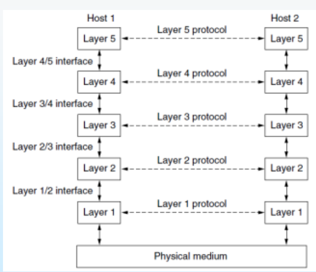
Network Software

- Communication Protocol Hierarchies
- Design Issues for the Layers (OSI Model)
- Connection-Oriented and Connectionless Services
- Service Primitives
- The Relationship of Services to Protocols

Protocol Hierarchies

- *"Abstraction—the hiding of details behind a well-defined interface—is the fundamental tool used by system designers to manage complexity"*
  Larry L. Peterson and Bruce S. Davie, Computer Networks
- To reduce design complexity networks are organized as a stack of layers
- The purpose of each layer is to offer certain services to the higher layers while shielding those layers from the details of how the offered services are actually implemented
- AKA: information hiding, abstract data types, data encapsulation, and object-oriented programming
- Conversation between layer *n* on one machine with layer *n* on another machine: the rules and conventions used in this conversation are collectively known as the **layer *n* protocol**

Layers, protocols, and interfaces

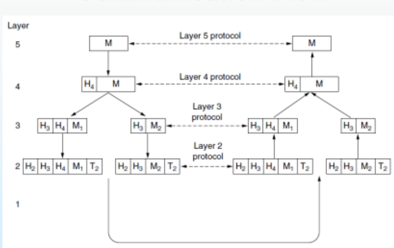


Real data is transfered only at the physical layer!
All other dotted lines are virtual!

Network Architecture

- A set of layers and protocols is called a **network architecture**
- Neither the details of the implementation nor the specification of the interfaces is part of the architecture
- A list of the protocols used by a certain system, one protocol per layer, is called a **protocol stack**
- **Typical flow:**
  - A message, *M*, is produced by an application process running in layer 5 and given to layer 4 for transmission
  - Layer 4 puts a **header** in front of the message to identify the message and passes the result to layer 3
  - The header includes control information, such as address/port, to allow layer 4 on the destination machine to deliver the message
  - Other examples of control information used in some layers are sequence numbers, sizes, and times
  - layer 3 must break up the incoming messages into smaller units, packets, prepending a layer 3 header to each packet

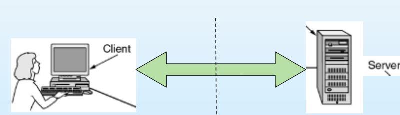
Communication Flow



- Layer 3 decides which lines to use and passes the packets to layer 2
- Layer 2 adds to each piece not only a **header** but also a **trailer**, and gives the resulting unit to layer 1 for physical transmission
- At the receiving machine the message moves upward, from layer to layer, with headers being stripped off as it progresses

Communication Protocol

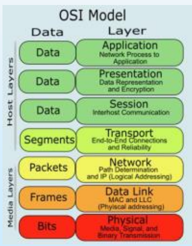
- Definition 1: A **protocol** is an agreement between the communicating parties on how communication is to proceed
- Definition 2: A **protocol** is a set of communication "rules" between two processes.
- Example: A "grades database query" protocol
  - (We may make a small project out of it later …)

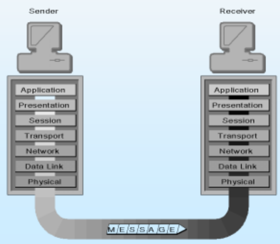


Client: HELLO	Server: READY
Client: NAME 051883261\n	Server: DAN HACKER\n
Client: GRADE MATH\n	Server: 87\n
Client: GRADE HISTORY\n	Server: 93\n
Client: END	Server: BYE

OSI Model

- **Open Systems Interconnection (OSI)**
- Proposed by the **International Standards Organization** (ISO)
- The OSI model has seven layers





Application Layer

- The closest layer to the user: Outlook, Explorer, Firefox, Skype (HTTP, POP, SMTP, FTP, TELNET).
- In this layer that a user interacts with the software application that does data transfer
- The main tasks:
  - Identify/authenticate the user who wants to communicate
  - determine whether the data and networks sources are available
  - synchronize communication between the two nodes

Presentation Layer

- Convert the data into a format that could be easily recognized by the application layers of other end users.
- For example: translation between ASCII and EBCDIC machines as well as between different floating point and binary formats. Integer size (16,32, or 64 bit?). Floating point representations.
- Compression/decompression, conversion, encryption/decryption, coding, decoding, etc.
- Converts the data obtained from the application layer into a format that can be easily identified by other network layers.

