# Telematics I

## Chapter 2
## Network Architecture – Services, Protocols & Layers

- Structuring communication systems into layers
- Services
- Protocols
- Reference models
- Standardization

Acknowledgement: Many slides have been taken from Prof. Karl's set of slide



## Goals of This Chapter

- Explain how to distinguish between
  - Functionalities that a communication system should offer along with requirements for these functions – *services*
  - Rules how to implement these functions – *protocols*
- Provide an example for a concrete service interface – *sockets*
- Group functions of communication system into well-defined sub-systems – *layers*
- Consider two reference models for such groupings – *ISO/OSI* and *TCP/IP*
- Discuss standardization issues

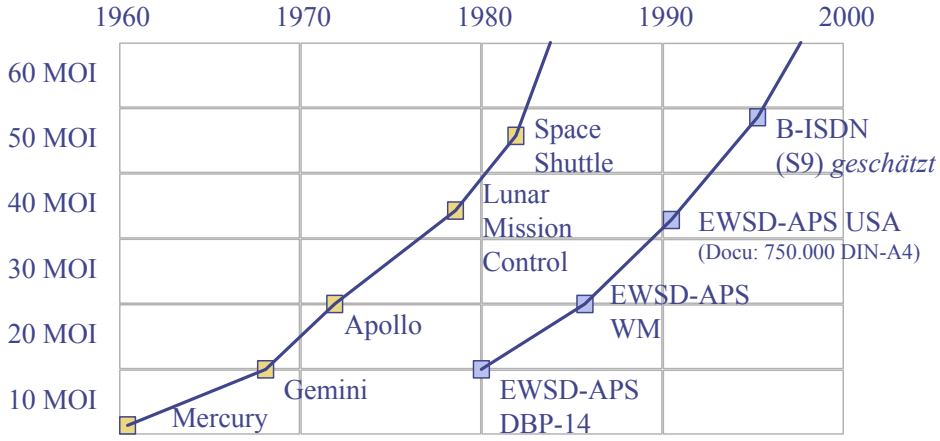
# Recall: Basic Required Functions

- Bit-to-signal and signal-to-bit conversion
- Grouping bits into packets
- Accessing a shared medium
- Switching, duplexing, multiplexing
- Controlling errors on a connection between two systems
- Forwarding incoming packets, consulting routing tables
- Constructing routing tables, maintaining them
- Controlling errors not detectable between two neighboring systems
- Protecting the network against overload
- Protecting end systems against overload
- Ensuring correct order and possibly timeliness of packets
- Making these functions accessible from application programs
- Controlling the actual hardware that connects a wire to a computer
- … *and more!*



# System Structure

- Any chance to build a single, monolithic system/piece of software that realizes all these functions, from wire to application? – No way!
  - To give an idea: Size of telecommunication and astronautic software (Siemens telephony switches)




*MOI = Million Object Code Instructions*

## System Structure – Layers

- Only feasible option: Build “virtual” subsystems of increasing capabilities and abstraction levels
  - Example: Multiplexing abstracted away the physical connections capability to support a single transmission into a ***logical link*** that can be used by several entities
- Subsystems are called ***layers***
- Each layer uses capabilities of a “lower” subsystem, adds own rules and procedures, to provide a more useful, advanced ***service***
  - Services expose a well defined ***interface*** to higher layers
  - Services are accessible at their ***Service Access Point (SAP)***
  - A service is a promise what is going to happen to data when delivered to the SAP



## Service

- Service: Any act or performance that one party can offer to another that is essentially intangible and does not result in the ownership of anything. Its production may or may not be tied to a physical product.

  *D. Jobber, Principles and Practice of Marketing*

- Focus is on the *output*, the *result* of the service
- NOT the means to achieve it

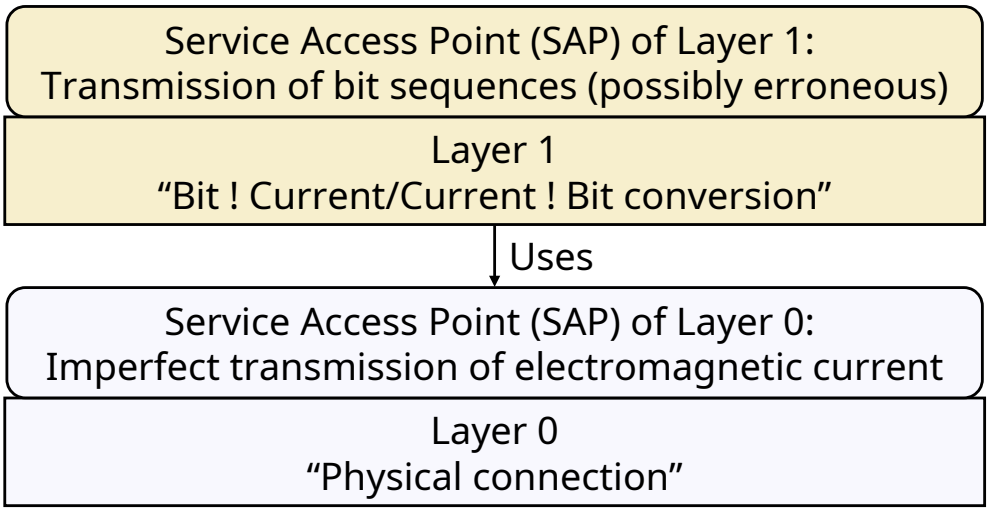
# Overview

- Structuring communication systems into layers
- ***Services***
  - Service primitives
  - Properties of service primitives
  - Example: sockets
- Protocols
- Reference models
- Standardization



# Services and Layers

- Layers provide services to their users, at their interface
  - Convention: consecutive numbering, higher numbers for more abstract layers/services
  - Service can be accessed at different places by different users






## Service Primitives

- Service primitives: set of operations available at the interface of a service
  - Formal definition of the service
- Example for Bit Transmission Layer
  - SEND_BIT – sender can ask for the delivery a of a bit to the receiver
  - RECEIVE_BIT – receive can wait for the arrival of a bit (blocking)
  - INDICATE_BIT – indicate to the receiver that bit is now available (asynchronous)

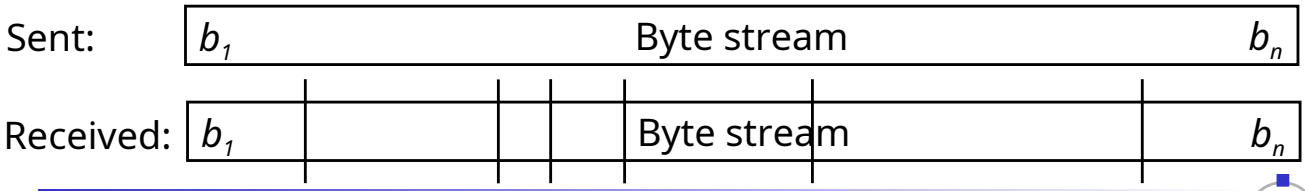


## Service Primitives – Main Groups

- Many different service primitives are conceivable
- Four main types/groups of such primitives are usually distinguished:
  - Request (Req) – Ask a layer to perform a particular service
  - Indication (Ind) – A layer indicates to its (higher-layer) user that “something has happened”, an (asynchronous) notification
  - Response (Res) – A higher-layer user answers an indication
    - E.g., by accepting the delivered data and confirming its receipt
  - Confirmation (Conf) – The original service requester is informed that the service request has been (successfully or unsuccessfully) completed
- Not all four types need to be available for all services
- Example: Phone calls, sending a facsimile, sending a postcard, etc.

