Slide 1

Distributed Systems

COMP3231 Operating Systems

2005 S2

Slide 2

Today

- ➜ Challenges in Distributed Systems
- ➜ Client Server Architecture
- ➜ Message Passing
- ➜ Remote Procedure Call
- ➜ Remote Method Invocation
- ➜ TCP/IP

There is an extra subject
Distributed Systems (COMP9243).

Slide 3

Distributed Systems

What is a *distributed system?*

- ➜ Andrew Tannenbaum defines it as follows:

  > *A distributed system is a collection of independent computers that appear to the users of the system as a single computer.*

- ➜ Is there any such system? Hardly!
- ➜ You can learn about the challenges in building "true" distributed systems in COMP9243

For the time being, we would like a weaker definition of distributed systems:

> *A distributed system is a collection of independent computers that are used jointly to perform a single task or to provide a single service.*

Slide 4

Examples of distributed systems

- ➜ Collection of Web servers: distributed database of hypertext and multimedia documents
- ➜ Distributed file system in a LAN (e.g., NFS as used at CSG)
- ➜ Point-of-sale system hooked up to a back office data center
- ➜ Domain Name Service (DNS)
- ➜ Cray T3E, UNICOS/mk

The Advantages and Challenges of Distributed Systems

Slide 5

What are economic and technical reasons for having distributed system?

Cost. Better price/performance as long as commodity hardware is used for the component computers

Performance. By using the combined processing and storage capacity of many nodes, performance levels can be reached that are out of the scope of centralised machines

Scalability. Resources such as processing and storage capacity can be increased incrementally

Inherent distribution. Some applications like the Web are naturally distributed

Reliability. By having redundant components, the impact of hardware and software faults on users can be reduced

Slide 6

Which problems are there in the use and development of distributed systems?

Limited software. Distributed software is harder to develop than conventional software; hence, it is more expensive and there is fewer software available

New component: network. Networks are needed to connect independent nodes and are subject to performance limits and constitute another potential point of failure

Security. It is easier to compromise distributed systems

Distributed Services
From Network OSes to Distributed Systems

Slide 7

What is a Network OS?

- ➜ Network of application systems
- ➜ Configuration with one or more servers
- ➜ Servers provide network wide services or applications
- ➜ Network OS is adjunct to local OS which supports interaction between application machines and servers
- ➜ User is aware of single machines, must deal with them explicitely

Network OSes provide the following:

- ➜ Services for remote login (`telnet`, `rsh`, and `ssh`)
- ➜ File transfer (`ftp`, `rcp`, and `scp`)

Slide 8

How far is a network OS away from a distributed system?

- ➜ Network OS lacks a single image view for any of its services
- ➜ Individual nodes are highly autonomous
- ➜ All distribution of tasks is explicit to the user

Slide 9

With extra software, network OSes may provide some distributed services:

Data sharing. Common data needs to be accessed and updated (e.g., distributed file systems, Web)

Device sharing. Common peripherals need to be used remotely (e.g., printers)

Flexibility. Workloads can be distributed or moved to less loaded machines (e.g., remote login)

Communication. Email, IM, and so on

However, the user usually is aware of the distribution.

Slide 10

Distributed Systems and Parallel Computing

- ➜ Parallel systems: improved performance by multiple processors per application
- ➜ There are two flavours:
  1. Shared-memory systems:
     - Multiple processor share a single bus and memory unit
     - SMP support in OS
     - Much simpler than distributed systems
     - Limited scalability
  2. Distributed memory systems:
     - Multiple nodes connected via a network
     - These are a form of distributed systems
     - Share many of the challenges discussed here
     - Better scalability & cheaper

Slide 11

Basic Problems and Challenges in Distributed Systems

The distributed nature of these systems brings some inherent challenges:

- ➜ Transparency ⇐
- ➜ Flexibility
- ➜ Reliability
- ➜ Performance
- ➜ Scalability ⇐

Slide 12

Transparency:

Concealment of the separation of the components of a distributed system (single image view).

