Communication components

- network: a set of computers connected by communication links
- Intranet: local area networks (LAN), in the same administrative domain
- Internet: wide area networks (WAN), collection of interconnected networks across administrative domains
- System area networks (SAN): distributed systems
- Communication rules: protocols

Circuit vs. Packet switching

- Circuit switching
  - example: telephony
  - resources are reserved and dedicated during the connection
- Packet switching
  - example: internet
  - entering data divided into packets
  - packets in network share resources
- Virtual circuit: cross between circuit switching and packet switching

Connection vs. Connectionless

- connection-oriented services: sender and receiver maintains a connection (using circuit switching for example)
- connectionless protocols: sender transmits each message when it is ready (similar to the mail system)
- a connection-oriented service can be implemented on top of a packet-switch network
Protocol Architecture

- in the network, computers must agree on the syntax (data format) and the semantics (data interpretation) of communication
- common approach: protocol functionality is distributed in multiple modules (layers) which are stacked
- layer N provides services to layer N+1, and relies on services of layer N-1
- communication is achieved by having similar layers at both end-points which understand each other

ISO/OSI protocol stack

- "officially": seven layers
- in practice four: application, transport, network, data link / physical

Application Layer

- process-to-process communication
- supports application functionality
- examples
  - file transfer protocol (FTP)
  - simple mail transfer protocol (SMTP)
  - hypertext transfer protocol (HTTP)
- user can add other protocols, for example a distributed shared memory protocol

Transport Layer

- transmission control protocol (TCP)
  - provides reliable byte stream service using retransmission
  - flow control
  - congestion control
- user datagram protocol (UDP)
  - provides unreliable unordered datagram service
Network Layer

- Internet protocol (IP)
  - understands the host address
  - responsible for packet delivery
  - provides routing function across the network
  - but can lose or misorder packets

Data Link/Physical Layer

- comes from the underlying network
- physical layer: transmits 0s and 1s in the wire
- data link layer: groups bits into frames and does error control using checksum + retransmission
- examples
  - Ethernet
  - ATM
  - Myrinet
  - phone/modem

Internet hierarchy

- FTP
- HTTP
- Finger
- SVM
- TCP
- UDP
- IP
- Ethernet
- ATM
- modem

The Network Layer: IP

- addressing: how hosts are named
- service model: how hosts interact with the network, what is the packet format
- routing: how a route from source to destination is chosen
IP Addressing

- Addresses
  - unique 32-bit address for each host
  - dotted-decimal notation: 128.112.102.65
  - three address formats: class A, class B and class C
- IP to physical address translation
  - network hardware recognizes physical addresses
  - Address Resolution Protocol (ARP) to obtain the translation
  - each host caches a list of IP-to-physical translation which expires after a while

IP packet

- IP transmits data in variable size chunks: datagrams
- may drop, reorder or duplicate datagrams
- each network has a Maximum Transmission Unit (MTU): which is the largest packet it can carry
- if packet is bigger than MTU it is broken into fragments which are reassembled at destination
- IP packet format:
  - source and destination addresses (128-bit in IPv6)
  - time to live: decremented on each hop, packet dropped when TTL=0
  - fragment information, checksum, other fields

ARP

- hosts broadcast a query packet asking for a translation for some IP address
- hosts which know the translation reply
- each host knows its own IP and physical translation
- reverse ARP (RARP) translates physical to IP and it is used to assign IP addresses dynamically

IP routing

- each host has a routing table which says where to forward packets for each network, including a default router
- how the routing table is maintained:
  - two-level approach: intra-domain and inter-domain
  - intra-domain: many approaches, ultimately call ARP
  - inter-domain: Boundary Gateway Protocol (BGP):
  - each domain designates a “BGP speaker” to represent it
  - speakers advertise which domain they can reach
  - routing cycles avoided
Transport Layer

- User Datagram Protocol (UDP): connectionless
  - unreliable, unordered datagrams
  - the main difference from IP: IP sends datagrams between hosts, UDP sends datagrams between processes identified as (host, port) pairs
- Transmission Control Protocol: connection-oriented
  - reliable; acknowledgment, timeout and retransmission
  - byte stream delivered in order (datagrams are hidden)
  - flow control: slows down sender if receiver overwhelmed
  - congestion control: slows down sender if network overwhelmed

TCP: Reliable communication

- each packet carries a sequence number
  - sequence number: last byte of data sent before this packet
- each packet also carries an acknowledge sequence number: first byte of data not yet received
  - no distinction between data and ack packets
- TCP keeps an average round-trip transmission time (RTT)
  - timeout if no ack received after twice the estimated RRT and resend data starting from the last ack
  - possible improvements:
    - ignore retransmitted packets when estimate RTT
    - double timeout on retransmission

TCP: Connection Setup

- TCP is a connection-oriented protocol
- three-way handshake:
  - client sends a SYN packet: “I want to connect”
  - server sends back its SYN + ACK: “I accept”
  - client acks the server’s SYN: “OK”

TCP: Sliding Window

- optimum transmission performance requires keeping the pipe full
- network capacity is equal to latency-bandwidth product
- sliding window: how much data to send without ack
- optimum window size is the network capacity
- sliding window protocol: agreement between sender and destination on how much data sender can send without waiting for ack such that id doesn’t overrun receiver’s buffer
Sliding Window Protocol