# Service Primitives – Unit of Data

- Two main possibilities how to treat data in service primitive design
  - Unit of data are well-delimited ***messages***
    - Only complete messages are sent
    - Only complete messages are received
    - "Half of messages" is not a useful concept
  - Unit of data is individual byte, sequence of bytes or ***byte stream*** is transmitted
    - Not inherently structured into messages
    - Sender transmits sequence of bytes $b_1, \ldots, b_n$
    - Receiver can make use of any sequence of sub-sequences








# Service Primitives – Correctness Requirements

- For both message and byte stream oriented (set of) service primitives, correctness requirements are important
  - ***Completeness***: All data that is sent is eventually delivered
  - ***Correctness***: If data is delivered, its is correct, i.e., the data that has been actually sent
    - Messages are not modified, original version is delivered
    - Byte sequence is free of errors
  - ***In order***: Byte sequence/sequence of messages is delivered in the order it has been sent
  - ***Dependable***: secure, available, …
  - ***Confirmed***: Reception of data is acknowledged to the sender
- Not all requirements are always necessary

## Connection-Oriented vs. Connection-Less Service

- Recall telephony vs. postal service
  - Service can require a preliminary setup phase, e.g., to determine receiver ! ***connection-oriented service***
    - Three phases: connect, data exchange, release connection
  - Alternative: Invocation of a service primitive can happen at any time, with all necessary information provided in the invocation ! ***connection-less service***
  - Note: This distinction does ***NOT*** depend on circuit or packet switching – connection-oriented services can be implemented on top of packet switching (and vice versa, even though a bit awkward)
- Connection-oriented services must provide primitives to handle connection
  - CONNECT – setup a connection to the communication partner
  - LISTEN – wait for incoming connection requests
  - INCOMING_CONN – indicate an incoming connection request
  - ACCEPT – accept a connection
  - DISCONNECT – terminate a connection



## Typical Examples of Services

- ***Datagram service***
  - Unit of data are messages
  - Correct, but not necessarily complete or in order
  - Connection-less
  - Usually insecure/not dependable, not confirmed
- ***Reliable byte stream***
  - Byte stream
  - Correct, complete, in order, confirmed
  - Sometimes, but not always secure/dependable
  - Connection-oriented
- Almost all possible combinations are conceivable!



# A Concrete Service Interface – BSD Sockets

- The most popular service interface are “BSD sockets”
- Model the network service like the access to a file
  - Sending data = Writing to a file
  - Receiving data = Reading from a file
  - Opening a file = Connecting to a communication peer
  - Files – in UNIX – are treated via so-called *file handles*
  - By analogy, file handles will represent communication peers

- Principle of socket programming is simple
  - Details are a bit nasty because of the intended generality of the socket application programmer interface (API)
  - And because of some imperfections of C as a programming language



# Datagram Communication Over Sockets

- Simplest possible service: unreliable datagrams

Sender

- `int s = socket (…);`
- `sendto (s, buffer, datasize, 0, to_addr, addr_length);`
- `to_addr` and `addr_length` specify the destination

Receiver

- `int s = socket (…);`
- `bind (s, local_addr, …);`
- `recv (s, buffer, max_buff_length, 0);`
- Will wait until data is available on socket `s` and put the data into `buffer`

# Byte Streams Over a Connection-Oriented Socket

- For reliable byte streams, sockets have to be connected first
- Receiver has to accept connection

**Client**

- `int s = socket (…);`
- `connect (s, destination_addr, addr_length);`
- `send (s,buffer, datasize, 0);`
- Arbitrary `recv()/send()`
- `close (s);`

- Connected sockets use a `send` without address information

**Server**

- `int s = socket (…);`
- `bind (s, local_addr, …);`
- `listen (s, …);`
- `int newsock = accept (s, *remote_addr, …);`
- `recv (newsock, buffer, max_buff_length, 0);`
- Arbitrary `recv()/send()`
- `close (newsock);`

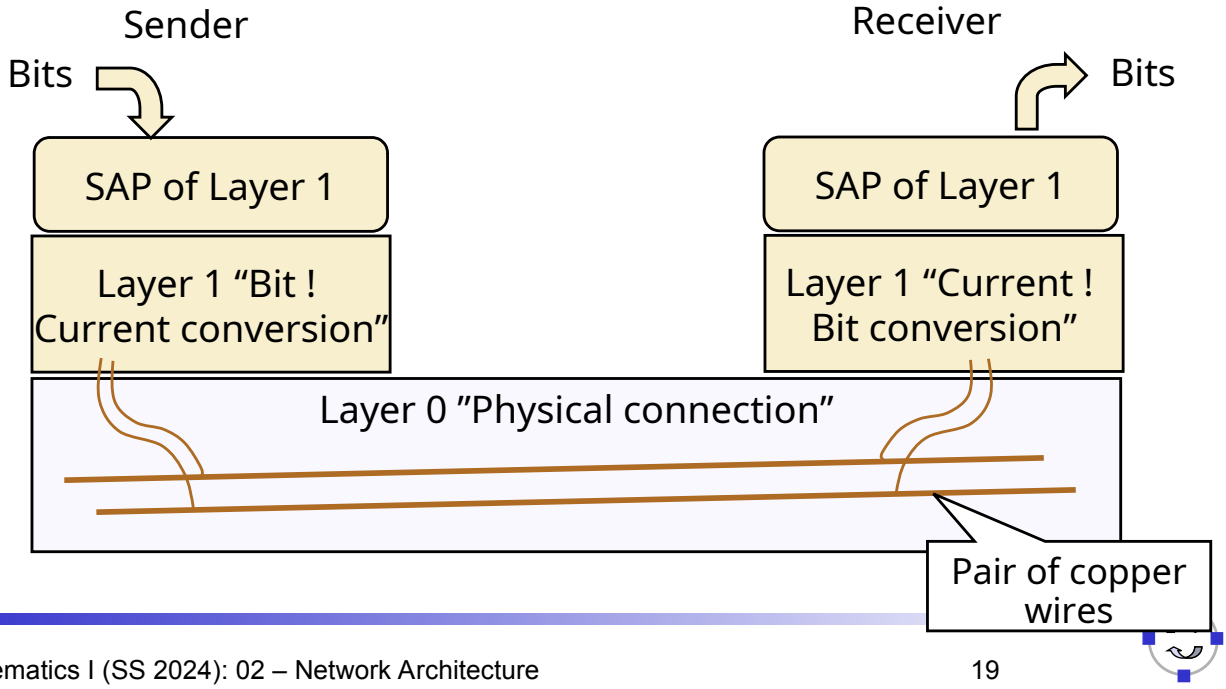


# Overview

- Structuring communication systems into layers
- Services
- ***Protocols***
- Reference models
- Standardization

# Layers are Distributed

- Previous example: “Bit sequence layer” has to be present at both transmitter (bit ! electrical current) and at receiver (electrical current ! bit)

