CS343 - Operating Systems

Module-9A

Virtual Machines

Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

http://www.iitg.ac.in/johnjose/

Overview

- Benefits and Features
- Building Blocks
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components

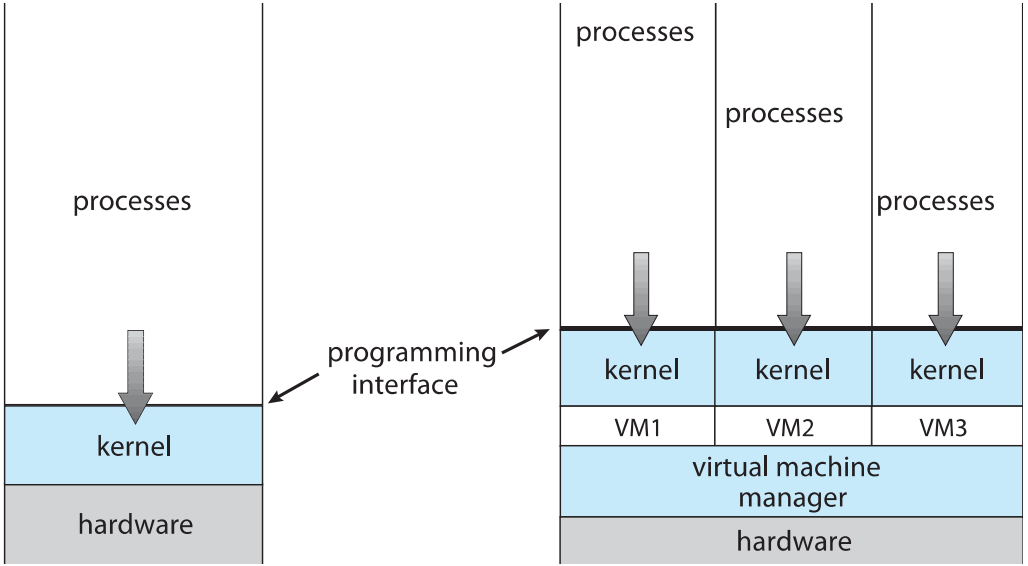
Objectives

- To explore the history and benefits of virtual machines
- To discuss the various virtual machine technologies
- To describe the methods used to implement virtualization
- To show the most common hardware features that support virtualization and explain how they are used by operating-system modules

Virtual Machine Concept

- Facilitate abstract hardware into several different execution environment
- A layer of virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run
  - **Host** – underlying hardware system
  - **Virtual machine manager** (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is **identical** to the host
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

System Models



Non-virtual machine

Virtual machine

Implementation of VMMs

- **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
  - IBM LPARs and Oracle LDOMs are examples
- **Type 1 hypervisors** – OS-like software built to provide virtualization
  - VMware ESX, Joyent SmartOS, and Citrix XenServer
- **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
  - Microsoft Windows Server with HyperV and RedHat Linux with KVM
- **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
  - VMware Workstation, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs

- **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
- **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
  - Used by Oracle Java and Microsoft.Net
- **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU
- **Application containment** - Provides virtualization like features by segregating applications from the OS, making them secure, manageable
  - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs

History

- IBM mainframes (1972) - allowed multiple users to share a batch system
- A VMM provides an environment for programs that is essentially identical to the original machine
- Programs running within that environment show only minor performance decreases
- The VMM is in complete control of system resources
- Intel CPUs (1990s) try virtualizing on general purpose PCs
  - **Xen** and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - A virus less likely to spread
  - Sharing is provided though via shared file system volume, network communication
- Freeze, suspend, running VM
  - Then can move or copy somewhere else and resume
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - Clone by creating copy and running both original and copy
- Run multiple, different OSes on a single machine

Benefits and Features

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- **Live migration** – move a running VM from one host to another!
  - No interruption of user access
- **Cloud computing -** Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

