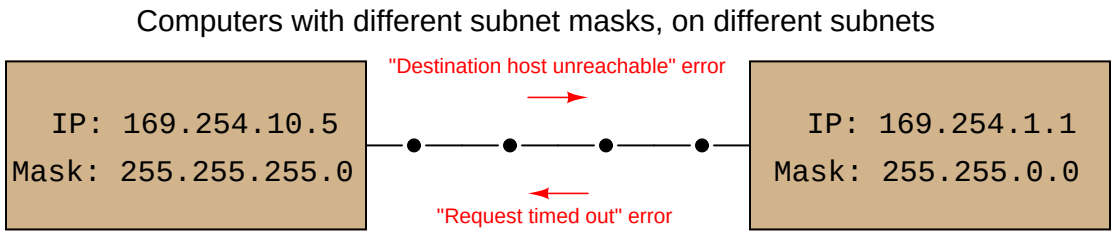
[5] Also called "netmasks" or simply "masks."

[6] These are real test cases I performed between two computers connected on a 10 Mbps Ethernet network. The error messages are those generated by the `ping` utility when communication was attempted between mis-matched computers.

In the next example, we see two computers with the same mask value, but with different address values in the octets designated by their masks. In other words, these two computers belong to different subnets: one to `167.254` and the other to `169.254`, and as a result they are not allowed to communicate with each other using Internet Protocol. The resulting error messages generated by the `ping` utility are shown in this diagram:



In the last example, we see two computers having different mask values as well as different IP addresses. The subnet of the left-hand computer is `169.254.10` while the subnet of the right-hand computer is `169.254`:



The computer on the left may only communicate with IP addresses matching in the first three octets (`169.254.10`). Seeing that the destination address for the second computer does not match in its third octet, `ping` returns a "Destination host unreachable" error message when executed from the left-hand computer.

When the computer on the right attempts to communicate with ("ping") the computer on the left, it is allowed to transmit to that computer because its mask only screens for agreement in the first two octets (`169.254`), which happen to match. However, the computer on the left is not allowed to transmit to the computer on the right because of its more restrictive subnet, and so `ping` running on the right-hand computer returns a "Request timed out" error message because it never receives a reply from the left-hand computer to any of its queries.

With just two computers connected by a single cable, the concept of subnetworks and masks seems useless, and indeed it is on this small of a scale. However, "subnetting" is a useful technique for managing high traffic loads on large networked systems using IP addresses, and so it is commonly seen in many local area networks (LANs) such as those found at industry and commercial sites.

While many IPv4-compliant computers designate both the IP address and the subnet mask values as sets of "dotted-decimal" numbers with each decimal (0-255) representing an "octet" of eight bits in the 32-bit IPv4 address space (e.g. IP = `169.254.5.1` and Mask = `255.255.0.0`), a more modern designation for subnets is to append the device's IP address with a forward slash character and a decimal number specifying how many bits are used to specify the subnet.

To illustrate by example, consider a computer having the IP address `169.254.5.1` and a mask value of `255.255.0.0` (thus specifying that it belongs to the subnetwork `169.254`), we could alternatively state that computer's IP address as `169.254.5.1/16`. The "16" means that the first sixteen bits of its IP address define its subnet. To cite another example, a computer with an IP address of `192.168.35.100` and a subnet mask of `255.255.255.0` could have its address alternatively written as `192.168.35.100/24`. The "24" tells us the first 24 bits (first three octets) of the IP address define its subnetwork: this computer belongs to the subnetwork `192.168.35`.

The `ping` diagnostic utility program may be used to search for unknown IP addresses on a known subnet. This may be done by "pinging" to the *broadcast address* for that subnet: an IP address formed by the known subnet numbers, followed by all binary 1's filling the unknown bit spaces. For example, you could use `ping` to search for devices on the subnet `156.71` (subnet mask `255.255.0.0`) by using the following command:

```
ping 156.71.255.255
```

3.4 Routing tables

Devices on an IP-compliant network need to know how to best route IP data packets from one location to another. In the case of special *router* devices managing traffic on the Internet, optimum packet routes are determined by a number of different criteria (e.g. degree of congestion in a route, the fewest number of "hops" from one router to another, geographical distance, etc.), updated continually by sophisticated algorithms operating within the routers. Data for these optimum routes are stored in lists called *routing tables*.

Personal computers also have routing tables, which may be modified by the user. An application where you may need to modify the routing table of a personal computer is the case of enabling that computer to communicate with a brand-new device installed on the industrial network, whose subnetwork ID differs from the other devices on that network. Many network-ready devices ship from the manufacturer with default subnets of `192.168.1`. It is quite possible that the subnet of the industrial network you intend to have the device operate on is different from this default factory-configuration. This in itself is not necessarily a problem, as IP addresses and subnet mask values of IP-enabled devices are always user-configurable. However, if the only method of configuring this new device is by communicating to it through an IP network connection, you are faced with a Catch-22: how do you communicate with it to alter its subnet, when its subnet prevents you from communicating with it?

One solution to this Catch-22 dilemma is to temporarily use a personal computer with a subnet matching the new device to configure that new device, then disconnecting the device from the personal computer after it has been configured for the new subnetwork, and subsequently plugging the device into the industrial network where it will function.

Another solution is to use one of the personal computers already residing on the industrial network to configure the new device, and specially enabling that one computer to talk to the device's default subnet. In this way, the new device may be plugged into the industrial network, configured for a new IP address and subnet while on that network, where it will subsequently communicate with existing devices on the proper subnet. This may be done through the `route` command-line utility. At a command prompt (just like when using the `ping` command), type "route" followed by arguments telling it to add the device's default address and subnet mask to the computer's routing table. Supposing our new device has a default IP address of `192.168.1.10` and a default mask of `255.255.255.0`, our `route` command would need to be entered as follows:

```
route add 192.168.1.10 mask 255.255.255.0
```

After issuing this command to the personal computer, it may be used to communicate with the new device to change its IP address and subnet mask values to match devices on the industrial network.

3.5 IP version 6

The next popular version of Internet Protocol (version 6, or *IPv6*) uses 128-bit addresses, giving 2^{128} address possibilities (in excess of 3.4×10^{38}), in stark contrast to IPv4's paltry 2^{32} address space. To put this enormous quantity into perspective, there are enough IPv6 addresses to designate nearly 57 *billion* of them for each and every gram of the Earth's mass[7]. While IPv4 addresses are typically written as four octets in decimal form (e.g. 169.254.10.5), this notation would be very cumbersome for writing IPv6 addresses. Thus, IPv6 addresses are written as a set of eight hexadecimal numbers (up to four characters per number) separated by colons, such as `4ffd:522:c441:d2:93b2:f5a:8:101f`. The phase-in of IPv6 to replace IPv4 has already started for certain portions of the Internet, but the full transition to IPv6 is expected to take many years. The IPv6 "loopback" virtual address for computers is `0:0:0:0:0:0:0:1`, or more simply[8] written as `::1`.

Note the "shorthand" notation used in the previous IPv6 addresses to eliminate extra characters: some of the 16-bit segments are truncated to less than four hexadecimal characters if the preceding (more-significant) characters are zero. Thus, you see `:522:` instead of `:0522:`, and `:d2:` instead of `:00d2:`. The loopback address of `::1` is the ultimate shorthand notation, collapsing all prior segments (which are all zero) into a pair of back-to-back colons.

IP version 6 supports subnetworks just as IPv4 does, but instead of denoting subnet masks in the same colon-delimited fashion as IPv6 addresses, IPv6 subnet masks are simply specified by the number of "1" bits beginning from the first (MSB). The rationale here is that subnet mask bits should be contiguous[9], with no zero bits separating one bits. This being the case, the subnet mask for any practical IP range may be specified as a simple number of 1's from the MSB down to the LSB[10].

For example, an IPv6 address written as `4ffd:522:c441:d2:93b2:f5a:8:101f/48` means that the first 48 bits of that address must match the first 48 bits of another address in order to be within the same subnetwork as the other address. In other words, any other address existing within the same subnetwork would also have to begin with `4ffd:522:c441`.

It should be noted that an updated version of the `ping` command (called `ping6`) is available to help diagnose IPv6 systems.

[7] According to Douglas Giancoli's *Physics for Scientists and Engineers* textbook, the mass of the Earth is 5.98×10^{24} kg, or 5.98×10^{27} grams. Dividing 2^{128} (the number of unique IPv6 addresses) by the Earth's mass in grams yields the number of available IPv6 address per gram of Earth mass. Furthermore, if we assume a grain of sand has a mass of about 1 milligram, and that the Earth is modeled as a very large collection of sand grains (not quite the truth, but good enough for a dramatic illustration!), we arrive at 57 *million* IPv6 addresses per grain of sand on Earth.

[8] The fully-written loopback address is actually `0000:0000:0000:0000:0000:0000:0000:0001`.

[9] While it is possible to use non-contiguous subnet mask values, the practice is frowned upon by most system administrators.

[10] Indeed, subnet masks for IPv4 can be specified in this manner as well, not just IPv6 subnet masks.

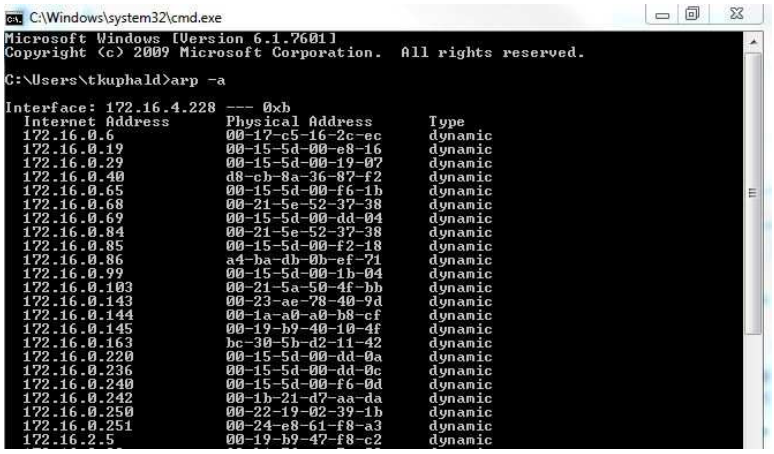
3.6 Address Resolution Protocol (ARP)

While Internet Protocol (IP) provides a universal addressing standard for devices operating on large-scale digital networks, individual devices usually possess MAC addresses unique to each device. As mentioned in a previous section, IP addresses are to MAC addresses as mailing addresses are to Social Security numbers: the IP address serves to route information sent over the network, while MAC addresses identify the individual devices themselves. Any digital network system dealing with both types of addresses must somehow "map" each MAC address to a corresponding IP address, and this is handled by a protocol called *Address Resolution Protocol*, or *ARP*.

Every node running the ARP protocol on a digital network maintains a table of equivalent addresses, MAC to IP. This table is called the *ARP cache*, the contents of which may be displayed by running the following command on the device's command-line interface:

```
arp -a
```

The `arp -a` command instructs the machine to print *all* (`-a`) ARP cache entries to the screen for your viewing. This, of course, only displays what that machine happens to know at that time. If the ARP cache has not been updated recently, addressing data found in the ARP cache may be out of date or even missing. Here is a partial screenshot of the `arp -a` command run on a Microsoft Windows computer, showing each IP ("internet") address in the ARP cache and its corresponding MAC ("physical") address:



One way to update the ARP cache on a machine with a command-line interface is to first issue a broadcast[11] ping request. Responses from active nodes on the network will populate the machine's ARP cache with address information, after which you may run the `arp -a` command to display those cache entries.

[11] The "ping" command is often used to test the response of a single IP node on a network, by issuing the command followed by the IP address of interest (e.g. `ping 192.168.35.70`). By contrast, a "broadcast" ping request attempts to contact a range of IP addresses within a subnet. For example, if we wished to ping all the IP addresses beginning with `192.168.35`, we would issue the command with all 1's in the last octet of the IP address field (e.g. `ping 192.168.35.255`).

3.7 Domain Name System (DNS)

The acronym *DNS* actually stands for two related things: *Domain Name System* and *Domain Name Server*. The first meaning of "DNS" refers to the system of exchanging numerical IP addresses with alphanumeric *Uniform Resource Locators* (*URLs*) which are easier for human beings to remember. When you use web browser software to navigate to a web site on the Internet, you have the option of entering the URL *name* of that site (e.g. `www.google.com`) or a numerical IP address (e.g. `75.125.53.104`). Special computers connected to the Internet called *Domain Name Servers*, and *Domain Name Resolvers* (*DNRs*) use the *Address Resolution Protocol* (*ARP*) to convert your target web site name to its actual IP address so that a connection may be made between that computer and yours.

ICANN, the same organization responsible for allotting IP addresses, also maintains databases for all registered domain names.

3.8 Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

At the next OSI Reference Model layer (layer 4) is a set of protocols specifying how reliable communication "connections" should be established between devices on a digital network. Rather than specifying addresses for routing data packets on a large network (OSI layer 3), layer 4 works on the level of virtual data "ports" at the transmitting and receiving devices. The purpose of these virtual ports is to manage multiple types of data transactions to and from the same IP address, such as in the case of a personal computer accessing a web page (using HTTP) and sending an email message (using SMTP) at the same time. An analogy to help understand the role of ports is to think of multiple packages delivered to different people at a common address such as a business office. The mailing address for the office is analogous to the IP address of a computer exchanging data over a network: it is how other computers on the network "find" that computer. The personal or department names written on the different packages are analogous to virtual ports on the computer: "places" within that node where specific messages are directed once they arrive.