There are different kinds of transparency

Location: Users unaware of location of resources

Migration: Resources can migrate without name change

Replication: Users unaware of existence of multiple copies

Failure: Users unaware of the failure of individual components

Concurrency: Users unaware of sharing resources with others

Parallelism: Users unaware of parallel execution of activities

Slide 13

Scalability:

- ➜ Centralised resources become performance bottlenecks:
  - components (single server),
  - tables (directories), or
  - algorithms (based on complete information).
- ➜ Bottleneck can be resources *or* communication with them

Helpful design rules:

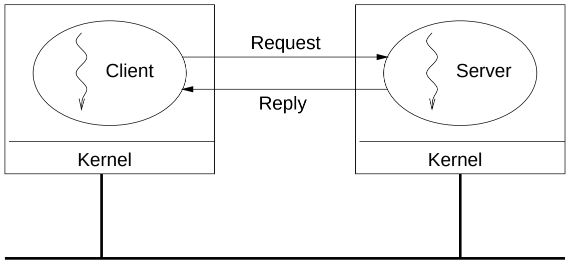
- ➜ Do not require any machine to hold complete system state
- ➜ Allow nodes to make decisions based on local info
- ➜ Algorithms must survive failure of nodes
- ➜ No assumption of a global clock

Scalability often conflicts with (small system) performance

Slide 14

CLIENT-SERVER ARCHITECTURE

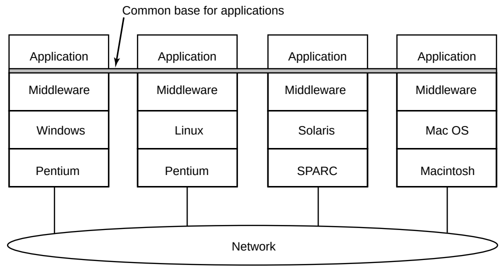
Basic architectural building block for distributed systems:



- ➜ Simple, connectionless request-reply protocol is sufficient
- ➜ Support in the form of stub generators etc. is possible

Slide 15

What is middleware?



- ➜ Abstraction layer in the middle, between OS and applications
- ➜ Less OS dependencies in the applications
- ➜ Varying degrees of transparency
- ➜ Typically two components: communications abstraction & services

---

Slide 16

The three most common communications abstractions:

① Message passing:
- Simple, light-weight
- Application has to do a lot of tedious work (e.g., marshalling)
- Sockets, Message Passing Interface (MPI)

② Remote Procedure Call (RPC):
- The idea: remote access apears as a local procedure call
- The called procedure is executed on the server
- Often stub generators or similar tool supported is available
- SUN RPC, XML-RPC, Simple Object Access Protocol (SOAP)

③ Remote Method Invocation (RMI):
- The idea:
  - Server is a remote object
  - remote access apears as a local method invocation
- IDL compiler or other form of stub generation supported
- CORBA, Java RMI, DCOM

Slide 17

Message Passing

- ➜ Messaging layer supports send and receive operations
- ➜ The application has to implement marshalling:
  - Conversion of in-memory to on-wire representation of data structures
  - Bridging architectural variants, e.g., byte order
- ➜ The application may also have to handle naming:
  - Bind names to remote services
  - Resolve names: name → location of service
  - Migration of services

Example: socket interface:

```
int socket (int domain, int type, int protocol);
int send (int s, const void *msg, size_t len, int flags);
int recv (int s,       void *buf, size_t len, int flags);
```

Slide 18

Name	Purpose
PF_UNIX, PF_LOCAL	Local communication
PF_INET	IPv4 Internet protocols
PF_INET6	IPv6 Internet protocols
PF_IPX	IPX - Novell protocols
PF_NETLINK	Kernel user interface device
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25 protocol
PF_ATMPVC	Access to raw ATM PVCs
PF_APPLETALK	Appletalk
PF_PACKET	Low level packet interface

Slide 19

Sample message format:

```
struct message {
   nodeid_t source;           /* system supplied */
   nodeit_t dest;             /* receiver identity */
   int      opcode;           /* which operation */
   int      count;            /* data size */
   char     object[N_NAME];   /* name of target object */
   char     data[BUF_SIZE];   /* data to be transferred */
};
```

Sample send code:

```
struct message msg;
msg.source = me ();
msg.dest   = somewhere;
msg.opcode = DO_SOMETHING_COOL;
msg.count  = N;
strncpy (&msg.object, "My cool object", N_NAME);
marshall_data (&msg.data, whatever, BUF_SIZE);
send (s, &msg, sizeof (msg));
```

Slide 20

There different flavours of point-to-point communication:

- ➜ Blocking versus non-blocking communication ⇐
- ➜ Reliable versus unreliable communication
- ➜ Buffered versus unbuffered messages ⇐

Slide 21

Blocking versus non-blocking communication:

➜ Blocking (synchronous):
- client blocked until reply arrives
- delivery guarantee
- latency can be significant (infinite?)

