SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

## SCHOOL OF COMPUTING

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# SCSA3008

# UNIT – III DISTRIBUTED DATABASE AND INFORMATION SYSTEMS

# SCSA3008_DISTRIBUTED DATABASE AND INFORMATION SYSTEMS

## COURSE OBJECTIVES

- To understand the role of databases and database management systems in managing organizational data and information.
- To understand the techniques used for data fragmentation, replication and allocation during the distributed database design process.
- To discuss the issues involved in resource management and process.
- To Perceive the building blocks and design of information systems.
- To acquire knowledge of information systems on Business operations.

## COURSE OUTCOMES

On completion of the course, student will be able to:

CO1 - Identify the introductory distributed database concepts and its structures.
CO2 - Produce the transaction management and query processing techniques in DDBMS.
CO3 - Develop in-depth understanding of relational databases and skills to Optimize database performance in practice.
CO4 - Critiques on each type of databases.
CO5 - Analyze, Design and present the information systems.
C06 - Designing of decision support system and tools for Business operations.

**UNIT 3**

**ADVANCED IN DISTRIBUTED SYSTEMS**

Authentication in distributed systems - Protocols based on symmetric cryptosystems - Protocols based on asymmetric cryptosystems - Password-based authentication - Unstructured overlays - Chord distributed hash table – Content addressable networks (CAN) - Tapestry - Some other challenges in P2P system design - Tradeoffs between table storage and route lengths - Graph structures of complex networks - Internet graphs - Generalized random graph networks.

## Authentication in distributed systems

- A distributed system is susceptible to a variety of threats mounted by intruders as well as legitimate users of the system. Authentication is needed for both authorization and accounting functions.
- The entities in a distributed system, such as users, clients, servers and processes, are collectively referred to as principals. A principal can impersonate another principal and authentication becomes an important requirement.
- Authentication is a process by which one principal verifies the identity of another principal. In one-way authentication, only one principal verifies the identity of the other principal. In mutual authentication, both communicating principals verify each other's identity.
- Authentication is a process of verifying that the principal's identity is as claimed.
- Authentication is based on the possession of some secret information, like password, known only to the entities participating in the authentication.
- When an entity wants to authenticate another entity, the former will verify if the latter possesses the knowledge of the secret.

- In a distributed system, authentication is carried out using a protocol involving message exchanges and these protocols are termed **authentication protocols**.
- **Authentication is identification plus verification**.
- Identification is the procedure whereby an entity claims a certain identity, while verification is the procedure whereby that claim is checked.
- Authentication is a process of verifying that the principal's identity is as claimed. The correctness of authentication relies heavily on the verification procedure employed.

There are three main types of authentications of interest in a distributed system:

- ***message content authentication*** — verifying that the content of a message received is the same as when it was sent;
- ***message origin authentication*** — verifying that the sender of a received message is the same one recorded in the sender field of the message;
- ***general identity authentication*** - verifying that a principal's identity is as claimed.

***message content authentication***
is commonly handled by tagging a key-dependent *message authentication code* (MAC) onto a message before it is sent. Message integrity can be confirmed upon reception by recomputing the MAC and comparing it with the one attached.

***message origin authentication***
is a subcase of GIA. A successful general identity authentication results in a belief held by the authenticating principal (the verifier) that the authenticated principal (the claimant) possesses the claimed identity. Hence subsequent claimant actions are attributable to the claimed identity.

***general identity authentication***
is needed for both authorization and accounting functions.

**Types of Principals**

**Hosts** : These are addressable entities at the network level. A host is usually identified by its name (for example, a fully qualified domain name) or its network address (for example, an IP address).

**Users** : These entities are ultimately responsible for all system activities. Users initiate and are accountable for all system activities. Most access control and accounting functions are based on users. Typical users include humans, as well as accounts maintained in the user database. Users are considered to be outside the system boundary.

**Processes** : The system creates processes within the system boundary to represent users. A process requests and consumes resources on the behalf of its user.

## A simple classification of authentication protocols

Authentication protocols can be categorized based on the type of:

- cryptography (symmetric vs. asymmetric),
- reciprocity of authentication (mutual vs. one-way),
- key exchange,
- real-time involvement of a third party (on-line vs. off-line),
- nature of trust required from a third party,
- nature of security guarantees, and storage of secrets.

### Cryptographic Techniques

- **Symmetric ("private key")** and **Asymmetric ("public key").**
- **Symmetric cryptography** uses a single private key to both encrypt and decrypt data. Any party that has the key can use it to encrypt and decrypt data. Symmetric cryptography algorithms are typically fast and are suitable for processing large streams of data.
- **Asymmetric cryptography**, also called public-key cryptography, uses a secret key that must be kept from unauthorized users and a public key that is made public.
- Both the public key and the private key are mathematically linked: data encrypted with the public key can be decrypted only by the corresponding private key, and data signed with the private key can only be verified with the corresponding public key.
- Both keys are unique to a communication session.

### Notations used

- We and define a system model that characterizes protocol executions.
- We assume a given set of constant symbols, which denote the names of principals, nonces, and keys.
- In symmetric key cryptography, let $X_k$ denote the encryption of X using a symmetric key k and $Y_{k-1}$ denote the decryption of Y using a symmetric key k.
- In asymmetric key cryptography, for a principal x, $K_x$ and $K^{-1}_x$ denote its public and private keys, respectively.
- A communication step whereby P sends a message M to Q is represented as P → Q: specify authentication protocols with precise syntax and semantics M, whereas a computation step of P is written as P: …, where "…".

**A Login Protocol**

For example, a typical login protocol between a host H and a user U is given in Algorithm (f denotes a one-way function, that is, given y, it is computationally infeasible to find an x such that f(x) = y).

$$
\begin{aligned}
U \rightarrow H: &\quad U \\
H \rightarrow U: &\quad \text{“Please enter password”} \\
U \rightarrow H: &\quad p \\
H: &\quad \text{compute } y = f(p) \\
: &\quad \text{Retrieve user record } (U, f(\text{password}_U)) \text{ from the database} \\
: &\quad \text{If y} = f(\text{password}_U), \text{ then accept; otherwise reject}
\end{aligned}
$$

Since authentication protocols for distributed systems directly use cryptosystems, their basic design principles also follow the type of cryptosystem used.

## Protocols Based upon Symmetric Cryptosystems

In a symmetric cryptosystem, knowing the shared key lets a principal encrypt and decrypt arbitrary messages. Without such knowledge, a principal cannot create the encrypted version of a message, or decrypt an encrypted message.
Hence, authentication protocols can be designed according to the following principle called SYM:

***“If a principal can correctly encrypt a message using a key that the verifier believes is known only to a principal with the claimed identity (outside of the verifier), this act constitutes sufficient proof of identity”.***

### Basic Protocol

Using the above principle, we immediately obtain the basic protocol (shown in Algorithm) where principal P is authenticating itself to principal Q. “k” denotes a secret key that is shared between only P and Q.

$$
\begin{aligned}
P: &\quad \text{Create a message } m = \text{“I am } P\text{.”} \\
: &\quad \text{Compute } m' = \{m, Q\}_k \\
P \rightarrow Q: &\quad m, m' \\
Q: &\quad \text{verify } \{m, Q\}_k = m' \\
: &\quad \text{if equal then accept; otherwise the authentication fails}
\end{aligned}
$$

In this protocol, the principal P prepares a message m, and encrypts the message and identity of Q using the symmetric key k and sends to Q both the plain text and encrypted messages. Principal Q, on receiving the message, encrypts the plain text message and its identity to get the encrypted message. If it is equal to the encrypted message sent by P, then Q has authenticated P, else the authentication fails.

**Weaknesses:**

- One major weakness of the protocol is its vulnerability to replays. An adversary could masquerade as P by recording the message m, m and later replaying it to Q.
- Since both plain text message m and its encrypted version m are sent together by P to Q, this method is vulnerable to known plain text attacks.
- Thus, the cryptosystem must be able to withstand known plain text attacks.

**Modified protocol with nonce**

To prevent replay attacks, we modify the protocol by adding a challenge-and response step using a **nonce**. A nonce is a large random or pseudo-random number that is drawn from a large space so that it is difficult to guess by an intruder. This property of a nonce helps ensure that old communications cannot be reused in replay attacks.

$$
\begin{aligned}
P \rightarrow Q: &\quad \text{“I am } P\text{.”} \\
Q: &\quad \text{generate nonce } n \\
Q \rightarrow P: &\quad n \\
P: &\quad \text{compute } m' = \{P, Q, n\}_k \\
P \rightarrow Q: &\quad m' \\
Q: &\quad \text{verify } \{P, Q, n\}_k = m' \\
: &\quad \text{if equal then accept; otherwise the authentication fails}
\end{aligned}
$$

Replay is foiled by the freshness of nonce n and because n is drawn from a large space.

**Weaknesses:**

- This protocol has scalability problems because each principal must store the secret key for every other principal it would ever want to authenticate.
- This presents major initialization (the predistribution of secret keys) and storage problems.
- Moreover, the compromise of one principal can potentially compromise the entire system.
- This protocol is also vulnerable to known plain text attacks.

## Wide-Mouth Frog Protocol

Principal A authenticates itself to principal B using a server S as follows:

$$A \rightarrow S : A, \{T_A, K_{AB}, B\}_{KAS}$$
$$S \rightarrow B : \{T_S, K_{AB}, A\}_{KBS}$$

A sends to S its identity and a packet encrypted with the key, $K_{AS}$, it shares with S. The packet contains the current timestamp, A's desired communication partner, and a randomly generated key $K_{AB}$, for communication between A and B.

S decrypts the packet to obtain $K_{AB}$ and then forwards this key to B in an encrypted packet that also contains the current timestamp and A's identity.
B decrypts this message with the key it shares with S and retrieves the identity of the other party and the key, $K_{AB}$.

**Weaknesses**:

- A global clock is required and the protocol will fail if the server S is compromised.

## Kerberos Authentication Service

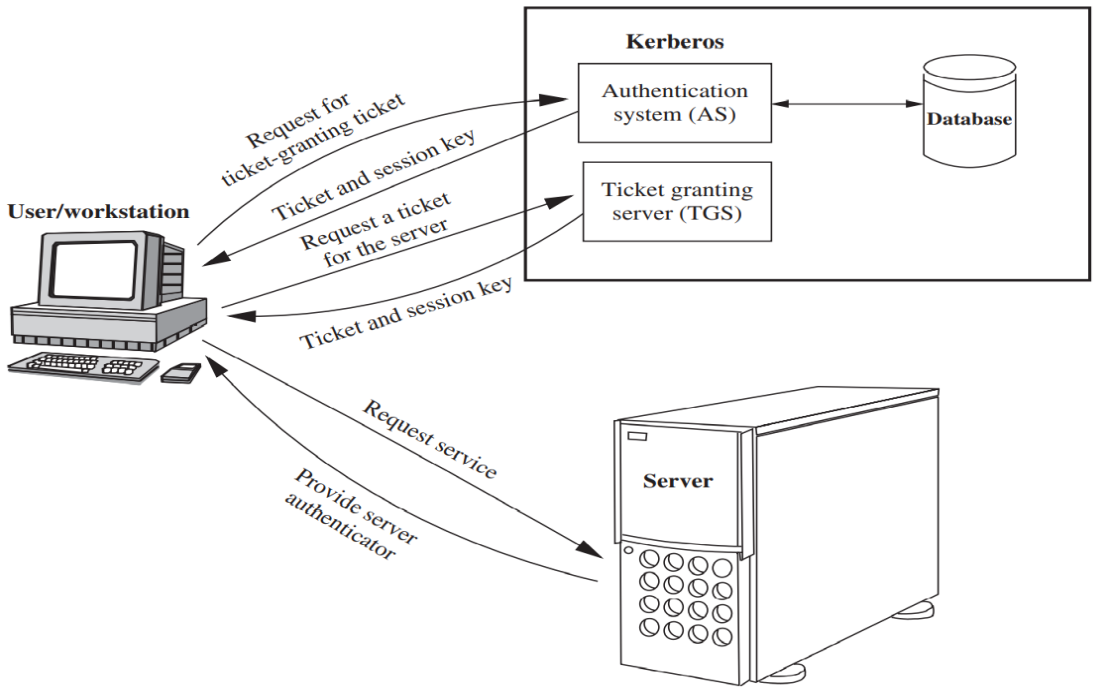
Kerberos primarily addresses client–server authentication using a symmetric cryptosystem, together with trusted third-party authentication servers. (Project Athena)
The basic components include ***authentication servers* (*Kerberos servers*) and *ticket-granting servers* (TGSs).**

## Initial Registration

- Every Client/user register with the Kerberos server by providing its user id, U and a password, $password_u$.
- The Kerberos server computes a key $k_u = f(password_u)$ using a one-way function f and stores this key in a database.
- $k_u$ is a secret key that depends on the password of the user and is shared by client U and Kerberos server only.

## The Authentication Protocol

1. **Initial authentication at login:** The Kerberos server authenticates user login at a host and installs a ticket for the ticket-granting server, TGS, at the login host.
2. **Obtain a ticket for the server:** Using the ticket for the ticket-granting server, the client requests the ticket-granting server, TGS, for a ticket for the server.
3. **Requesting service from the server:** The client uses the server ticket obtained from the TGS to request services from the server.




***Steps in Authentication in Kerberos***

## Initial Authentication at Login

Initial authentication at login uses a Kerberos server and is shown below. Let U be a user

who is attempting to log into a host H.

| | | | |
|---|---|---|---|
| (1) | $U \rightarrow H$ | : | $U$ |
| (2) | $H \rightarrow$ Kerberos | : | $U, TGS$ |
| (3) | Kerberos | : | retrieve $k_U$ and $k_{TGS}$ from database |
| | | : | generate new session key $k$ |
| | | : | create a ticket-granting ticket |
| | | : | $tick_{TGS} = \{U, TGS, k, T, L\}_{K_{TGS}}$ |
| (4) | Kerberos $\rightarrow H$ | : | $\{TGS, k, T, L, tick_{TGS}\}_{k_U}$ |
| (5) | $H \rightarrow U$ | : | "Password?" |
| (6) | $U \rightarrow H$ | : | *password* |
| (7) | $H$ | : | compute $k'_U = f(password)$ |
| | | : | recover $k$, $tick_{TGS}$ by decrypting $\{TGS, k, T, L, tick_{TGS}\}_{k_U}$ with $k'_U$ |
| | | : | if decryption fails, abort login, otherwise, retain $tick_{TGS}$ and $k$ |
| | | : | erase *password* from the memory |

**Obtain a ticket for the server**

The client executes the steps shown below to request a ticket for the server from the TGS. Basically, the client sends the ticket tickTGS to the TGS, requesting a ticket for the server S. (T1 and T2 are timestamps.)

| | | | |
|---|---|---|---|
| (1) | $C \rightarrow TGS$ | : | $S, tick_{TGS}, \{C, T_1\}_k$ |
| (2) | $TGS$ | : | recover $k$ from $tick_{TGS}$ by decrypting with $k_{TGS}$, |
| | | : | recover $T_1$ from $\{C, T_1\}_k$ by decrypting with $k$ |
| | | : | check timelines of $T_1$ with respect to local clock |
| | | : | generate new session key $k$ |
| | | : | Create server ticket $tick_S = \{C, S, k, T, L\}_{k_S}$ |
| (3) | $TGS \rightarrow C$ | : | $\{S, k, T, L, tick_S\}_k$ |
| (4) | $C$ | : | recover $k$, $tick_S$ by decrypting the message with $k$ |

**Requesting service from the server**

Client C sends the ticket and the authenticator to server. The server decrypts $tick_S$ and recovers k. It then uses k to decrypt the authenticator $\{C,T2\}_k$ and checks if the timestamp is current and the client identifier matches with that in the $tick_S$ before granting service to the client. If mutual authentication is required, the server returns an authenticator.

$$
\begin{array}{lrl}
(1) & C \rightarrow S: & tick_S, \{C, T_2\}_k \\
(2) & S: & \text{recover } k \text{ from } tick_S \text{ by decrypting it with } k_S \\
& : & \text{recover } T_2 \text{ from } \{C, T_2\}_k \text{ by decrypting with } k \\
& : & \text{check timeliness of } T_2 \text{ with respect to the local clock} \\
(3) & S \rightarrow C: & \{T_2 + 1\}_k
\end{array}
$$

**Weaknesses**:

- Kerberos makes no provisions for host security; it assumes that it is running on trusted hosts with an untrusted network. If host security is compromised, then Kerberos is compromised as well.
- Kerberos uses a principal's password (encryption key) as the fundamental proof of identity. If a user's Kerberos password is stolen by an attacker, then the attacker can impersonate that user with impunity.
- In Kerberos version 4, authenticators are valid for a particular time. If an attacker sniffs the network for authenticators, they have a small-time window in which they can re-use it and gain access to the same service.
- Kerberos version 5 introduced pre-authentication to solve this problem

## Protocols based on Asymmetric Cryptosystems

In an asymmetric cryptosystem, each principal P publishes its public key $k_p$ and keeps secret its private key $k_p^{-1}$. Thus only P can generate $\{m\}_{kp-1}$ for any message m by signing it using $k_p^{-1}$. The signed message $\{m\}_{kp-1}$ can be verified by any principal with the knowledge of $k_p$ (assuming a commutative asymmetric cryptosystem).

Asymmetric authentication protocols can be constructed using a design principle called ASYM, which is as follows:

***If a principal can correctly sign a message using the private key of the claimed identity,***

***this act constitutes a sufficient proof of identity.***

This ASYM principle follows the proof-by-knowledge principle for authentication, in that a principal's knowledge is indirectly demonstrated through its signing capability.

## Basic Protocol

Q sends a random number n to P and challenges it to encrypt with its private key. P encrypts (P Q n) with its private key $k_p^{-1}$ and sends it to Q. Q verifies the received message by decrypting it with P's public key $k_p$ and checking with the identity of P, Q, and n.

$$
\begin{array}{rcl}
P \rightarrow Q: & & \text{“I am } P\text{.”} \\
Q: & & \text{generate nonce } n \\
Q \rightarrow P: & & n \\
P: & & \text{compute } m = \{P, Q, n\}_{k_p^{-1}} \\
P \rightarrow Q: & & m \\
Q: & & \text{verify } (P, Q, n) = \{m\}_{k_p} \\
: & & \text{if equal, then accept; otherwise, the authentication fails}
\end{array}
$$

## A modified protocol with a Certification Authority

The basic protocol requires that Q has the knowledge of P's public key. A problem arises if Q does not know P's public key. This problem is alleviated by postulating a centralized certification authority (CA) that maintains a database of all published public keys. If a user A does not have the public key of another user B, A can request B's public key from the CA.

$P \rightarrow Q$ : "I am $P$."
$Q$ : generate nonce $n$
$Q \rightarrow P$ : $n$
$P$ : compute $m = \{P, Q, n\}_{k_p^{-1}}$
$P \rightarrow Q$ : $m$
$Q \rightarrow CA$ : "I need $P$'s public key."
$CA$ : retrieve public key $k_P$ of $P$ from key database
Create certificate $c = \{P, k_P\}_{k_{CA}^{-1}}$
$CA \rightarrow Q$ : $P, c$
$Q$ : recover $P, k_P$ from $c$ by decrypting with $k_{CA}$
verify $(P, Q, n) = \{m\}_{k_P}$
: if equal, then accept; otherwise, the authentication fails

## SSL Protocol

The secure sockets layer (SSL) protocol was developed by Netscape and is the standard Internet protocol for secure communications. The secure hypertext transfer protocol (HTTPS) is a communications protocol designed to transfer encrypted information between computers over the World Wide Web. HTTPS is http using a secure socket layer (SSL). SSL resides between TCP/IP and upper-layer applications, requiring no changes to the application layer. SSL is used typically between server and client to secure the connection. One advantage of SSL is that it is application protocol independent. A higher-level protocol can layer on top of the SSL protocol transparently. SSL protocol allows client–server applications to communicate in a way so that eavesdropping, tampering, and message forgery are prevented.

The SSL protocol, in general, provides the following features:

- **End point authentication:** The server is the "real" party that a client wants to talk to, not someone faking the identity.
- **Message integrity:** If the data exchanged with the server has been modified along the way, it can be easily detected.
- **Confidentiality:** Data is encrypted. A hacker cannot read your information by simply looking at the packets on the network.

- The record protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies MAC, encrypts, adds a header, and transmits the resulting unit into a TCP segment.

- Received data are decrypted, verified, decompressed, and reassembled, and then delivered to high-level users.

The SSL Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

The following steps, shown in below figure, are involved in the SSL handshake:

1. The SSL client sends a "client hello" message that lists cryptographic information such as the SSL version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations.

2. The SSL server responds with a "server hello" message that contains the CipherSuite chosen by the server from the list provided by the SSL client, the session ID and another random byte string. The SSL server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a "client certificate request" that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs).

3. The SSL client verifies the digital signature on the SSL server's digital certificate and checks that the CipherSuite chosen by the server is acceptable.
4. The SSL client, using all data generated in the handshake so far, creates a premaster secret for the session that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The premaster secret itself is encrypted with the server's public key.

5. If the SSL server sent a "client certificate request", the SSL client sends another signed piece of data which is unique to this handshake and known only to the client and server, along with the encrypted premaster secret and the client's digital certificate, or a "no digital certificate alert". This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.

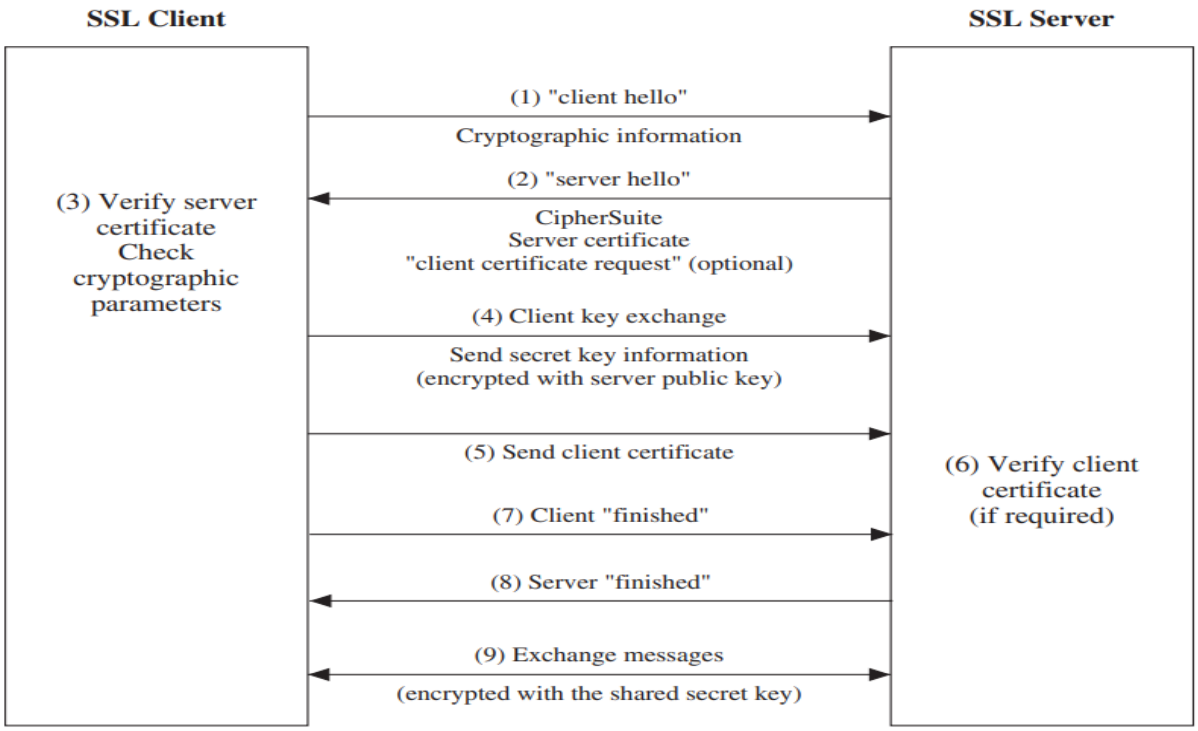
6. The SSL server verifies the signature on the client certificate.

7. The SSL client sends the SSL server a "finished" message, which is encrypted with

the secret key, indicating that the client part of the handshake is complete.

8. The SSL server sends the SSL client a "finished" message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.

9. For the duration of the SSL session, the SSL server and SSL client can now exchange messages that are encrypted with the shared symmetric secret key.




***SSL handshake Protocol***

## How SSL provides Authentication

- For client authentication, the server uses the public key in the client certificate to decrypt the data the client sends during step 5 of the handshake.
- The exchange of finished messages confirms that authentication is complete.
- If any of the authentication steps fails, the handshake fails and the session terminates.
- The exchange of digital certificates during the SSL handshake is a part of the authentication process. (CA $x$ issues the certificate to the SSL client, and CA $y$ issues the certificate to the SSL server.)

## Password-based Authentication

The use of passwords is very popular to achieve authentication because of low cost and convenience. However, people tend to pick a password that is convenient, i.e., short and easy to remember. (Vulnerable to a password-guessing attack)

### *Off-line dictionary attack*:

An adversary builds a database of possible passwords, called a dictionary. The adversary picks a password from the dictionary and checks if it works. This may amount to generating a response to a challenge or decrypting a message using the password or a function of the password. After every failed attempt, the adversary picks a different password from the dictionary and repeats the process.

### *Preventing Off-line Dictionary Attacks:*

A password-based authentication protocol aims at preventing off-line dictionary attacks by producing a cryptographically strong shared secret key, called the session key. This session key can be used by both entities to encrypt subsequent messages for a secret session.

Two password-based authentication protocols:

- Encrypted key exchange (EKE) protocol.
- Secure remote password (SRP) protocol.

### Encrypted key exchange (EKE) protocol

Bellovin and Merritt developed a password-based encrypted key exchange (EKE) protocol using a combination of symmetric and asymmetric cryptography.

(1) $A$: $(E_A, D_A)$, $K_{pwd} = f(pwd)$. {* $f$ is a function. *}
(2) $A \rightarrow B: A, \{K_{pwd}\}_{E_A}$.
(3) $B$: Compute $E_A = \{\{E_A\}_{K_{pwd}}\}_{K_{pwd}^{-1}}$ and generate a random secret key $K_{AB}$.
(4) $B \rightarrow A$: $\{\{K_{AB}\}_{E_A}\}_{K_{pwd}}$.
(5) $A$: $K_{AB} = \{\{\{\{K_{AB}\}_{E_A}\}_{\{K_{pwd}\}}\}_{K_{pwd}^{-1}}\}_{D_A}$. Generate a unique challenge $C_A$.
(6) $A \rightarrow B: \{C_A\}_{K_{AB}}$.
(7) $B$: Compute $C_A = \{\{C_A\}_{K_{AB}}\}_{K_{AB}^{-1}}$ and generate a unique challenge $C_B$.
(8) $B \rightarrow A$: $\{C_A, C_B\}_{K_{AB}}$.
(9) $A$: Decrypt message sent by $B$ to obtain $C_A$ and $C_B$. Compare the former with own challenge. If they match, go to the next step, else abort.
(10) $A \rightarrow B$: $\{C_B\}_{K_{AB}}$.

**Steps in Encrypted key exchange (EKE) protocol**

- Step 1: *A* generates a public/private key pair ($E_A$, $D_A$) and derives a secret key $K_{pwd}$ from his password *pwd*.
- Step 2: *A* encrypts his public key $E_A$ with $K_{pwd}$ and sends it to *B*.
- Steps 3 and 4: *B* decrypts the message and uses $E_A$ together with $K_{pwd}$ to encrypt a session key $K_{AB}$ and sends it to *A*.
- Steps 5 and 6: *A* uses this session key to encrypt a unique challenge $C_A$ and sends the encrypted challenge to *B*.
- Step 7: *B* decrypts the message to obtain the challenge and generates a unique challenge $C_B$.
- Step 8: *B* encrypts $\{C_A, C_B\}$ with the session key $K_{AB}$ and sends it to *A*.
- Step 9: *A* decrypts this message to obtain $C_A$ and $C_B$ and compares the former with the challenge it had sent to *B*. If they match, *B* is authenticated.
- Step 10: *A* encrypts *B*'s challenge $C_B$ with the session key $K_{AB}$ and sends it to *B*. When *B* receives this message, it decrypts the message to obtain $C_B$ and uses it to authenticate *A*.
- The resulting session key is stronger than the shared password and can be used to encrypt sensitive data.
- A Drawback: The EKE protocol suffers from the plain-text equivalence (the user and the host have access to the same secret password or hash of the password).

## Secure remote password (SRP) protocol

Wu combined the technique of zero-knowledge proof with asymmetric key exchange protocols to develop a verifier-based protocol, called secure remote password (SRP) protocol.SRP protocol eliminates plain-text equivalence.

All computations in SRP are carried out on the finite field $F_n$, where $n$ is a large prime. Let $g$ *be* a generator of $F_n$. Let $A$ be a user and $B$ be a server. Before initiating the SRP protocol, $A$ and $B$ do the following:

- $A$ and $B$ agree on the underlying field.
- $A$ picks a password $pwd$ , a random salt $s$ and computes the verifier $v = g^x$ , where $x = H(s, pwd)$ is the long-term private-key and $H$ is a cryptographic hash function.
- $B$ stores the verifier $v$ and the salt $s$.

$$
\begin{aligned}
&(1)\quad A \to B: \quad A.\\
&(2)\quad B \to A: \quad s.\\
&(3)\quad A: \quad x := H(s, pwd);\ K_A := g^a.\\
&(4)\quad A \to B: \quad K_A.\\
&(5)\quad B: \quad K_B := v + g^b.\\
&(6)\quad B \to A: \quad K_B, r.\\
&(7)\quad A: \quad S := (K_B - g^x)^{a+rx} \text{ and } B: \quad S := (K_A v^r)^b.\\
&(8)\quad A, B: \quad K_{AB} := H(S).\\
&(9)\quad A \to B: \quad C_A := H(K_A, K_B, K_{AB}).\\
&(10)\quad B \text{ verifies } C_A \text{ and computes } C_B := H(K_A, C_A, K_{AB}).\\
&(11)\quad B \to A: \quad C_B.\\
&(12)\quad A \text{ verifies } C_B. \text{ Accept if verification passes; abort otherwise.}
\end{aligned}
$$

**Steps in Secure remote password (SRP) protocol**

- Step 1: $A$ sends its username "A" to server $B$.
- Step 2: $B$ looks-up $A$'s verifier $v$ and salt $s$ and sends $A$ his salt.
- Steps 3 and 4: $A$ computes its long-term private-key $x = H(s, pwd)$, generates an ephemeral public-key $K_A = g^a$ where $a$ is randomly chosen from the interval $1 < a < n$ and sends $K_A$ to $B$.
- Steps 5 and 6: $B$ computes ephemeral public-key $K_B = v + g^b$ where $b$ is randomly chosen from the interval $1 < a < n$ and sends $K_B$ and a random number $r$ to $A$.
- Step 7: $A$ computes $S = (K_B - g^x)^{a+rx} = g^{ab+brx}$ and $B$ computes $S = (K_A v^r)^b = g^{ab+brx}$.
- Step 8: Both $A$ and $B$ use a cryptographically strong hash function to compute a session key $K_{AB} = H(S)$.

- Step 9: $A$ computes $C_A = H(K_A, K_B, K_{AB})$ and sends it to $B$ as an evidence that it has the session key. $C_A$ also serves as a challenge.
- Step 10: $B$ computes $C_A$ itself and matches it with $A$'s message. $B$ also computes $C_B = H(K_A, C_A, K_{AB})$.
- Step 11: $B$ sends $C_B$ to $A$ as an evidence that it has the same session key as $A$.
- Step 12: $A$ verifies $C_B$, accepts if the verification passes and aborts otherwise.
- None of the protocol are messages encrypted in the SRP protocol. Since neither the user nor the server has access to the same secret password or hash of the password, SRP eliminates plain-text equivalence.

# Unstructured overlays

## Structured and Unstructured Overlays:

A core mechanism in P2P networks is searching for data, and this mechanism depends on how (i) the data, and (ii) the network, are organized. Search algorithms for **P2P networks tend to be data-centric**, as opposed to the host-centric algorithms for traditional networks. P2P search uses the P2P overlay, which is a logical graph among the peers, that is used for the object search and object storage and management algorithms. The P2P overlay is the application layer overlay, where communication between peers is point-to-point (representing a logical all-to-all connectivity,) once a connection is established. The P2P overlay can be **structured or unstructured**, i.e., no particular graph structure is used. Object storage and search strategies are intricately linked to the overlay
Structure as well as to the data organization mechanisms.

**Structured P2P overlays:**

- E.g., hypercubes, meshes, butterfly networks, de Bruijn graphs.
- rigid organizational principles for object storage and object search.

**Unstructured P2P overlays:**

- Loose guidelines for object search and storage.
- No particular graph structure is used.
- Search mechanisms are ad-hoc, variants of flooding and random walk.

## Properties:

Unstructured overlays have the serious disadvantage that queries may take a long time to find a file or may be unsuccessful even if the queried object exists. The message overhead of a query search may also be high.

The following are the main advantages of unstructured overlays such as the one used by Gnutella:

- Exact keyword queries, range queries, attribute-based queries, and other complex queries can be supported because the search query can capture the semantics of the data being sought; and the indexing of the files and data is not bound to any non-semantic structure.

- Unstructured overlays can accommodate high churn, i.e., the rapid joining and departure of many nodes without affecting performance.

The following are advantages of unstructured overlays if certain conditions are satisfied:

- Unstructured overlays are efficient when there is some degree of data replication in the network.
- Users are satisfied with a best-effort search.
- The network is not so large as to lead to scalability problems during the search process.

## Gnutella

Gnutella uses a fully decentralized architecture. In Gnutella logical overlays, nodes index only their local content. The actual overlay topology can be arbitrary as nodes join and leave randomly. A node joins the Gnutella network by forming a connection to some nodes found in standard Gnutella directory-like databases. (Note that the function of joining the network cannot be said to be fully decentralized.) Users communicate with each other, performing the role of both server and client, termed as **servent**. The following are the main message types used by Gnutella:

- Ping messages are used to discover hosts, and allow a new host to announce itself.
- Pong messages are the responses to Pings. The Pong messages indicate the port and (IP) address of the responder, and some information about the amount of data (the number and size of files) that node can make available.
- Query messages. The search strategy used is flooding. Query messages contain a search string and the minimum download speed required of the potential responder, and are flooded in the network.
- QueryHit messages are sent as responses if a node receiving a Query detects a local match in response to a query. A QueryHit contains the port and address (IP), speed, the number of files found, and related information.The path traced by a Query is recorded in the message, so the QueryHit follows the same path in reverse.

# CHORD DISTRIBUTED HASH TABLE

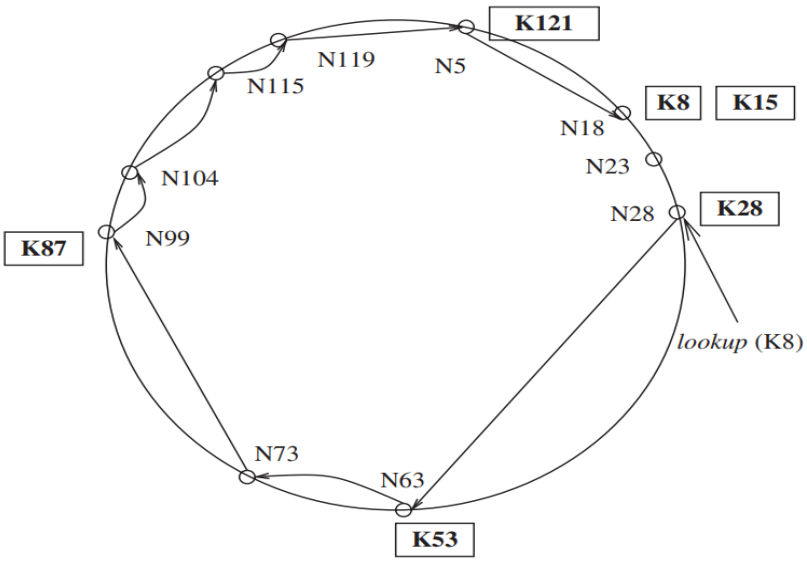
The Chord protocol, proposed by Stoica et al., uses a flat key space to associate the mapping between network nodes and data objects/files/values. The node address as well as the data object/file/value is mapped to a logical identifier in the common key space using a consistent hash function. These mappings are illustrated in the below example Figure. Both these mappings should ensure that the keys are distributed roughly equally among the nodes. This also insures that with high probability, the overhead of key management when nodes join or leave the P2P network is low. Specifically, when a node joins or leaves the network having n nodes, only O(1/n) keys need to be moved from one location to another.

The Chord key space is flat, thus giving applications flexibility in mapping their files/data to keys. Chord supports a single operation, lookup(x), which maps a given key x to a network node. Specifically, Chord stores a file/object/value at the node to which the file/object/value's key maps. Two steps are involved:

- Map the object/file/value to its key in the common address space.
- Map the key to the node in its native address space using lookup. The design of lookup is the main challenge.

In Chord, a node's IP address is hashed to an m-bit identifier that serves as the node identifier in the common key (identifier) space. Similarly, the file/data key is hashed to an m-bit identifier that serves as the key identifier, m is sufficiently large so that the probability of collisions during the hash is negligible. The Chord overlay uses a logical ring of size $2^m$. The identifier space is ordered on the logical ring modulo $2^m$. Henceforth in this section, we will assume modulo $2^m$ arithmetic. A key k gets assigned to the first node such that its node identifier equals or follows the key identifier of k in the common identifier space. The node is the successor of k, denoted succ(k).

An example chord ring with m = 7, showing mappings to the Chord address space, and a query lookup using a simple scheme :




lookup(K8) initiated at node 28,

Nodes N5, N18, N23, N28, N63, N73, N99, N104, N115, and N119 are shown. Six keys, K8, K15, K28, K53, K87, and K121, are stored among these nodes as follows:

Succ(8)=18, Succ(15)=18, Succ(28)=28, Succ(53)=63, Succ(87)=99 and Succ(121)=5.

## Simple lookup

A simple key lookup algorithm that requires each node to store only 1 entry in its routing table works as follows. Each node tracks its successor on the ring, in the variable *successor*; a query for key x is forwarded to the successors of nodes until it reaches the first node such that that node's identifier y is greater than the key x, modulo $2^m$. The result, which includes the IP address of the node with key y, is returned to the querying node along the reverse of the path that was followed by the query. This mechanism requires O(1) local space but O(n) hops, where n is the number of nodes in the P2P network.

(variables)
**integer**: *successor* ⟵ initial value;

(1) *i.Locate_Successor*(*key*), where $key \neq i$:
(1a) **if** $key \in (i, successor]$ **then**
(1b) **return**(*successor*)
(1c) **else return** (*successor.Locate_Successor*(*key*)).

***Algorithm : A simple object location algorithm in Chord at node i***

The pseudo-code for this simple lookup is given in Algorithm. The following convention is assumed. Notation (x,y) represents the left-open right-closed segment of the Chord logical ring modulo m. Notation x.Proc(·) is a RPC to execute Proc on node x while x.var is a RPC to read the variable var at process x.

## Scalable lookup

A scalable lookup algorithm that uses O (log n) message hops at the cost of O(m) space in the local routing tables, uses the following idea. Each node i maintains a routing table, called the finger table, with at most O(log n) distinct entries, such that the $x^{th}$ entry ($1 \leq x \leq m$) is the node identifier of the node succ($i+2^{x-1}$). This is denoted by *i.finger[x]* = succ($i+2^{x-1}$). This is the first node whose key is greater than the key of node i by at least $2^{x-1}$ mod $2^m$. Note that each finger table entry would have to contain the IP address and port number in addition to the node identifier, in order that i can communicate with *i.finger[x]*; henceforth we will assume this implicitly without showing these entries.

The size of the finger table is bounded by m entries. Due to the logarithmic structure, the finger table has more information about nodes closer ahead of it in the Chord overlay, than about nodes further away. Given any key whose node is to be located, the highly scalable logarithmic search shown in Algorithm below.

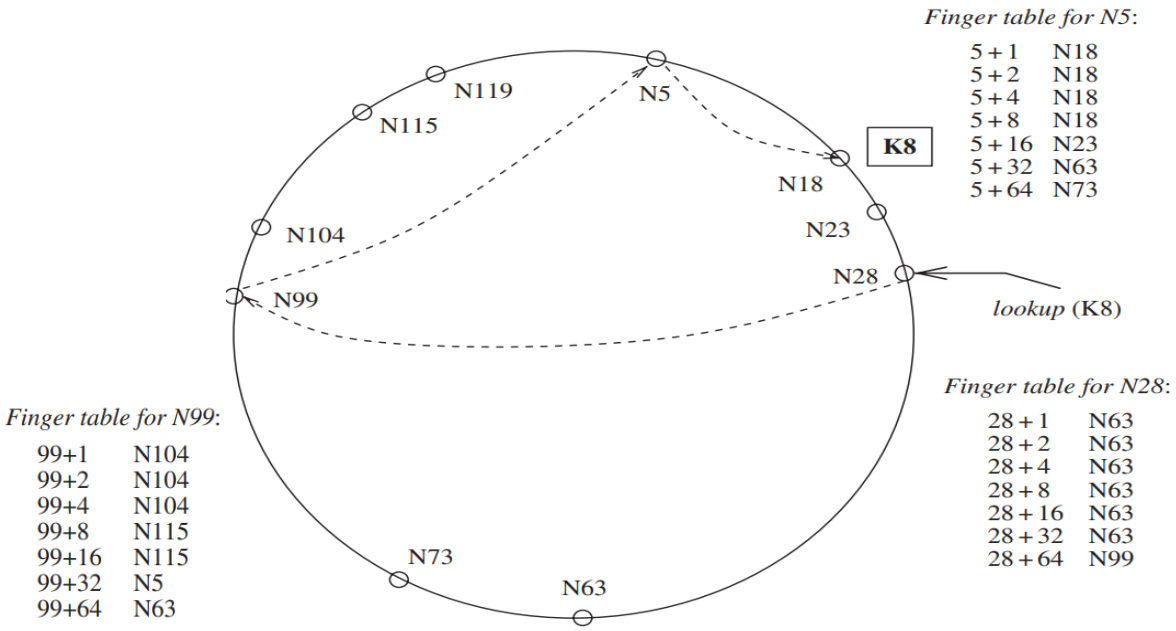
```
(variables)
integer: successor ⟵ initial value;
integer: predecessor ⟵ initial value;
integer finger[1...m];

(1) i.Locate_Successor(key), where key ≠ i:
(1a) if key ∈ (i, successor] then
(1b)        return(successor)
(1c) else
(1d)        j ⟵ Closest_Preceding_Node(key);
(1e) return (j.Locate_Successor(key)).

(2) i.Closest_Preceding_Node(key), where key ≠ i:
(2a) for count = m down to 1 do
(2b)        if finger[count] ∈ (i, key] then
(2c)            break();
(2d) return(finger[count]).
```

For a query on key key at node i, if key lies between i and its successor, the key would reside at the successor and the successor's address is returned. If key lies beyond the successor, then node i searches through the m entries in its finger table to identify the node

j such that j most immediately precedes key, among all the entries in the finger table. As j is the closest known node that precedes key, j is most likely to have the most information on locating key, i.e., locating the immediate successor node to which key has been mapped.




*The use of the finger tables in answering the query lookup(K8) at node N28 is illustrated in Figure above. The finger tables of N28, N99, and N5 are provided.*

## *Content Addressible networks (CAN)*

A content-addressible network (CAN) is essentially an indexing mechanism that maps objects to their locations in the network. The CAN project originated from the observation that the bottleneck to designing a scalable P2P network is this indexing mechanism. An efficient and scalable CAN is useful not only for object location in P2P networks, but also for large-scale storage management systems and wide-area name resolution services that decouple name resolution and the naming scheme. All these applications inherently require efficient and scalable addition of and location of objects using arbitrary location-independent names or keys for the objects.

A CAN supports three basic operations: insertion, search, and deletion of (key, value) tuples. (A "value" is an object in the context of a CAN.) A good CAN design is distributed, fault-tolerant, scalable, independent of the naming structure, implementable at the application layer, and autonomic, i.e., self-organizing and self-healing. Although CAN is a generic phrase, it also specifically denotes the particular design of a CAN proposed by Ratnasamy et al.

CAN is a logical d-dimensional Cartesian coordinate space organized as a d-torus logical topology, i.e., a virtual overlay d-dimensional mesh with wrap-around. A two-dimensional torus was shown in Figure 1.5(a) in Chapter 1. The entire space is partitioned dynamically among all the nodes present, so that each node i is assigned a disjoint region *r(i)* of the space. As nodes arrive, depart, or fail, the set of participating nodes, as well as the assignment of regions to nodes, change.

CAN is a logical d-dimensional Cartesian coordinate space organized as a d-torus logical topology, i.e., a virtual overlay d-dimensional mesh with wrap-around. A two-dimensional torus was shown in Figure 1.5(a) in Chapter 1. The entire space is partitioned dynamically among all the nodes present, so that each node i is assigned a disjoint region r(i) of the space. As nodes arrive, depart, or fail, the set of participating nodes, as well as the assignment of regions to nodes, change.

For any object v, its key k(v) is mapped using a deterministic hash function to a point P in the Cartesian coordinate space. The (k,v) pair is stored at the node that is presently assigned the region that contains the point P. In other words, the (k,v) pair is stored at node i if presently the point P corresponding to (k,v) lies in region r(i). Analogously, to retrieve object v, the same hash function is used to map its key k to the same point P. The node that is presently assigned the region that contains P is accessed (using a CAN routing algorithm) to retrieve v. The three core components of a CAN design are the following:

1. Setting up the CAN virtual coordinate space, and partitioning it among the nodes as they join the CAN.
2. Routing in the virtual coordinate space to locate the node that is assigned the region containing P.
3. Maintaining the CAN due to node departures and failures.

## CAN Initialization

1. Each CAN is assumed to have a unique DNS name that maps to the IP address of one or a few bootstrap nodes of that CAN. A bootstrap node is responsible for tracking a partial list of the nodes that it believes are currently participating in the CAN. These are reasonable assumptions, and perhaps the most “non-distributed” portions of the CAN design.
2. To join a CAN, the joiner node queries a bootstrap node via a DNS lookup, and the

bootstrap node replies with the IP addresses of some randomly chosen nodes that it believes are participating in the CAN. 3.

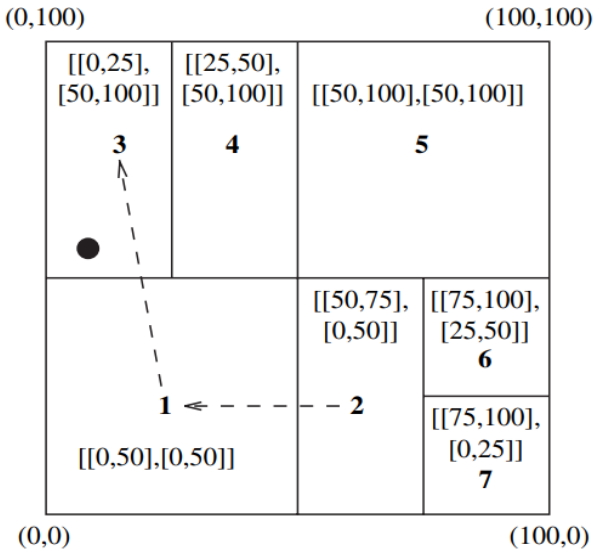
3. The joiner chooses a random point P in the coordinate space. The joiner sends a request to one of the nodes in the CAN, of which it learnt in step 2, asking to be assigned a region containing P. The recipient of the request routes the request to the owner old_owner(P)of the region containing P, using the CAN routing algorithm.
4. The old_owner(P) node splits its region in half and assigns one half to the joiner. The region splitting is done using an a priori ordering of all the dimensions, so as to decide which dimension to split along. This also helps to methodically merge regions, if necessary. The (k,v) tuples for which the key k now maps to the zone to be transferred to the joiner, are also transferred to the joiner.
5. The joiner learns the IP addresses of its neighbors from old_owner (P)The neighbors are old_owner (P) and a subset of the neighbors of old_owner (P). old_owner (P)also updates its set of neighbors. The new joiner as well as old_owner (P) inform their neighbors of the changes to the space allocation, so that they have correct information about their neighborhood and can route correctly. In fact, each node has to send an immediate update of its assigned region, followed by periodic HEART-BEAT refresh messages, to all its neighbors.

When a node joins a CAN, only the neighboring nodes in the coordinate space are required to participate in the joining process. The overhead is thus of the order of the number of neighbors, which is O(d) and independent of n, the number of nodes in the CAN.

## CAN routing

CAN routing uses the straight-line path from the source to the destination in the logical Euclidean space. This routing is realized as follows. Each node maintains a routing table that tracks its neighbor nodes in the logical coordinate space. In d-dimensional space, nodes x and y are neighbors if the coordinate ranges of their regions overlap in d − 1 dimensions, and abut in one dimension.

All the regions are convex and can be characterized as follows.
Let region(x) = $[[x^1_{min}, x^1_{max}],\ldots,[x^d_{min}, x^d_{max}]]$. Let region(y) = $[[y^1_{min}, y^1_{max}],\ldots,[y^d_{min}, y^d_{max}]]$. Nodes x and y are neighbors if there is some dimension j such that $x^j_{max} = y^j_{min}$ and for all other dimensions i $[[x^i_{min}, x^i_{max}]$ and $[[y^i_{min}, y^i_{max}]$ overlap. An example of neighboring nodes in two-dimensional space is shown below:




The routing table at each node tracks the IP address and the virtual co-ordinate region of each neighbor. To locate value *v*, its key *k(v)* is mapped to a point (P) whose coordinates are used in the message header. Knowing the neighbors' region coordinates, each node follows simple greedy routing by forwarding the message to that neighbor having coordinates that are closest to the destination's coordinates. To implement greedy routing to a destination node *x*, the present node routes a message to that neighbor among the neighbors $k \in$ Neighbors, given by:

$$argmin_{k \in Neighbors}[\min |\vec{x} - \vec{k}|].$$

Here, $^{->}$x and $^{->}$k are the coordinates of nodes x and k. Assuming equal-sized zones in d-dimensional space, the average number of neighbors for a node is *O(d)*. The average path length is $(d/4) \cdot n^{1/d}$. The implication on scaling is that each node has about the same number of neighbors and needs to maintain about the same amount of state information, irrespective of the total number of nodes participating in the CAN. In this respect, the CAN structure is superior to that of Chord. Also note that unlike in Chord, there are typically many paths for any given source-destination pair. This greatly helps for fault-tolerance. Average path length in CAN scales as $O(n^{1/d})$ as opposed to *log n* for Chord.

**CAN Complexity:**

The time overhead for a new joiner is *O(d)* for updating the new neighbors in the CAN, and $O(d/4) \cdot log(n)$ for routing to the appropriate location in the coordinate space. This is also the overhead in terms of the number of messages. The time overhead and the overhead in terms of the number of messages for a node departure is $O(d^2)$, because the TAKEOVER protocol uses a message exchange between each pair of neighbors of the departed node.

# Tapestry

The Tapestry P2P overlay network provides efficient scalable location independent routing to locate objects distributed across the Tapestry nodes. Much of the design is adapted from an earlier design of Plaxton trees. The notable enhancements of Tapestry include dealing with node churn as well as dynamic addition and deletion of objects. As in Chord, nodes as well as objects are assigned identifiers obtained by mapping from their native name spaces to a common large identifier space using a uniformly distributed hash function such as SHA-1. The hashed node identifiers are termed VIDs (*the acronym for virtual i.d.s*) and the hashed object identifiers are termed as GUIDs (*acronym for globally unique i.d.s*). For brevity, a specific node *v's* virtual identifier is denoted $v_{id}$ and a specific object O's GUID is denoted $O_G$.

## Overlay and routing:

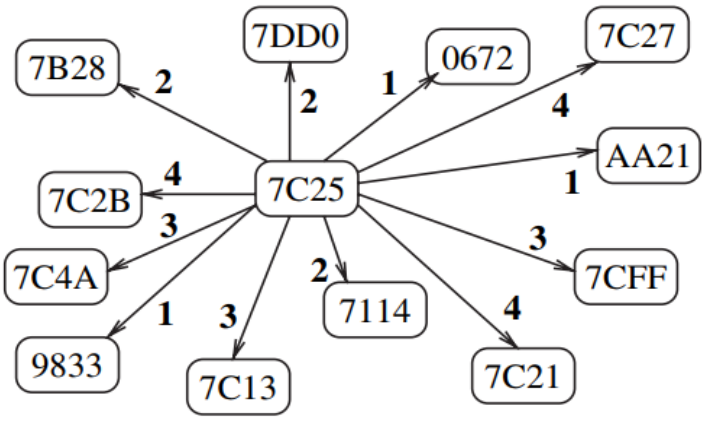
### Root and surrogate root

Tapestry uses a common identifier space specified using *m* bit values. This identifier is typically expressed in hexadecimal notation, i.e., *base b = 16*, and presently Tapestry recommends *m = 160*. Each identifier $O_G$ in this common overlay space is mapped to a set of unique nodes that exists in the network, termed as the identifier's root set denoted $O_{GR.}$ Typically, $|O_{GR}|$ is a small constant, and the main purpose of having $|O_{GR}| > 1$ is to increase fault-tolerance. In our discussion, we assume $|O_{GR}| =$, and refer to a root of $O_G$ *and* $O_{GR..}$

If there exists a node *v* such that $v_{id} = O_{GR}$ , then *v* is the root of identifier $O_G$. If such a node does not exist, then a globally known deterministic rule is used to identify another unique node sharing the largest common prefix with $O_G$, that acts as the *surrogate root*. To access object O, the goal is to reach the root $O_{GR}$ (whether real or surrogate). Routing to $O_{GR}$ is done using distributed routing tables that are constructed using prefix routing information.

## Object publication and object search:

Prefix routing in Tapestry is somewhat analogous to prefix routing within the telephone network, or to address allocation in the Internet using classless interdomain routing (CIDR). Unlike the telephone numbers or CIDR-assigned IP addresses, Tapestry's VIDs are in a virtual space without correlation to topology, however, topological information can be used to select nodes that are "close" as per some metric.




Some example links of the Tapestry routing mesh at node with identifier 7C25. Three links from each level 1 through 4 are labeled by the level. The nodes in the router table at $v_{id}$ are the neighbors in the overlay, and these are exactly the nodes with which $v_{id}$ communicates. A part of the routing mesh at one node is shown in the above Figure. For each forward pointer from node *v* to *v`*, there is a backward pointer from *v`* to *v*.

(variables)
**integer** $Table[1 \ldots log_b 2^m, 1 \ldots b]$; // routing table

(1) $NEXT\_HOP(i, O_G = d_1 \circ d_2 \ldots \circ d_{log_b M})$ executed at node $v_{id}$ to route to $O_G$:
// $i$ is (1 + length of longest common prefix), also level of the table

(1a) **while** $Table[i, d_i] = \perp$ **do** // $d_j$ is $i$th digit of destination

(1b) $d_i \longleftarrow (d_i + 1) \bmod b$;

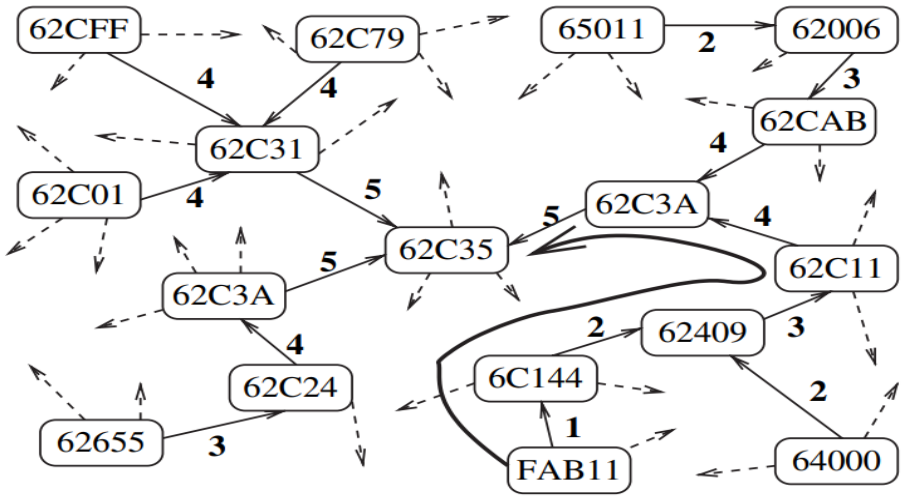
(1c) **if** $Table[i, d_i] = v$ **then** // node $v$ also acts as next hop
// (special case)

(1d) **return** $(NEXT\_HOP(i+1, O_G))$ // locally examine next digit of
// destination

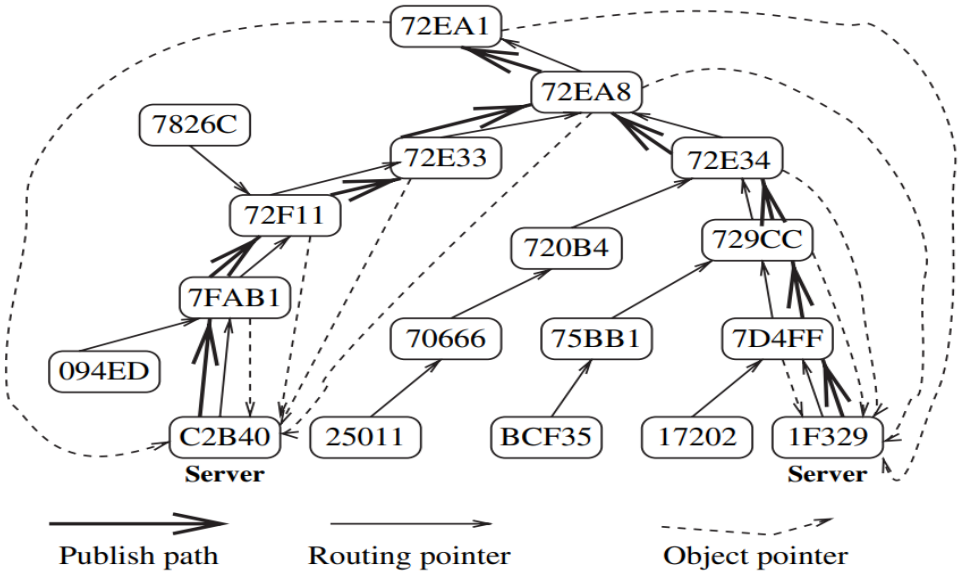
(1e) **else return**$(Table[i, d_i])$. // node $Table[i, d_i]$ is next hop

***An Example of Routing in Tapestry***

The logic for determining the next hop at a node with node identifier *v*, $1 \leq v \leq n$, based on the $i^{th}$ digit of $O_G$, i.e., based on the digit in the $i^{th}$ most significant position in $O_G$.




An example of routing from FAB11 to 62C35. The numbers on the arrows show the level of the routing table used. The dashed arrows show some unused links.




Consider the object $O_G$ which has identifier 72EA1 and two replicas at 1F329 and C2B40, as shown in the example in above Figure. A query for the object from 094ED will find the object pointer at 7FAB1. A query from 7826C will find the object pointer at 72F11. A query from BCF35 will find the object pointer at 729CC.Overall, experiments have shown that Tapestry continues to perform well with high probability, despite dynamic node insertions and failures.

## Complexity

- A search for an object is expected to take ($\log_b 2^m$) hops. However, the routing tables are optimized to identify nearest neighbor hops (as per the space metric). Thus, the

latency for each hop is expected to be small, compared to that for CAN and Chord protocols.

- The size of the routing table at each node is $c \cdot b \cdot \log_b 2^m$, where c is the constant that limits the size of the neighbor set that is maintained for fault-tolerance.

The larger the Tapestry network, the more efficient is the performance. Hence, it is better that different applications share the same overlay.

## Some other challenges in P2P system design

### Fairness: a game theory application

P2P systems depend on all the nodes cooperating to store objects and allowing other nodes to download from them. However, nodes tend to be selfish in nature; thus, there is a tendancy to download files without reciprocating by allowing others to download the locally available files. This behavior, termed as *leaching or free-riding*, leads to a degradation of the overall P2P system performance. Hence, penalties and incentives should be built in the system to encourage sharing and maximize the benefit to all nodes.

We now examine the classical problem, termed the *prisoners' dilemma*, from game theory, that has some useful lessons on how selfish agents might cooperate. This problem is an example of a non-zero-sum-game.

In the prisoners' dilemma, two suspects, A and B, are arrested by the police. There is not enough evidence for a conviction. The police separate the two prisoners, and, separately, offer each the same deal: if the prisoner testifies against (betrays) the other prisoner and the other prisoner remains silent, the betrayer gets freed and the silent accomplice gets a 10-year sentence. If both testify against the other (betray), they each receive a 2-year sentence. If both remain silent, the police can only sentence both to a small 6-month term on a minor offence.

Rational selfish behavior dictates that both A and B would betray the other. This is not a Pareto-optimal solution, where a Pareto-optimal solution is one in which the overall good of all the participants is maximized. In the above example, both A and B staying silent results in a Pareto-optimal solution. The dilemma is that this is not considered the rational behavior of choice.