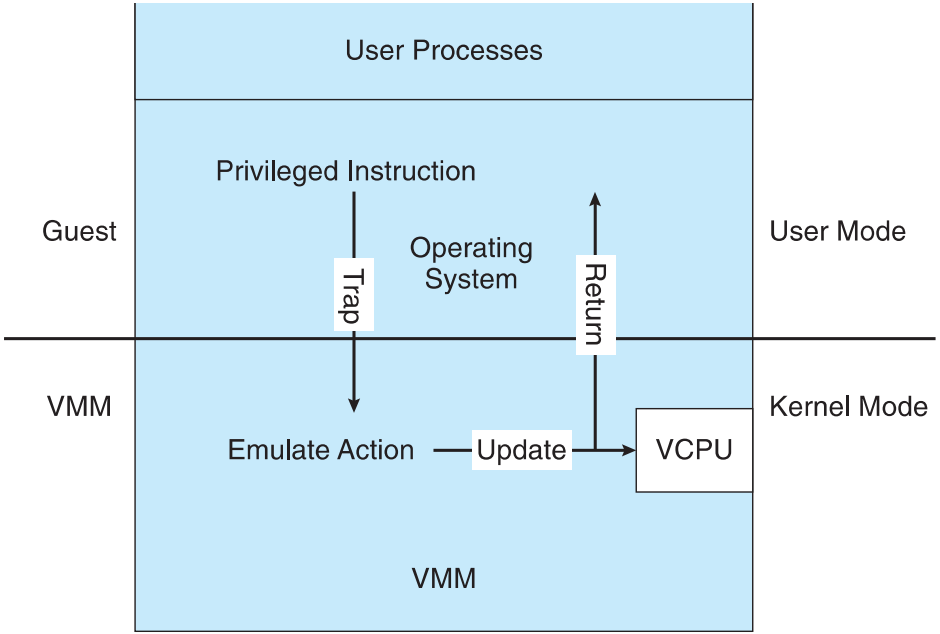
Building Blocks

- Generally difficult to provide an **exact** duplicate of underlying machine when CPU has only dual mode (user and kernel modes)
- Most VMMs implement **virtual CPU** (**VCPU**) to represent state of CPU per guest
- When guest context switched onto CPU by VMM, information from VCPU loaded and stored

Building Block – Trap and Emulate

- Dual mode CPU means guest executes in user mode
  - Kernel runs in kernel mode
  - Not safe to let guest kernel run in kernel mode too
  - So VM needs two modes – virtual user mode and virtual kernel mode
    - Both of which run in real user mode
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode

Trap-and-Emulate Virtualization Implementation



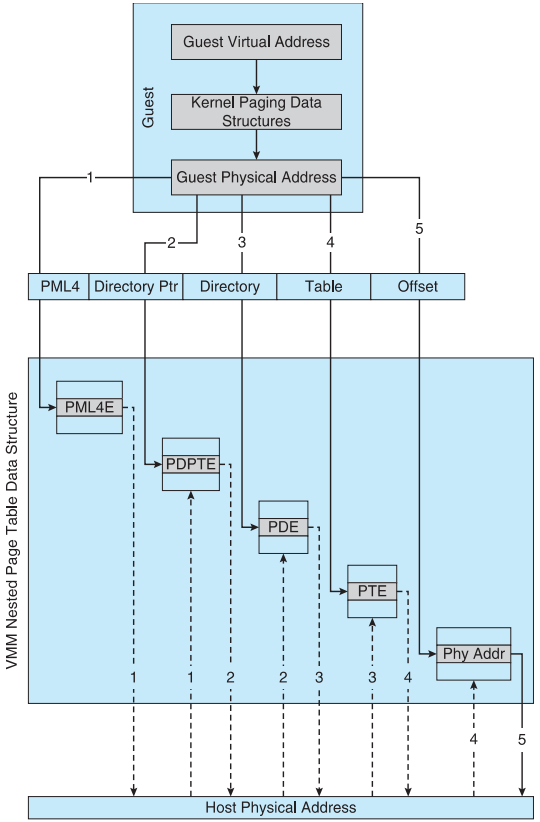
Nested Page Tables

- Memory management another general challenge to VMM implementations
- VMM keeps page-table state for both guests believing they control the page tables, but VMM control the tables.
- Common method is **nested page tables** (**NPTs**)
- Each guest maintains page tables to translate virtual to physical addresses
- VMM maintains per guest NPTs to represent guest's page-table state
- When a guest works on CPU, →VMM makes that guest's NPTs the active system page tables
- When guest tries to change page table, → VMM makes equivalent change to NPTs and its own page tables
  - Can cause many more TLB misses and slower performance

Hardware Support for VMM

- All virtualization needs some HW support
- Generally define more CPU modes – guest and host
- In guest mode, guest OS thinks it is running natively, sees devices as defined by VMM for that guest

Nested Page Tables

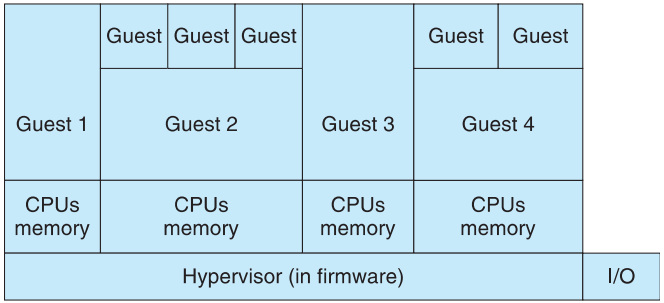


Types of Virtual Machines and Implementations

- Whatever the type, a VM has a lifecycle
  - Created by VMM
  - Resources assigned to it (number of cores, amount of memory, networking details, storage details)
  - In type 0 hypervisor, resources usually dedicated
  - Other types dedicate or share resources, or a mix
  - When no longer needed, VM can be deleted, freeing resource
- Steps simpler, faster than with a physical machine install
  - Can lead to **virtual machine sprawl** with lots of VMs, history and state difficult to track

Types of VMs – Type 0 Hypervisor

- There are partitions and domains
- A HW feature implemented by firmware
- Each guest has dedicated HW
- I/O a challenge as difficult to have enough devices, controllers to dedicate to each guest
- Sometimes VMM implements a **control partition** running daemons that other guests communicate with for shared I/O