➜ Non-blocking (asynchronous):
- client can perform other processing
- client must not modify message buffer until transmitted
  - kernel buffers message
  - kernel interrupts client when buffer processed

Several factors may influence a decision:

➜ Blocked client not a problem if multitasked/mulitthreaded

➜ Kernel buffering is overhead

➜ Interrupts are overhead, and tricky to program

Slide 22

Reliable versus unreliable communication

➜ Generally, messages may get lost (network failure, node down, server crashed, . . .)

➜ Unreliable communication:
- Messaging layer does not make any guarantees
- Application has to handle message loss

➜ Reliable communication:
- Messaging layer guarantees delivery if possible
- Advantage:
  - Application code gets simpler
- Disadvantage:
  - Application-specific protocol properties cannot be exploited
    ⇒ often more expensive

Slide 23

REMOTE PROCEDURE CALL (RPC)

Idea: Replace I/O oriented message passing model by execution of a procedure call on a remote node:

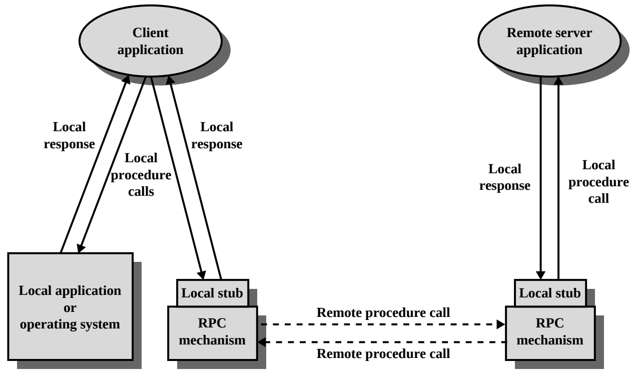
➜ Based on blocking messages

➜ Message-passing details hidden from application

➜ Procedure call parameters used to transmit data

➜ Client calls local “stub” which does messaging and marshalling

Slide 24



Slide 25

Sample Stub: often generated from a high-level specification

```
read(int fd, void *buf, size_t count) {
   int result;
   msg.dest            = FS_SERVER_ID;
   msg.opcode          = FS_READ;
   msg.count           = 2*sizeof(int);
   ((int*)msg.data)[0] = fd;
   ((int*)msg.data)[1] = count;
   send (s, &msg, sizeof (msg));  /* send request  */
   recv (s, &msg, sizeof (msg));  /* receive reply */
   result = *((int *) msg.data);
   if (result >=0)
      bcopy(&msg.data[1], buf, result);
   return result;
}
```

Slide 26

Application side:

- ➜ Just calls

```
result = read (this_fd, &my_buf, n);
```

- ➜ The procedure call hides all the marshalling and messaging complexity

Slide 27

Parameter Marshalling

- ➜ stub must pack ("marshal") parameters into message structure
- ➜ message data must be pointer free
  by-reference data must be passed by-value
- ➜ may have to perform other conversions:
  - byte order (big endian vs little endian)
  - floating point format...
  - convert everything to standard ("network") format, or
  - message indicates format, receiver converts if necessary
- ➜ stubs may be generated automatically from interface specs

Slide 28

POSSIBLE PROBLEMS WITH RPC

RPC can fail in ways not possible for "real" procedure calls:

- ➜ Cannot locate service (down, wrong version, migrating)
- ➜ request lost
- ➜ reply lost
- ➜ server crash
- ➜ client crash

Need error values for functions that cannot fail locally.
⇒ Limits the illusion of "procedure call" (lack of transparency)

Disjoint address space:

- ➜ Concurrent access to global program variables (`errno`)
- ➜ Need for stub to know size of all parameters (open arrays)
- ➜ Arbitrary (pointer) data structures cannot be marshalled

Slide 29

Remote Message Invocation (RMI)

The transition from Remote Procedure Call (RPC) to Remote Method Invocation (RMI) is a transition from the server metaphor to the object metaphor.