Transmission Control Protocol (*TCP*) and *User Datagram Protocol* (*UDP*) are two methods used to manage data flow through "ports" on a DTE device, with TCP being the more complex and robust of the two. Both TCP and UDP rely on IP addressing to specify which devices send and receive data, which is why you will often see these protocols listed in conjunction with IP (e.g. TCP/IP and UDP/IP). TCP and UDP are both useless on their own: a protocol specifying port locations without an IP address would be as meaningless as a package placed in the general mail system with just a name but with no street address. The combination of an IP address and a TCP or UDP port number is called a *socket*. Conversely, IP anticipates the presence of a higher-level protocol such as TCP or UDP by reserving a portion of its "datagram" (packet) bit space for a "protocol" field[12] to specify which high-level protocol generated the data in the IP packet.

TCP is a complex protocol specifying not only which virtual "ports" will be used at the sending and receiving devices, but also how data integrity will be guaranteed. Data communicated via TCP are sent in blocks of bits called *segments*. In the same way that the IP algorithm encapsulates data into self-contained "packets" with the necessary routing data to ensure proper delivery to the destination, the TCP algorithm encapsulates data with "header" bits specifying such details as sequence number, acknowledgment identification, checksum (for error-detection), urgency of the message, and optional data. TCP takes responsibility for ensuring both devices on the network are ready to exchange data, breaks the data block into segments of an appropriate size to be encapsulated inside IP packets on the transmitting end, checks each segment for corruption (using the CRC method) upon receipt, reassembles those segments into the whole data block at the receiving end, and terminates the connection between devices upon completion. If a TCP segment successfully arrives at the receiving device, the TCP protocol running at the receiving device acknowledges this to the transmitting device. If the transmitting device fails to receive an acknowledgment within a specified time, it automatically re-sends the segment. If a TCP segment arrives corrupted, the TCP protocol running at the receiving device requests a re-send from the transmitting device.

Another feature of TCP is end-to-end *flow control*, enabling the receiving device to halt the incoming data stream for any reason, for example if its buffer memory becomes full. This is analogous

[12] Both IPv4 and IPv6 reserve eight bits for this purpose.

to the XON/XOFF control signals used in simple serial communication protocols such as EIA/TIA-232. In the case of large networks, TCP flow control manages the flow of data segments even when the communication paths are dynamic and liable to change.

The action of TCP to divide a large data set into smaller segments is akin to an author dividing up a typed paper manuscript in preparation to mail to a publisher, with page numbers showing the proper order. The action of IP to address and route these segments along a large network is akin to the postal system providing street addresses and standard envelope labeling, which the author uses to bundle up each manuscript segment.

If IP (Internet Protocol) is the "glue" that holds the Internet together, TCP (Transmission Control Protocol) is what makes that glue strong enough to be practically useful. Without TCP, data communication over the wildly disparate physical networks that comprise the world-wide Internet would be far less reliable.

UDP is a much simpler protocol for managing data ports, lacking not only the segment sequencing of TCP but also lacking many of the data-integrity features of TCP. Unlike TCP, the UDP algorithm does not sequence the data block into numbered segments at the transmitting end or reassemble those numbered segments at the receiving end, instead relegating this task to whatever application software is responsible for generating and using the data. Likewise, UDP does not error-check each segment and request re-transmissions for corrupted or lost segments, once again relegating these tasks to the application software running in the receiving computer. It is quite common to see UDP applied in industrial settings, where communication takes place over much smaller networks than the world-wide Internet, and where the IP data packets themselves tend to be much smaller as well. Another reason UDP is more common in industrial applications is that it is easier to implement in the "embedded" computer hardware at the heart of many industrial devices. The TCP algorithm requires greater computational power and memory capacity than the UDP algorithm, and so it is much easier to engineer a single-chip computer (i.e. microcontroller[13]) to implement UDP than it would be to implement TCP.

Information-technology professionals accustomed to the world of the Internet may find the use of UDP rather than TCP in industrial networks alarming. However, there are good reasons why industrial data networks do not necessarily need all the data-integrity features of TCP. For example, many industrial control networks consist of seamless Ethernet networks, with no routers (only hubs and/or switches) between source and destination devices. With no routers to go through, all IP packets communicated between any two specified devices are guaranteed to take the exact same data path rather than being directed along alternate, dynamic routes. The relative stability and consistency of communication paths in such industrial networks greatly simplifies the task of packet-reassembly at the receiving end, and makes packet corruption less likely. In fact, many industrial device messages are short enough to be communicated as single IP packets, with no need for data segmentation or sequencing at all!

[13]This limitation is fast becoming obsolete as microcontroller designs continue to advance in memory capacity and computational power.

Using another utility program on a personal computer called `netstat` (available for both Microsoft Windows and UNIX operating systems) to check active connections[14], we see the various IP addresses and their respective port numbers (shown by the digits following the colon after the IP address) as a list, organized by TCP connections and UDP connections:

```
C:\Documents and Settings\btc>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:2869           0.0.0.0:0              LISTENING
  TCP    127.0.0.1:1025         0.0.0.0:0              LISTENING
  TCP    127.0.0.1:5152         0.0.0.0:0              LISTENING
  TCP    169.254.1.2:23         169.254.1.1:1116       ESTABLISHED
  TCP    169.254.1.2:139        0.0.0.0:0              LISTENING
  UDP    0.0.0.0:445            *:*
  UDP    0.0.0.0:500            *:*
  UDP    0.0.0.0:1062           *:*
  UDP    0.0.0.0:4500           *:*
  UDP    0.0.0.0:7725           *:*
  UDP    127.0.0.1:123          *:*
  UDP    127.0.0.1:1063         *:*
  UDP    127.0.0.1:1066         *:*
  UDP    127.0.0.1:1900         *:*
  UDP    169.254.1.2:123        *:*
  UDP    169.254.1.2:137        *:*
  UDP    169.254.1.2:138        *:*
  UDP    169.254.1.2:1900       *:*

C:\Documents and Settings\btc>
```

An interesting distinction between TCP and UDP is evident in this screenshot. Note how each of the TCP connections has an associated "state" (either LISTENING or ESTABLISHED), while none of the UDP connections has any state associated with it. Recall that TCP is responsible for initiating and terminating the connection between network devices, and as such each TCP connection must have a descriptive state. UDP, on the other hand, is nowhere near as formal as TCP in establishing connections or ensuring data transfer integrity, and so there are no "states" to associate with any of the UDP connections shown. Data arriving over a UDP connection simply shows up unannounced, like an impolite guest. Data arriving over a TCP connection is more like a guest who announces in advance when they will arrive, and also says "good bye" to the host as they depart.

Many different port numbers have been standardized for different applications at OSI Reference Model layers above 4 (above that of TCP or UDP). Port 25, for example, is always used for SMTP (Simple Mail Transfer Protocol) applications. Port 80 is used by HTTP (HyperText Transport Protocol), a layer-7 protocol used to view Internet "web" pages. Port 443 is used by HTTPS, an encrypted (secure) version of HTTP. Port 107 is used by TELNET applications, a protocol whose purpose it is to establish command-line connections between computers for remote administrative work. Port 22 is used by SSH, a protocol similar to TELNET but with significantly enhanced security. Port 502 is designated for use with Modbus messages communicated over TCP/IP.

[14] In this particular case, I typed `netstat -an` to specify *all* (`a`) ports with *numerical* (`n`) IP addresses and port numbers shown.

Chapter 4

Derivations and Technical References

This chapter is where you will find mathematical derivations too detailed to include in the tutorial, and/or tables and other technical reference material.

4.1 The OSI Reference Model

Digital communication is a fairly complex topic, and there are a great many ways in which one digital device might communicate information to another. In order to help describe how different methods (protocols) function in forming a complete communication system, the *OSI Reference Model* was developed. This is not a standard itself, but rather a taxonomical system for *describing* and *classifying* various aspects of any communications system.

For example, one communication system might use discrete voltage pulses in a non-return-to-zero (NRZ) format to represent 0 and 1 bit states, while another might use analog audio tone frequency voltage signals to do the same. According to the OSI model, the choice over electrical signal types is a *Layer 1* matter. The patterns in which bit states make up larger serial data frames is another matter which must be standardized in any communication system, and according to the OSI model that is the domain of *Layer 2*. At the highest level of the OSI model we have *Layer 7* which regards the practical meaning of the bits communicated, whether those be alphanumeric codes representing an internet web page's contents, Modbus control functions, or anything else.

Layer 7 **Application**	This is where digital data takes on practical meaning in the context of some human or overall system function. *Examples: HTTP, FTP, HART, Modbus*
Layer 6 **Presentation**	This is where data gets converted between different formats. *Examples: ASCII, EBCDIC, MPEG, JPG, MP3*
Layer 5 **Session**	This is where "conversations" between digital devices are opened, closed, and otherwise managed for reliable data flow. *Examples: Sockets, NetBIOS*
Layer 4 **Transport**	This is where complete data transfer is handled, ensuring all data gets put together and error-checked before use. *Examples: TCP, UDP*
Layer 3 **Network**	This is where the system determines network-wide addresses, ensuring a means for data to get from one node to another. *Examples: IP, ARP*
Layer 2 **Data link**	This is where basic data transfer methods and sequences (frames) are defined within the smallest segment(s) of a network. *Examples: CSMA/CD, Token passing, Master/Slave*
Layer 1 **Physical**	This is where data bits are equated to electrical, optical, or other signals. Other physical details such as cable and connector types are also specified here. *Examples: EIA/TIA-232, 422, 485, Bell 202*

Students of philosophy might see a parallel between the OSI model's description of digital communication versus the ancient Greek philosopher Aristotle's classification of *causes*. Aristotle proposed four different ways to describe what causes a thing to exist or to occur. I have arranged these four causes in an order analogous to the OSI model's layers:

Aristotle's Four Causes

Layer 4 **Final**	This is the end-purpose for which something exists or has happened.
Layer 3 **Efficient**	This is the means by which something came into existence or occurred.
Layer 2 **Formal**	This is the structure or form taken by something.
Layer 1 **Material**	This is the fundamental substance something is made of.

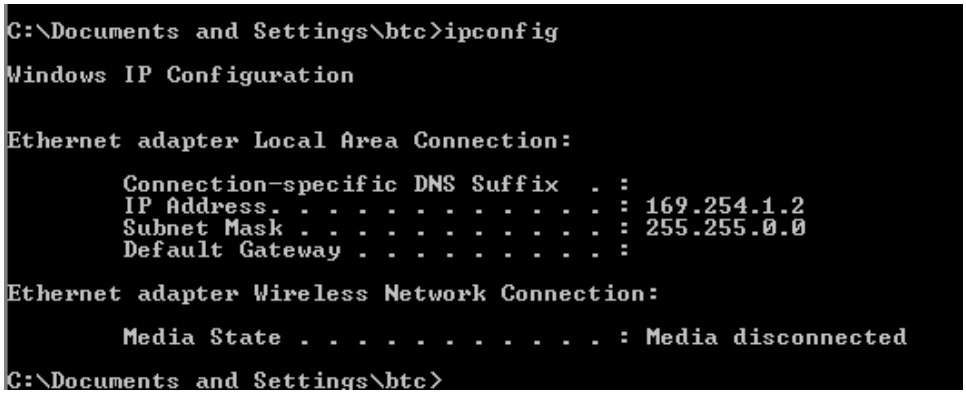
For example, we might classify the four causes of a textbook as follows:

- **Final cause** – to educate and enlighten the reader
- **Efficient cause** – a human author wrote the book, and it came into existence through the operation of a printing press
- **Formal cause** – a series of characters and images arranged into words, paragraphs, diagrams, graphs, equations, etc. following one or more standards of language (e.g. English, international math notation, Cartesian-coordinate graphs)
- **Material cause** – paper and ink

All of these causes describe different aspects of what was required for a book to come into existence. Similarly, with the OSI model, each level of the model is like one of Aristotle's causes in that it describes something necessary for the communication to take place.

