COMPUTER NETWORKING REVIEW

EN.600.424

Fall 2016

Lecture Notes

OVERVIEW

- In order to understand network security, one has to understand networks/networking
- Computer networking is a really broad topic
  - Communication media
  - Communication protocols
  - Traffic engineering/theory
  - Architectures (e.g., Client/Server, P2P)
  - Applications
  - Etc, etc, etc
- Security issues stem from every single sub-topic
- But our review today focuses on the core topic of *protocols*

WHAT IS A PROTOCOL?

- A protocol is the set of rules that govern the interaction of two or more parties
- In the context of networking, it defines how two nodes communicate
  - When a party can communicate
  - What a party can communicate, *including message structure*
  - How a party responds to received communications
- The behavior is guaranteed when the rules are followed

OVERLOADED TERM

- Actually, a protocol often refers to two separate things
- **FIRST**, the rules/specification referred to on the previous slide
- **SECOND**, the computer module that *implements* the rules

COMMON CONTEMPORARY PROTOCOLS

- HTTP – HyperText Transfer Protocol
- IP – Internet Protocol
- SMTP – Simple Mail Transport Protocol

ONE PROTOCOL IS NOT ENOUGH

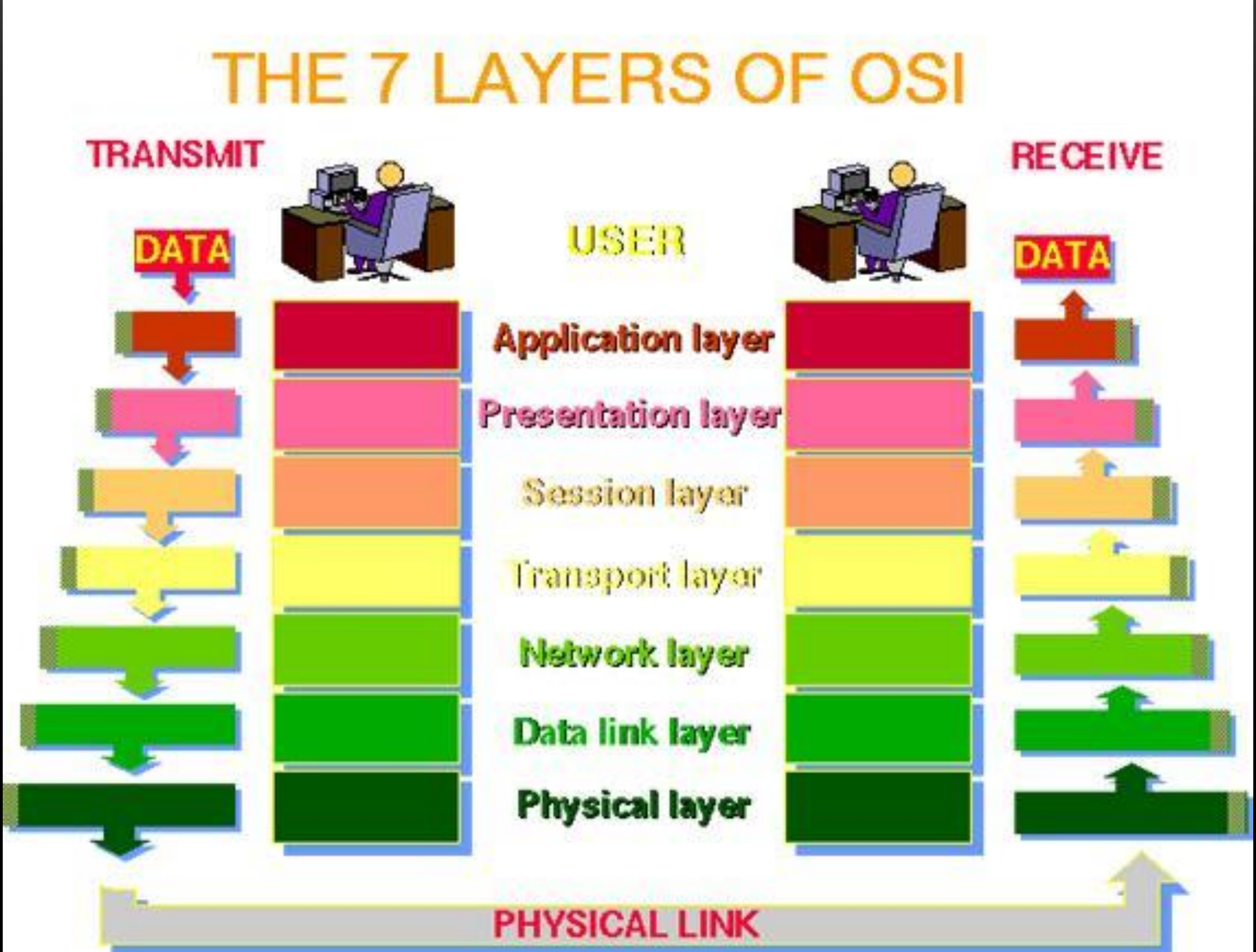
- There are too many rules for any one protocol to handle
- Also, behavior/rules need to change for different hardware/goals
- For example, consider HTTP
  - HTTP protocol shouldn’t need to worry about the IP protocol rules
  - HTTP definitely shouldn’t need to worry about Ethernet rules
  - And HTTP should work even after a switch from Ethernet to Wifi