Session Layer

- In practice, this layer is often not used or services within this layer are sometimes incorporated into the transport layer
- Establishing, maintaining and terminating the connection between two end nodes (not used in TCP/IP)
- Controls the communication between the source user and the destination user and also decides the time of communication
- It determines one-way or two-way communications and manages the dialog between both parties; for example, making sure that the previous request has been fulfilled before the next one is sent
- Any error report related to application layer, presentation layer and session layer, are provided by this layer

Transport Layer

- Responsible for delivering the data or the messages between the two nodes
- Divide the data in packets at the sender side
- Re-assemble packets at the receiver side
- Third task: error free data transmission
  - Uses checksums for error correction or rejection
  - Drop corrupt packets and requests retransmission
- Fourth task: guarantee data integrity
  - Make sure all packets have arrived
- UDP, SPX, TCP are some of the protocols that operate on this layer with one exception: UDP is unreliable

Network Layer

- Provide switching technologies and routing technologies:
  It is the *network* layer's job to figure out the *network* topology, handle routing and to prepare data for transmission
- Establishes the route between the sending and receiving nodes for data transmission (also known as virtual circuits)
- Encapsulation of transport data into network layer protocol data units
- Also responsible for handling errors, packet sequencing, controlling network congestion and addressing
- In short: this layer is responsible for the setting up the required network for transferring data from one node to other.

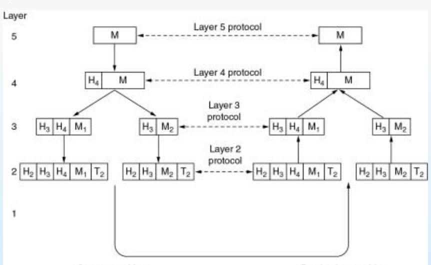
Data Link Layer

- Encoding and decoding of data frames into bits (as the physical layer may use waves or other type of media). At the receiving side: Collects a stream of bits into a larger aggregate called a *frame.*
- Segmentation of upper layer datagrams (packets) into frames in sizes that can be handled by the communications hardware
- Takes care of any errors in the physical layer (electricity presence, voltage drop, no power, connection, etc.)
- Provides reliable transit of the data through a physical network
- Synchronization of various physical devices that will transmit the data
- It makes sure that the frames are transferred in correct order and asks for retransmission in case of error
- The frame formatting issues such as stop and start bits, bit order, parity and other functions are handled here. Management of big-endian/little-endian issues are also managed at this layer.
- Usually implemented on Hardware (network interface card):

Physical Layer

- Deals with the physical components of a network
- Activation, maintenance and deactivation of various physical links that act in data transmission
- Electrical signals, voltage levels, cables, data transmission rates, etc., are some of the major elements defined by the physical layer
- It is also responsible for passing and receiving bytes from the physically connected medium
- Implemented on hardware (network interface card)

Information Flow



The peer processes in layer 4 (for example) conceptually think of their communication as being "horizontal," using the layer 4 protocol
Each one is likely to have procedures called something like *SendToOtherSide* and *GetFromOtherSide*, even though these procedures actually communicate with lower layers across the 3/4 interface, and not with the other side.

Design Issues - Accuracy

- Packet traveling through the network: there is a chance that some bits will be flipped, or even get lost, or new ones will be added:
  - fluke electrical noise
  - random wireless signals
  - hardware flaws
  - software bugs (and so on …)
- Is it possible to detect and even fix these errors?
- Must separate between two targets:
- **Error Detection**
  - Easy mechanisms for detecting errors (with very high probability)
- **Error Correction**
  - This is possible but very costly (space, time, resources)

Design Issues - Reliability

- Finding a working path through a network:
  - Usually there are multiple paths between a source and destination
- In a large network, there may be broken links, hosts, and routers
- If the network is down in Germany: packets sent from London to Rome via Germany will not get through, but packets from London to Rome via Paris may get through … ?
- A network should automatically detect the problem and make this decision. This topic is called **routing**. How this is done? We'll see later …
- Not all communication channels preserve the order of packets sent on them, and packets can also get completely lost

