
CS 519

Operating Systems

Spring 2000

Lecture 4

Communication Management

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

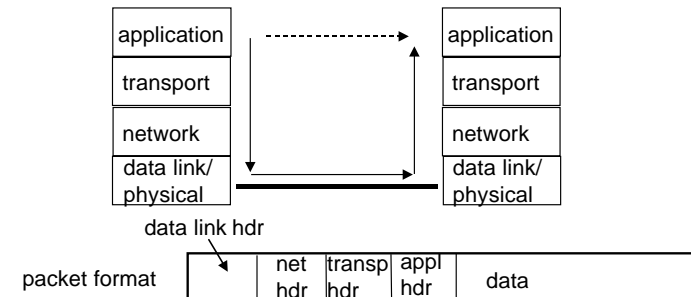
Connection vs. Connectionless

- _ 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

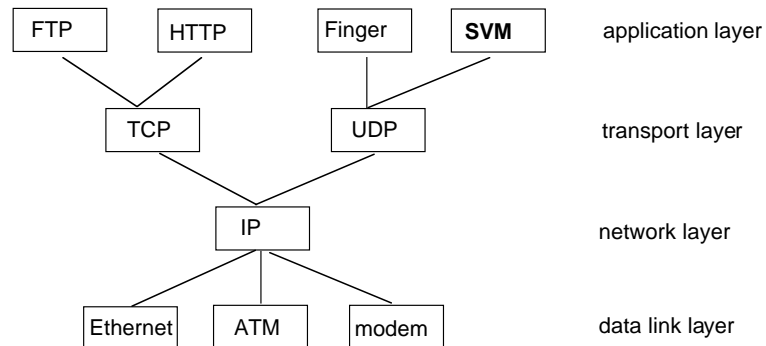
Network Layer

- _ 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



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

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 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

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 it 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 = $\min(\text{receiver window, congestion window})$
- _ 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

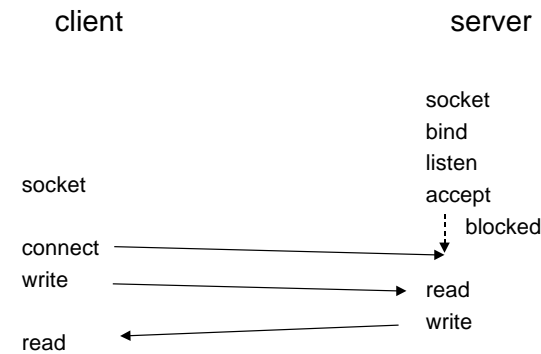
Client-Server Model

- _ 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

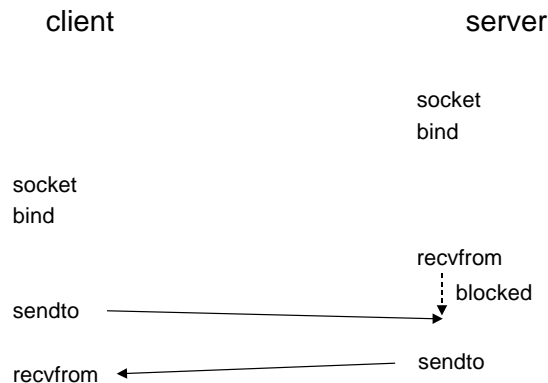
Sockets

- _ 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/rcvfrom* to transmit data connection-oriented or connectionless via sockets

Connection-oriented server



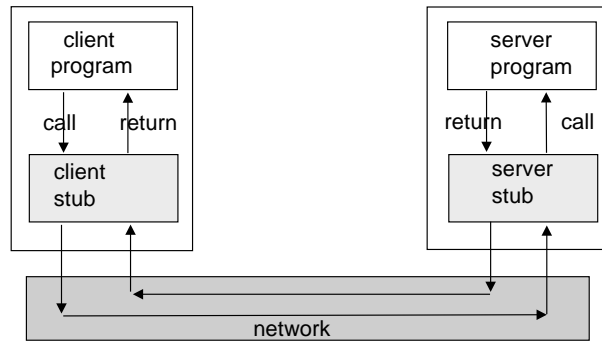
Connectionless server



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



Rutgers University, CS 519, Spring 2000

29

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

Rutgers University, CS 519, Spring 2000

30

Client Stub Example

```
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();
}
```

Rutgers University, CS 519, Spring 2000

31

Server Stub Example

```
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;
        }
    }
}
```

Rutgers University, CS 519, Spring 2000

32

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

```
sender                                receiver
int send_buffer[1024];               int receive_buffer[1024];
recv_id=import(receiver,exp_id);     exp_id=export(buffer, sender);

send(recv_id, send_buffer);          recv(exp_id);
```

- _ 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

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