PROTOCOL *STACKS*

- Object-oriented design has been around long before object-oriented programming
  - Modularity
  - Abstraction
  - Information hiding
- Protocols are designed in an object-oriented fashion
  - Protocols are combined to solve more complex problems
  - Each protocol should focus on one purpose/goal (High Cohesion)
  - Different component protocols can be swapped (Low coupling)
- We call a group of protocols that work together a *protocol stack*
- In computer networking, a *network protocol stack* or a *network stack*

OSI MODEL

- Good object-oriented design is implementation independent
- ISSO defined a guide for any given network stack called the OSI Model
- It has seven layers:
  - 7: Application
  - 6: Presentation
  - 5: Session
  - 4: Transport
  - 3: Network
  - 2: Data Link
  - 1: Physical


THE 7 LAYERS OF OSI
TRANSMIT
RECEIVE
DATA
USER
DATA
Application layer
Presentation layer
Session layer
Transport layer
Network layer
Data link layer
Physical layer
PHYSICAL LINK

THE OSI MODEL IN PRACTICE

- Like most OO-designs, the abstraction often breaks down
- Many stacks have multiple protocols in “one layer”, and none in another
- Modularity/abstraction/information hiding break down
- The TCP/IP stack really only uses the following layers:
  - Application (Layer 7; example: HTTP)
  - Transport (Layer 4; TCP)
  - IP (Layer 3; IP)
  - Data Link (Layer 2; example: Ethernet)
- NOTE: It’s common to just refer to a layer by it’s number (e.g., a layer-4 protocol)

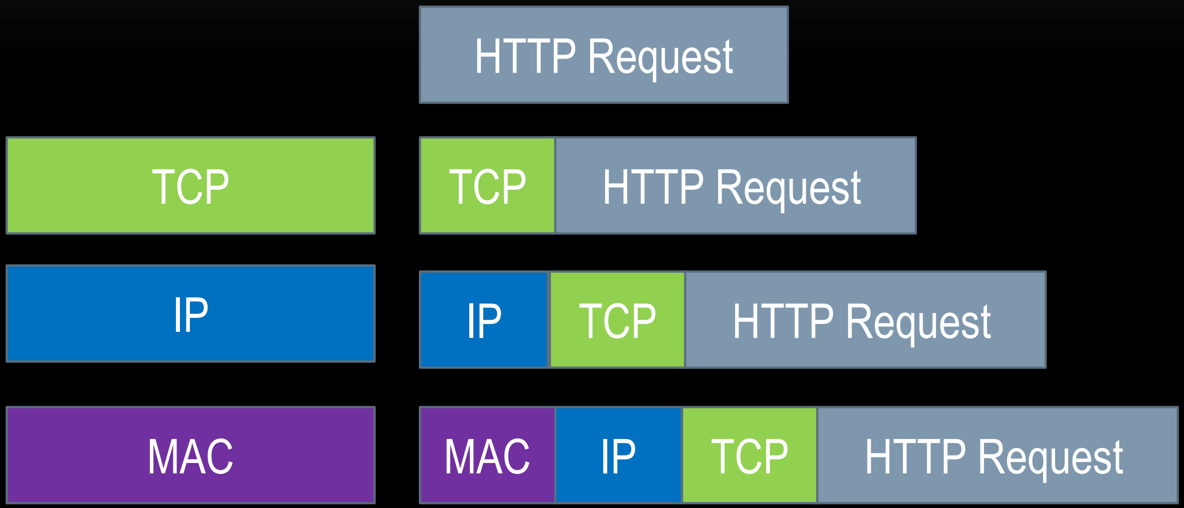
TCP/IP STACK

- For our purposes, we will focus on TCP/IP and TCP/IP-like stacks
- The TCP and IP layers are, obviously, fixed for layers 3 and 4.
- But layers 7 and 2 vary widely
- Millions of networked applications work over TCP/IP at layer 7
- Many layer 2 protocols such as WiFi, Ethernet
  - Networked applications work over WiFi or Ethernet without any change
  - Sometimes called a MAC protocol (Media Access Protocol)
  - TCP/IP work over a walkie-talkie with an appropriate MAC protocol

HOW DOES DATA MOVE IN A STACK?