4.2 Command-line diagnostic utilities

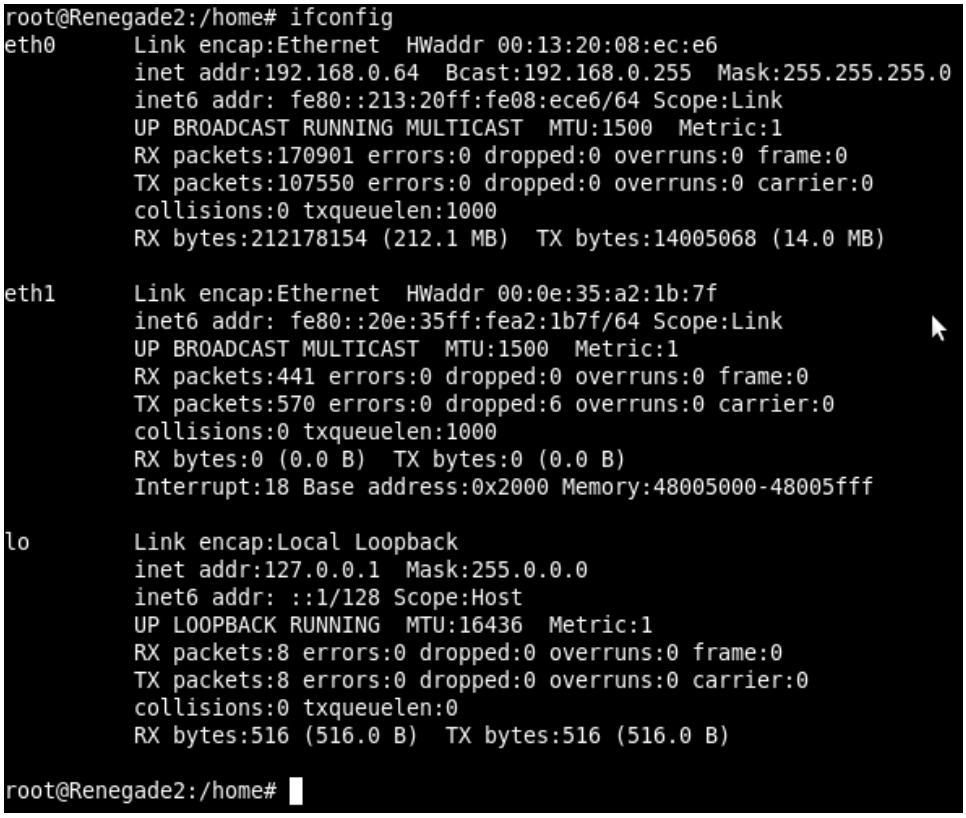
In addition to `ping` and `arp`, another utility program useful for troubleshooting network connections from a Microsoft Windows computer's command line interface[1] is `ipconfig`. When executed in a command-line terminal[2], `ipconfig` returns a listing of all available (configured and operating) network interfaces on that computer:



[1] In UNIX-based operating systems the program used to access the command line is often called `terminal` or `xterm`. In Microsoft Windows systems it is simply called `cmd`.

[2] In Microsoft Windows, the basic command-line terminal is located by typing `cmd` into the search field. `PowerShell` is a fuller-featured alternative to the basic command line terminal.

The equivalent command for UNIX operating systems is the legacy utility `ifconfig` shown in the screenshot below:



A more modern utility for UNIX operating systems is `ip`, the results of running the `ip a` command shown below:

```
warrior@stc237instruct:/mnt/z/.IETTI/Grade_CSV_files$ ip a
17: eth0: <> mtu 1500 group default qlen 1
    link/ether 84:1b:77:f2:9f:67
    inet 169.254.158.196/16 brd 169.254.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
15: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 group default qlen 1
    link/ether a4:bb:6d:b6:d5:7a
    inet 10.209.6.9/16 brd 10.209.255.255 scope global dynamic
       valid_lft 690885sec preferred_lft 690885sec
1: lo: <LOOPBACK,UP> mtu 1500 group default qlen 1
    link/loopback 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host dynamic
       valid_lft forever preferred_lft forever
12: wifi0: <> mtu 1500 group default qlen 1
    link/ieee802.11 84:1b:77:f2:9f:63
    inet 169.254.94.104/16 brd 169.254.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
14: wifi1: <> mtu 1500 group default qlen 1
    link/ieee802.11 84:1b:77:f2:9f:64
    inet 169.254.142.165/16 brd 169.254.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
7: wifi2: <> mtu 1500 group default qlen 1
    link/ieee802.11 86:1b:77:f2:9f:63
    inet 169.254.148.155/16 brd 169.254.255.255 scope global dynamic
       valid_lft forever preferred_lft forever
```

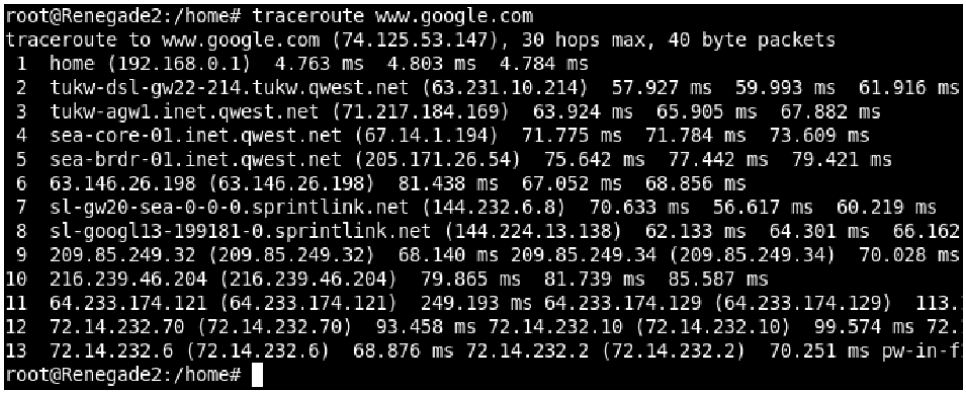
Some of the more interesting details contained in the outputs of both `ifconfig` and `ip` are the IPv6 addresses (in addition to IPv4 addresses), Ethernet MAC addresses (listed as hardware address "Hwaddr" or as "link/ether"), Ethernet performance data (e.g. number of collisions), IP performance data (e.g. number of IP packets received and transmitted), and details on the "loopback" address *lo* (IPv4 `127.0.0.1` or IPv6 `::1`).

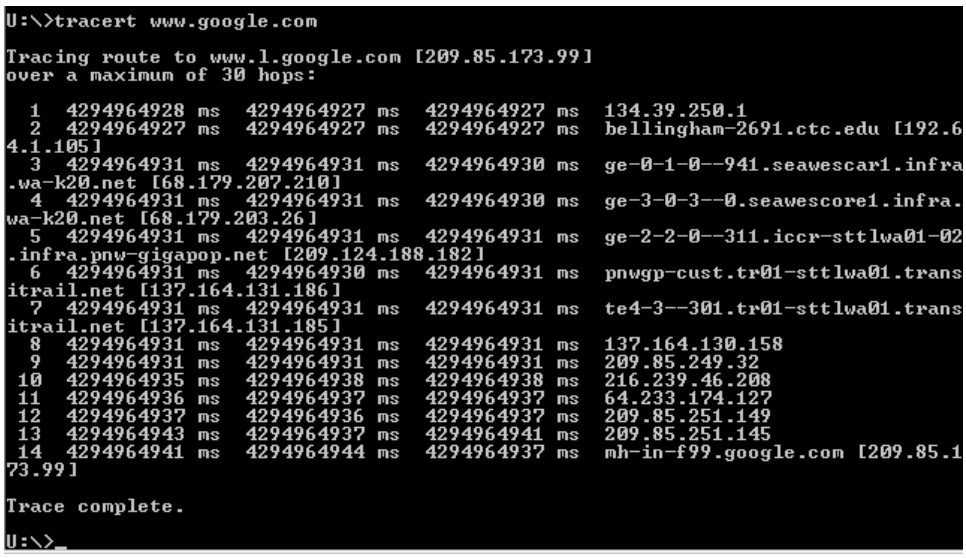
A utility intended to reveal the DNS name of a computer given its IP address, or visa versa, is `nslookup`. The same command works on Microsoft Windows and UNIX operating systems alike. Here, we see the UNIX version used to identify four IP addresses of the popular Google search engine web site, followed by the Microsoft Windows version:

```
root@Renegade2:/home# nslookup www.google.com
Server:         192.168.0.1
Address:        192.168.0.1#53

Non-authoritative answer:
www.google.com  canonical name = www.l.google.com.
Name:   www.l.google.com
Address: 74.125.53.103
Name:   www.l.google.com
Address: 74.125.53.147
Name:   www.l.google.com
Address: 74.125.53.99
Name:   www.l.google.com
Address: 74.125.53.104
```

```
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

U:\>nslookup www.google.com
Server:  btc2000-dc1.bellingham-tech.edu
Address:  172.16.0.240

Non-authoritative answer:
Name:    www.l.google.com
Addresses:  209.85.173.104, 209.85.173.103, 209.85.173.147, 209.85.173.99
Aliases:  www.google.com


U:\>_
```

Another utility used to explore network connections is `traceroute` (spelled `tracert` on Microsoft Windows operating systems). This utility sends a test packet to the designated destination address, returning information on all the "hops" the IP packet takes between computers along the network to reach its destination and the amount of time taken to make the trip. Execution of `traceroute` on a UNIX computer and `tracert` on a Microsoft Windows computer are shown here:





4.3 ASCII character codes

ASCII characters consist of seven-bit digital words. The following table shows all 128 possible combinations of these seven bits, from 0000000 (ASCII "NUL" character) to 1111111 (ASCII "DEL" character). For ease of organization, this table's columns represent the most-significant three bits of the seven-bit word, while the table's rows represent the least-significant four bits. For example, the capital letter "C" would be encoded as 1000011 in the ASCII standard.

↓ LSB / MSB →	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	”	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	ˆ	n	˜
1111	SI	US	/	?	O	_	o	DEL

It is worth noting that the ASCII codes for the Arabic numerals 0 through 9 are simply the four-bit binary representation of those numbers preceded by 011. For example, the number six (0110) is represented in ASCII as 0110110; the number three (0011) in ASCII as 0110011; etc. This is useful to know, for example, if you need to program a computer to convert single decimal digits to their corresponding ASCII codes: just take each four-bit numerical value and add forty-eight (0x30 in hexadecimal) to it.

Chapter 5

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

5.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

5.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

5.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Packet

Encapsulation

Ethernet

Internet Protocol (IP)

Transmission Control Protocol (TCP)

User Datagram Protocol (UDP)

Router

IP address

MAC address

Ping

Loopback address

Broadcast address

Subnetwork

Mask

Routing table

Address Resolution Protocol (ARP)

DNS

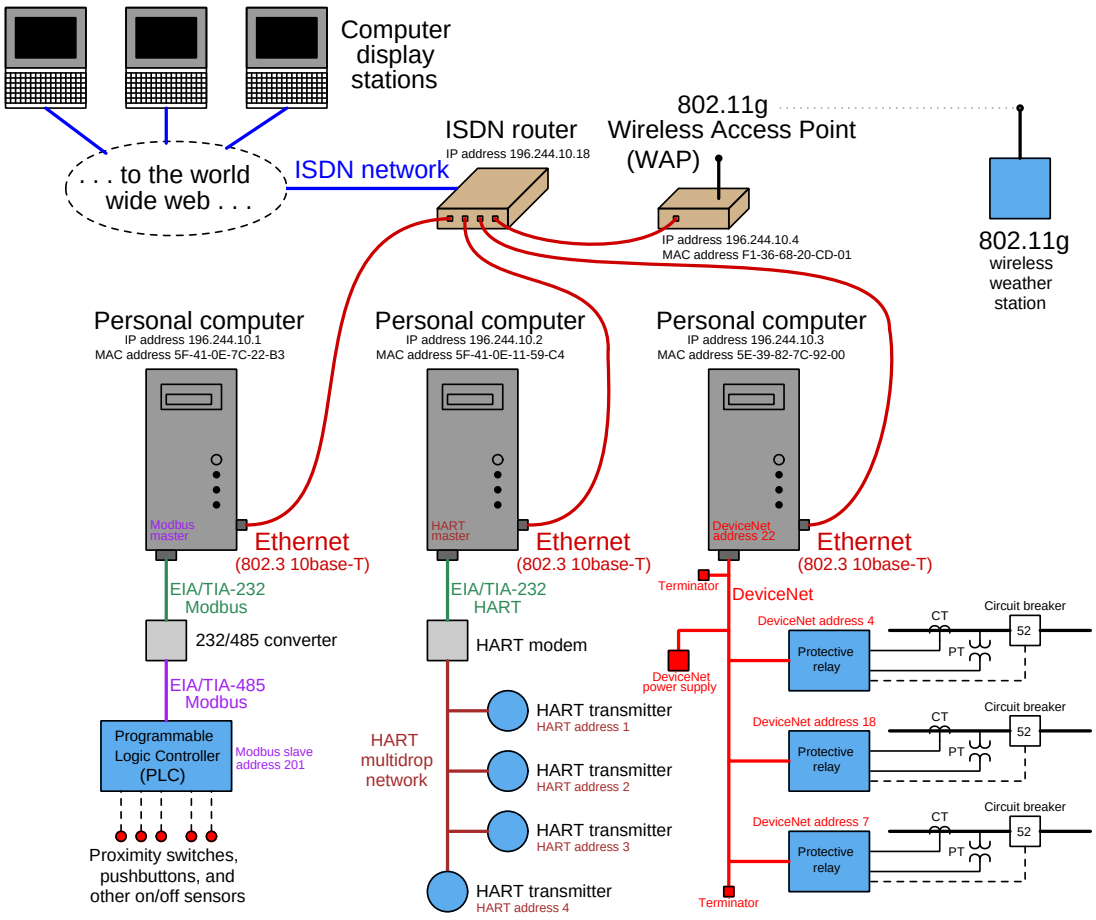
Netstat

Port

Unix

5.1.3 Industrial SCADA system

Examine this network diagram of an industrial SCADA (Supervisory Control And Data Acquisition) system, comprised of different technologies for acquiring the data, but ultimately communicating to computer display stations located somewhere on the Internet (world wide web):



Identify some of the different OSI layer 1 (physical) network standards you see in this system, as well as OSI layer 2 (data link) addressing schemes. Then, explain how all this data, in all its different forms, gets shuttled over the same ISDN cable to the Internet using TCP/IP packets.