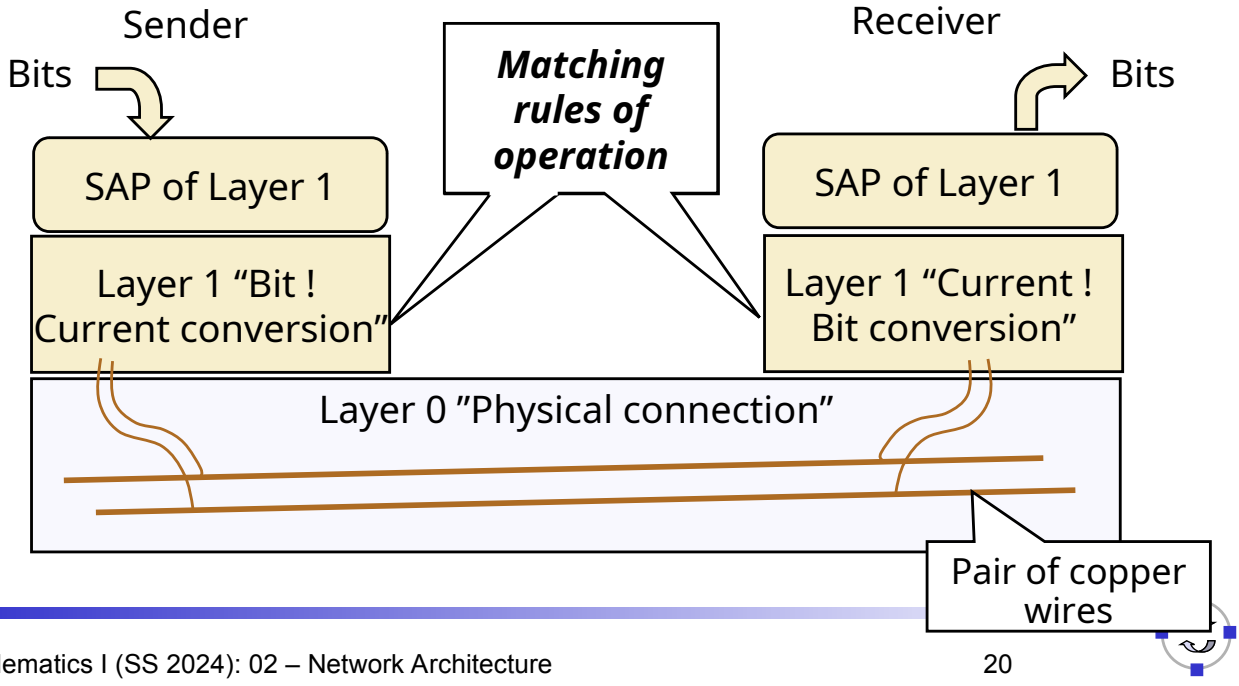







# Distributed Layers Need to Follow Rules – Protocols

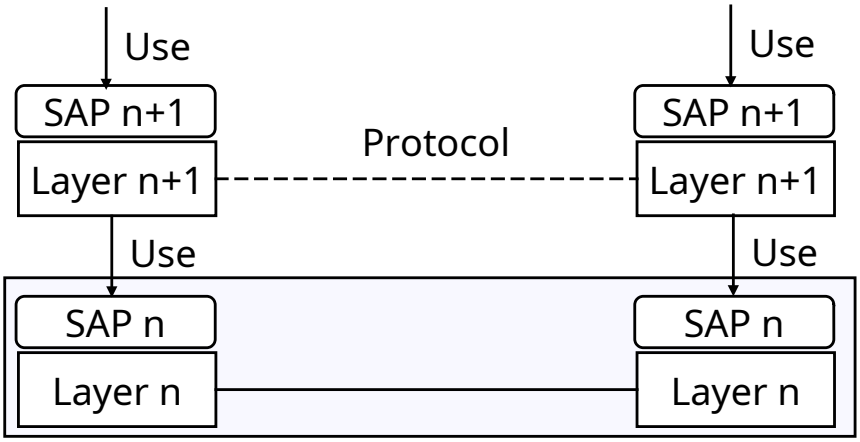
- Distributed parts of the layer 1 implementation have to follow the same ***set of rules – protocols***
  - Suppose sender represented a “1” by “current there”; receiver by “no current”



# Protocols

- Protocols are a set of rules
  - Describe how two (or more) remote parts of a layer cooperate to *implement the service* of the given layer
  - These remote parts are called ***peer protocol entities*** or simply ***peers***
  - Protocols define format, order of messages sent and received among entities, as well as actions taken on message transmission and receipt
  - Use the service of the underlying layer to exchange data with peer

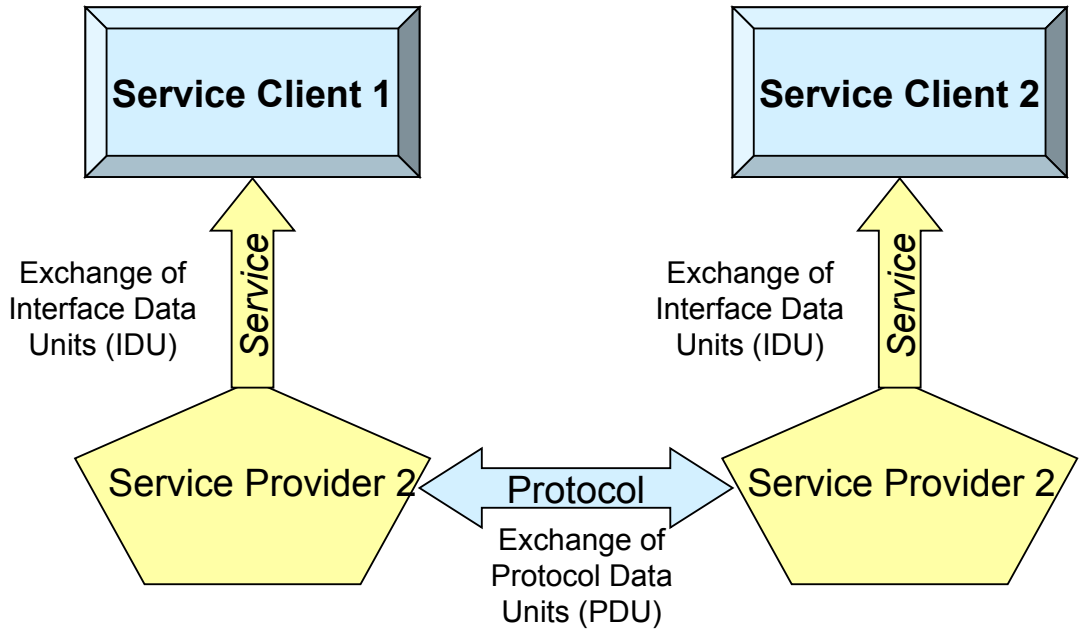
- ***Protocol is implementation of the service***
  - Not visible to service user





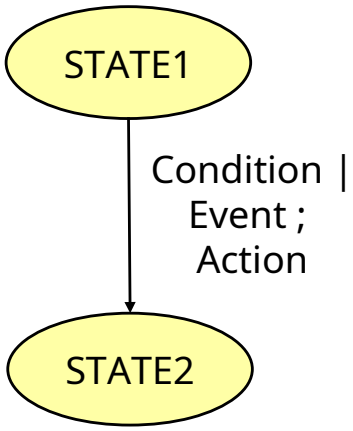



# Service and Protocol



## Protocol Specification

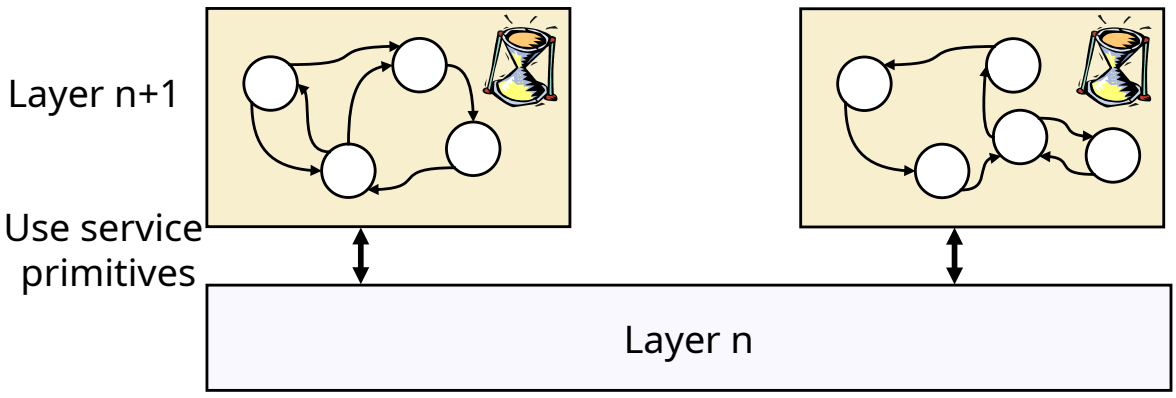
- The formal behavior, the rules which constitute the protocol have to be precisely specified
- One popular method: ***(Extended) Finite State Machine (FSM)***
- A ***protocol instance/protocol engine*** at each entity
- Protocol instance has several states
  - E.g., for a protocol implementing a connection oriented service: IDLE, CONNECTED, RELEASING_CONNECTION
- Events/transitions between states
  - Message arrivals
  - Real time / timer events
- Transition can have conditions
- Actions during transition are
  - Send new message
  - Set timer, delete timer, …





