Motivation

• So far, we talked about mechanisms and policies that
  – Virtualize the underlying machine
  – Support multi-programming of a single machine
• With the proliferation of cheap but powerful machines (devices) and ubiquitous network connections
  – Computing is occurring on more than one machine

Why Distributed Systems?

• Distributed system vs. mainframe
  – Microprocessors offer better price/performance
  – More scalable => more computing power
  – Inherent distribution, e.g. computer-supported cooperative work
  – Reliability, incremental growth
• Distributed system vs. independent PCs
  – Some applications require sharing of data, e.g. airline reservations
  – Sharing of hardware, e.g. expensive devices (color laser printer)
  – Easier human-to-human communication, e.g. electronic mail
  – Spread the workload over the machines in the most effective way
Distributed System

- **Distributed system** is a collection of loosely coupled processors interconnected by a communications network.
- **Reasons for distributed systems**
  - Resource sharing
    - sharing and printing files at remote sites
    - processing information in a distributed database
    - using remote specialized hardware devices
  - Computation speedup – load sharing
  - Reliability – detect and recover from site failure, function transfer, reintegrate failed site
  - Communication – message passing

A Distributed System

- **Network Structure (LAN)**
  - Local-Area Network (LAN) – designed to cover small geographical area (campus networks or enterprise networks).
    - Multisaccess bus, ring, or star network
    - Speed ≈ 100 megabits/second, or higher
    - Broadcast is fast and cheap
    - Nodes:
      - usually workstations and/or personal computers
      - a few (usually one or two) mainframes

- **Ethernet**
  - **Ethernet**
    - (a) classic Ethernet
    - (b) switched Ethernet
Network Structure (WAN)

- Wide-Area Network (WAN) – links geographically separated LANs
  - Point-to-point connections over long-haul lines (optical fiber), microwave, and satellite
- Hosts and routers
  - Hosts are PCs, laptops, servers, and mainframes
  - Routers are specialized switching computers that forward packets and connect networks together.

The Internet

ISO 7-layer Network Model
The ISO Network Message

Internet protocol stack

- **application**: supporting network applications
  - FTP, SMTP, STTP
- **transport**: host-host data transfer
  - TCP, UDP
- **network**: routing of datagrams from source to destination
  - IP, routing protocols
- **link**: data transfer between neighboring network elements
  - Ethernet, Wireless LAN
- **physical**: bits “on the wire”

Fundamental Services

- **Naming service (name resolution)** - How do two processes locate each other to communicate?
  - DNS: IP address <–> host name
  - ARP: IP address <–> MAC address
- **Routing service** - How are messages sent through the network?
- **Connection strategies** - How do two processes send a sequence of messages?
- **Contention** - How to access a shared medium, i.e., how do we resolve conflicting demands for its use?
Domain Name System (DNS)
- DNS is mainly used to translate a hostname into an IP address
  - DNS uses hierarchy to allow local management
  - Split up the data into a hierarchical database

Address Resolution Protocol (ARP)
- A host broadcasts on a LAN a query packet asking for a translation from IP address to MAC address (Media Access Control)
- Hosts which know the translation reply
- Each host knows its own IP and MAC (physical) translation
  - Each host caches a list of IP-to-MAC translations
- Reverse ARP (RARP) translates MAC to IP address and it is used to assign IP addresses dynamically

Routing Services
- Provided by IP layer
- Routing is difficult for large systems, and for systems that change rapidly.
  - The Internet is both large and dynamic
  - Intra-domain routing protocol
    - RIP (distance vector based)
    - OSPF (link state based)
  - Inter-domain routing protocol
    - BGP (path vector policy-based)

Connection Strategies
- Circuit Switching
  - what you get when you make a phone call
  - good when you require constant bit rate
  - good for reserving bandwidth (refuse connection if bandwidth not available)
- Packet Switching
  - what you get when you send a bunch of letters
  - network bandwidth consumed only when sending
  - packets are routed independently
  - packetizing may reduce delays (using parallelism)
Contention
Several hosts may want to transmit information over a shared link simultaneously. Techniques to avoid repeated collisions include:

- **CSMA/CD** - Carrier sense with multiple access (CSMA); collision detection (CD)
  - CSMA: listen before transmit
  - Collisions can still occur due to propagation delay
  - Collisions detected within short time, then
  - Stop transmitting, wait for a random time
  - CSMA/CD is used successfully in Ethernet

- **Token passing** - A unique message type, known as a token, continuously circulates in the system
  - A site that wants to transmit information must wait until the token arrives
  - When the site completes its round of message passing, it retransmits the token

Sockets
A process can ask the OS to create a “socket”, which will be one endpoint of a network connection

- Thread is like a virtual processor
- Address space is like a virtual memory
- A socket is like a virtual network interface card

Ports
- Each socket on a computer has a unique number
  - A process can associate a specific port number with a socket
  - When sending messages through a socket
    - The destination port number is included in each message
    - This allows the destination machine to know which process should receive the message

Client/Server Model
The prevalent model for structuring distributed computation is the client/server paradigm

- a server is a program (or collection of programs) that provides a service to other programs
  - e.g., file server, name server, web server, mail server ...
  - server/service may span multiple nodes (clusters)
  - often, nodes are called servers too
  - e.g., the web server runs on a Dell server computer
- a client is a program that uses the service
  - the client first binds to the server
  - locates it, establishes a network connection to it
  - the client then sends requests (with data) to perform actions, and the server sends responses (with data)
  - e.g., web browser sends a “GET” request, server responds with a web page
Remote Procedure Call

- Why RPC?
  - Procedure call is an accepted and well-understood mechanism for control transfer within a program
    - Presumably, accepted is equivalent to "good" – clean semantics
  - Providing procedure call semantics for distributed computing makes distributed computing much more like programming on a single machine
    - Don't have to worry about remote execution except …
  - Abstraction helps to hide:
    - The possibly heterogeneous nature of the hardware platform
    - The fact that the distributed machines do not share memory

RPC Structure

- Stubs make RPCs look "just" like normal procedure calls
- Binding
  - Naming
  - Location
- Marshalling & Unmarshalling
  - Translate internal data ↔ message representation
  - How to transmit pointer-based data structure (e.g. graph)?
    - Serialization
  - How to transmit data between heterogeneous machines?
    - Virtual data types
- Send/receive messages

RPC Structure (Cont'd)

Remote Procedure Call

- Steps in making a remote procedure call
  - the stubs are shaded gray
RPC binding

- Binding is the process of connecting the client to the server
  - the server, when it starts up, exports its interface
  - identifies itself to a network name server
  - tells RPC runtime that it is alive and ready to accept calls
  - the client, before issuing any calls, imports the server
    - RPC runtime uses the name server to find the location of the server and establish a connection
  - The import and export operations are explicit in the server and client programs
    - a slight breakdown in transparency

RPC marshalling

- Marshalling is the packing of procedure parameters into a message packet
  - the RPC stubs call type-specific procedure to marshal or unmarshal the parameters of an RPC
    - the client stub marshals the parameters into a message
    - the server stub unmarshals the parameters and uses them to invoke the service’s procedure
  - on return:
    - the server stub marshals the return value
    - the client stub unmarshals the return value, and returns them to the client program

RPC transparency

- One goal of RPC is to be as transparent as possible
  - make remote procedure calls look like local procedure calls
  - we’ve seen that binding breaks this transparency
- What else breaks transparency?
  - failures: remote nodes/networks can fail in more ways than with local procedure calls
  - performance: remote communication is inherently slower than local communication