Design Issues – Flow Control

- **Congestion**: how to keep a fast sender from swamping a slow receiver?
- Overloading of the network is called **congestion:** too many computers want to send too much traffic, and the network cannot deliver it all
- One strategy is for each computer to reduce its demand when it experiences congestion
- **Starvation**: fast receivers against slow senders (fast clients vs. slow server)
- **Quality of service** is the name given to mechanisms that reconcile these competing demands
- Applications: video streaming, VOIP, media recorders ("buffer overrun")
  - Balancing senders and receivers speeds in such cases is very crucial

Design Issues – Security

- Network must be secured by defending it against different kinds of threats:
- **Confidentiality:** prevent unauthorized access to information (snooping)
- **Authentication:** prevent someone from impersonating someone else (Phishing)
- **Integrity:** prevent surreptitious changes to messages: ''debit my account $10'' → ''debit my account $1000''
- Solution designs are heavily based on **cryptography**

Connection-Oriented and Connectionless Services

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Remote login
	Unreliable connection	Digitized voice
Connection-less	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Registered mail
	Request-reply	Database query

Connection-Oriented

- Connection is established, the sender, receiver, and subnet conduct a **negotiation** about the parameters to be used, such as
  - Maximum message size
  - Quality of service required, and other issues
- Typically, one side makes a proposal and the other side can accept it, reject it, or make a counter proposal.
- A **circuit** is another name for a connection with associated resources (after the telephone model …)
- Reliability: do not lose data – e.g., the receiver acknowledge the receipt of each message
- so the sender is sure that it arrived
- TCP – Transmission Control Protocol is connection oriented
- Text documents, email, image attachments

Connectionless Service

- In contrast to connection-oriented service, **connectionless** service is modeled after the postal system
- Each message (letter/package) carries the full destination address and each one is routed through the intermediate nodes inside the system independent of all the subsequent messages
- UDP – User Datagram Protocol – unreliable
- Unreliable (meaning not acknowledged) connectionless service is often called **datagram** service, in analogy with telegram (service, which also does not return an acknowledgement to the sender)
- Video streaming, Video conference, VOIP, Digital TV transmission (Idan+)

Co-existence of both kinds

- reliable communication may not be available in a given layer
- For example, Ethernet does not provide reliable communication. Packets can occasionally be damaged in transit
- It is up to higher protocol levels to recover from this problem. In particular, many reliable services are built on top of an unreliable datagram service. Second,
- Both reliable and unreliable communication usually coexist.

Connection-oriented Service Primitives

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

- Minimal example of service primitives that provide a reliable byte stream
- A service is formally specified by a set of **primitives** (operations) available to user processes to access the service
- These primitives tell the service to perform some action or report on an action taken by a peer entity (usually as operating system calls)
- Modeled after the Berkeley socket interface

Service Primitives (2)

- **LISTEN** is usually implemented by a block system call - the server process is blocked until a request for connection appears
- **CONNECT** is usually implemented by a connection request to a server
  - The **CONNECT** call may need to specify the server's address
  - The operating system then typically sends a packet to the peer asking it to connect
- The client process is suspended until there is a response
- When the packet arrives at the server, the operating system sees that the packet is requesting a connection
  - It checks to see if there is a listener
  - If so it unblocks the listener (wake-up call)
  - The server process may accept the connection with the **ACCEPT** call
- This sends a response back to the client process to accept the connection

Service Primitives (3)

- Next step: **RECEIVE**
  - The server prepares to accept the first client request
  - The **RECEIVE** call blocks the server
- Then the client executes **SEND** to transmit its request (data or action) followed by the execution of **RECEIVE** by the server (and then blocks)
- The arrival of the request packet at the server machine unblocks the server so it can handle the request
- After it has done the work, the server uses **SEND** to return the answer to the client
- The arrival of this packet unblocks the client, which can now inspect the answer. If the client has additional requests, it can proceed immediately.
- When the client is done, it executes **DISCONNECT** to terminate the connection. Usually, a **DISCONNECT** is a blocking call, suspending the client and sending a packet to the server saying that the connection is no longer needed

Service Primitives (4)

- When the server gets the client disconnect packet, it also issues a server **DISCONNECT** of its own, acknowledging the client and releasing the connection
- When the server's packet gets back to the client machine, the client process is released and the connection is broken
- In a nutshell, this is how connection-oriented communication works: