What is a Distributed System

- Motivation: sharing resources.
- Definition of distributed system:
  - a collection of independent computers that appears to its users as a single coherent system.
  - a system of networked computers that coordinate their activity only by message passing.
- Resources in a distributed system
  - managed by a server program.
  - accessed by communication via service interface provided by the server program.
- Different from “distributed computing” and “distributed application”, multi-processor system
- In a distributed system, each computer has its own memory, has its own clock, and each computer runs its own operating systems.
- Characteristics of distributed systems
  - Concurrency; No global clock; Independent failures
    - Network failure; Computer failure

Resource sharing and the web

- Client/Server model
  - Clients are active and servers are passive
- Caching technique vs. buffering
- The WWW (World Wide Web)
  - the "hypertext" structure among the documents.
  - Open system
  - Standard technological components:
    - HTML (HyperText Markup Language)
    - URL (Uniform Resource Locators)
    - HTTP (HyperText Transfer Protocol).
- URL
  - scheme:scheme-specific-identifier
- HTTP
  - A request-reply protocol
  - Specify content types in request
  - One resource per request
  - Dynamic pages, Downloaded code (mobile code)
### Architectural models
- Platform does not provide a view of a single coherent system
- Solution: Middleware
  - Masking heterogeneity
- Limitation of Middleware: “End-to-end argument”
  - Some functions require knowledge only the applications have
  - Example: Careful file transfer, delivery guarantee
- Client-server model
- Client-server group model: Partition or replicate resources
- Proxy server and caches
- Mobile code: push model
- An example in client/server model: e-mail

### Interaction Model
- Distributed algorithm vs. simple algorithm
- Difficulties in distributed algorithm
  - Complex: client/server group model
  - Hard to predict: executing rate, transmission rate
  - No general state
- Performance of communication channels
  - Latency; Bandwidth; Jitter
- Computer clocks
  - clock drift rate: difference from perfect clock
  - Correction: Time server or Logical clock
Networking and internetworking

- Packet transmission
  - Message: a logical unit of information, a sequence of data items of arbitrary length.
  - Packet: a sequence of binary data of restricted length, with addressing information.
  - Packet size

- Difference
  - Port address;
  - IP address
  - Physical address

Switching schemes

- Broadcast
  - No switching logic, i.e., Ethernet, wireless networks

- Circuit switching
  - Communicate through a number of intervening exchanges, i.e., POTS

- Packet switching [1960s]
  - Store-and-forward network
  - May be lost, vary in latency. A few ten microseconds-a few milliseconds.
  - Short Internet packet takes up to 200 milliseconds to arrive at its destination

- Frame relay
  - Video conference: < 50 milliseconds.
  - Combine the advantages of circuit switching to packet-switching.
  - Example: ATM
Internetworking Protocol

- Overview
  - Unreliable, best-effort delivery service: post office
  - Connectionless: datagram
    - transported independently → datagrams sent by the same source to the same destination could arrive out of order

- mask

Routing algorithms

- implemented by a program in the network layer at each router node
- two responsibilities
  - route of each incoming packet: hop-by-hop basis
  - update its knowledge of the network
- RIP
  - Two-node loop instability
    - Solution: defining infinity; split horizon
  - Three-node instability
    - defining infinity
- Open Shortest Path First protocol
  - link state routing method
  - Three steps
  - Can avoid problems in RIP. Why?
Transport layer in TCP/IP suite

- Difference between UDP and TCP
- Delivery of a message from one process to another process.
  - Service-point addressing: port address
  - Segmentation and reassemble
  - Connection mechanism
  - Flow control
  - Error control
  - Congestion control

Flow control in TCP

- See sample solution of hw1
Error control in TCP

- Error detection and error correction
- Checksum
- Acknowledgment
  - Next sequence number expected
  - One Ack for every two in-order data segment
  - One Ack for each out-of-order segment
  - One Ack for each duplicate segment
- time-out
- Retransmission

Congestion, Congestion Control

- Congestion: the load on the network is greater than the capacity of the network.
- Congestion control: mechanisms that detect, prevent and handle network congestion.

Congestion control vs. flow control
Congestion Control

- Two implementation points
  - Routers, switches: queuing discipline
  - End hosts

Queuing disciplines at routers

See sample solution of Hw1

Congestion control in TCP

- Additive Increase/Multiplicative Decrease (AIMD)
Congestion control in TCP

- Slow start

(a) Slow start in normal mode

- Fast Retransmit

 congestion control in TCP

Congestion control in TCP

- Fast Retransmit

 congestion control in TCP

- Fast Retransmit
RED Gateway

- RED Gateway vs. DECbit
- See sample solution for hw1

Ethernet

- broadband or baseband signalling
- Carrier-Sense Multiple Access with Collision Detection (CSMA/CD)
- All nodes are continuously ‘listening’ to the medium for packets that are addressed to them.
- Packets $\rightarrow$ frames
  - Prefix: hardware timing purposes
  - the destination address, the sending address;
  - length of data (46—1,500 bytes), data of variable length,
  - checksum
**Ethernet**

- **Packet collision**
  - Carrier sensing: not enough
  - Time = 0
    - Message almost there at time T when B starts – collision!
  - Time = T'

- **Collision detection**
  - Sender's responsibility to detect
  - Time = 2T

- **Minimum packet length in collision detection**
- Send jamming signal, delay and try again
- Delay time is selected using binary exponential back-off

**Types of communication**

- **Persistent communication** – Stores message until communicated to user
- **Transient communication** – Stored only when sending and receiving processes are alive
  - Transport level protocols provide transient communication
- **Asynchronous** – Sender continues after sending message
- **Synchronous** – Sender blocks until message is stored at receiver's local buffer, delivered to receiver or processed by receiver

Example of **Persistent Asyn. Comm.**: email system
c) Transient asynchronous communication: UDP, one-way RPCs.

(d) Receipt-based transient synchronous communication

(e) Delivery-based transient synchronous communication at message delivery: Asyn. RPCs

(f) Response-based transient synchronous communication: PRCs, RMIs.

IPC mechanisms

- Pipes
  - processes must be related through a common ancestor
  - impossible in a distributed environment

- Sockets

- message queues: Message-oriented Middleware (MOM)
Socket creation: socket()

- \( s = \text{socket}(\text{domain, type, protocol}); \)
  - \textit{domain}: AF_UNIX, AF_INET, or AF_NS
  - \textit{type}: SOCK_STREAM, SOCK_DGRAM, etc
  - \textit{protocol}: TCP or UDP. Auto selected if 0
  - Return a socket descriptor (a small integer for later reference)
  - Ex: \( s = \text{socket}(\text{AF_INET}, \text{SOCK_STREAM}, 0); \)

Connection Establishment

- Asymmetric, involving a server and a client
  - Server: create \( \rightarrow \) bind \( \rightarrow \) listen \( \rightarrow \) accept
  - Client: create \( \rightarrow \text{bind} \rightarrow \text{connect} \)
  - \textit{connect}(s, address, len)
    - \( s \): socket descriptor
    - \textit{address}: server address
    - \textit{len}: the length of the address
System Call: listen()
- listen(s, max_num)
  - s: socket descriptor
  - max_num: the maximum number of outstanding connections which may be queued awaiting acceptance by the server process
  - If the queue is full, a connection will be ignored (instead of refused). Why?
Java Sockets

Client
- socket()
- writeUTF()
- readUTF()
- close()

Server
- socket()
- accept()
- accept()
- accept()
- Start a thread
- Wait for new connection

Java API for the Internet protocols

- java.net.InetAddress
  - Static InetAddress.getByName(String host)
  - host name: “java.sun.com”
- getHostAddress(): IP address string in textual presentation.
- getHostName(): the host name for this IP address.
- 32-bit integers for port number
- Socket types
  - UDP socket
  - TCP socket
**TCP socket**

- TCP is a connection-oriented protocol, a connection is established first.
- Server listens connection request
- Client asks for a connection
- Two types of TCP sockets: ordinary sockets and server sockets
- A client process constructs an ordinary socket and then it asks for a connection with the server.
- A server socket receives a connection request, it constructs an ordinary socket with an unused port number which completes the connection.
- No limit on data size.
- Streams: one in each direction

**Message-oriented Middleware (MOM)**

- Main features
  - intermediate-term storage for messages: persistent !
  - neither sender nor receiver is required to be active
  - Message queue → eliminate the need for programs to be logically connected: asynchronous !
  - takes minutes