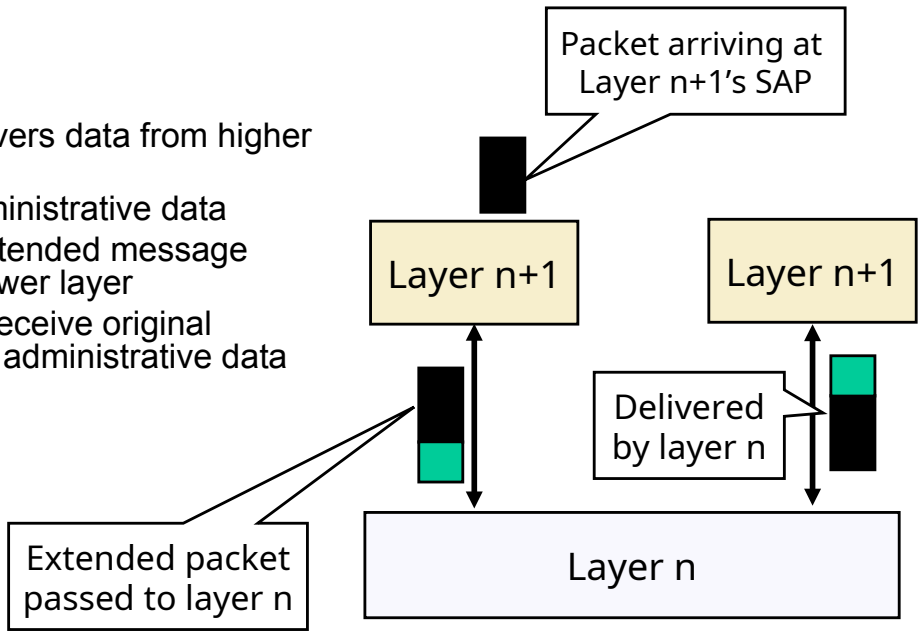

## Protocols and FSMs

- Finite state machines implement actual behavioral rules of a protocol
- Have to communicate with their remote peer
  - Cannot do so directly, have to use service of the underlying communication layer
  - Via service primitives, which can also provide arriving data to the protocol
  - E.g., indications from lower layer are events to higher layer protocol engine



# Protocols and Messages

- When using lower-layer services to communicate with the remote peer, administrative data is usually included in those messages
- Typical example
  - Protocol receivers data from higher layer
  - Adds own administrative data
  - Passes the extended message down to the lower layer
  - Receiver will receive original message plus administrative data
- ***Encapsulating***
  - ***Header*** or ***trailer***

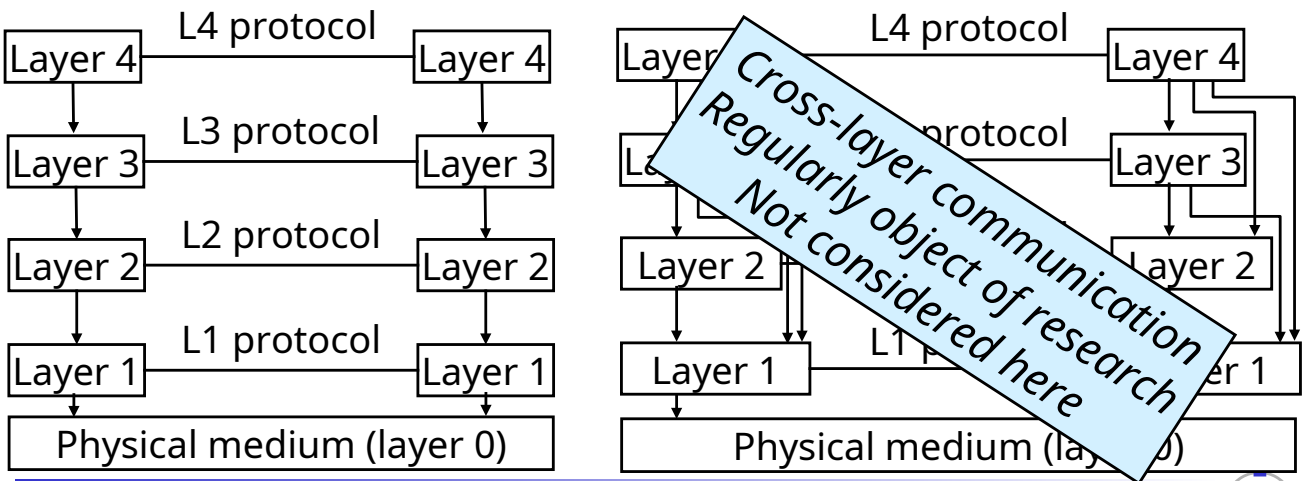






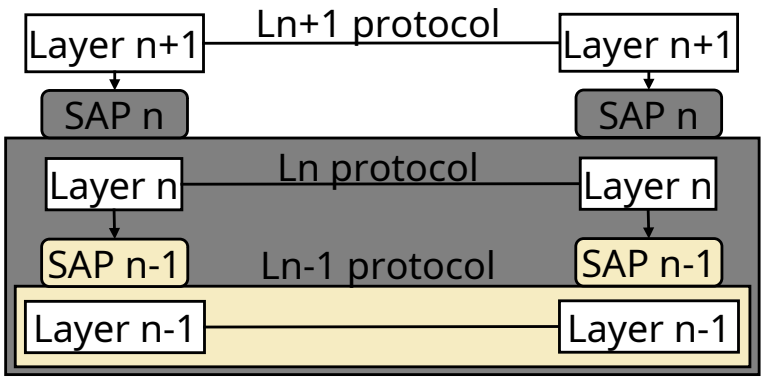

# Protocol Stacks

- Typically, there are several layers and thus several protocols in a real system
- Layers/protocols are arranged as a ***(protocol) stack***
  - One atop the other, ***only using*** services from directly beneath
  - This is called ***strict layering***



# Layers do not Care About Distributed Lower Layers

- A given layer n+1 does not care about the fact that its lower layer is actually distributed, has remote FSMs, …
  - Layer n+1 imagines layer n as something that "just works", has service access points where they are necessary
  - In reality, layer n of course is distributed in turn, relying on yet lower layers
  - At the end, the physical medium (layer 0) is transporting data/signals






# Protocol Mechanisms: What Do Protocols Do for a Living?

- All or some of the following:
  - *Addressing/naming:* manage identifiers
  - *Fragmentation:* divide large message into smaller chunks to fit lower layer
  - *Re-sequencing:* reorder out-of-sequence messages
  - *Error control:* detection and correction of errors and losses (e.g. retransmission, forward error correction)
  - *Flow control:* avoid flooding/overwhelming of slower receiver
  - *Congestion control:* avoid flooding of slower network nodes/links
  - *Resource allocation:* administer bandwidth, buffers among contenders
  - *Multiplexing:* combine several higher-layer sessions into one "channel"
  - *Compression:* reduce data rate by encoding
  - *Privacy, authentication:* security policy (others are listening)

# Overview

- Structuring communication systems into layers
- Services
- Protocols
- ***Reference models***
  - ISO/OSI
  - TCP/IP
- Standardization