Challenges

- Identify where the diagnostic utility `ping` could be used to test nodes in this heterogeneous network.

- Identify where the diagnostic utility `ping` could *not* be used to test nodes in this heterogeneous network.

5.1.4 Different protocols

One day an industrial electrician goes to connect a PLC (programmable logic controller) to an operator display panel and notices the communication ports on both the PLC and on the display panel are labeled *Modbus*, which the electrician figures is some sort of networking standard. Upon inspection of the screw terminals the electrician also notices the wiring is similar to RS-485 (EIA/TIA-485), with TD(+), TD(−), RD(+), and RD(−) terminals. Later on, when reading the user manuals for both devices, the electrician notices the ports described as being "RS-485" as well as being "Modbus."

Asking a more experienced electrician about this, the answer is that the ports are *both* EIA/TIA-485 and Modbus. These two network standards are not exclusive, but complementary.

Elsewhere in the world, a new computer network technician is going to download some software from a website, and notices it is possible to use either *HTTP* (Hyper-Text Transfer Protocol) or *FTP* (File Transfer Protocol) to do the job. Looking behind the computer, the technician notices a regular Ethernet cable (twisted-pair) plugged into the network port. Later, the technician asks someone more experienced, "Which network standard am I using when I download files, HTTP, FTP, or Ethernet?" The answer is similar to that given to the instrument technician: HTTP and FTP are alternatives to each other, both being complementary to Ethernet as parts of a complete protocol "pathway" from file to user. It is never a question of HTTP *or* Ethernet, just as it is never a question of Modbus *or* RS-485.

How would you explain either situation, using the OSI seven-layer model as a guide?

Challenges

- A commonly-heard criticism of the OSI model is that "no communications standard fully agrees with it," i.e. no single standard has specifications in all seven layers of the model. Explain why this is not really a problem at all, and how it represents a fundamental misunderstanding of the OSI model.
- *ASCII* is a layer-6 standard for encoding alphanumeric text in digital form. Give an example where this standard works in conjunction with lower-level standards to communicate text data between two computers.
- *S-HTTP* is a layer-7 standard used for encrypting and decrypting digital data over networks. This is what you are using when you access a web page beginning with `https://`. Give an example where this standard works in conjunction with ASCII and other lower-level standards to securely communicate a credit card number between two computers during an online purchase transaction.

5.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[4]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[5] on an answer key!

[4] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[5] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

5.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

5.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[6] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[6] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[7] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[8] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[9] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[7] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[8] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[9] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

[10] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

5.2.3 IP address space

The fourth version of the *IP* (Internet Protocol) standard, known as IPv4, specifies an address that is 32 bits wide. The address is usually expressed in the form of four "octets" translated into decimal form and separated by periods. Here is an example of an IPv4 address:

```
196.252.70.183
```

What are the largest and the smallest IPv4 addresses possible in this format? How many total unique addresses does this work out to be?

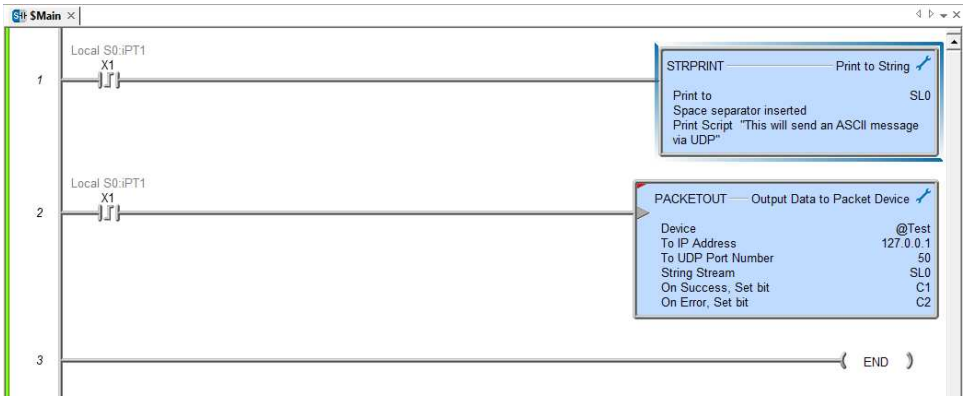
The next version of IP is version 6 (version 5 was experimental). IPv6 uses a 128-bit wide address. How many "octets" does it take to express an IPv6 address? How many unique addresses can be represented in the IPv6 field?

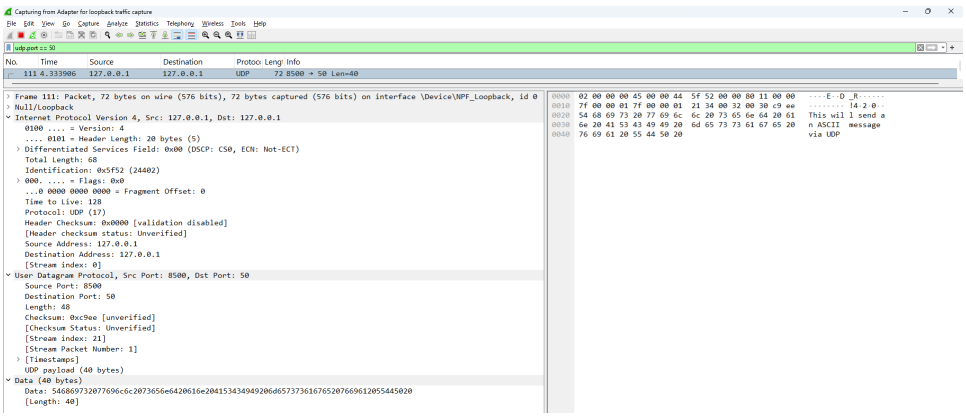
Challenges

- Are there any special IP addresses reserved for specific purposes?
- How do the address spaces of IPv4 and IPv6 compare to that of Ethernet MAC address space?
- Why are IP addresses required in addition to Ethernet MAC addresses in a network where most devices are Ethernet-based?
- Explain what `192.168.25.7/24` means as an IP address.
- Explain what `192.168.25.7/8` means as an IP address.

5.2.4 PLC transmitting ASCII message via UDP

The following ladder-diagram PLC program runs on a simulator, on a personal computer rather than on a real PLC. The program is designed to transmit an ASCII text message to IP loopback address 127.0.0.1 where it is monitored by `Wireshark` on that same computer:





Why do you suppose a rising-edge contact instruction is used to activate both the `STRPRINT` and `PACKETOUT` instructions?

Explain why it is useful to configure this PLC simulator's `PACKETOUT` instruction to send specifically to address 127.0.0.1.

For this example `Wireshark` has a filter parameter set, `udp.port == 50`. Explain why this is helpful when using `Wireshark` to monitor the simulated PLC's transmissions.

Where may we spot evidence of the *Space separator inserted* option of the `STRPRINT` instruction in action, from the data captured by `Wireshark`?

The `PACKETOUT` PLC instruction has an option for the IP address to be contained within a variable in the PLC's memory rather than have it be a fixed (constant) address as shown here.

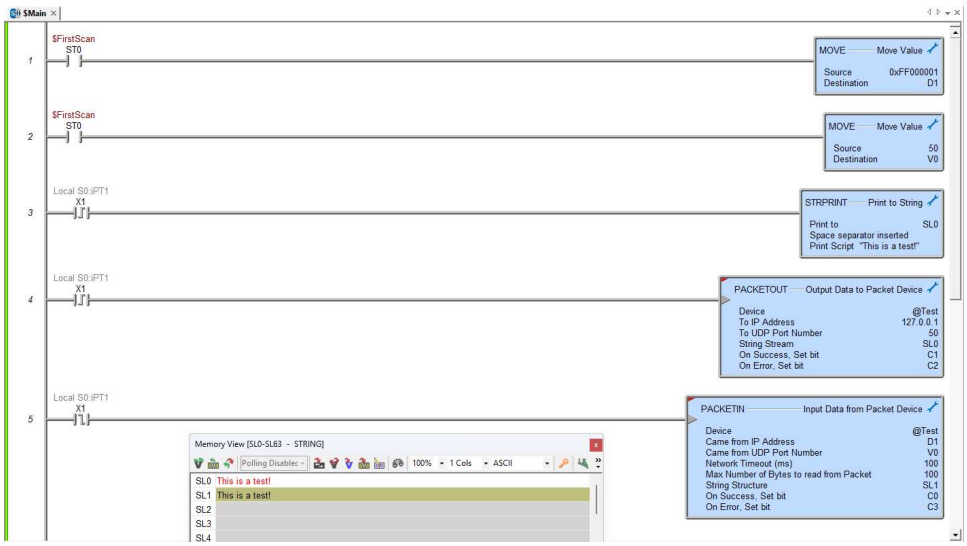
What type of variable do you think would work best for this, to contain an IPv4 address?

Challenges

- Show how to decode the ASCII text message from the hexadecimal characters shown by `Wireshark`.

5.2.5 PLC transmitting ASCII message to itself

The following ladder-diagram PLC program runs on a simulator, on a personal computer rather than on a real PLC. The program is designed to transmit an ASCII text message to IP address 127.0.0.1 where it is monitored by `Wireshark` on that same computer:



Explain how the IP address is set in this PLC's `PACKETIN` instruction.

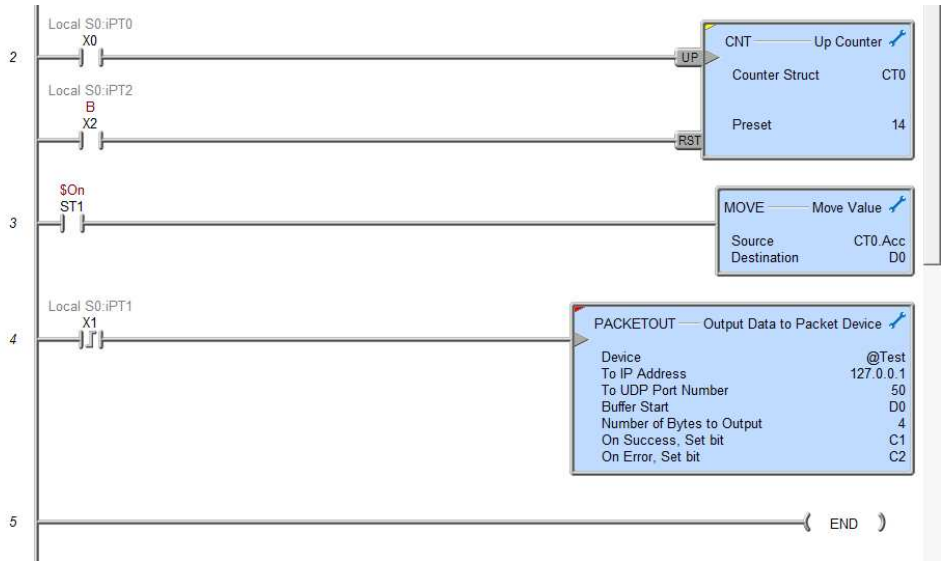
How is the PLC editor's *Memory View* feature being used to confirm the program is working as it should?

Challenges

- Show how to decode the ASCII text message from the hexadecimal characters shown by `Wireshark`.

5.2.6 PLC transmitting integer data over network

The following ladder-diagram PLC program runs on a simulator, on a personal computer rather than on a real PLC. The program is designed to transmit the accumulator value of its counter instruction to IP address 127.0.0.1 where it is monitored by `Wireshark` on that same computer:




```
0000  02 00 00 00 45 00 00 20  9e 42 00 00 80 11 00 00
0010  7f 00 00 01 7f 00 00 01  00 32 00 32 00 0c 54 7b
0020  9e f4 0e 00
```

In the `Wireshark` screen-capture shown above, the counter's accumulated value happened to be 980126 (decimal), this value highlighted in the hexdump (4 bytes representing the PLC's 32-bit counter value).

Several other counter values were experimentally tested using this same PLC program, with their respective data payloads shown in `Wireshark`. As with the previous example, the counter accumulator values are all shown in decimal, with the data payloads all shown in hexadecimal:

PLC counter accumulator value	**UDP data payload shown by Wireshark**
0	`0x 00 00 00 00`
23	`0x 17 00 00 00`
1024	`0x 00 04 00 00`
2048	`0x 00 08 00 00`
4096	`0x 00 10 00 00`
32767	`0x FF 7F 00 00`
32768	`0x 00 80 00 00`
65536	`0x 00 00 01 00`
980126	`0x 9E F4 0E 00`

Explain why the data payload hexadecimal values appear as they do, seemingly mis-matched to their respective decimal counter values.

Challenges

- Show how to decode the ASCII text message from the hexadecimal characters shown by `Wireshark`.