Types of VMs – Type 1 Hypervisor

- Datacenter managers control OSes by controlling the Type 1 hypervisor
- Consolidation of multiple OSes and apps onto less HW
- Move guests between systems to balance performance
- Snapshots and cloning
- Special purpose operating systems that run natively on HW
  - Rather than providing system call interface, create run and manage guest OSes
  - Guests generally don’t know they are running in a VM
  - Provide other traditional OS services like CPU and memory management

Types of VMs – Type 1 Hypervisor

- Another variation is a general purpose OS that also provides VMM functionality
  - RedHat Enterprise Linux with KVM, Windows with Hyper-V, Oracle Solaris
  - Perform normal duties as well as VMM duties
  - Typically less feature rich than dedicated Type 1 hypervisors
- Treat guests OSes as just another process

Types of VMs – Type 2 Hypervisor

- Less interesting from an OS perspective
  - Very little OS involvement in virtualization
  - VMM is simply another process, run and managed by host
    - Even the host doesn't know they are a VMM running guests
  - Tend to have poorer overall performance because can't take advantage of some HW features
  - But also a benefit because require no changes to host OS
    - Student could have Type 2 hypervisor on native host, run multiple guests, all on standard host OS such as Windows, Linux, MacOS

Types of VMs – Paravirtualization

- Does not fit the definition of virtualization – VMM not presenting an exact duplication of underlying hardware
- Xen, (leader in paravirtualized space) has simple device abstractions
  - Good communication between guest and VMM about device I/O
  - Each device has circular buffer shared by guest and VMM via shared memory
- Memory management does not include nested page tables
  - Each guest has own read-only tables
  - Guest uses call to hypervisor when page-table changes needed
- Guest had to be modified to use run on paravirtualized VMM

Programming Environment Virtualization

- Programming language is designed to run within custom-built virtualized environment
- For example Oracle Java has many features that depend on running in **Java Virtual Machine** (**JVM**)
- In this case virtualization is defined as providing APIs that define a set of features made available to a language and programs written in that language to provide an improved execution environment
- JVM compiled to run on many systems (including some smart phones even)
- Programs written in Java run in the JVM no matter the underlying system

Types of VMs – Emulation

- Emulation allows guest to run on different CPU
- Necessary to translate all guest instructions from guest CPU to native CPU
  - Emulation, not virtualization
- Useful when host system has one architecture, guest compiled for other architecture
  - Company replacing outdated servers with new servers containing different CPU architecture, but still want to run old applications
- Performance challenge – order of magnitude slower than native code
  - New machines faster than older machines so can reduce slowdown
- Very popular – especially in gaming where old consoles emulated on new

Types of VMs – Application Containment

- Some goals of virtualization are segregation of apps, performance and resource management, easy start, stop, move, and management of them
- Can do those things without full-fledged virtualization
  - If applications compiled for the host operating system, don't need full virtualization to meet these goals
- Oracle **containers** / **zones** for example create virtual layer between OS and apps
  - Only one kernel running – host OS
  - OS and devices are virtualized, providing resources within zone with impression that they are only processes on system
  - Each zone has its own applications; networking stack, addresses, and ports; user accounts, etc

Virtualization and Operating-System Components

- Lot of design challenges at operating system aspects for implementing virtualization
  - CPU scheduling, memory management, I/O, storage, and unique VM migration feature
  - How do VMMs schedule CPU use when guests believe they have dedicated CPUs?
  - How can memory management work when many guests require large amounts of memory?

CPU Scheduling

- Even single-CPU systems act like multiprocessor ones when virtualized
  - One or more virtual CPUs per guest
- Generally VMM has one or more physical CPUs and number of threads to run on them
- When enough CPUs for all guests → VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs → CPU **overcommitment**
  - VMM can use standard scheduling algorithms to put threads on CPUs

Memory Management

- Extra management efficiency from VMM during oversubscription
- Double-paging, in which the guest page table indicates a page is in a physical frame but the VMM moves some of those pages to backing store
- Deduplication by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests
- **Balloon** memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available

Input/Output

- Easier for VMMs to integrate with guests because I/O has lots of variation
  - Already somewhat segregated / flexible via device drivers
  - VMM can provide new devices and device drivers
- But overall I/O is complicated for VMMs
- Networking also complex as VMM and guests all need network access
  - VMM can **bridge** guest to network (allowing direct access)
  - And / or provide **network address translation** (**NAT**)
  - NAT address local to machine on which guest is running, VMM provides address translation to guest to hide its address