Why is this important?

- ➜ There is no inherent link between procedure calls and issuing server requests, but
- ➜ There certainly is an intimate link between method invocations and the use of objects.
- ➜ RPC: explicit handling of host identification to determine the destination
- ➜ RMI: addressed to a particular state-encapsulating entity (object)
- ➜ Objects are first-class citizens
- ➜ More natural resource management and error handling
- ➜ But still only a small evolutionary step

Slide 30

References:

- ➜ Objects are identified by object references
- ➜ Distributed objects are identified by remote object reference
- ➜ The latter are more difficult to implement (why?)

Interfaces:

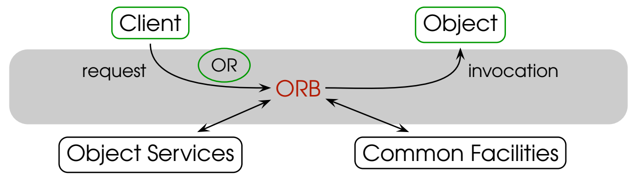
- ➜ Access to objects is controlled by interfaces
- ➜ Which contain the signatures of a set of methods
- ➜ Signatures include argument and result types of a method

```
struct Person {
  string name;
  string place;
  long   year;};
interface PersonList {
  readonly attribute string listname;
  void addPerson (in Person p);
  void getPerson (in string name, out Person p);
  long number ();};
```


Slide 31

The Architecture of CORBA

- ➜ The concept of an Object Request Broker (ORB) is the
- ➜ centerpiece of OMG's Common Object Request Broker Architecture (CORBA).



Tasks of an ORB:

- ➜ Find object implementation
- ➜ Prepare object implementation (activation)
- ➜ Communicate data

Slide 32

Corba

- ➜ First version did not specify protocol between client and server ORB
- ➜ Non-CORBA objects can be integrated using *object adapter*

Drawbacks:

- ➜ Each object is located on a single server only
- ➜ Mostly used on small scale systems

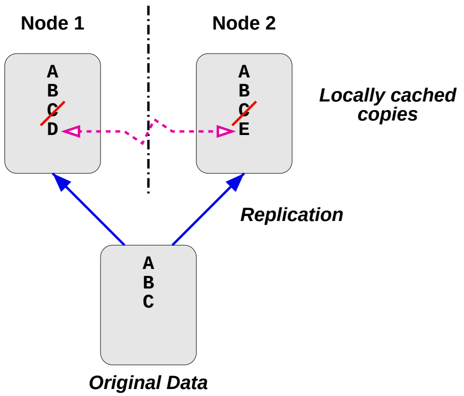
Slide 33

DATA CONSISTENCY

Common problem in distributed systems: data consistency:

- ➜ Consistency of virtual shared-memory systems
- ➜ Consistency of distributed file services
- ➜ Consistency of naming information
- ➜ Consistency of a snapshot of the global state of a system

Slide 34



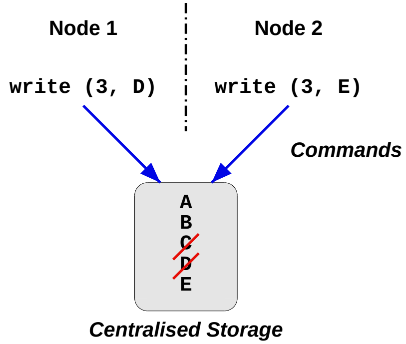
Slide 35

Why do these consistency problems arise?

- ➜ Absence of strict central control
- ➜ Presence of caches
- ➜ Replication of data

Central control:

- ➜ Central bottleneck
- ➜ Does not scale



Slide 36

CONCRETE EXAMPLES

Let's look at the consistency problem in

- ➜ distributed shared memory and
- ➜ distributed file systems.