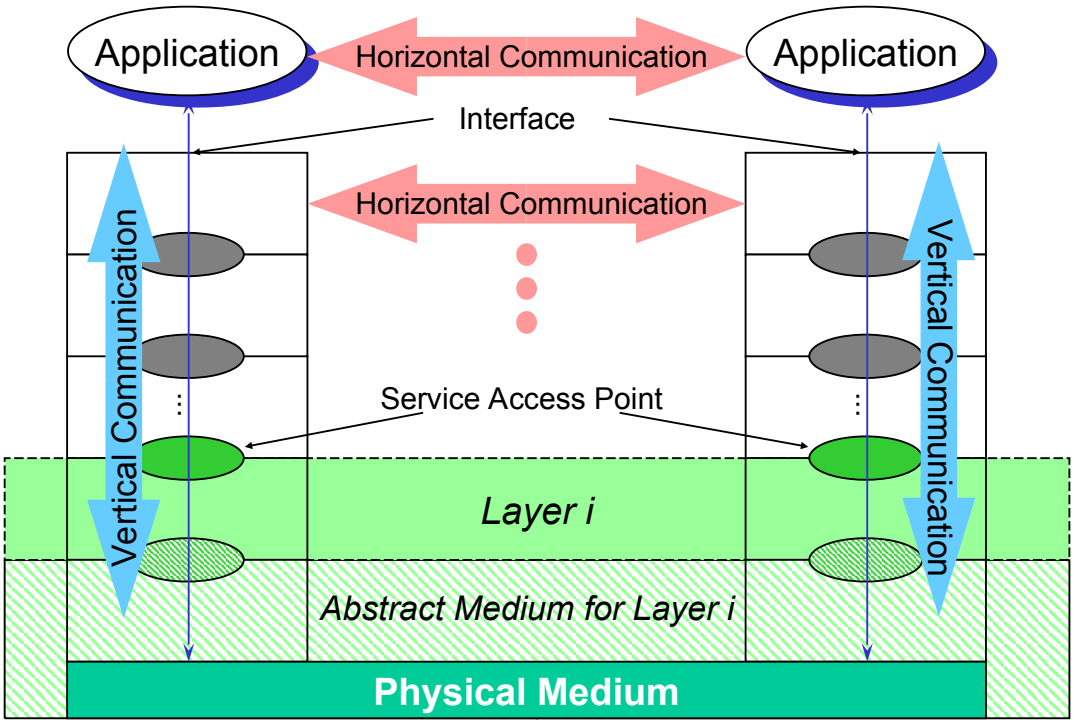


# How to Structure Functions / Layers in Real Systems?

- Many functions have to be realized
  - Modularization and layering ease maintenance & updating of a system
- How to actually group them so as to obtain a real, working communication system?
  - Explicit structure allows identification, and specification of relationships of complex system's pieces
  - Layered ***reference model*** for discussion
  - Layering structure and according protocols define the ***communication architecture***



# Layered Communication System






# Layering Considered Harmful?

- Benefits of layering:
  - Need layers to manage complexity: don't want to reinvent Ethernet-specific protocol for each application
  - Change of implementation of one layer's service is transparent to the rest of a system
  - Common functionality: "Ideal" network
- But:
  - Layer N may duplicate lower layer functionality (e.g. error recovery)
  - Different layers may need same information
  - Layer N may need to peek into layer N+x

- Two main ***reference models*** exist
  - ISO/OSI reference model
    (International Standards Organization Open Systems Interconnection)
  - TCP/IP reference model
    (by IETF – Internet Engineering Taskforce)

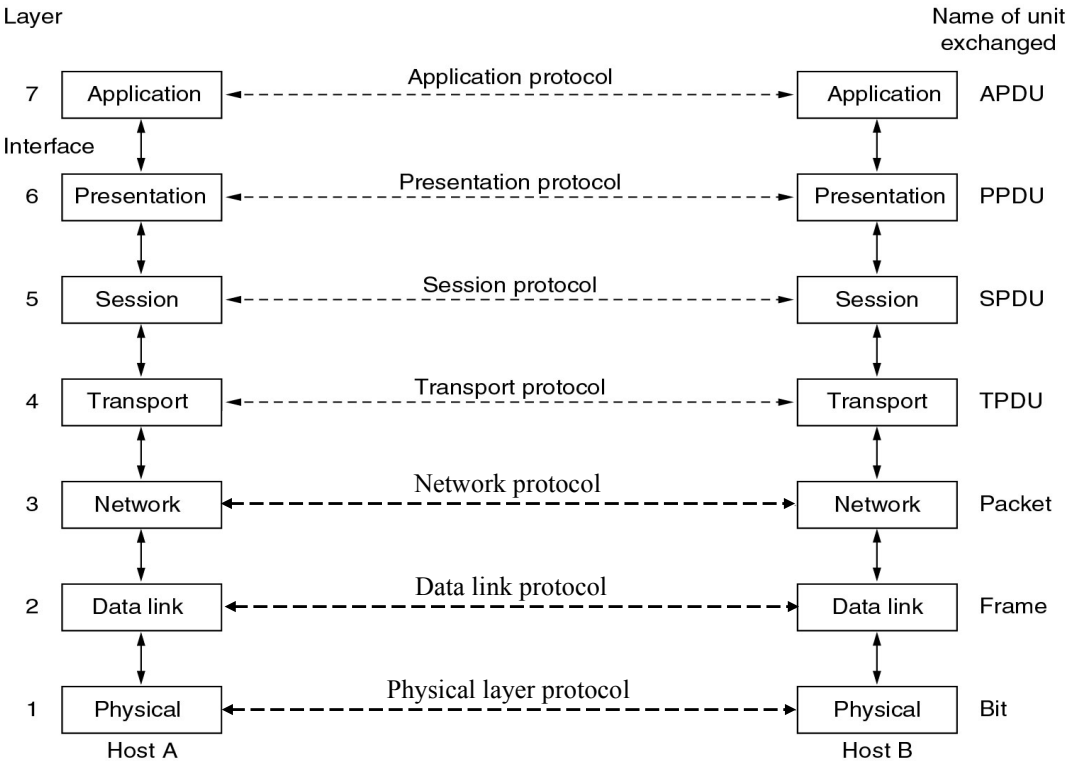
# ISO/OSI Reference Model

- Basic design principles
  - One layer per abstraction
  - Each layer has a well-defined function
  - Choose layer boundaries such that information flow across the boundary is minimized (minimize inter-layer interaction)
  - Enough layers to keep separate things separate, few enough to keep architecture manageable

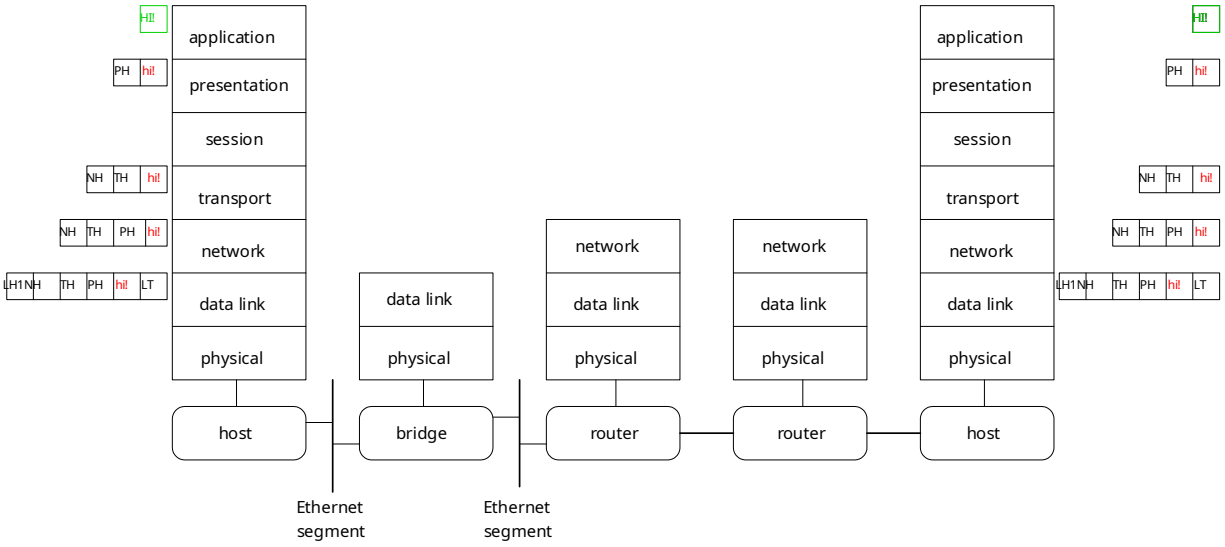
- Result: 7-layer model
  - Not strictly speaking an architecture, because protocol details are not specified
  - Only general duties of each layer are defined



# ISO/OSI 7-Layer Reference Model (two entities)




# ISO/OSI 7-layer reference model (complete network)




- **Sending side** layer N takes interface data unit (IDU) from layer N +1, adds layer N header, and passes result to layer N - 1
- **Receiving side** layer N takes IDU from layer N – 1, strips layer N headers, processes and passes rest to layer N + 1



# OSI Layers (1)

- Physical Layer (Ph):
  - Provides a bit transparent interface to the physical media. Specifies the mechanical, electrical, functional and procedural means to support physical connection between open systems.
  - Physical connection does not imply connection-oriented operation!
  - Different real media can be used, different control procedures are needed
  - In-sequence delivery of Bits assured
  - Error detection sometimes included
- Link Layer (L):
  - Supports transmission of service data units (SDU) bigger than “word” among systems connected to a single physical path
  - Essential function is block synchronization.
  - Sometimes, error detection or even error control is also provided
  - In the case of half/ duplex or multipoint links, the medium access has to be controlled, and peer systems have to be addressed

# OSI Layers (2)

- Network Layer (N):
  - Creates a logical path between open systems connected to individual, possibly different, subnetworks
    - This logical path can go through several, possibly different, intermediate subnetworks
  - The Network Layer supports routing; thus N- Service users are not concerned with the path
  - The N- Service is uniform regardless the variation of subnetwork technologies, topologies, QoS, organization
  - Network address ~ end system address
- Transport Layer (T):
  - Completes the transmission part of the OSI stack. Supports transmission with required QoS, in an economic way between (T)- Users (usually processes in an end system), regardless of network structure.
  - Several classes of protocol with different functionality are defined (connection-oriented / connectionless; reliable / unreliable)



# OSI Layers (3)

- Session Layer (S):
  - Supports the synchronization of the dialogue and managing the exchange of data (potentially spanning over multiple transport layer connections)
  - Quarantine data delivery - a whole group of transmitted S-SDUs is delivered on explicit request of the sending party
  - Interaction management allows to explicitly define which of the S-Users obtains the right to transmit
  - Connection resetting to pre-defined synchronization points
- Presentation Layer (P):
  - Supports translation of data and data structures into a unique representation
  - Only the syntax is modified in order to maintain the semantics
  - Selection of one of the generally agreed transfer syntaxes
  - Local syntax of each end systems is translated into/ from the selected transfer syntax

## OSI Layers (4)

- Application Layer (A):
  - Supports directly the end user via providing a variety of application services.
  - This can be of:
    - General type (e. g. Remote Procedural Calls, Transaction Processing,....), or
    - Specific type (e. g. virtual terminal, file transfer access and management, job transfer.....)
  - A classical example: virtual terminal (functions of a real terminal are mapped into the virtual functions)



## ISO/OSI Reference Model – Critique

- The reference model as such, in its structuring of functions into layers, is very influential until today
- Actual protocols developed for it are irrelevant in practice
- ISO failed in gaining actual market acceptance for its model
  - Bad timing – competing approaches already in market, lack of industry support
  - Bad technology – too big, too complex; some design flaws
  - Bad implementations – initial implementations low quality
  - Bad politics – ISO/OSI conceived of as a bureaucratic thing
- In the Internet protocol model, the upper three OSI layers are summarized in one (Internet-) application layer



## Architectural Requirements for the Internet

- Generality
  - Support ANY set of diverse applications,
  - Either datagrams (video...) or virtual connections (web...).
- Heterogeneity
  - Interconnect ANY set of network technologies
- Robustness
  - More important than efficiency
- Extensibility
  - More important than efficiency
- Scalability
  - A later discovery: How many ARPAnets could the world support? A few hundred, maybe… ?



## Towards the End-to-End Principle

Foundation of the Internet architecture:

- Dumb network, smart end systems
  - *(Exact opposite of telephone network!)*
- Dumb networks: require only least common service
  - Datagram service: no connection state in routers
  - Best effort: all packets treated equally.
  - Can lose, duplicate, reorder packets.
- Smart hosts:
  - Maintain state to enhance service for applications.
  - “Fate-sharing”-- If a host crashes and loses communication state, applications that are communicating share this fate.



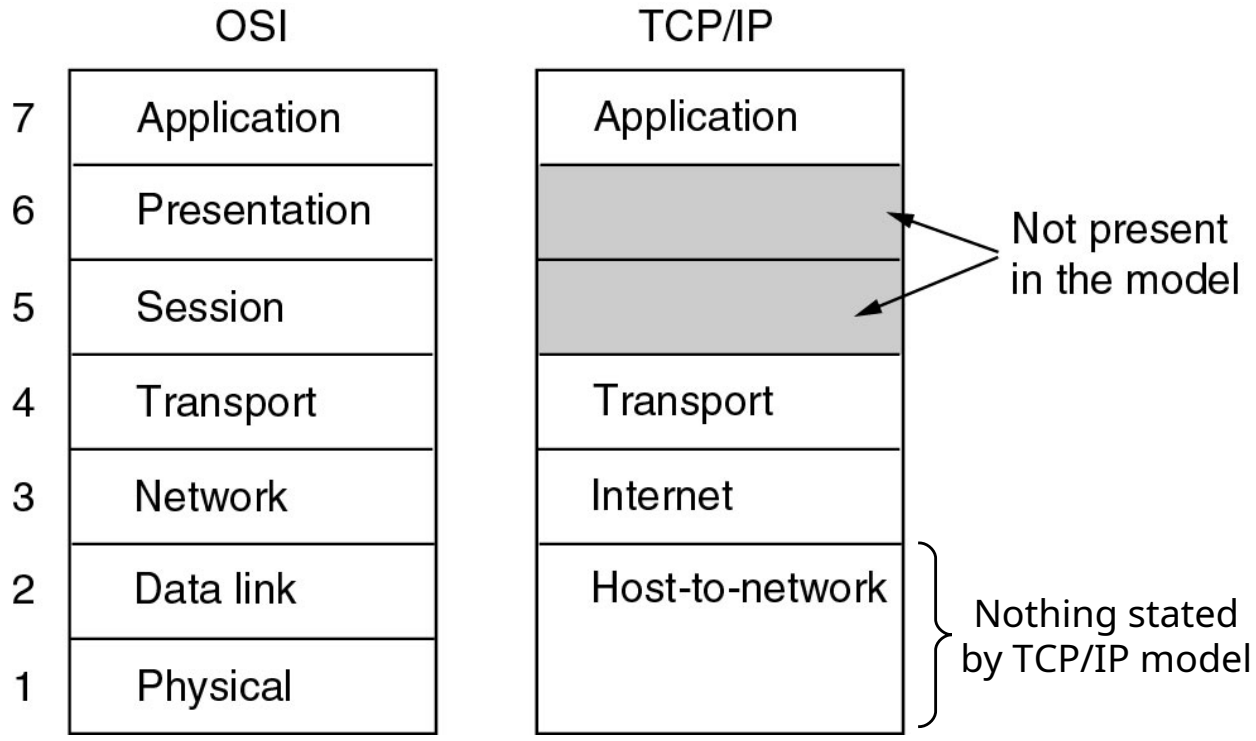
# TCP/IP Reference Model

- Historically based on the ARPANET, later to become the Internet
  - Started out as little university networks, which had to be interconnected
- In effect, only really defines two layers
  - **Internet layer**: packet switching, addressing, routing & forwarding. Particularly for hierarchically organized networks ("networks of networks") – **Internet Protocol (IP)** defined
  - **Transport layer**: two services & protocols defined
    - Reliable byte stream: **Transport Control Protocol (TCP)**
    - Unreliable datagram: **User Datagram Protocol (UDP)**

    In addition, (de)multiplexing
  - Lower and higher layers not really defined
    - "Host to host" communication assumed as a given
    - Applications assumed