Storage Management

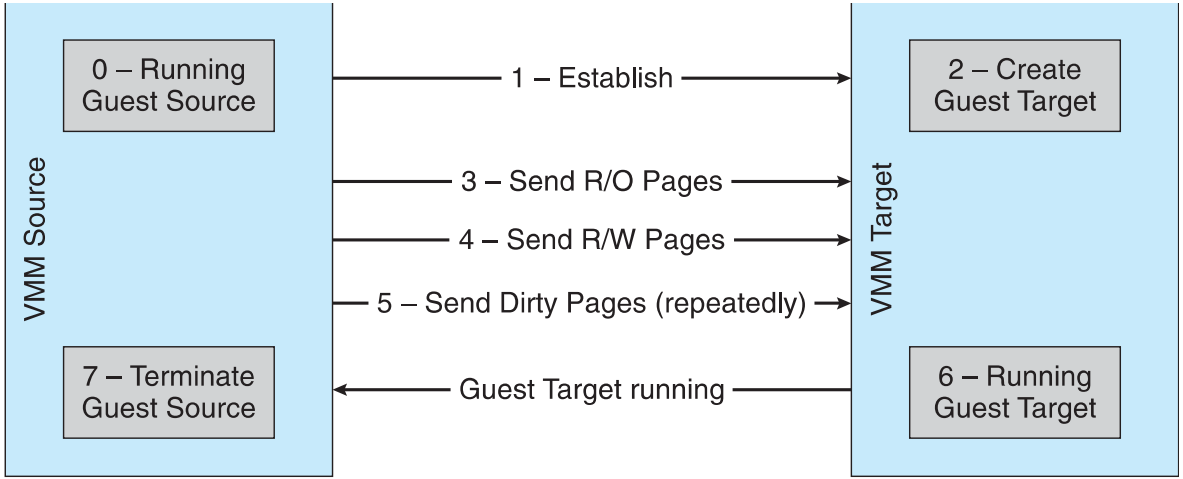
- Need to support many guests per VMM (not by standard disk partition)
- Type 1 → storage of guest root disks and config information within file system provided by VMM as a **disk image**
- Type 2 → store as files in file system provided by host OS
- Duplicate file → create new guest
- Move file to another system → move guest
- **Physical-to-virtual** (**P-to-V**) convert native disk blocks into VMM format
- **Virtual-to-physical** (**V-to-P**) convert virtual format to native disk format

Live Migration

- Running guest can be moved between systems, without interrupting user access to the guest or its apps
- Very useful for resource management, maintenance downtime, etc

1. The source VMM establishes a connection with the target VMM
2. The target creates a new guest by creating a new VCPU, etc
3. The source sends all read-only guest memory pages to the target
4. The source sends all read-write pages to the target, marking them clean
5. The source repeats step 4, as during that step some pages were probably modified by the guest and are now dirty
6. Source VMM sends VCPU's final state details, sends final dirty pages, and tells target to start running the guest
7. Once target acknowledges that guest running, source terminates guest

Live Migration of Guest Between Servers



johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/

CS343 - Operating Systems

Module-9B

Distributed Systems

Dr. John Jose

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Guwahati, Assam.

http://www.iitg.ac.in/johnjose/

Overview

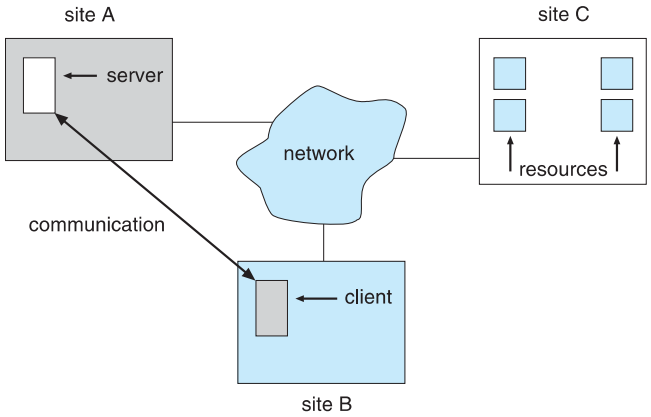
- Network Operating Systems vs Distributed Systems
- Network Structure
- Communication Structure and Protocols
- Design Issues
- Distributed File System

Objectives

- To provide a high-level overview of distributed systems and the networks that interconnect them
- To discuss the general structure of distributed operating systems
- To explain general communication structure and communication protocols
- To describe issues concerning the design of distributed systems

Distributed System

- **Distributed system** is collection of loosely coupled processors interconnected by a communications network
- Processors variously called **nodes, computers, machines, hosts**
  - **Site** is location of the processor
  - Generally a **server** has a resource a **client** node at a different site wants to use



Distributed System

- **Resource sharing**
  - Sharing and printing files at remote sites
  - Processing information in a distributed database
  - Using remote specialized hardware devices
- **Computation speedup** – **load sharing** or **job migration**
- **Reliability** – detect and recover from site failure, function transfer, reintegrate failed site
- Communication – **message** passing
- Better user interfaces and easier maintenance by moving from large system to multiple smaller systems performing distributed computing

Types of Distributed Operating Systems

- Network Operating Systems
- Distributed Operating Systems

Network-Operating Systems

- Users are aware of multiplicity of machines
- Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine (telnet, ssh)
  - Remote Desktop (Microsoft Windows)
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism
- Users must change paradigms – establish a **session**, give network-based commands
  - More difficult for users

Distributed-Operating Systems

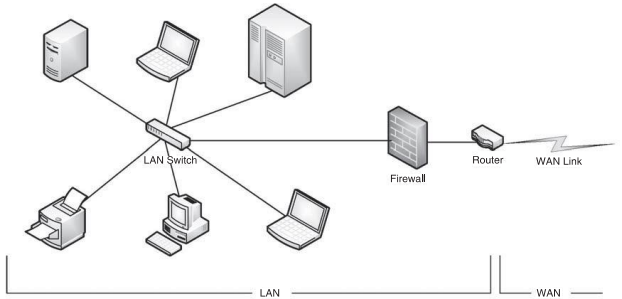
- Users not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
- **Data Migration** – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- **Computation Migration** – transfer the computation, rather than the data, across the system
  - Via remote procedure calls (RPCs)
  - Via messaging system