5.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

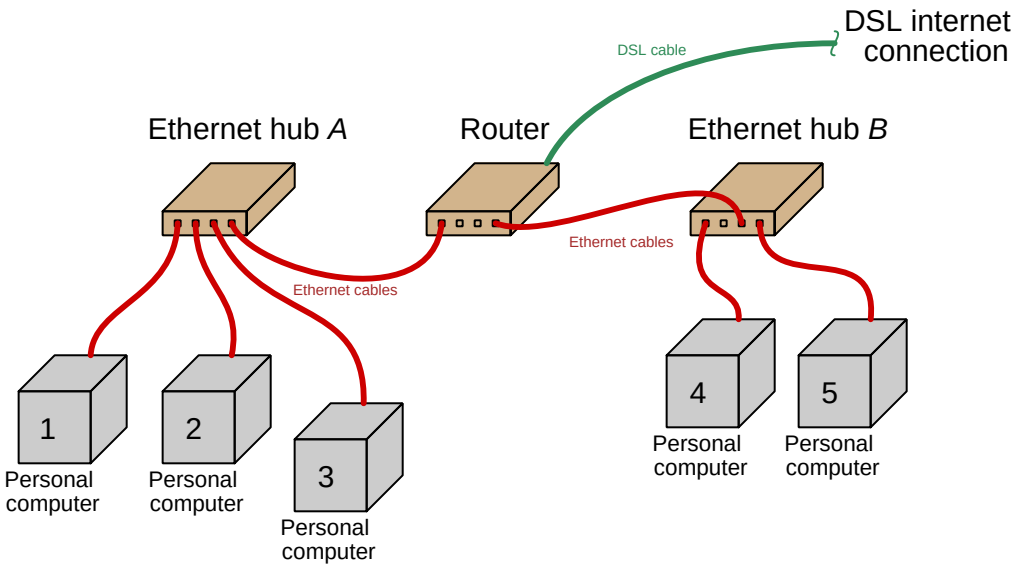
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

5.3.1 Tracing message routes

This Ethernet network has a problem in it somewhere:



- 1 can "ping" 3
- 4 can "ping" `www.google.com`
- 4 can "ping" 5
- 2 cannot "ping" `www.google.com`

An excellent diagnostic strategy is to trace paths of data flow in a complex system, looking for places of intersection. Successful paths prove all points along the pathway are functioning. Places of overlap between unsuccessful paths prove that section is suspect.

Apply this strategy to the problem at hand, and use the results to narrow the field of possible faults.

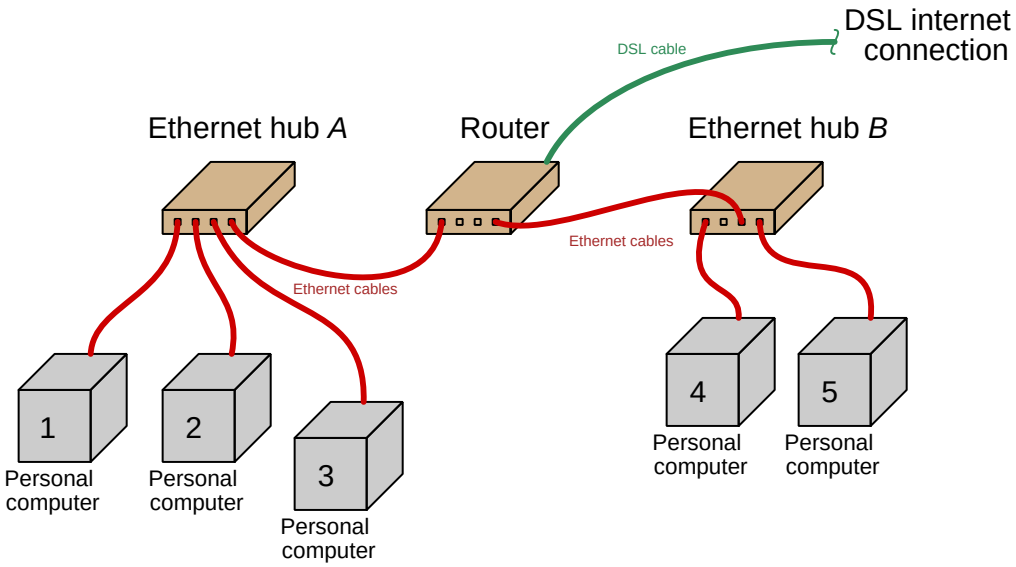
Finally, identify a good "next ping" test to do.

Challenges

- Suppose you were to swap hubs A and B, and repeat these same tests. What would this demonstrate, if anything?

5.3.2 Possible faults in a network

The following Ethernet network has a problem. Someone trying to access the Internet from personal computer #4 cannot do so, and has called you to troubleshoot the problem:



Your first diagnostic test is to "ping" computer #4 from computer #5, and you find that this test is successful. Your next test is to check Internet connectivity at computer #5 by "pinging" `http://www.google.com`, and you find that test is successful as well.

Identify the likelihood of each specified fault for this network. Consider each fault one at a time (i.e. no coincidental faults), determining whether or not each fault is compatible with *all* measurements and symptoms in this network.

- Hub A failed =
- Hub B failed =
- Router failed =
- Internet service provider failed =
- Cable failed between computer #4 and Hub B =
- Cable failed between Hub A and Router =
- Cable failed between Hub B and Router =
- Security settings (e.g. firewall) in computer #4 =

Finally, identify the *next* diagnostic test or measurement you would make on this system. Explain how the result(s) of this next test or measurement help further identify the location and/or nature of the fault.

Challenges

- Identify a case where `ping` would fail even though two computers connect to each other using proper cabling and non-faulted DCE devices.

Chapter 6

Projects and Experiments

The following project and experiment descriptions outline things you can build to help you understand circuits. With any real-world project or experiment there exists the potential for physical harm. *Electricity can be very dangerous in certain circumstances, and you should follow proper safety precautions at all times!*

6.1 Recommended practices

This section outlines some recommended practices for all circuits you design and construct.

6.1.1 Safety first!

Electricity, when passed through the human body, causes uncomfortable sensations and in large enough measures[1] will cause muscles to involuntarily contract. The overriding of your nervous system by the passage of electrical current through your body is particularly dangerous in regard to your heart, which is a vital muscle. Very large amounts of current can produce serious internal burns in addition to all the other effects.

Cardio-pulmonary resuscitation (CPR) is the standard first-aid for any victim of electrical shock. This is a very good skill to acquire if you intend to work with others on dangerous electrical circuits. You should never perform tests or work on such circuits unless someone else is present who is proficient in CPR.

As a general rule, any voltage in excess of 30 Volts poses a definitive electric shock hazard, because beyond this level human skin does not have enough resistance to safely limit current through the body. "Live" work of any kind with circuits over 30 volts should be avoided, and if unavoidable should only be done using electrically insulated tools and other protective equipment (e.g. insulating shoes and gloves). If you are unsure of the hazards, or feel unsafe at any time, stop all work and distance yourself from the circuit!

A policy I strongly recommend for students learning about electricity is to *never come into electrical contact*[2] *with an energized conductor, no matter what the circuit's voltage*[3] *level!* Enforcing this policy may seem ridiculous when the circuit in question is powered by a single battery smaller than the palm of your hand, but it is precisely this instilled habit which will save a person from bodily harm when working with more dangerous circuits. Experience has taught me that students who learn early on to be careless with safe circuits have a tendency to be careless later with dangerous circuits!

In addition to the electrical hazards of shock and burns, the construction of projects and running of experiments often poses other hazards such as working with hand and power tools, potential

[1] Professor Charles Dalziel published a research paper in 1961 called "The Deleterious Effects of Electric Shock" detailing the results of electric shock experiments with both human and animal subjects. The threshold of perception for human subjects holding a conductor in their hand was in the range of 1 milliampere of current (less than this for alternating current, and generally less for female subjects than for male). Loss of muscular control was exhibited by half of Dalziel's subjects at less than 10 milliamperes alternating current. Extreme pain, difficulty breathing, and loss of all muscular control occurred for over 99% of his subjects at direct currents less than 100 milliamperes and alternating currents less than 30 milliamperes. In summary, it doesn't require much electric current to induce painful and even life-threatening effects in the human body! Your first and best protection against electric shock is maintaining an insulating barrier between your body and the circuit in question, such that current from that circuit will be unable to flow through your body.

[2] By "electrical contact" I mean either directly touching an energized conductor with any part of your body, or indirectly touching it through a conductive tool. The only physical contact you should ever make with an energized conductor is via an electrically insulated tool, for example a screwdriver with an electrically insulated handle, or an insulated test probe for some instrument.

[3] Another reason for consistently enforcing this policy, even on low-voltage circuits, is due to the dangers that even some low-voltage circuits harbor. A single 12 Volt automobile battery, for example, can cause a surprising amount of damage if short-circuited simply due to the high current levels (i.e. very low internal resistance) it is capable of, even though the voltage level is too low to cause a shock through the skin. Mechanics wearing metal rings, for example, are at risk from severe burns if their rings happen to short-circuit such a battery! Furthermore, even when working on circuits that are simply too low-power (low voltage and low current) to cause any bodily harm, touching them while energized can pose a threat to the circuit components themselves. In summary, it's generally wise (and *always* a good habit to build) to "power down" *any* circuit before making contact between it and your body.

contact with high temperatures, potential chemical exposure, etc. You should never proceed with a project or experiment if you are unaware of proper tool use or lack basic protective measures (e.g. personal protective equipment such as safety glasses) against such hazards.

Some other safety-related practices should be followed as well:

- All power conductors extending outward from the project must be *firmly* strain-relieved (e.g. "cord grips" used on line power cords), so that an accidental tug or drop will not compromise circuit integrity.
- All electrical connections must be sound and appropriately made (e.g. soldered wire joints rather than twisted-and-taped; terminal blocks rather than solderless breadboards for high-current or high-voltage circuits). Use "touch-safe" terminal connections with recessed metal parts to minimize risk of accidental contact.
- Always provide overcurrent protection in any circuit you build. *Always.* This may be in the form of a fuse, a circuit breaker, and/or an electronically current-limited power supply.
- Always ensure circuit conductors are rated for more current than the overcurrent protection limit. *Always.* A fuse does no good if the wire or printed circuit board trace will "blow" before it does!
- Always bond metal enclosures to Earth ground for any line-powered circuit. *Always.* Ensuring an equipotential state between the enclosure and Earth by making the enclosure electrically common with Earth ground ensures no electric shock can occur simply by one's body bridging between the Earth and the enclosure.
- Avoid building a high-energy circuit when a low-energy circuit will suffice. For example, I always recommend beginning students power their first DC resistor circuits using small batteries rather than with line-powered DC power supplies. The intrinsic energy limitations of a dry-cell battery make accidents highly unlikely.
- Use line power receptacles that are GFCI (Ground Fault Current Interrupting) to help avoid electric shock from making accidental contact with a "hot" line conductor.
- Always wear eye protection when working with tools or live systems having the potential to eject material into the air. Examples of such activities include soldering, drilling, grinding, cutting, wire stripping, working on or near energized circuits, etc.
- Always use a step-stool or stepladder to reach high places. Never stand on something not designed to support a human load.
- When in doubt, *ask an expert.* If anything even seems remotely unsafe to you, do not proceed without consulting a trusted person fully knowledgeable in electrical safety.

6.1.2 Other helpful tips

Experience has shown the following practices to be very helpful, especially when students make their own component selections, to ensure the circuits will be well-behaved:

- Avoid resistor values less than 1 kΩ or greater than 100 kΩ, unless such values are definitely necessary[4]. Resistances below 1 kΩ may draw excessive current if directly connected to a voltage source of significant magnitude, and may also complicate the task of accurately measuring current since any ammeter's non-zero resistance inserted in series with a low-value circuit resistor will significantly alter the total resistance and thereby skew the measurement. Resistances above 100 kΩ may complicate the task of measuring voltage since any voltmeter's finite resistance connected in parallel with a high-value circuit resistor will significantly alter the total resistance and thereby skew the measurement. Similarly, AC circuit impedance values should be between 1 kΩ and 100 kΩ, and for all the same reasons.

- Ensure all electrical connections are low-resistance and physically rugged. For this reason, one should avoid *compression splices* (e.g. "butt" connectors), solderless breadboards[5], and wires that are simply twisted together.

- Build your circuit with **testing** in mind. For example, provide convenient connection points for test equipment (e.g. multimeters, oscilloscopes, signal generators, logic probes).

- Design permanent projects with **maintenance** in mind. The more convenient you make maintenance tasks, the more likely they will get done.

- **Always document and save your work**. Circuits lacking schematic diagrams are more difficult to troubleshoot than documented circuits. Similarly, circuit construction is simpler when a schematic diagram precedes construction. Experimental results are easier to interpret when comprehensively recorded. Consider modern videorecording technology for this purpose where appropriate.

- **Record your steps** when troubleshooting. **Talk to yourself** when solving problems. These simple steps clarify thought and simplify identification of errors.