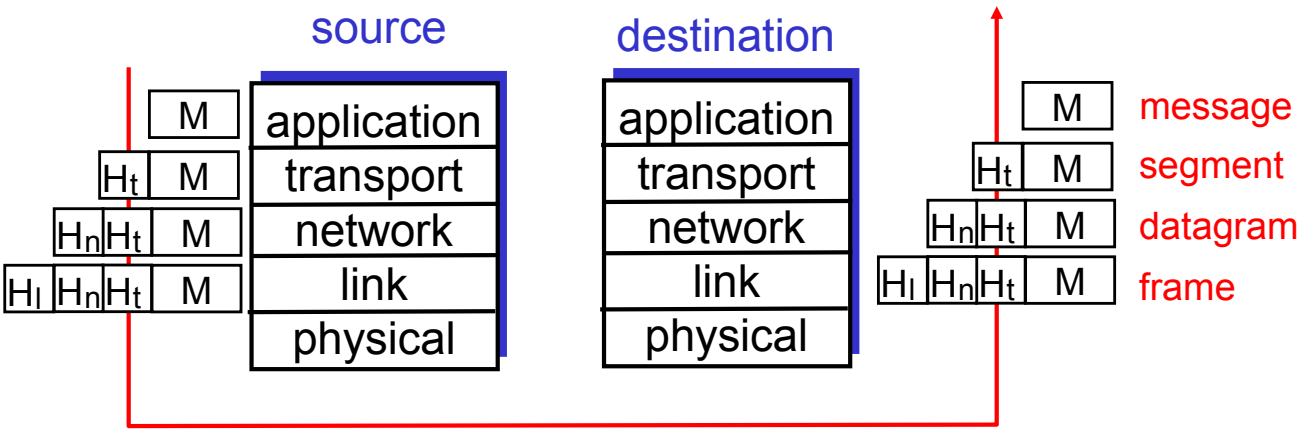
# TCP/IP Protocol Stack




# Protocol Layering and Data in the Internet

Each layer takes data from (layer) above

- Adds header information to create new data unit
- Passes new data unit to layer below

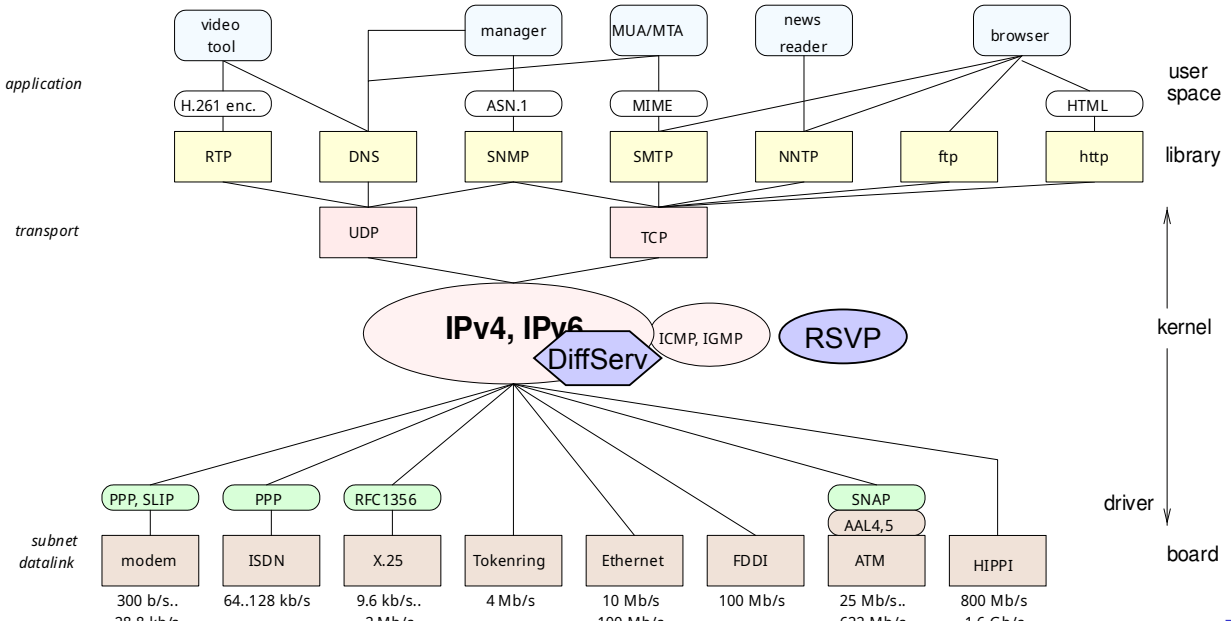







# TCP/IP Suite of Protocols

- Many application specific protocols over IP
- IP (with best effort service model) over many media specific (L2) protocols
  → „Hourglass“ model of IP
- Quality of Service added to IP as an ‚afterthought‘



## History of the Internet: Pre 1970

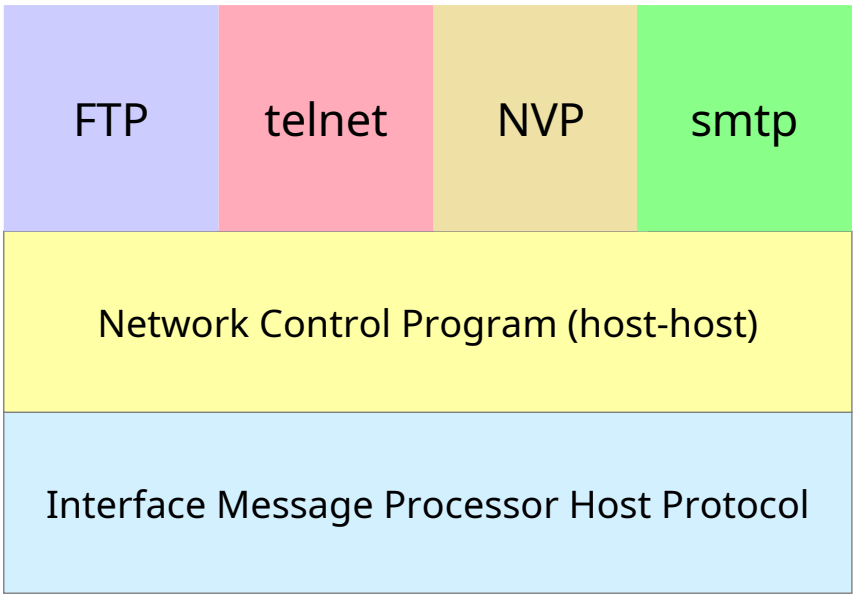
- 1830: Telegraph
- 1876: Circuit switching
- MIT: Len Kleinrock, J. C. R. Licklider, Larry Roberts
  - Kleinrock: Paper on theory of packet switching, 1961
  - Licklider: Memos on "Galactic Network", 1962
  - Roberts: Plan for the "ARPANET", 1967
- Paul Baran
  - Report on packet switching for secure voice, 1964
- NPL (UK): Donald Davies & Roger Scantlebury:
  - Paper on a packet-switching network, 1967
- 1968: ARPA selected Bolt, Beranek, and Newman (BBN) to design, build, and operate the IMPs (interface message processor) and the ARPAnet infrastructure.
- 1968: ARPAnet has 4 nodes (UCLA, SRI, U. Utah, UCSB) connected by IMPs and 50 kb/s lines



## ARPANet: Protocols and Stack

- IMP-Host Interface: defined by BBN Report 1822
- Communication Service:
  - Reliable delivery to a specified host system of a message of arbitrary bit length.
    - I.E., a “reliable datagram” service
  - The 8-bit byte not universal yet; computers used 8, 12, 16, 18, 24, 32, 36, 48, ... bit words.
  - ARPAnet host addresses were 24 bits

# ARPANet: Protocols and Stack








# History of the Internet: 1970‘s

- Multiple access networks:
  - ALOHA (wireless), Ethernet (cable, 10Mb/s)
  - IBM System Network Architecture (1974), DECnet (1975)
- **1971:** 15 nodes and 23 hosts: UCLA, SRI, UCSB, U of Utah, BBN, MIT, RAND, SDC, Harvard, Lincoln Lab, Stanford, UIU(C), CWRU, CMU, NASA/Ames
- **1972:** first public demonstration at ICCC
- **1973:** TCP/IP design
- **1973:** first satellite link from California to Hawaii
- **1973:** first international connections to the ARPANET: England (UCL) and Norway
- **1979:** ARPAnet  100 nodes



## History of the Internet: 1980‘s

- Proliferation of local area networks: Ethernet and token rings
- Late 1980’s: fiber optic networks; fiber distributed data interface (FDDI) at 100 Mb/s
- **1980’s:** DARPA funded Berkeley Unix, with TCP/IP
- **1981:** Minitel deployed in France
- **1980-81:** BITNET (IBM protocols) and CSNET (NSF-funded 200 sites)
- **Jan. 1, 1983:** flag day: NCP to TCP (10 years after TCP design)
- **early 1980’s:** split ARPANET (research), MILNET (military)
- **1984:** Domain Name Service replaces hosts.txt file
- **1986:** NSFNET created (56 kb/s backbone)
- **Nov. 1, 1988:** Internet worm
- **1989:** Internet passes 100,000 nodes
- **1989:** first proposal for World-WideWeb
- **1989:** NSFNET backbone upgraded to T1 (1.544Mbps)