Distributed-Operating Systems

- **Process Migration** – run an entire process, or parts of it, at different sites
- **Load balancing** – distribute processes across network to even the workload
- **Computation speedup** – subprocesses can run concurrently on different sites
- **Hardware preference** – process execution may require specialized processor
- **Software preference** – required software may be run at only a particular site
- **Data access** – run process remotely, rather than transfer all data locally

Network Structure

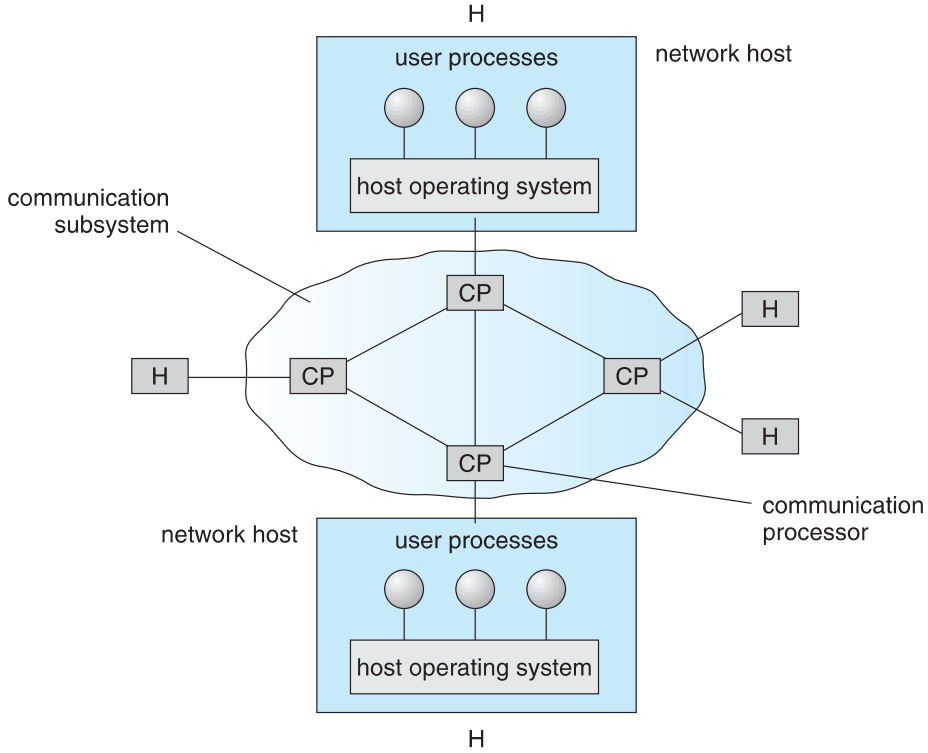
- **Local-Area Network** (**LAN**) – designed to cover small geographical area
  - Multiple topologies like star or ring
  - Speeds from 1Mbps per second to 40 Gbps for fastest Ethernet over twisted pair copper or optical fibre
  - Consists of multiple computers (mainframes through mobile devices), peripherals (printers, storage arrays), routers (specialized network communication processors) providing access to other networks



Network Structure

- **Wide-Area Network** (**WAN**) – links geographically separated sites
  - Point-to-point connections over long-haul lines
    - Implemented via **connection processors** known as **routers**
  - Internet WAN enables hosts world wide to communicate
    - Hosts differ in all dimensions but WAN allows communications
  - WANs and LANs interconnect, similar to cell phone network:
    - Cell phones use radio waves to cell towers
    - Towers connect to other towers and hubs

Communication Processors in a Wide-Area Network



Communication Structure – Design Issues

- **Naming and name resolution.**
  - How do two processes locate each other to communicate?
- **Routing strategies**
  - How are messages sent through the network?
- **Connection strategies**
  - How do two processes send a sequence of messages?
- **Contention**
  - The network is a shared resource, so how do we resolve conflicting demands for its use?

Naming and Name Resolution

- Name systems in the network
- Address messages with the process-id
- Identify processes on remote systems by <**host-name**, **identifier**> pair
- **Domain name system** (**DNS**) – specifies the naming structure of the hosts, as well as name to address **resolution** (Internet)

Routing Strategies

- **Fixed routing** - A path from A to B is specified in advance; path changes only if a hardware failure disables it
  - Since the shortest path is usually chosen, communication costs are minimized
  - Fixed routing cannot adapt to load changes
  - Ensures that messages will be delivered in the order in which they were sent
- **Virtual routing**- A path from A to B is fixed for the duration of one session. Different sessions involving messages from A to B may have different paths
  - Partial remedy to adapting to load changes
  - Ensures that messages will be delivered in the order in which they were sent

Routing Strategies

- **Dynamic routing** - The path used to send a message from site *A* to site *B* is chosen only when a message is sent
  - Usually a site sends a message to another site on the link least used at that particular time
  - Adapts to load changes by avoiding routing messages on heavily used path
  - Messages may arrive out of order – managed by a sequence number