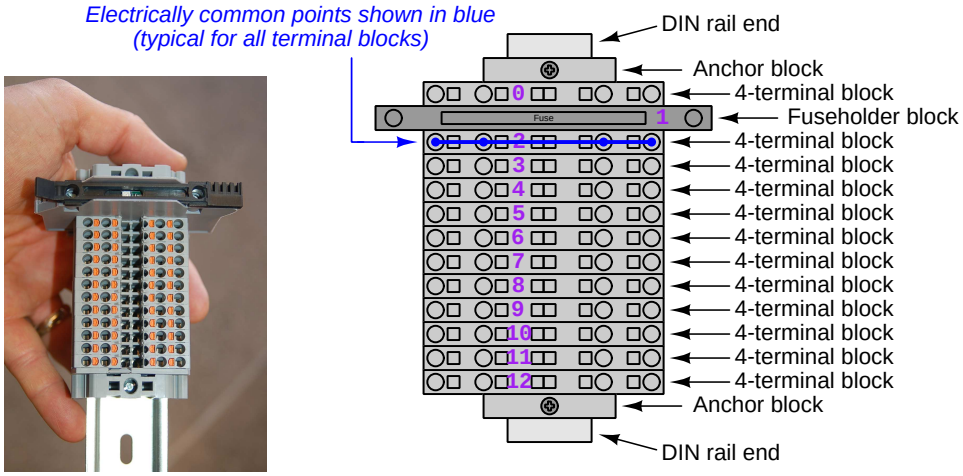
[4] An example of a necessary resistor value much less than 1 kΩ is a *shunt resistor* used to produce a small voltage drop for the purpose of sensing current in a circuit. Such shunt resistors must be low-value in order not to impose an undue load on the rest of the circuit. An example of a necessary resistor value much greater than 100 kΩ is an electrostatic *drain resistor* used to dissipate stored electric charges from body capacitance for the sake of preventing damage to sensitive semiconductor components, while also preventing a path for current that could be dangerous to the person (i.e. shock).

[5] Admittedly, solderless breadboards are very useful for constructing complex electronic circuits with many components, especially DIP-style integrated circuits (ICs), but they tend to give trouble with connection integrity after frequent use. An alternative for projects using low counts of ICs is to solder IC sockets into prototype printed circuit boards (PCBs) and run wires from the soldered pins of the IC sockets to terminal blocks where reliable temporary connections may be made.

6.1.3 Terminal blocks for circuit construction

Terminal blocks are the standard means for making electric circuit connections in industrial systems, and are also quite useful for student-built experiments, providing a robust alternative to solderless breadboards[6]. Terminal blocks provide highly reliable connections rated for significant voltage and current magnitudes, and they force the builder to think carefully about component placement which is an important mental practice. Terminal blocks designed to mount on standard 35 mm DIN rail[7] come in many types and capabilities.

Every student of electricity should build their own terminal block array consisting of several terminal blocks where each block has at least 4 connection points all electrically common to each other[8] and at least one terminal block that is a fuse holder for overcurrent protection. A pair of anchoring blocks hold all terminal blocks securely on the DIN rail, preventing them from sliding off the rail. Each of the terminals should bear a unique number starting from 0. An example is shown in the following photograph and illustration:



Screwless terminal blocks (using internal spring clips to clamp wire and component lead ends) are preferred over screw-based terminal blocks, as they reduce assembly and disassembly time, and also minimize repetitive wrist stress from twisting screwdrivers. Some screwless terminal blocks require the use of a special tool to release the spring clip, while others such as the Phoenix Contact model 3211019 blocks shown in the photograph provide buttons[9] for this task which may be pressed using the tip of any suitable tool such as a small screwdriver or multimeter test probe tip.

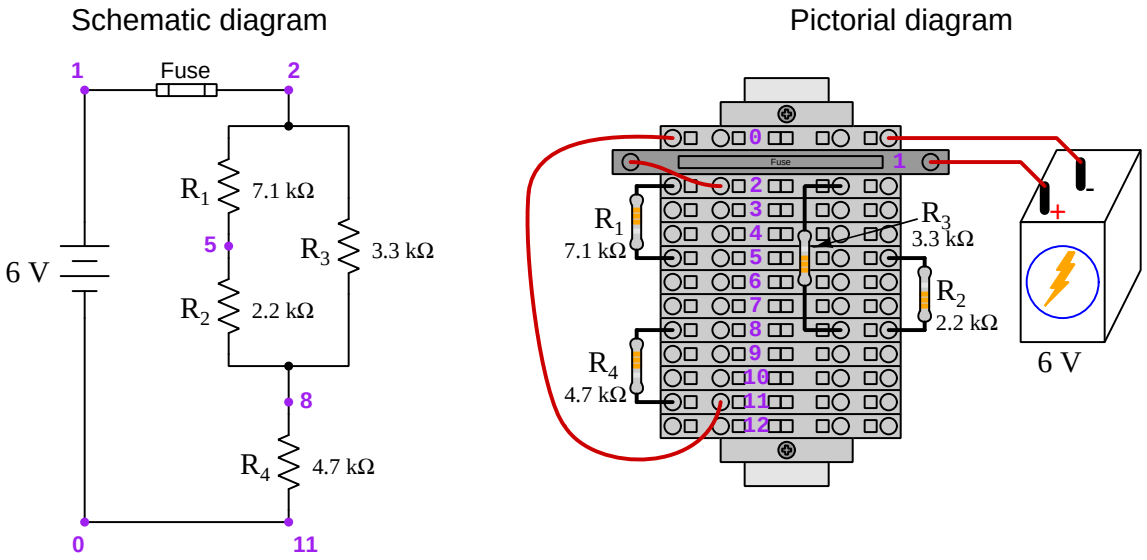
[6] Solderless breadboard are preferable for complicated electronic circuits with multiple integrated "chip" components, but for simpler circuits I find terminal blocks much more practical. An alternative to solderless breadboards for "chip" circuits is to solder chip sockets onto a PCB and then use wires to connect the socket pins to terminal blocks. This also accommodates *surface-mount* components, which solderless breadboards do not.

[7] DIN rail is a metal rail designed to serve as a mounting point for a wide range of electrical and electronic devices such as terminal blocks, fuses, circuit breakers, relay sockets, power supplies, data acquisition hardware, etc.

[8] Sometimes referred to as *equipotential*, *same-potential*, or *potential distribution* terminal blocks.

[9] The small orange-colored squares seen in the above photograph are buttons for this purpose, and may be actuated by pressing with any tool of suitable size.

The following example shows how such a terminal block array might be used to construct a series-parallel resistor circuit consisting of four resistors and a battery:



Numbering on the terminal blocks provides a very natural translation to SPICE[10] netlists, where component connections are identified by terminal number:

```
* Series-parallel resistor circuit
v1 1 0 dc 6
r1 2 5 7100
r2 5 8 2200
r3 2 8 3300
r4 8 11 4700
rfuse 1 2 0.01
rwire 0 11 0.01
.op
.end
```

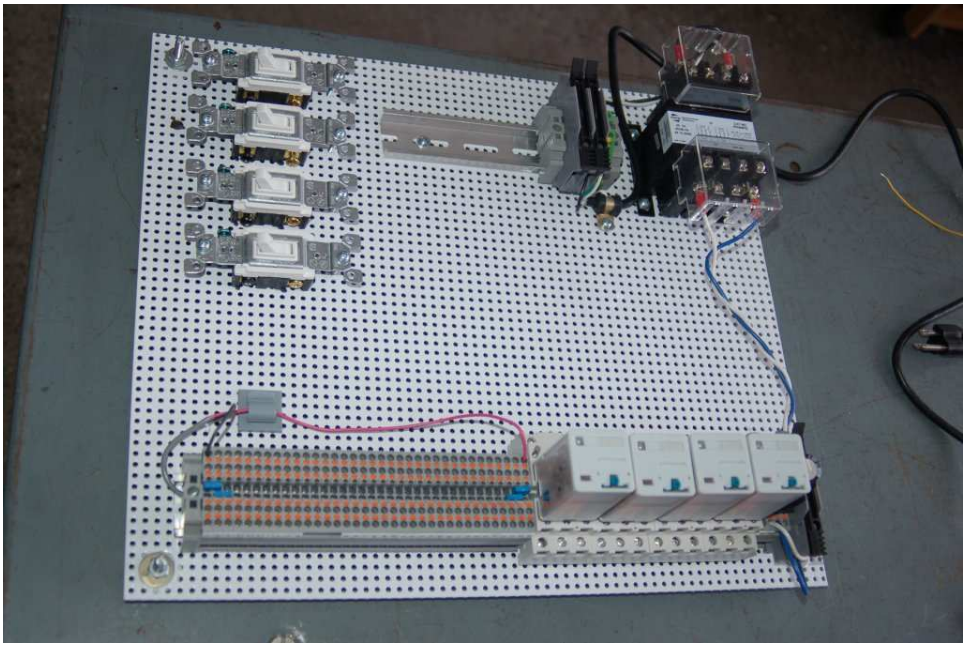
Note the use of two resistances `rfuse` and `rwire` to describe fuse resistance between terminals 1 and 2 and "jumper" wire resistance between terminals 0 and 11, respectively. Being resistances, SPICE requires a resistance value for each, and here we see they have both been set to an arbitrarily low value of 0.01 Ohm appropriate to their real-life resistance values.

Listing all components and wires along with their numbered terminals happens to be a useful documentation method for any circuit built on terminal blocks, independent of SPICE. Such a

[10] SPICE is computer software designed to analyze electrical and electronic circuits. Circuits are described for the computer in the form of *netlists* which are text files listing each component type, connection node numbers, and component values.

"wiring sequence" may be thought of as a *non-graphical description* of an electric circuit, and is exceptionally easy to follow.

An example of a more elaborate terminal block array is shown in the following photograph, with terminal blocks and "ice-cube" style electromechanical relays mounted to DIN rail, which is turn mounted to a perforated subpanel[11]. This "terminal block board" hosts an array of thirty five undedicated terminal block sections, four SPDT toggle switches, four DPDT "ice-cube" relays, a step-down control power transformer, bridge rectifier and filtering capacitor, and several fuses for overcurrent protection:



Four plastic-bottomed "feet" support the subpanel above the benchtop surface, and an unused section of DIN rail stands ready to accept other components. Safety features include electrical bonding of the AC line power cord's ground to the metal subpanel (and all metal DIN rails), mechanical strain relief for the power cord to isolate any cord tension from wire connections, clear plastic finger guards covering the transformer's screw terminals, as well as fused overcurrent protection for the 120 Volt AC line power and the transformer's 12 Volt AC output. The perforated holes happen to be on $\frac{1}{4}$ inch centers with a diameter suitable for tapping with 6-32 machine screw threads, their presence making it very easy to attach other sections of DIN rail, printed circuit boards, or specialized electrical components directly to the grounded metal subpanel. Such a "terminal block board" is an inexpensive[12] yet highly flexible means to construct physically robust circuits using industrial wiring practices.

[11] An electrical *subpanel* is a thin metal plate intended for mounting inside an electrical enclosure. Components are attached to the subpanel, and the subpanel in turn bolts inside the enclosure. Subpanels allow circuit construction outside the confines of the enclosure, which speeds assembly. In this particular usage there is no enclosure, as the subpanel is intended to be used as an open platform for the convenient construction of circuits on a benchtop by students. In essence, this is a modern version of the traditional *breadboard* which was literally a wooden board such as might be used for cutting loaves of bread, but which early electrical and electronic hobbyists used as platforms for the construction of circuits.

[12] At the time of this writing (2019) the cost to build this board is approximately $250 US dollars.

6.1.4 Conducting experiments

An *experiment* is an exploratory act, a test performed for the purpose of assessing some proposition or principle. Experiments are the foundation of the *scientific method*, a process by which careful observation helps guard against errors of speculation. All good experiments begin with an *hypothesis*, defined by the American Heritage Dictionary of the English Language as:

> An assertion subject to verification or proof, as (a) A proposition stated as a basis for argument or reasoning. (b) A premise from which a conclusion is drawn. (c) A conjecture that accounts, within a theory or ideational framework, for a set of facts and that can be used as a basis for further investigation.

Stated plainly, an hypothesis is an *educated guess* about cause and effect. The correctness of this initial guess matters little, because any well-designed experiment will reveal the truth of the matter. In fact, *incorrect* hypotheses are often the most valuable because the experiments they engender lead us to surprising discoveries. One of the beautiful aspects of science is that it is more focused on the process of *learning* than about the status of *being correct*[13]. In order for an hypothesis to be valid, it must be testable[14], which means it must be a claim possible to refute given the right data. Hypotheses impossible to critique are useless.

Once an hypothesis has been formulated, an experiment must be designed to test that hypothesis. A well-designed experiment requires careful regulation of all relevant variables, both for personal safety and for prompting the hypothesized results. If the effects of one particular variable are to be tested, the experiment must be run multiple times with different values of (only) that particular variable. The experiment set up with the "baseline" variable set is called the *control*, while the experiment set up with different value(s) is called the *test* or *experimental*.

For some hypotheses a viable alternative to a physical experiment is a *computer-simulated experiment* or even a *thought experiment*. Simulations performed on a computer test the hypothesis against the physical laws encoded within the computer simulation software, and are particularly useful for students learning new principles for which simulation software is readily available[15].

[13] Science is more about clarifying our view of the universe through a systematic process of error detection than it is about proving oneself to be right. Some scient*ists* may happen to have large egos – and this may have more to do with the ways in which large-scale scientific research is *funded* than anything else – but scientific *method* itself is devoid of ego, and if embraced as a practical philosophy is quite an effective stimulant for humility. Within the education system, scientific method is particularly valuable for helping students break free of the crippling fear of *being wrong*. So much emphasis is placed in formal education on assessing correct retention of facts that many students are fearful of saying or doing anything that might be perceived as a mistake, and of course making mistakes (i.e. having one's hypotheses disproven by experiment) is an indispensable tool for learning. Introducing science in the classroom – *real* science characterized by individuals forming actual hypotheses and testing those hypotheses by experiment – helps students become self-directed learners.