- Only guarantee is that a message will be inserted in receivers’ queue. But no guarantees about when, or even if the message will actually be read
Message Brokers

- Issue: message format
  - How to make sure the receiver understands sender’s message?
- One format?
  - Application are too diverse.
- Act as an application level gateway
  - E.g. change delimiters at the end of records

External data representation and marshalling

- Big Endian vs. Little Endian
- Solutions
  - transmitted in the sender’s format together with an indication of the format used
  - converted to an agreed external format
Java object serialization

- Serializable objects
  - Implement the “java.io.Serializable” interface
  - You can implement one or more of the methods readObject(), writeObject() to custom serialization.

- Externalizable objects
  - Implement the “java.io.Externalizable” interface
  - The programmer takes full responsibility for the serialization and deserialization of objects.

- Serialization will preserve the state of all fields in the object graph except for fields marked transient or static or fields contained in superclasses that are not serializable.

- Visibility modifiers (e.g., private, protected, etc.) on fields do not affect serialization.

- Any subclasses of a serializable class are serializable classes, and any data inside a serializable class are also serializable data.
RPCs

- RPC vs. LPC:
  - Direct variable access is not allowed in distributed situation.
  - Error handling: In RPC, failures of the remote server and failures of network.
  - Performance: RPCs operate much slower than LPCs.
  - Authentication: insecure networks, authentication is necessary.

- RPC uses client/server model

- response-based transient synchronous communication

- Remote procedures appear local through stub functions.

- Two stubs: client stub and server stub.

- In RPC, stubs are compiled and linked with the applications.

Steps in one RPC

- Before call a remote procedure, client initiates a connection to server.

- When client process calls a remote procedure, client stub:
  - Retrieves the required parameters from the client address space.
  - Translates the parameters into a standard network data representation (NDR) format for transmission over the network.
  - Calls functions in the RPC client run-time library to send the request and its parameters to the server.

- At the server side,
  - The server RPC run-time library functions accept the request and call the server stub procedure.
  - The server stub retrieves the parameters from the network buffer and converts them from the network transmission format to the format the server needs.
  - The server stub calls the actual procedure on the server.
  - After the remote procedure returns its data to the server stub, the server stub converts return value to the network message and call the RPC run-time library functions.
  - The server RPC run-time library functions transmit the reply message to the client computer.

- At the client side,
  - The client RPC run-time library receives the return values and returns them to the client stub.
  - The client stub converts the data into the format used by the client. The stub returns the result to the calling program.
  - The calling procedure continues.
RPC message

- written in Interface definition language (IDL), also called RPC language
- transaction identifier, xid.
  - used for client RPC layer to matching reply messages with call messages, and may be used by server process to detect retransmissions.
- Body of an RPC call message:
  - RPC version number: always equal to 2 here.
  - Remote program number: (in hexadecimal)
  - Remote program version number
  - Remote procedure number
  - two authentication fields: the credential and verifier
  - the procedure parameters
- Body of a reply message
  - Requirement: contain enough information to distinguish different error conditions
  - accepted reply message or rejected reply message

Other Uses of RPC Protocol

- Batching:
  - a client sends a large sequence of call messages to a server. The client doesn’t wait for a reply from the server, and the server does not send replies to batch calls. A sequence of batch calls is terminated by a simple remote procedure call operation. And server will send a reply message to that last call message.

- Broadcast:
  - the client sends a broadcast call to the network and waits for numerous replies. Servers that support broadcast protocols only reply when the call is successfully processed, and not reply if some error happens. Broadcast calls use the Port Mapper RPC service.
**DES Authentication Verifiers**

- **Content**: an encrypted timestamp
- **Rules**:
  - The server can decrypt this timestamp
  - If it is close to the real time, then the client must have encrypted it correctly.
  - The only way the client could encrypt it correctly is to know the "conversation key" K.
  - If the client knows K, then it must be the real client.
- K is generated by the client, and client sends it to the server in its first RPC call, using a public key scheme. (Diffie-Hellman with 192-bit keys, next week)
- agree on the current time
  - Network Time Protocol
  - a simple time request
- **1st transaction**: the client sends an encrypted timestamp and "window" to the server.
- Additional check in 1st transaction: the client sends an encrypted "window verifier", equal to the window minus 1.
- For any other transaction, the server checks for two things: (1) the timestamp is greater than the previous one. (2) compare current real time with the timestamp plus window.
- The client check the verifier returned from the server: the encrypted timestamp minus one second.