Routing Strategies

- **Router** is communications processor responsible for routing messages
- Must have at least 2 network connections
- Maybe special purpose or just function running on host
- Checks its tables to determine where destination host is, where to send messages
  - Static routing – table only changed manually
  - Dynamic routing – table changed via **routing protocol**

Routing Strategies

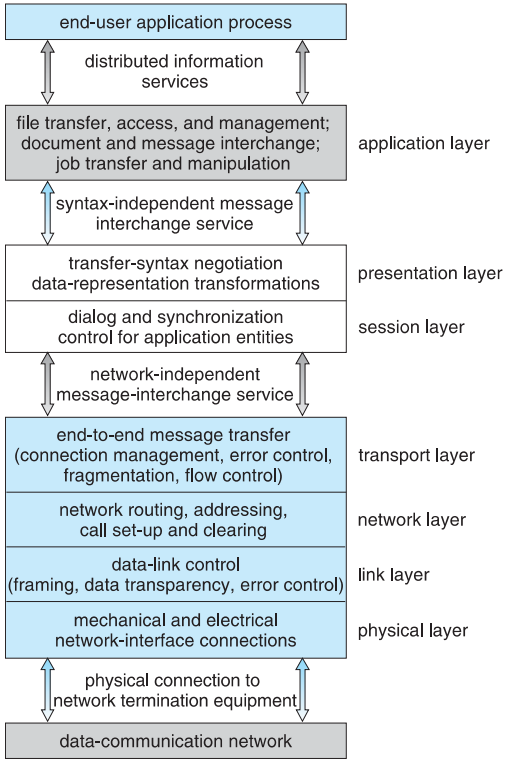
- More recently, routing managed by intelligent software more intelligently than routing protocols
  - **OpenFlow** is device-independent, allowing developers to introduce network efficiencies by decoupling data-routing decisions from underlying network devices
- Messages vary in length – simplified design breaks them into **packets** (or **frames**, or **datagrams**)
- **Connectionless message** is just one packet
  - Otherwise need a connection to get a multi-packet message from source to destination

Connection Strategies

- **Circuit switching** - A permanent physical link is established for the duration of the communication (i.e., telephone system)
- **Message switching** - A temporary link is established for the duration of one message transfer (i.e., post-office mailing system)
- **Packet switching** - Messages of variable length are divided into fixed-length packets which are sent to the destination
  - Each packet may take a different path through the network
  - The packets must be reassembled into messages as they arrive
- Circuit switching requires setup time, but incurs less overhead for shipping each message, and may waste network bandwidth
- Message and packet switching require less setup time, but incur more overhead per message

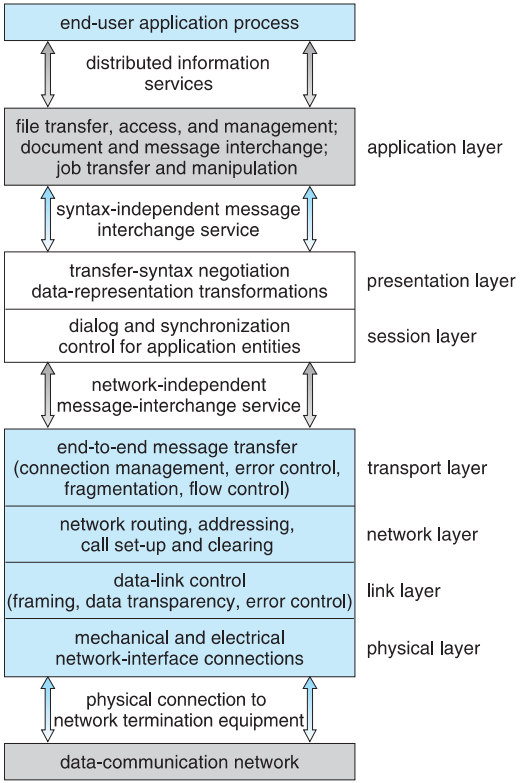
Communication Protocol

- **Layer 1: Physical layer** – handles the mechanical and electrical details of the physical transmission of a bit stream
- **Layer 2: Data-link layer** – handles the frames, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer



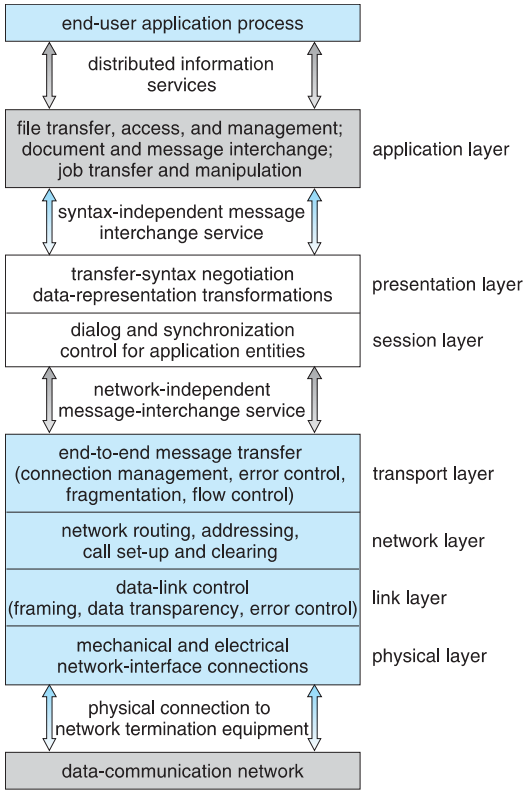
Communication Protocol

- **Layer 3: Network layer** – provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels
- **Layer 4: Transport layer** – responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow

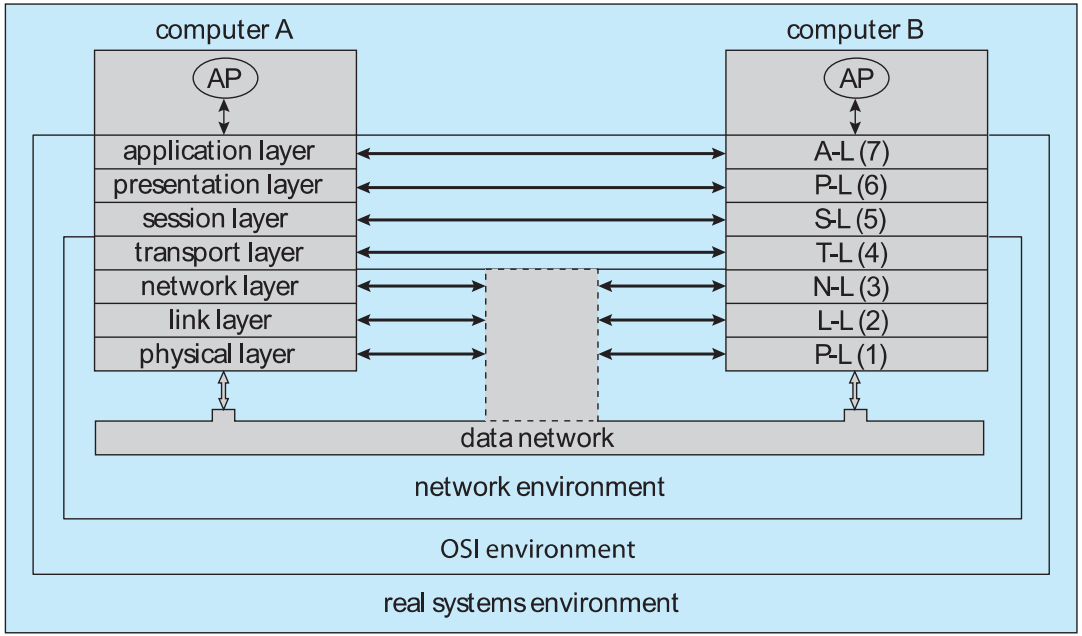


Communication Protocol

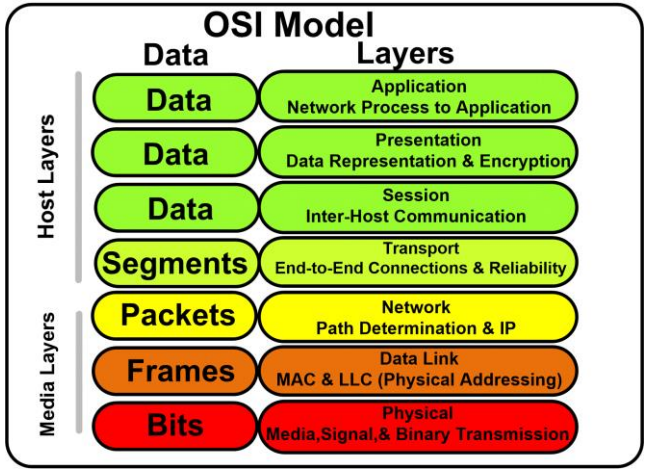
- **Layer 5: Session layer** – implements sessions, or process-to-process communications protocols
- **Layer 6: Presentation layer** – resolves the differences in formats among the various sites in the network, including character conversions
- **Layer 7: Application layer** – interacts directly with the users, deals with file transfer, remote-login protocols and electronic mail, as well as schemas for distributed databases