[14] This is the principle of *falsifiability*: that a scientific statement has value only insofar as it is liable to disproof given the requisite experimental evidence. Any claim that is unfalsifiable – that is, a claim which can *never* be disproven by any evidence whatsoever – could be completely wrong and we could never know it.

[15] A very pertinent example of this is learning how to analyze electric circuits using simulation software such as SPICE. A typical experimental cycle would proceed as follows: (1) Find or invent a circuit to analyze; (2) Apply your analytical knowledge to that circuit, predicting all voltages, currents, powers, etc. relevant to the concepts you are striving to master; (3) Run a simulation on that circuit, collecting "data" from the computer when complete; (4) Evaluate whether or not your hypotheses (i.e. predicted voltages, currents, etc.) agree with the computer-generated results; (5) If so, your analyses are (provisionally) correct – if not, examine your analyses and the computer simulation again to determine the source of error; (6) Repeat this process as many times as necessary until you achieve mastery.

Thought experiments are useful for detecting inconsistencies within your own understanding of some subject, rather than testing your understanding against physical reality.

Here are some general guidelines for conducting experiments:

- The clearer and more specific the hypothesis, the better. Vague or unfalsifiable hypotheses are useless because they will fit *any* experimental results, and therefore the experiment cannot teach you anything about the hypothesis.
- Collect as much data (i.e. information, measurements, sensory experiences) generated by an experiment as is practical. This includes the time and date of the experiment, too!
- *Never* discard or modify data gathered from an experiment. If you have reason to believe the data is unreliable, write notes to that effect, but never throw away data just because you think it is untrustworthy. It is quite possible that even "bad" data holds useful information, and that someone else may be able to uncover its value even if you do not.
- Prioritize *quantitative* data over *qualitative* data wherever practical. Quantitative data is more specific than qualitative, less prone to subjective interpretation on the part of the experimenter, and amenable to an arsenal of analytical methods (e.g. statistics).
- Guard against your own bias(es) by making your experimental results available to others. This allows other people to scrutinize your experimental design and collected data, for the purpose of detecting and correcting errors you may have missed. Document your experiment such that others may independently replicate it.
- Always be looking for sources of error. No physical measurement is perfect, and so it is impossible to achieve *exact* values for any variable. Quantify the amount of uncertainty (i.e. the "tolerance" of errors) whenever possible, and be sure your hypothesis does not depend on precision better than this!
- Always remember that scientific confirmation is provisional – no number of "successful" experiments will prove an hypothesis true for all time, but a single experiment can disprove it. Put into simpler terms, *truth is elusive but error is within reach.*
- Remember that scientific method is about *learning*, first and foremost. An unfortunate consequence of scientific triumph in modern society is that science is often viewed by non-practitioners as an unerring source of truth, when in fact science is an ongoing process of challenging existing ideas to probe for errors and oversights. This is why it is perfectly acceptable to have a failed hypothesis, and why the only truly failed experiment is one where nothing was learned.

The following is an example of a well-planned and executed experiment, in this case a physical experiment demonstrating Ohm's Law.

```
Planning Time/Date = 09:30 on 12 February 2019

HYPOTHESIS: the current through any resistor should be exactly proportional
to the voltage impressed across it.

PROCEDURE: connect a resistor rated 1 k Ohm and 1/4 Watt to a variable-voltage
DC power supply.  Use an ammeter in series to measure resistor current and
a voltmeter in parallel to measure resistor voltage.

RISKS AND MITIGATION: excessive power dissipation may harm the resistor and/
or pose a burn hazard, while excessive voltage poses an electric shock hazard.
30 Volts is a safe maximum voltage for laboratory practices, and according to
Joule's Law a 1000 Ohm resistor will dissipate 0.25 Watts at 15.81 Volts
(P = V^2 / R), so I will remain below 15 Volts just to be safe.

Experiment Time/Date = 10:15 on 12 February 2019

DATA COLLECTED:
  (Voltage)    (Current)         (Voltage)   (Current)
   0.000 V   =  0.000 mA          8.100    =  7.812 mA
   2.700 V   =  2.603 mA          10.00 V  =  9.643 mA
   5.400 V   =  5.206 mA          14.00 V  =  13.49 mA

Analysis Time/Date = 10:57 on 12 February 2019

ANALYSIS: current definitely increases with voltage, and although I expected
exactly one milliAmpere per Volt the actual current was usually less than
that.  The voltage/current ratios ranged from a low of 1036.87 (at 8.1 Volts)
to a high of 1037.81 (at 14 Volts), but this represents a variance of only
-0.0365% to +0.0541% from the average, indicating a very consistent
proportionality -- results consistent with Ohm's Law.

ERROR SOURCES: one major source of error is the resistor's value itself.  I
did not measure it, but simply assumed color bands of brown-black-red meant
exactly 1000 Ohms.  Based on the data I think the true resistance is closer
to 1037 Ohms.  Another possible explanation is multimeter calibration error.
However, neither explains the small positive and negative variances from the
average.  This might be due to electrical noise, a good test being to repeat
the same experiment to see if the variances are the same or different.  Noise
should generate slightly different results every time.
```

The following is an example of a well-planned and executed *virtual* experiment, in this case demonstrating Ohm's Law using a computer (SPICE) simulation.

```
Planning Time/Date = 12:32 on 14 February 2019

HYPOTHESIS: for any given resistor, the current through that resistor should be
exactly proportional to the voltage impressed across it.

PROCEDURE: write a SPICE netlist with a single DC voltage source and single
1000 Ohm resistor, then use NGSPICE version 26 to perform a "sweep" analysis
from 0 Volts to 25 Volts in 5 Volt increments.

    * SPICE circuit
    v1 1 0 dc
    r1 1 0 1000
    .dc v1 0 25 5
    .print dc v(1) i(v1)
    .end

RISKS AND MITIGATION: none.

DATA COLLECTED:
       DC transfer characteristic  Thu Feb 14 13:05:08  2019
    ------------------------------------------------------------
    Index   v-sweep         v(1)            v1#branch
    ------------------------------------------------------------
    0       0.000000e+00    0.000000e+00    0.000000e+00
    1       5.000000e+00    5.000000e+00    -5.00000e-03
    2       1.000000e+01    1.000000e+01    -1.00000e-02
    3       1.500000e+01    1.500000e+01    -1.50000e-02
    4       2.000000e+01    2.000000e+01    -2.00000e-02
    5       2.500000e+01    2.500000e+01    -2.50000e-02

Analysis Time/Date = 13:06 on 14 February 2019

ANALYSIS: perfect agreement between data and hypothesis -- current is precisely
1/1000 of the applied voltage for all values.  Anything other than perfect
agreement would have probably meant my netlist was incorrect.  The negative
current values surprised me, but it seems this is just how SPICE interprets
normal current through a DC voltage source.

ERROR SOURCES: none.
```

As gratuitous as it may seem to perform experiments on a physical law as well-established as Ohm's Law, even the examples listed previously demonstrate opportunity for real learning. In the physical experiment example, the student should identify and explain why their data does not perfectly agree with the hypothesis, and this leads them naturally to consider sources of error. In the computer-simulated experiment, the student is struck by SPICE's convention of denoting regular current through a DC voltage source as being *negative* in sign, and this is also useful knowledge for future simulations. Scientific experiments are most interesting when things *do not* go as planned!

Aside from verifying well-established physical laws, simple experiments are extremely useful as educational tools for a wide range of purposes, including:

- Component familiarization (e.g. *Which terminals of this switch connect to the NO versus NC contacts?*)
- System testing (e.g. *How heavy of a load can my AC-DC power supply source before the semiconductor components reach their thermal limits?*)
- Learning programming languages (e.g. *Let's try to set up an "up" counter function in this PLC!*)

Above all, the priority here is to inculcate the habit of hypothesizing, running experiments, and analyzing the results. This experimental cycle not only serves as an excellent method for self-directed learning, but it also works exceptionally well for troubleshooting faults in complex systems, and for these reasons should be a part of every technician's and every engineer's education.

6.1.5 Constructing projects

Designing, constructing, and testing projects is a very effective means of practical education. Within a formal educational setting, projects are generally chosen (or at least vetted) by an instructor to ensure they may be reasonably completed within the allotted time of a course or program of study, and that they sufficiently challenge the student to learn certain important principles. In a self-directed environment, projects are just as useful as a learning tool but there is some risk of unwittingly choosing a project beyond one's abilities, which can lead to frustration.

Here are some general guidelines for managing projects:

- Define your goal(s) before beginning a project: what do you wish to achieve in building it? What, exactly, should the completed project *do*?
- Analyze your project prior to construction. Document it in appropriate forms (e.g. schematic diagrams), predict its functionality, anticipate all associated risks. In other words, *plan ahead.*
- Set a reasonable budget for your project, and stay within it.
- Identify any deadlines, and set reasonable goals to meet those deadlines.
- Beware of *scope creep*: the tendency to modify the project's goals before it is complete.
- Document your progress! An easy way to do this is to use photography or videography: take photos and/or videos of your project as it progresses. Document failures as well as successes, because both are equally valuable from the perspective of learning.

6.2 Experiment: PLC data transmission using IP

Using either a PLC simulation[16] or a real PLC, conduct an experiment to test the behavior of a data-transmission instruction capable of sending arbitrary data packaged as a TCP or UDP message. Use `Wireshark` or some similar data-capture application to monitor the PLC program's output.

- What condition(s) cause the instruction to transmit data to the network?
- What options exist for IP addresses?
- What options exist for TCP or UDP port numbers?

Challenges

- ???.
- ???.

[16] If using a simulation on a PC, it is recommended to transmit to the loopback address (127.0.0.1), in which case you will need to configure the packet-sniffing application to monitor loopback traffic, and also you will benefit from setting a port number filter so that you only see those messages related to your simulated PLC and not other loopback traffic on your computer.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

B.1 First principles of learning

- **Anyone can learn anything** given appropriate time, effort, resources, challenges, encouragement, and expectations. Dedicating time and investing effort are the student's responsibility; providing resources, challenges, and encouragement are the teacher's responsibility; high expectations are a responsibility shared by both student and teacher.

- **Transfer is not automatic.** The human mind has a natural tendency to compartmentalize information, which means the process of taking knowledge learned in one context and applying it to another usually does not come easy and therefore should never be taken for granted.

- **Learning is iterative.** The human mind rarely learns anything perfectly on the first attempt. Anticipate mistakes and plan for multiple tries to achieve full understanding, using the lessons of those mistakes as feedback to guide future attempts.

- **Information is absorbed, but understanding is created.** Facts and procedures may be memorized easily enough by repeated exposure, but the ability to reliably apply principles to novel scenarios only comes through intense personal effort. This effort is fundamentally creative in nature: casting new concepts in one's own words, running mental experiments to perceive patterns and trends, etc. These acts of making knowledge "one's own" need not be perfect in order to be effective, as the value lies in the activity and not necessarily the finished product. Depth of understanding and permanence of memory are both directly related to mental effort during the learning process – i.e. *easy learning is shallow and fleeting.*

- **Education trumps training.** There is no such thing as an entirely isolated subject, as all fields of knowledge are connected. Training is narrowly-focused and task-oriented. Education is broad-based and principle-oriented. When preparing for a life-long technical career, education beats training every time.

- **Character matters.** Poor habits are more destructive than deficits of knowledge or skill. This is especially true in collective endeavors, where a team's ability to function depends on trust between its members. Simply put, no one wants an untrustworthy person on their team. An essential component of education then, is character development.

- **People learn to be responsible by bearing responsibility.** An irresponsible person is someone who has never *had* to be responsible for anything that mattered enough to them. Just as anyone can learn anything, anyone can become responsible if the personal cost of irresponsibility becomes high enough.

- **What gets measured, gets done.** Accurate and relevant assessment of learning is key to ensuring all students learn. Therefore, it is imperative to measure what matters.

- **Failure is nothing to fear.** Every human being fails, and fails in multiple ways at multiple times. Eventual success only happens when we don't stop trying.