**Portmapper program protocol**

- **Broadcasting**

  ![Diagram](image)

- **PMAPPROC_CALLIT**: allows a client to call a remote procedure without knowing its port number. broadcasting.
  - Its parameters are the program number, version number, procedure number, and parameters of the remote procedure.
  - Note that:
    - This procedure only sends a reply if the procedure was successfully executed.
    - The portmapper communicates with the remote program using UDP only.
    - The procedure returns the remote program's port number, and the reply message from the remote procedure.

- **Steps for Sun RPC**
  - Define the RPC Interface in a .x file. Such as MyRPCService.x
  - Use rpcgen to compile the .x file: % rpcgen MyRPCService.x.
  - Code the server implementation: you can use implementation template and fill in the details.
  - Build the server: compile server stub, server implementation and link to RPC library to build an executable file.
  - Write a client: establish a connection to corresponding server process via clnt_create. Then, compile & link the client implementation and client stub.
  - Run server and client
Java RMI

- locate remote objects: obtain a reference to the object.
- two mechanisms
  - register its remote objects with RMI's simple naming facility: rmiregistry
  - pass and return remote object references
- java.rmi.Naming
  - bind(String name, Remote obj)
  - lookup(String name): rmi://host:port/objectname
    - default port: 1099
- One difference between RPC stubs and RMI stubs:
  - In RPC, stubs are compiled and linked with the client application. RMI stubs need not be compiled into the client; it can be downloaded at runtime.
- Some advantages of Java RMI
  - Object oriented
  - Mobile behaviour or dynamic invocation
  - Safe and secure
  - Distributed Garbage Collection
    - reference-counting algorithm
    - request-reply way with at-most-once invocation semantics
  - Write once, Run Anywhere

Steps for use RMI to develop a distributed application

- Design and implement the components of your distributed application
  - Defining and implementing the remote interfaces
  - Implementing remote objects
  - Implementing the clients
- Compile sources and generate stubs.
- Make classes network accessible.
- Start the application
  - To start the registry
    - Windows users: start rmiregistry (in java\bin directory);
    - Unix users: rmiregistry &
  - To start the server: java SumServiceServer
  - To start the client: java SumServiceClient localhost
- Example: a service that calculate sum of two integers.
Security

- Confidentiality: protection against disclosure to unauthorized individuals
- Integrity: protection against modification or corruption
- Availability: protection against interference with the means to access the resources.

- Situation
  - distributed systems are open
  - the attackers are quite knowledgeable
  - secret has limit lifetime, the design of your security systems are available to attackers
  - Only small portion of people are trustable

- Attacks
  - Interruption; Interception; Modification; Fabrication

- Passive attacks, active attacks

Cryptography

- Plaintext; Encryption algorithm; keys; Ciphertext; Decryption algorithm
- Three points:
  - two general operations: substitution, transposition
  - The number of keys used.
    - Same key: symmetric, single-key, secret-key, or conventional encryption.
    - Two keys: asymmetric, two-key, or public-key encryption
  - The way used to process the plaintext
    - block cipher; stream cipher

- Two requirements for using conventional encryption:
  - Strong encryption algorithm
  - secret key must be secure
DES Encrypt Alg.

1. perform initial permutation (IP) on one input block. IP(Input Block) → (L₀, R₀)
2. Then 16 iterations of same operation.
   - Ri₋₁ → Li
   - XOR(Li₋₁, f(Ri₋₁, kᵢ)) → Rᵢ
   - kᵢ is ‘round key’; f is called “S-box Function”. It is used to achieve a big degree of “message diffusion”.
3. Finally, swap the left-half block and right-half block and perform an inverse initial permutation on it.
   - IP⁻¹(R₁₆, L₁₆) → output block.