Slide 37

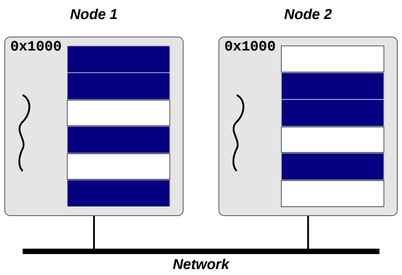
DISTRIBUTED SHARED MEMORY (DSM)

- ➜ A set of processes on different hosts share part of their address space
- ➜ DSM system guarantees that updates by one process are available to others
- ➜ Data exchange often at the granularity of individual memory pages
- ➜ Easy to use (no marshalling, but synchronisation primitives required)

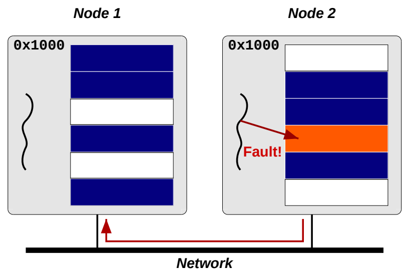
How does it work?

- ➜ Virtual memory management is extended
- ➜ In case of a page fault, the page may be requested from a remote node

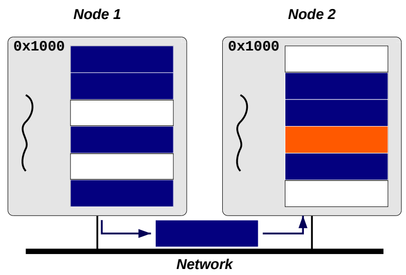
Slide 38



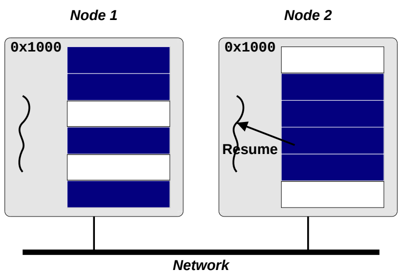
Slide 39



Slide 40



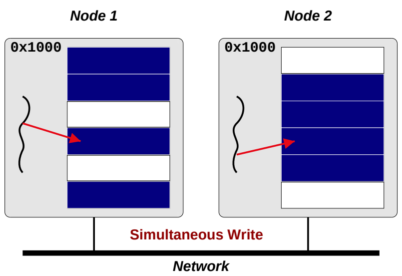
Slide 41



Slide 42

Consistency problem:

- ➜ Concurrent write access to the same page



- ➜ Simplest solution: multiple-reader/single-writer policy
- ➜ Lock whole page ⇒ expensive
- ➜ Access maybe to same page, but different memory location

Slide 43

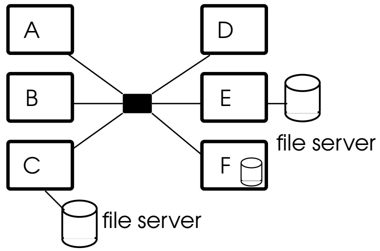
DISTRIBUTED FILE SYSTEMS (DFS)

In a DFS

- ➜ multiple clients share
- ➜ multiple file servers, which may support
- ➜ differing types of file systems.

The client-side structure of the file system may

- ➜ consist of mixture of directories & files from local & remote devices and
- ➜ be different for each client.

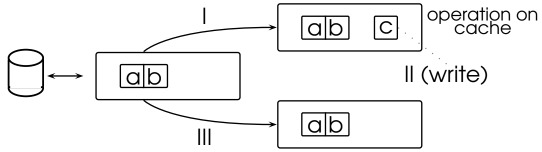


Slide 44

Semantics of file access:

UNIX semantics:

- ➜ A READ after a WRITE returns the value just written
- ➜ When two WRITES follow in quick succession, the second persists
- ➜ Trivial with a single file server and without caching, but. . .



- ➜ Caches are needed for performance & write-through is expensive
- ➜ Multiple file servers aggravate the problem

⟹ transparency for a UNIX system is problematic

Slide 45

How can we solve this problem?

① We stay faithful to the semantics and compromise on performance, or

② we search for an alternative semantics that can be implemented more efficiently.

➜ The second alternative is quite popular (NFS & CODA)

➜ What are feasible semantics?