- To send, data is inserted (pushed) at the top-most protocol
- The receiving protocol
  - Processes the data, potentially splitting, recoding, etc
  - Derives one or more chunks of data
  - Typically affixes a header to each, but sometimes a footer and/or other meta-data
  - Each chunk, along with the meta-data is a “packet”
  - The packet is inserted (pushed) down to the next layer
- When data is received, the process is reversed

TCP/IP STACK EXAMPLE



DIVISION OF LABOR IN TCP/IP

- At the lowest layer, the MAC protocol simply connects two endpoints. Typically:
  - Has its own addressing scheme (MAC address)
  - Controls who talks when
  - Provides error detection and *error correction*
- IP (Internetwork Protocol)
  - Connects many different networks of different media types
  - Global addressing scheme
- TCP
  - Reliable, in-order delivery (Session)
  - Multiplexing

INTEROPERABILITY

- No one company writes all TCP modules; How do they work together?
- Protocol specifications are approved by the IETF (Internet Engineering Task Force)
  - You can find the specifications in RFC’s (Request For Comments)
  - RFC 793 was the first specification of TCP (1981)
- So long as an implementation follows the spec, it will be interoperable

RFC 793 OVERVIEW

- Data broken into “segments” in section 2.2
- Network layers in section 2.5 (a little different from our usage)
- Section 2.6 lays out critical goal: Reliability
  - Data is delivered reliably (i.e., delivery is assured)
  - Data is delivered in-order
  - How? Sequence numbers and acknowledgements on segments
- Section 2.7 identifies another goal: Multiplexing
  - Different flows get different ports
- Section 2.8 indicates that this is a *stream* based protocol

TCP Header Format

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |          Source Port          |       Destination Port        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Sequence Number                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Acknowledgment Number                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Data |           |U|A|P|R|S|F|                               |
   | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
   |       |           |G|K|H|T|N|N|                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |         Urgent Pointer        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Options                    |    Padding    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             data                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                            TCP Header Format

          Note that one tick mark represents one bit position.

                               Figure 3.
```

PROTOCOLS AND STATE MACHINES

- It is often useful to model a protocol as a finite state machine (FSM)
  - The protocol starts in an initial state
  - When it receives data, it processes the data and moves to a new state
- For TCP, a state machine is defined in section 3.2
- If you don’t know what a FSM is, or how it works, you should probably look it up

PLAYGROUND NETWORKING

- We will be introducing the Playground Overlay network next class
- Key concepts
  - Playground is an overlay network
  - A Chaperone node connects one or more Playground Gates
  - Gates can communicate with each other using the Gate to Gate (G2G) Protocol
    - A combined layer-2/layer-3 protocol called the Gate to Gate Protocol
    - It DOES provide ports
    - It does NOT provide sessions, error correction, etc
- At the end of lab 1, you'll modify your webserver to work over playground
  - Even with just G2G, HTTP should work alright

LOOKING AHEAD TO LAB #2

- The writing assignment is to create an RFC for your own transport layer
  - It must provide reliable full-duplex communications
  - It must identify the beginning and ending of sessions
  - It must detect and correct errors (up to a certain error rate)
  - It does not provide ports
  - Your submission can be similar to TCP, but it can be quite different
- I will provide you with a tool that take in XML and produces an RFC-formatted doc
- One submission will be accepted as the class specification

IMPLEMENTING LAB #2

- Once a specification is selected, each student will implement this protocol
- The protocol will plug in to Playground Gates in a stack
- This will enable network applications to run over Playground
  - Even in the presence of errors
  - With clearly marked sessions
- I will provide you with helper classes for:
  - Creating stacked Twisted protocols
  - Defining, serializing and de-serializing packets

PACKET DEFINITION

- When working with low-level code like TCP, headers are a pain
- You have to read/write at the bit level… yuck
- To make this easier, Playground provides a simple method for packets
- You start by defining a packet structure using the MessageDefinition class

```
class MyPacket(MessageDefinition):
    PLAYGROUND_IDENTIFIER = "netsec.fall2016.MyPacket"
    PLAYGROUND_VERSION = "1.0"
    BODY = [ ("src", STRING),
             ("port", UINT2) ]
```

CREATING AND SERIALIZING PACKETS

```
packet = MyPacket()

packet.src = “20164.1.2.3”

packet.port = 80

transport.write(packet.__serialize__())
```

DESERIALIZING PACKETS

```
def recv(self, data):
       pkt = MyPacket.Deserialize(data)
       print pkt.src
       print pkt.port
```

IN YOUR PRFC

- Do not write a message header in your Playground RFC (PRFC)
- Instead, just write the message definition
- I’ll have some examples posted soon