- receiver decides how much memory to dedicate to this connection
- receiver continuously advertises current window size = allocated memory - unread data
- sender stops sending when the unack-ed data = receiver current window size

TCP: Congestion Control

- detect network congestion then slow down sending enough to alleviate congestion
- detecting congestion: TCP interprets a timeout as a symptom of congestion (can be mistaken in wireless communication)
- transmission window size = min( receiver window, congestion window)
- Congestion window
  - when all is well: increases slowly (additively)
  - when congestion: decrease rapidly (multiplicatively)
  - slow restart: size = 1, multiplicatively until timeout

Distributed computing

- so far we looked at TCP/IP protocols
- how to use network protocols for distributed computing
  - client-server model
  - sockets
  - remote procedure calls (RPC)
  - user-level communication

Client-Server Model

- typical client-server interaction
  - server waits for requests from clients
  - client issues request to server and waits for result
  - server receives the request and performs the service
  - sender replies to the client with the result of the service
  - client resumes the execution using the result
- client and server can run as different processes or in the same process
- if in the same process: either different threads or client must handle asynchronous requests to act as server
Sockets

- Communication abstraction in UNIX:
  - `socket` system call creates an end-point for communication: TCP or UDP protocol
  - `bind` gives an identity to a socket: (host IP, port)
  - `connect` establishes a connection between a local socket (client) and a remote socket (server)
  - `listen` and `accept` are used by a server under TCP to accept connection requests and create a new socket for each connection (see example)
  - `write/read` or `sendto/recvfrom` to transmit data
- Connection-oriented or connectionless via sockets

Connection-oriented server

Connectionless server

Remote Procedure Call (RPC)

- Idea: make communication look like a procedure call
- Simple abstraction, easy to connect to language mechanisms
- Interfaces to servers can be specified as a set of named operations with designated types
- RPC implementation reduces to reliable, blocking message passing
- RPC differs from a local procedure call
- How to make RPC fast?
- Non-blocking RPC: asynchronous RPC, queued RPC
RPC Structure

- client program
- server program
- client stub
- server stub
- call
- return
- network

RPC implementation

- a stub procedure in the caller's address space
- creates a message that identifies the procedure being called and includes parameters (parameter marshaling)
- identifies the location of the server
- sends the message and waits for reply
- when the reply message arrives return to the calling program providing the returned values
- at the server (callee), another stub program which receives the message and calls the corresponding local procedure

Client Stub Example

```c
void remote_add(Server s, int *x, int *y, int *z) {
    s.sendInt(AddProcedure); s.sendInt(*x); s.sendInt(*y); s.flush()
    status = s.receiveInt(); /* if no errors */
    *sum = s.receiveInt();
}
```

Server Stub Example

```c
void serverLoop(Client c) {
    while (1) {
        int Procedure = c.receiveInt();
        switch (Procedure) {
            case AddProcedure:
                int x = c.receiveInt();
                int y = c.receiveInt();
                int sum;
                add("x", "y", *sum);
                c.sendInt(StatusOK);
                c.sendInt(sum);
                break;
        }
    }
}
```
RPC semantics

- different from a local procedure call semantics
- global variables are not accessible inside the RPC
- call-by-copy, not value or reference
- communication errors that may leave client uncertain about whether the call really happened
  - various semantics possible: at-least-once, at-most-once, exactly-once
  - difference is visible unless the call is idempotent

TCP/IP in LAN

- using traditional TCP/IP communication in local area networks is expensive
  - socket calls are system calls
  - permission is checked at every send
  - data is copied both at the sender and at the receiver from user/kernel to kernel/user address spaces
  - buffer management adds overhead
- alternative solutions: user-level communication

User-level communication

- basic idea: remove the kernel from the critical path of sending and receiving messages
  - user-memory to user-memory: zero copy
  - permission is checked once when the mapping is established
  - buffer management left to the application
- Advantages
  - low-latency
  - low overhead
  - approach raw bandwidth provided by the network

Virtual Memory-mapped communication

- receiver exports the receive buffers
- sender must import a receive buffer before sending
  - the permission of sender to write into the receive buffer is checked once when the export/import handshake is performed (usually at the beginning)
  - sender can directly communicate with the network interface to send data into imported buffers without kernel intervention
  - at the receiver the network interface stores the received data directly into the exported receive buffer with no kernel intervention
Virtual-to-physical address

- in order to store data directly into the application address space (exported buffers), the NI must know the virtual to physical translations
- one solution is to pin the receive buffers in memory

sender
int send_buffer[1024];
recv_id=import(receiver, exp_id);
send(recv_id, send_buffer);

receiver
int receive_buffer[1024];
exp_id=export(buffer, sender);
recv(exp_id);

Software TLB in network interface

- the network interface incorporates a TLB (NI-TLB) which is kept consistent with the virtual memory system
- when a message arrives, NI attempts a virtual to physical translation using NI-TLB
- if a translation is missing in NI-TLB, the processor is interrupted to bring the page in: the kernel increments the reference count for that page to avoid swapping
- when a page entry is evicted from the NI-TLB, the kernel is informed to decrement the reference count
- swapping prevented while DMA in progress