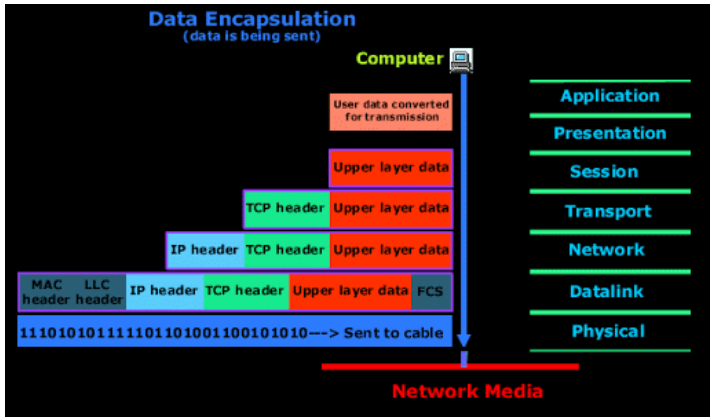


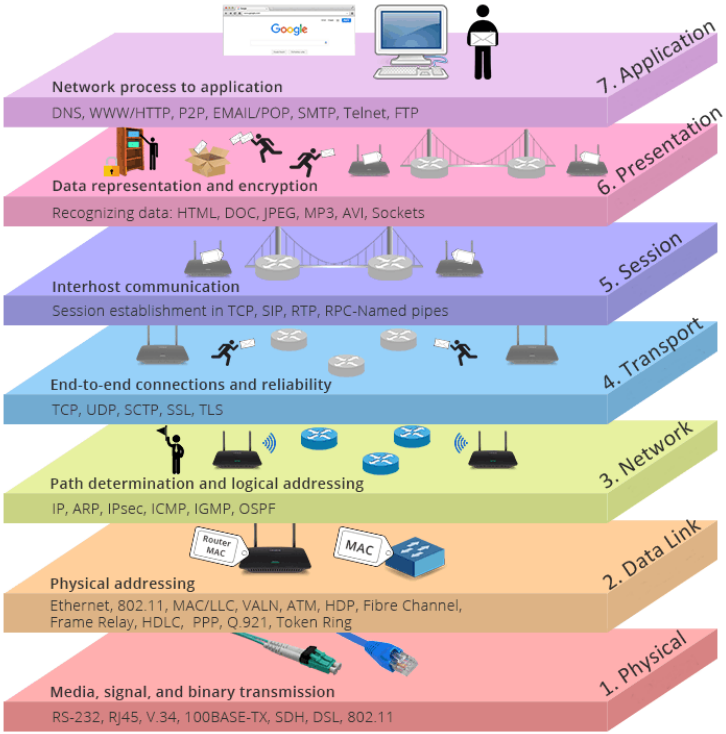
Communication Via ISO Network Model



Communication Via ISO Network Model







Robustness- Failure Detection

- Detecting hardware failure is difficult
- To detect a link failure, a **heartbeat** protocol can be used
- Assume Site A and Site B have established a link
  - At fixed intervals, each site will exchange an I-am-up message indicating that they are up and running
- If Site A does not receive a message within the fixed interval, it assumes either (a) the other site is not up or (b) the message was lost
- Site A can now send an Are-you-up? message to Site B
- If Site A does not receive a reply, it can repeat the message or try an alternate route to Site B

Robustness- Failure Detection

- If Site A does not ultimately receive a reply from Site B, it concludes some type of failure has occurred
- Types of failures:
  Site B is down
  The direct link between A and B is down
  The alternate link from A to B is down
  The message has been lost
- However, Site A cannot determine exactly **why** the failure has occurred

Robustness- Reconfiguration

- When Site A determines a failure has occurred, it must reconfigure the system:
  - If the link from A to B has failed, this must be broadcast to every site in the system
  - If a site has failed, every other site must also be notified indicating that the services offered by the failed site are no longer available
- When the link or the site becomes available again, this information must again be broadcast to all other sites

Design Issues

- **Transparency** – the distributed system should appear as a conventional, centralized system to the user
- **Fault tolerance** – the distributed system should continue to function in the face of failure
- **Scalability** – as demands increase, the system should easily accept the addition of new resources to accommodate the increased demand

Distributed File System

- **Distributed file system** (**DFS**) – a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources
- A DFS manages set of dispersed storage devices
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces
- There is usually a correspondence between constituent storage spaces and sets of files
- Challenges include:
  - Naming and Transparency
  - Remote File Access

DFS Structure

- **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients
- **Server** – service software running on a single machine
- **Client** – process that can invoke a service using a set of operations that forms its client interface
- A client interface for a file service is formed by a set of primitive file operations (create, delete, read, write)
- Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files

Naming and Transparency

- **Naming** – mapping between logical and physical objects
- **Multilevel mapping** – abstraction of a file that hides the details of how and where on the disk the file is actually stored
- A **transparent** DFS hides the location where in the network the file is stored
- **Location transparency** – file name does not reveal the file's physical storage location
- **Location independence** – file name does not need to be changed when the file's physical storage location changes
- For a file being **replicated** in several sites, the mapping returns a set of the locations of this file's replicas; both the existence of multiple copies and their location are hidden

Remote File Access

- **Remote-service mechanism - r**educe network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally
  - If needed data not already cached, a copy of data is brought from the server to the user
  - Accesses are performed on the cached copy
  - Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches
  - **Cache-consistency problem** – keeping the cached copies consistent with the master file

Cache Update Policy

- **Write-through** – write data through to disk as soon as they are placed on any cache
- **Delayed-write** (**write-back**) – modifications written to the cache and then written through to the server later
  - Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all
  - Poor reliability; unwritten data will be lost when a user machine crashes
  - Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan
  - **write-on-close**, writes data back to the server when the file is closed
    - Best for files that are open for long periods and frequently modified

Consistency

- Is locally cached copy of the data consistent with the master copy?
- **Client-initiated approach**
  - Client initiates a validity check
  - Server checks whether the local data are consistent with the master copy
- **Server-initiated approach**
  - Server records, for each client, the (parts of) files it caches
  - When server detects a potential inconsistency, it must react

johnjose@iitg.ac.in
http://www.iitg.ac.in/johnjose/