Slide 46

Session semantics:

➜ Changes to an open file are only locally visible

➜ When a file is closed, changes are propagated to the server (and other clients)

➜ Easy to implement, but there are tradeoffs. For example,

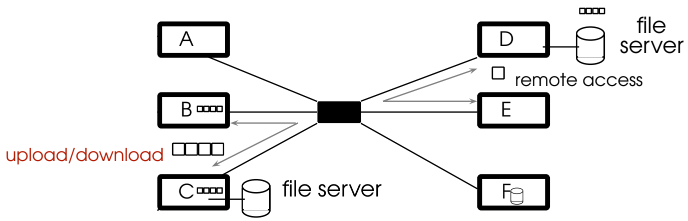
- parent and child processes cannot share file pointers if running on different machines.

```
% cat twoecho
#!/bin/sh
/bin/echo "a"
/bin/echo "b"
% twoecho >output
% cat output
a
b
```

Slide 47

Implementation of session semantics:

➜ Upload/download model



① Download the whole file to the client (often more efficient)

② Update in cache

③ Upload file to server on `close()`

Slide 48

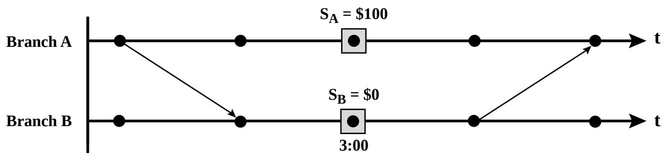
Consistency of Global State

Why is determining the global state of a system difficult?

➜ Lack of global clock/synchronisation

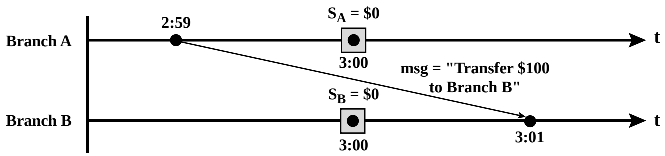
➜ Messages may be in the network

Slide 49



➜ Sum at 3:00 is $100

Slide 50



➜ Message currently in transit

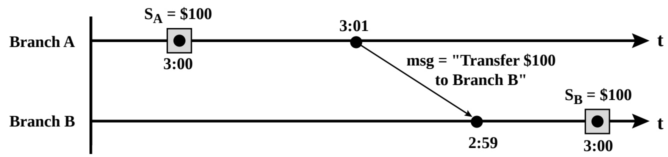
➜ Sum at 3:00 is $0

What can we do?

➜ Include record of transfers

➜ Check against receipts

Slide 51



➜ Clocks are not synchronised

➜ Sum at 3:00 is $200

What can we do?

➜ Define a notion of consistent global state

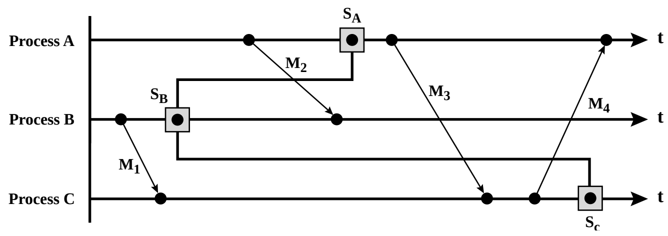
➜ Make sure we only take consistent distributed snapshots

Slide 52

When is a distributed snapshot conistent?

➜ Snapshot of a process includes all messages that have been sent or received since the last snapshot

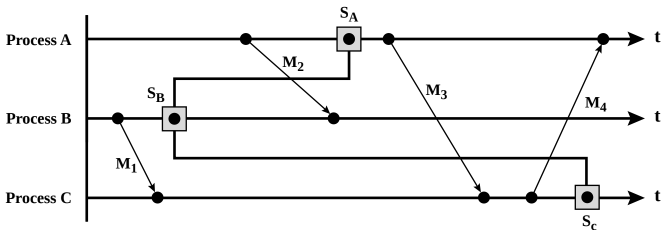
➜ Distributed snapshot is a collection of snapshots, one for each process



Slide 53

A distributed snapshot is consistent

- ➜ if any message recorded as received
- ➜ is recorded as sent by the originating process



No messages out of thin air!