B.2 Proven strategies for instructors

- Assume every student is capable of learning anything they desire given the proper conditions. Treat them as capable adults by granting real responsibility and avoiding artificial incentives such as merit or demerit points.
- Create a consistent culture of high expectations across the entire program of study. Demonstrate and encourage patience, persistence, and a healthy sense of self-skepticism. Anticipate and de-stigmatize error. Teach respect for the capabilities of others as well as respect for one's own fallibility.
- Replace lecture with "inverted" instruction, where students first encounter new concepts through reading and then spend class time in Socratic dialogue with the instructor exploring those concepts and solving problems individually. There is a world of difference between observing someone solve a problem versus actually solving a problem yourself, and so the point of this form of instruction is to place students in a position where they *cannot* passively observe.
- Require students to read extensively, write about what they learn, and dialogue with you and their peers to sharpen their understanding. Apply Francis Bacon's advice that "reading maketh a full man; conference a ready man; and writing an exact man". These are complementary activities helping students expand their confidence and abilities.
- Use artificial intelligence (AI) to challenge student understanding rather than merely provide information. Find productive ways for AI to critique students' clarity of thought and of expression, for example by employing AI as a Socratic-style interlocutor or as a reviewer of students' journals. Properly applied, AI has the ability to expand student access to critical review well outside the bounds of their instructor's reach.
- Build frequent and rapid feedback into the learning process so that students know at all times how well they are learning, to identify problems early and fix them before they grow. Model the intellectual habit of self-assessing and self-correcting your own understanding (i.e. a cognitive *feedback loop*), encouraging students to do the same.
- Use "mastery" as the standard for every assessment, which means the exam or experiment or project must be done with 100% competence in order to pass. Provide students with multiple opportunity for re-tries (different versions of the assessment every time).
- Require students to devise their own hypotheses and procedures on all experiments, so that the process is truly a scientific one. Have students assess their proposed experimental procedures for risk and devise mitigations for those risks. Let nothing be pre-designed about students' experiments other than a stated task (i.e. what principle the experiment shall test) at the start and a set of demonstrable knowledge and skill objectives at the end.
- Have students build as much of their lab equipment as possible: building power sources, building test assemblies[1], and building complete working systems (no kits!). In order to provide

[1] In the program I teach, every student builds their own "Development Board" consisting of a metal chassis with DIN rail, terminal blocks, and an AC-DC power supply of their own making which functions as a portable lab environment they can use at school as well as take home.

this same "ground-up" experience for every new student, this means either previous students take their creations with them, or the systems get disassembled in preparation for the new students, or the systems grow and evolve with each new student group.

- Incorporate external accountability for you and for your students, continuously improving the curriculum and your instructional methods based on proven results. Have students regularly network with active professionals through participation in advisory committee meetings, service projects, tours, jobshadows, internships, etc. Practical suggestions include requiring students to design and build projects for external clients (e.g. community groups, businesses, different departments within the institution), and also requiring students attend all technical advisory committee meetings and to dialogue with the industry representatives at those meetings.
- Repeatedly explore difficult-to-learn concepts across multiple courses, so that students have multiple opportunities to build their understanding.
- Relate all new concepts, whenever possible, to previous concepts and to relevant physical laws. Challenge each and every student, every day, to *reason* from concept to concept and to explain the logical connections between. Challenge students to verify their conclusions by multiple approaches (e.g. double-checking their work using different methods). Ask *"Why?"* often.
- Maintain detailed records on each student's performance and share these records privately with them. These records should include academic performance as well as professionally relevant behavioral tendencies.
- Hold mandatory "check-in" meetings between all program faculty and each new student during their first term. Offer these to all other students as an option, except for any students continuing to manifest unprofessional behaviors, poor academic performance, or who have some other need for a face-to-face meeting with faculty.
- Address problems while they are small, before they grow larger. This is equally true for tutoring technical concepts as it is for helping students build professional habits.
- Build rigorous quality control into the curriculum to ensure every student masters every important concept, and that the mastery is retained over time. This includes (1) review questions added to every exam to re-assess knowledge taught in previous terms, (2) cumulative exams at the end of every term to re-assess all important concepts back to the very beginning of the program, and (3) review assessments in practical (hands-on) coursework to ensure critically-important skills were indeed taught and are still retained. What you will find by doing this is that it actually boosts retention of students by ensuring that important knowledge gets taught and is retained over long spans of time. In the absence of such quality control, student learning and retention tends to be spotty and this contributes to drop-out and failure rates later in their education.
- Finally, *never rush learning.* Education is not a race. Give your students ample time to digest complex ideas, as you continually remind yourself of just how long it took you to achieve mastery! Long-term retention and the consistently correct application of concepts are always the result of *focused effort over long periods of time* which means there are no shortcuts to learning.

B.3 Proven strategies for students

The single most important piece of advice I have for any student of any subject is to take responsibility for your own development in all areas of life including mental development. Expecting others in your life to entirely guide your own development is a recipe for disappointment. This is just as true for students enrolled in formal learning institutions as it is for auto-didacts pursuing learning entirely on their own. Learning to think in new ways is key to being able to gainfully use information, to make informed decisions about your life, and to best serve those you care about. With this in mind, I offer the following advice to students:

- **Approach all learning as valuable**. No matter what course you take, no matter who you learn from, no matter the subject, there is something useful in every learning experience. If you don't see the value of every new experience, you are not looking closely enough!
- **Continually challenge yourself**. Let other people take shortcuts and find easy answers to easy problems. The purpose of education is to stretch your mind, in order to shape it into a more powerful tool. This doesn't come by taking the path of least resistance. An excellent analogy for an empowering education is productive physical exercise: becoming stronger, more flexible, and more persistent only comes through intense personal effort.
- **Master the use of language**. This includes reading extensively, writing every day, listening closely, and speaking articulately. To a great extent language channels and empowers thought, so the better you are at wielding language the better you will be at grasping abstract concepts and articulating them not only for your benefit but for others as well.
- **Do not limit yourself to the resources given to you**. Read books that are not on the reading list. Run experiments that aren't assigned to you. Form study groups outside of class. Take an entrepreneurial approach to your own education, as though it were a business you were building for your future benefit.
- **Express and share what you learn**. Take every opportunity to teach what you have learned to others, as this will not only help them but will also strengthen your own understanding[2].
- Realize that **no one can give you understanding**, just as no one can give you physical fitness. These both must be *built*.
- **Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable**. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[3] effort, and never give up! That concepts don't immediately come to you is not a sign of something wrong, but rather of something right: that you have found a worthy challenge!

[2] On a personal note, I was surprised to learn just how much my own understanding of electronics and related subjects was strengthened by becoming a teacher. When you are tasked every day with helping other people grasp complex topics, it catalyzes your own learning by giving you powerful incentives to study, to articulate your thoughts, and to reflect deeply on the process of learning.

[3] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

B.4 Design of these learning modules

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits. Every effort has been made to embed the following instructional and assessment philosophies within:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[4] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[5] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[6] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[4] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[5] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[6] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

To high standards of education,

Tony R. Kuphaldt

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

Giancoli, Douglas C., *Physics for Scientists & Engineers*, Third Edition, Prentice Hall, Upper Saddle River, NJ, 2000.

Horak, Ray, *Telecommunications and Data Communications Handbook*, John Wiley & Sons, Inc., New York, NY, 2007.

Horak, Ray, *Webster's New World Telecom Dictionary*, Wiley Publishing, Inc., Indianapolis, IN, 2008.

Newton, Harry, *Newton's Telecom Dictionary*, CMP Books, San Francisco, CA, 2005.

Postel, John, *Internet Protocol – DARPA Internet Program Protocol Specification*, RFC 791, Information Sciences Institute, University of Southern California, Marina Del Ray, CA, September 1981.

Postel, John, *Transmission Control Protocol – DARPA Internet Program Protocol Specification*, RFC 793, Information Sciences Institute, University of Southern California, Marina Del Ray, CA, September 1981.

Postel, John, *User Datagram Protocol*, RFC 768, Information Sciences Institute, University of Southern California, Marina Del Ray, CA, August 1980.

Spurgeon, Charles E., *Ethernet: The Definitive Guide*, O'Reilly Media, Inc., Sebastopol, CA, 2000.

Weiguo, Lin, "Address Resolution Protocol (ARP), RFC 826 ", College of Computing, CUC, 2009-2013.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

24 April 2026 – edited the Tutorial to mention `ip` as a newer version of `ifconfig` for Unix-type operating systems.

22 February 2026 – edited the Tutorial in several places for greater clarity.

5 January 2026 – added some text and color fill to image_3936 showing data encapsulation.

30 December 2025 – converted Tutorial subsections (on IP) into sections, and also corrected some typographical errors.

26 December 2025 – added some new Quantitative Reasoning questions based on a messages sent by a PLC. Also added the ASCII code reference table as a Technical Reference in this module.

13 February 2025 – added some comments to the Tutorial about OSI layers 3 and 4.

16-19 September 2024 – divided the Introduction chapter into sections, one with recommendations for students, one with a listing of challenging concepts, and one with recommendations for instructors. Also added some details on subnetting for IPv6.

15 February 2024 – minor edits to the text in the "Encapsulation" section of the Tutorial for clarity, and minor edits to the "Internet Protocol (IP)" section on subnet masks.

27 March 2023 – fixed a very simple typographical error in the Case Tutorial chapter, where I forgot a closing parenthesis.

29 November 2022 – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

5 June 2022 – added a Case Tutorial chapter.

10 May 2021 – commented out or deleted empty chapters.

5 October 2020 – added to the Introduction chapter.

5 September 2020 – significantly edited the Introduction chapter to make it more suitable as a pre-study guide and to provide cues useful to instructors leading "inverted" teaching sessions.

27 June 2020 – added another Foundational Concept.

25 June 2020 – added a Technical Reference section on the OSI Reference Model.

22 June 2020 – document first created.

Index

Adding quantities to a qualitative problem, 90
Address Resolution Protocol, 27, 36
Annotating diagrams, 72, 89
ARP, 27, 36, 37

Bitwise AND, 30
Breadboard, solderless, 78, 79
Breadboard, traditional, 82
Broadcast address, IP, 33

Cache, ARP, 36
Cardio-Pulmonary Resuscitation, 76
Checking for exceptions, 90
Checking your work, 90
Code, computer, 99
Command line terminal, Windows, 44
CPR, 76

Dalziel, Charles, 76
Data packet, 24
Department of Defense, US, 24
Destination host unreachable, error message, 32
DHCP, 29
Dimensional analysis, 89
DIN rail, 79
DIP, 78
DNR, 37
DNS, 37
Domain Name Resolver, 37
Domain Name Server, 37
Domain Name System, 37
Dynamic Host Configuration Protocol, 29

Echo Request, ICMP, 28
Edwards, Tim, 100
EIA/TIA-232 serial communication, 27, 39
Electric shock, 76
Electrically common points, 77
Enclosure, electrical, 82
Equipotential points, 77, 79
Experiment, 83
Experimental guidelines, 84

Google, Internet search engine, 47
Graph values to solve a problem, 90
Graphic User Interface, 29
Greenleaf, Cynthia, 51
GUI, 29

How to teach with these modules, 97
HTTP, 40
HTTPS, 40
Hwang, Andrew D., 101

IANA, 29
IC, 78
ICANN, 29, 37
ICMP, 28
Identify given data, 89
Identify relevant principles, 89
Ifconfig, utility program, 45
Intermediate results, 89
Internet Control Message Protocol, 28
Internet Protocol, 24, 26
Inverted instruction, 97
IP, 24, 26
IP packet, 25
Ip, utility program, 46
Ipconfig, utility program, 44
IPv4, 26
IPv6, 35

Knuth, Donald, 100

Lamport, Leslie, 100
Limiting cases, 90

Loopback address, 29

MAC address, Ethernet, 26
Mask, subnet, 30
Metacognition, 56
Modbus TCP, 40
Moolenaar, Bram, 99
Murphy, Lynn, 51

Netstat, utility program, 40

Open-source, 99
OSI reference model, 24

Packet switching, 24
Ping, utility program, 28, 30
Ping6, utility program, 35
Port number (OSI layer 4), 40
Postel, Jon, 29
Potential distribution, 79
PowerShell, Windows, 44
Problem-solving technique: divide and conquer, 28
Problem-solving: annotate diagrams, 72, 89
Problem-solving: check for exceptions, 90
Problem-solving: checking work, 90
Problem-solving: dimensional analysis, 89
Problem-solving: graph values, 90
Problem-solving: identify given data, 89
Problem-solving: identify relevant principles, 89
Problem-solving: interpret intermediate results, 89
Problem-solving: limiting cases, 90
Problem-solving: qualitative to quantitative, 90
Problem-solving: quantitative to qualitative, 90
Problem-solving: reductio ad absurdum, 90
Problem-solving: simplify the system, 89
Problem-solving: thought experiment, 84, 89
Problem-solving: track units of measurement, 89
Problem-solving: visually represent the system, 89
Problem-solving: work in reverse, 90
Project management guidelines, 87

Qualitatively approaching a quantitative problem, 90

Reading Apprenticeship, 51
Reductio ad absurdum, 90, 96, 97
Request timed out, error message, 32
Router, 26
RS-232 serial communication, 27, 39

Safety, electrical, 76
SCADA system, 59
Schoenbach, Ruth, 51
Scientific method, 56, 83
Scope creep, 87
Shunt resistor, 78
Simplifying a system, 89
SMTP, 40
Socket, 38
Socrates, 96
Socratic dialogue, 97
Solderless breadboard, 78, 79
SPICE, 51, 84
SPICE netlist, 80
SSH, 40
Stallman, Richard, 99
Subnet mask, 30
Subpanel, 82
Surface mount, 79

TCP, 24, 38
TCP segment, 25
TELNET, 40
Terminal block, 77–80, 82
Terminal Control Protocol, 24
Thought experiment, 84, 89
Torvalds, Linus, 99
Transmission Control Protocol, 38

UDP, 38, 39
Units of measurement, 89
User Datagram Protocol, 38, 39

Visualizing a system, 89

Windows PowerShell, 44
Wiring sequence, 81
Work in reverse to solve a problem, 90
WYSIWYG, 99, 100

X-windows, 29