Decryption algorithm

- uses same three steps.
- The only different is the order of round keys: k₁₆, k₁₅, …, k₁.

check the correctness

S-box function

- Non-linear property can avoid DC attacks. DC attacks a cipher by exploring the linear difference between two plaintext messages and the linear difference between their corresponding ciphertext messages.
- a longer key: Triple DES
  - Drawbacks: slow in software, smaller block size.
The Advanced Encryption Standard

- Rijndael Cipher: block cipher with a variable block size and variable key size
- At each round, four different transformations:
  - SubBytes(): non-linear property
  - ShiftRows(): message diffusion
  - MixColumns(): message diffusion
  - AddRoundedKey(): randomness

Cipher operation modes

- electronic codebook (ECB);
- cipher block chaining (CBC) mode;
- output feedback (OFB) mode;
- cipher feedback (CFB) mode;
- counter (CTR) mode

Electronic codebook (ECB) mode

- encrypt each message segment independently, unique ciphertext for a segment
- Possible attack on some fixed pattern: stable frequency
- deterministic
CBC mode
- “initial vector” (IV). An IV is a random n-bit block. IV is not secret.
- the ciphertext messages sent to the receiver will include the IV.
- Has encryption/decryption algorithms

CFB Mode
- the encryption function of the underlying block cipher is used at the encryption side and the decryption side

OFB Mode
- decryption identical to Encryption
- Note the difference between it and CFB

CTR Mode
- Ctr\textsubscript{i}: initial random value. Ctr\textsubscript{i}=Ctr\textsubscript{i-1}+1
- the algorithms at sender and receiver sides are same

Key channel establishment
- Authentication servers
- Public-key techniques
- Trent: authentication server.
- Alice and Bob: two principals want to communicate with each other.
- Malice: attacker
- K\textsubscript{AT}: a key shared between Alice and Trent;
- K\textsubscript{BT}: is the key shared between Bob and Trent.
- The first protocol: “From Alice to Bob”
  - 1. Alice sends to Trent: Alice, Bob, \{K\}\textsubscript{KAT}
  - 2. Trent sends to Bob: Alice, Bob, \{K\}\textsubscript{KBT}
  - 3. Bob sends to Alice: \{Hi Alice, I’m Bob!\}\textsubscript{K}.
- Drawback: Bob may not trust Alice
- Fix: “session key from Trent”
  - 1. Alice sends to Trent: Alice, Bob
  - 2. Trent sends to Alice: \{K\}_\textsubscript{KAT}, \{K\}_\textsubscript{KBT};
  - 3. Alice to Bob: Trent, Alice, \{K\}_\textsubscript{KBT}
  - 4. Bob sends to Alice: \{Hi Alice, I’m Bob!\}\textsubscript{K}.
Problem: no protection on the identities

**Attack**: Malice can interrupt it and modifies Bob’s identity with his identity, and then the key generated will be known to Alice and Malice.

To fix it, Alice can encrypt Bob’s identity with her key. But not encrypt her identity, why?
- this fix is not enough, another attack is that Malice interrupts the Alice’s request message and sends a message: Alice, {Malice}$_{KAT}$ to Trent. Why Malice has {Malice}$_{KAT}$?
- Also at the last step, Malice needs send an ACK with Bob’s identity. Why Malice knows it’s Bob in the first message?
- Yet another attack is: Malice modifies the message from Trent to Alice into {K’}$_{KAT}$

**Message Authentication Protocol**: prevent modifying messages.
- main idea: a binding between the session keys and its intended users.
  1. Alice sends to Trent: Alice, Bob;
  2. Trent sends to Alice: {Bob, K}$_{KAT}$, {Alice, K}$_{KBT}$;
  3. Alice decrypts {Bob, K}$_{KAT}$, checks Bob’s identity, and sends to Bob: Trent, {Alice, K}$_{KBT}$;
  4. Bob decrypts {Alice, K}$_{KBT}$, checks Alice’s identity, and sends an encrypted Ack message to Alice.