Slide 54

TCP/IP Protocol Architecture

Collection of protocols issued as Internet standard by the Internet Activity Board

TCP/IP Layers:

- ➜ Physical Layer
- ➜ Network access layer
- ➜ Internet Layer
- ➜ Host-to-host (transport-) layer
- ➜ Application layer

Slide 55

Physical Layer:

Covers physical interface between data transfer device (computer) and transmission medium (network):

- ➜ specifies characteristics of network
- ➜ data transfer rate
- ➜ nature of signals
- ➜ data rate

Slide 56

Network access layer:

Exchange of data between server/workstation and network

- ➜ sender provides network with address of receiver
- ➜ sender may invoke network services (eg priorities)
- ➜ type of network determines software used at this layer:
  - circuit switching
  - packet switching
  - LAN
- ➜ upper layers need not be concerned about network specifics

Slide 57

Internet Layer:

- ➜ Internet Protocol (IP) provides routing functions across multiple networks
- ➜ Protocol implemented in end systems (server/workstations) and routers
- ➜ Router: processor
  - which connects two networks
  - relays data from from one network to the other

Slide 58

Host-to-Host or Transport Layer:

- ➜ Reliability of data transfer:
  - all data arrives
  - data arrives in the correct order
- ➜ independent of applciation
- ➜ Transmission Contron Protocol (TCP) most commonly used

Slide 59

TCP

- ➜ applications send data stream
- ➜ TCP chops it up into packages
- ➜ packages then passed to IP layer
- ➜ TCP checks to avoid package loss
- ➜ waits for acknowledgement, otherwise resends
- ➜ uses checksum to ensure correct transmission of package

Slide 60

TCP header:

```
                     16                   31
+--------------------+---------------------+
|     Source Port    |   Destination Port  |
+------------------------------------------+
|            Sequence  Number              |
+------------------------------------------+
|           Acknowledgment Number          |
+------------+-----------------------------+
|Header Len. | Res.  |     Window          |
+------------+-------+---------------------+
| Checksum           | Urgent Pointer      |
+--------------------+---------------------+
| Options   ........... Padding            |
+------------------------------------------+
|                  Data                    |
```

Slide 61

User Datagram Protocol (UDP)

- ➜ small protocol overhead
- ➜ no guaranteed delivery
- ➜ no guaranteed preservation of sequence
- ➜ no protection against duplication
- ➜ Example application: SNMP (Simple Network Management Protocol)

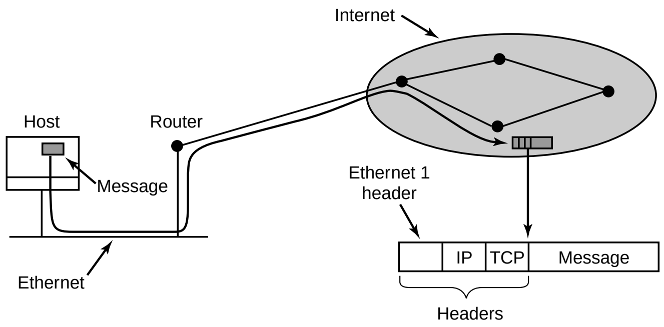
```
+---------------------------------------+
| Source Port       | Destination Port  |
+-------------------+-------------------+
| Segment Length    |    Checksum       |
+-------------------+-------------------+
```

Slide 62

IP and IPv6

- ➜ IPv4 is version 4 of the Internet Protocol (IP)
- ➜ first widely used IP version
- ➜ first published 1981
- ➜ IPv4 uses 32-bit addresses
- ➜ address space too limited

Slide 63



Slide 64

TCP/IP Operation

Process on Host A associated with Port 3 wants to send data to Process on host B, Port 2:

① Process A
- hands down message to TCP layer
- instructs it to send to Host B, Port 3

② TCP
- chops message up, if necessary
- adds control information to each package (TCP header)
- hands it down to IP layer
- instructs it to send to Host B

③ IP
- adds control information (IP header)
- hands it down to Network layer (eg, Ethernet logic)
- instructs it to send it to router

Slide 65

④ Network

- adds control information (Network header)

⑤ IP module in router directs package to Host B

⑥ Network strips off network header, passes it to IP layer

⑦ IP layer strips off IP header, passes it to TCP

⑧ TCP strips off TCP header, passes it to application