## History of the Internet: 1990‘s

- High-speed networks: Asynchronous Transfer Mode (ATM) at 150 Mb/s and higher
- Focus on new applications
- Wireless local area networks
- Commercialization
- **1990:** Original ARPANET disbanded
- **Fall 1991:** CSNET discontinued
- **1991:** Gopher released by University of Minnesota
- **1992:** NSFNET backbone upgraded to T3 (44.736Mbps)
- **March 1992:** First MBONE audio multicast
- **November 1992:** First MBONE video multicast
- **February 1993:** NCSA Mosaic
- **June 1993:** 1,776,000 hosts
- **April 30, 1995:** NSFNET backbone disbanded

## TCP/IP Reference Model – Critique

- No clear distinction between service, protocol, interface
  - Reliable byte stream is *equated* with TCP, although there is a clear difference
  - Particularly below IP
- Very specialized stack, does not allow a generalization to other technologies/situations
- Great void below IP
- Many ad hoc, wildly hacked solutions in many places, without careful design
  - Mobility support is a typical area where problems result later on



## ISO/OSI versus TCP/IP

- ISO/OSI: Very useful model, non-existing protocols
- TCP/IP: Non-existing model, very useful protocols

- Hence: Use a simplified ISO/OSI model, but treat the TCP/IP protocol stack in the context of this model
  - With suitable add-ons especially for the lower layers

| | |
|---|---|
| 5 | Application layer |
| 4 | Transport layer |
| 3 | Network layer |
| 2 | Data link layer |
| 1 | Physical layer |

## Overview

- Structuring communication systems into layers
- Services
- Protocols
- Reference models
- ***Standardization***



## Standardization

- To build large networks, standardization is necessary
- Traditional organization from a telecommunication/ telephony background
  - Well established, world-wide, relatively slow “time to market”
- Internet
  - Mostly centered around the Internet Engineering Task Force (IETF) with associated bodies (Internet Architectural Board IAB, Internet Research Task Force IRTF, Internet Engineering Steering Group IESG)
  - Consensus oriented, heavy focus on working implementations
  - Hope is quick time to market, but has slowed down considerably in recent years
- Manufacturer bodies



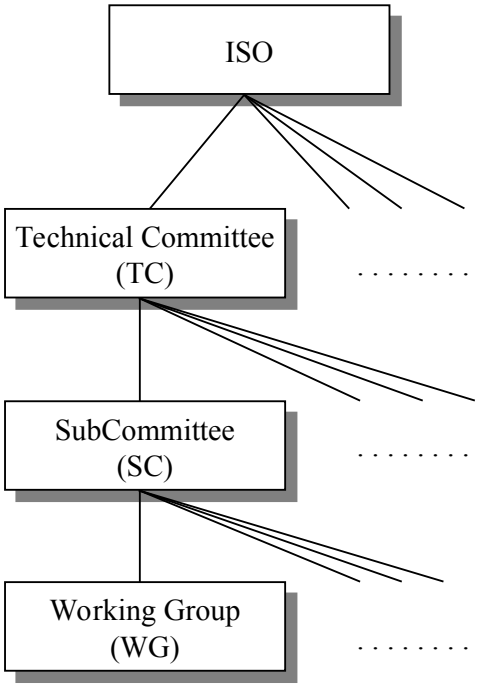
# Standardization – Traditional Organizations

- ITU – International Telecommunication Union (formerly CCITT und CCIR)
- CCITT – Consultative Committee on International Telegraphy and Telephony (Comité Consultatif International Télégraphique et Téléphonique)
- CCIR – Consultative Committee on International Radio
- CEPT – Conférence Européenne des Administrations des Postes et des Télécommunications
- ISO – International Organization for Standardization
- DIN – Deutsches Institut für Normung
  - German partner organization of ISO



# ISO Standardization Work

- Working Group (WG) Meetings:
  - Every 6-9 months
  - National organizations have time to accept proposed concepts
  - Then: actual standarization process
    - DP: Draft Proposal
    - DIS: Draft International Standard
    - IS: International Standard
- Move a proposal to a higher level by incorporating all dissenting voices and international consensus
- Very slow process




## Making the Standards for the Internet

- **Internet Architecture Board:** IAB
  - architectural oversight
  - elected by ISOC
- **Internet Engineering Steering Group (IESG)**
  - approves standards
- **Internet Engineering Task Force (IETF)**
  - “prepares” standards
- **Internet Society:** ISOC
  - Conferences
  - “hosts” IANA
- **Internet Assigned Number Authority:** IANA
  - keeps track of numbers
  - delegates Internet address assignment



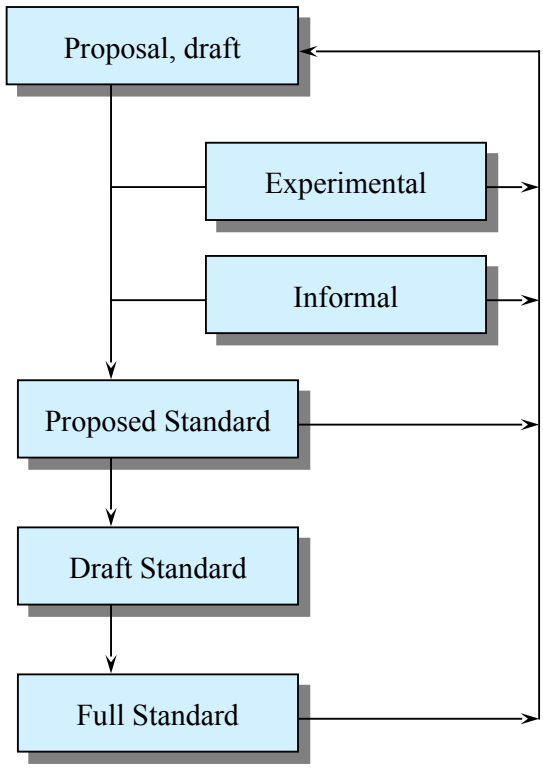
## The Internet Engineering Task Force (IETF)

- Small focused efforts preferred to larger comprehensive ones
- Competing efforts allowed (part of the system)
- Published goals and milestones
- No formal voting
- “Rough consensus (and running code!)”
- Disputes resolved by discussion and demonstration (mostly. . . )
- Mailing list and face-to-face meetings
- Open, no-fee membership
- Standardization only after several implementations
- Specifications available without charge by ftp
  ($, €, ...: ITU, IEEE)
- Since early 2000s: getting very large and commercially influenced



# The Internet Engineering Task Force (IETF)

- IETF is organized in areas and Working Groups
  - Volunteers from industry, academia, government organizations participate
- Drafts/proposal can be made by anybody
  - An „on-demand“ process
- To move beyond draft status, two independent, inter-operating implementations are required
- Informal voting in working groups
  - „Humming“
  - Three meetings a year
- Result:
  - RFC – request for comment, the actual standard
  - FYI – informal or informational






# RFCs and Drafts

- “Request for Comments”, since 1969
- Most RFCs are not standards!
- Internet drafts: working documents, but often used for prototypes
- Edited, but not refereed
- Numbered sequentially (March 2024: 9565 RFCs)
- Check the April 1 ones. . . (e.g. RFC 3514: "The Security Flag in the IPv4 Header")
- https://www.ietf.org/rfc/rfc-index

# Conclusions

- To keep complexity of communication systems tractable, division in subsystems with clearly assigned responsibilities is necessary – *layering*
- Each layer (and the communication system as a whole) offers a particular *service*
  - Services become more abstract and more powerful the higher up in the layering hierarchy
- To provide a service, a layer has to be distributed over remote devices
- Remote parts of a layer use a *protocol* to cooperate
  - Make use of service of the underlying layer to exchange data
  - Protocol is a horizontal relationship, service a vertical relationship
- Two important reference models how to group functionalities into layers, which services to offer at each layer, how to structure protocols