**Message replay attack on Message Authentication Protocol**
- Malice has old ciphertext messages: {Bob,K’}$_{KAT}$, and {Alice,K’}$_{KBT}$, and knows the old key K’.
- Two mechanisms to check if the message received is an old message.
  - challenge-response, or handshake, or Needham-Schroeder Symmetric-key Authentication protocol
  - Timestamp: DES Authentication Verifiers

**challenge-response**
- 1. Alice sends to Trent: Alice, Bob, N$_A$; (N$_A$: random number)
- 2. Trent sends to Alice: {NA, Bob, K, {Alice, K}$_{KBT}$}$_{KAT}$;
- 3. Alice sends to Bob: Trent, {Alice, K}$_{KBT}$;
- 4. Bob sends to Alice: {I’m Bob! N$_B$}$_{K}$;
- 5. Alice sends to Bob: {I’m Alice! N$_B$-1}$_{K}$;

**Attack on this protocol**: Malice interrupts the messages 3,4,5, and replaces them with his own version.
- 3’. Malice to Bob: Trent, {K’, Alice}$_{KBT}$

**Fix**: challenge-response between Trent and Bob (more message flow)
Timestamp
- 1. Alice sends to Trent: Alice, Bob;
- 2. Trent sends to Alice: \{Bob, K,T, \{Alice,K,T\}_{KBT}\}_{KAT};
- 3. Alice sends to Bob: \{Alice, K,T\}_{KBT};
- 4,5. same as in “Challenge Response” protocol.

One problem is good-quality time value and reasonable window size.

Public key techniques
- mathematical functions
- smaller trust base
- 100 or 1000 times processing power for secret-key
- Applications: digital signature (RSA); key exchange (DH key exchange, RSA); encryption/decryption (RSA).

RSA: block cipher; block value: [0,n-1]
- En: \(C=P^e \pmod n\); De: \(P=C^d \pmod n\).
- Public-key: \(\{e,n\}\); private-key is \(\{d,n\}\)
- Key generation
  - 1. Select two prime numbers, for example \(p=7\), and \(q=17\).
  - 2. Calculate \(n=p*q=119\).
  - 3. Calculate \(\phi(n)=96\).
  - 4. Select \(e\) s.t. \(e\) is relatively prime to \(\phi(n)\) and \(<=\ \phi(n)\), in this case, \(e=5\).
  - 5. Determine \(d\) such that \(d*e=1 \pmod {96}\) and \(d <= \phi(n)\). The correct value for \(d\) is 77 because \(77*5=385=4*96+1\).
- Huge computation

DH Key exchange
- two public numbers: a prime number \(q\) and an integer \(a\), where \(a\) is a primitive root of \(q\).
- User A selects a random integer \(X_A < q\) and calculates its public key \(Y_A = a^{X_A} \mod q\).
- Similarly, B selects \(X_B\) and calculates its public key \(Y_B\)
- The Man-in-the-Middle Attack
- Fix: authentication service
NS Public-key authentication protocol

- KA: Alice’s public key; KA⁻¹: Alice’s private key.
  1. Alice sends to Trent: Alice, Bob;
  2. Trent sends to Alice: {KB, Bob} KT⁻¹;
  3. Alice sends to Bob: {NA, Alice} KB; (NA is a random number: Alice’s secret information).
  4. Bob sends to Trent: Bob, Alice;
  5. Trent sends to Bob: {KA, Alice} KT⁻¹;
  6. Bob sends to Alice: {NA, NB} KA; (NB is Bob’s secret information).
  7. Alice sends to Bob: {NB} KB.

Attack: 1 is for Alice-Malice; 2 is for Malice-Bob

- 1-3. Alice sends to Malice: {NA, Alice} KM
- 2-3. Malice sends to Bob: {NA, Alice} KB
- 2-6. Bob sends to Alice (Interrupted by Malice): {NA, NB} KA
- 1-6. Malice sends to Alice: {NA, NB} KA
- 1-7. Alice sends to Malice: {NB} KM
- 2-7. Malice sends to Bob: {NB} KB

Data Integrity techniques

- Symmetric techniques: keyed hash function technique
- Asymmetric techniques: digital signatures
- A hash function is a deterministic function that maps a big string of arbitrary length to a hashed value.
  - A hashed value is a bit string of a fixed length.
- Properties of a hash function:
  - Mixing-transformation
  - Collision resistance
  - Pre-image resistance
  - Practical efficiency
- Birthday attack or square-root attack on hash function
- The SHA-1 Secure Hash Function
  - Input: bit length less than 2^64. Its output is a 160-bit message digest.
  - Step 1: Append padding bits.
  - Step 2: Append length. (avoid padding attack)
  - Step 3: Initialize buffer.
  - Step 4: Process message in 512-bit blocks.