In the iterative prisoners' dilemma, the game is played multiple times, until an

"equilibrium" is reached. Each player retains memory of the last move of both players (in more general versions, the memory extends to several past moves). After trying out various strategies, both players should converge to the ideal optimal solution of staying silent. This is *Pareto-optimal*.

The commonly accepted view is that the tit-for-tat strategy, described next, is the best for winning such a game. In the first step, a prisoner cooperates, and in each subsequent step, he reciprocates the action taken by the other party in the immediately preceding step.

The BitTorrent P2P system has adopted the tit-for-tat strategy in deciding whether to allow a download of a file in solving the leaching problem. Here, cooperation is analogous to allowing others to upload local files, and betrayal is analogous to not allowing others to upload. The term choking refers to the refusal to allow uploads. As the interactions in a P2P system are long-lived, as opposed to a one-time decision to cooperate or not, optimistic unchoking is periodically done to unchoke peers that have been choked. This optimistic action roughly corresponds to the re-initiation of the game with the previously choked peer after some time epoch has elapsed.

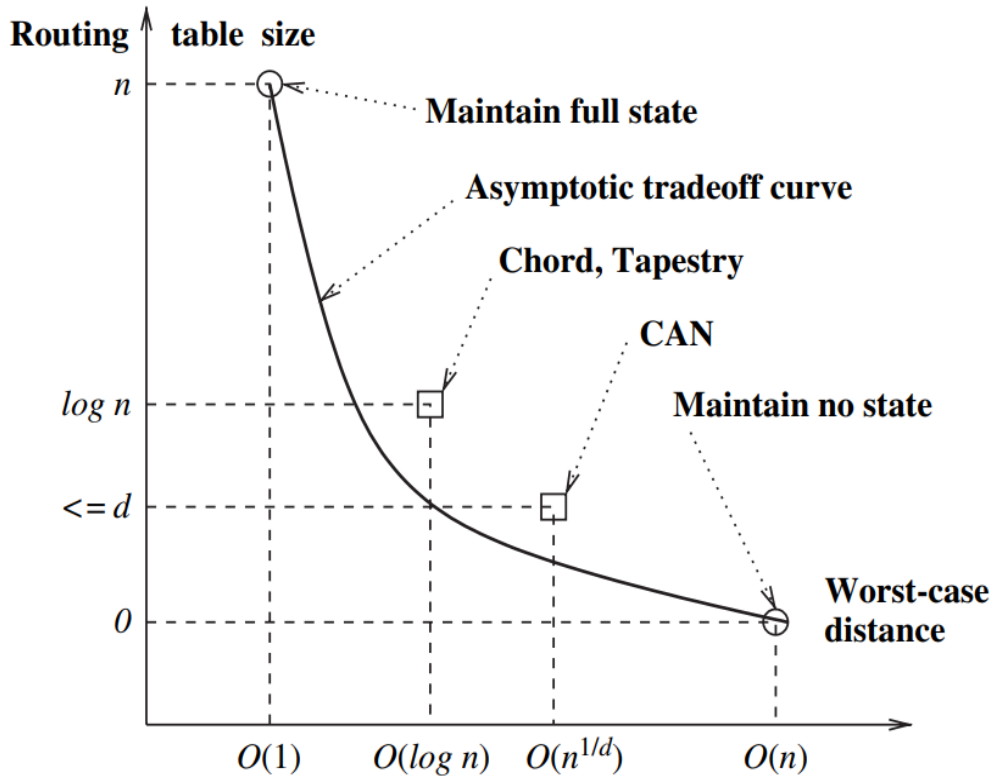
## Tradeoffs between table storage and route lengths

Chord, CAN, and Tapestry are three well-known representative protocols for managing structured P2P overlays. Despite their seeming differences, Xu et al. showed that the routing function they perform can be expressed in a uniform way by generalizing the function of classless interdomain domain routing (CIDR) used by the IP protocol.

| Protocol | Chord | CAN | Tapestry |
|---|---|---|---|
| Routing table size | $k = O(log_2 n)$ | $k = O(d)$ | $k = O(log_b n)$ |
| Worst case distance | $O(log_2 n)$ | $O(n^{1/d})$ | $O((b-1) \cdot log_b n)$ |
| $n$, common name space | $2^k$ | $x^d$ | $b^x$ |
| $S_i$ | $[2^{i-1}, 2^i)$ | $[x^{i-1}, x^i)$ | $[j \cdot b^{x-lvl+1}, (j+1) \cdot b^{x-lvl+1})$ |
| $J_i$ | $2^{i-1}$ | $kx^{i-1}$ | $suffix(J_{(lvl-1)\cdot b+j}, x - lvl + 1)$ |

***Comparison of representative P2P overlays.***

*d is the number of dimensions in CAN. b is the base in Tapestry.*




*Fundamental asymptotic tradeoffs between router table size and network diameter.*

Chord, CAN, and Tapestry are all congestion-free algorithms. A strongly uniform algorithm is node-congestion-free. When the routing algorithms are weakly uniform, $\Omega(log_2 n)$ and $\Omega(n^{1/d})$ are the lower bounds on the diameter in networks with routing tables of sizes $O(log\ n)$ and $d$, respectively. As Chord, CAN, and Tapestry are strongly uniform, they achieve the asymptotic lower bounds in the tradeoff.

## GRAPH STRUCTURES OF COMPLEX NETWORKS

P2P overlay graphs can have different structures. An intriguing question is to characterize the structure of overlay graphs. This question is a small part of a much wider challenge of how to characterize large networks that grow in a distributed manner without any coordination. Such networks exist in the following:

- Computer science: the WWW graph (WWW), the Internet graph that models individual routers and interconnecting links (INTNET), and the autonomous systems (AS) graph in the Internet.
- Social networks (SOC), the phone call graph (PHON), the movie actor collaboration graph (ACT), the author collaboration graph (AUTH), and citation networks (CITE).

- Linguistics: the word co-occurrence graph (WORDOCC), and the word synonym graph (WORDSYN).
- The power distribution grid (POWER).
- Nature: in protein folding (PROT), where nodes are proteins and an edge represents that the two proteins bind together, and in substrate graphs for various bacteria and micro-organisms (SUBSTRATE), where nodes are substrates and edges are chemical reactions in which substrates participate.

It is widely intuited that such complex graphs must display some organizational principles that are encoded in their topology in some subtle ways. This has driven research on a unification theory to determine a suitable model in which all such uncontrolled graphs are instantiations.

The first logical attempt to model large networks without any known design principles is to use random graphs. The random graph model, also known as the Erdos–Renyi (ER) model, assumes *n* nodes and a link between each pair of nodes with probability *p*, leading to $n(n-1)p/2$ edges. Many interesting mathematical properties have been shown for random graphs. However, the complex networks encountered in practice are not entirely random, and show some, somewhat intangible, organizational principles.

Three ideas have received much investigative attention in recent times

- **Small world networks** Even in very large networks, the path length between any pair of nodes is relatively small. This principle of a "small world" was popularized by sociologist Stanley Milgram by the "six degrees of separation" uncovered between any two people. As the average distance between any pair of nodes in the ER model grows logarithmically with n, the ER graphs are small worlds.
- **Clustering Social networks** are characterized by cliques. The degree of cliques in a graph can be measured by various clustering coefficients. The random graph model has a clustering coefficient of exactly *p*. As most real networks have a much larger clustering coefficient, this random graph model (ER) is unsatisfactory.
- **Degree distributions** Let $P(k)$ be the probability that a randomly selected node has *k* incident edges. In many networks – such as INTER, AS, WWW, i.e., $P(k)$ is distributed with a power law tail. Such networks that are free of any characteristic scale, i.e., whose degree characterization is independent of n, are called scale-free networks.

  In a random graph, the degree distribution is Poisson-distributed with a peak of

*P(<k>)*, where *<k>*, which is a function of n, is the average degree in the graph. Thus, random graphs are not scale-free. While some real networks have an exponential tail, the actual form of *P(k)* is still very different from that for a Poisson distribution.

Current empirical measurements show the following properties of some commonly occurring graphs:

**WWW** In-degree and out-degree distributions both follow power laws; it is a small world; and is a directed graph, but does show a high clustering coefficient.
**INTNET** Degree distributions follow power law; small world; shows clustering.
**AS** Degree distributions follow power law; small world; shows clustering.
**ACT** Degree distributions follow power law tail; small world (similar path length as ER); shows high clustering.
**AUTH** Degree distributions follow power law; small world; shows high clustering.
**SUBSTRATE** In-degree and out-degree distributions both follow power laws; small world; large clustering coefficient.
**PROT** Degree distribution has a power law with exponential cutoff.
**PHON** In-degree and out-degree distributions both follow power laws.
**CITE** In-degree follows power law, out-degree has an exponential tail.
**WORDOCC** Two-regime power-law degree distribution; small world; high clustering coefficient.
**WORDSYN** Power-law degree distribution; small world; high clustering coefficient.
**POWER** Degree distribution is exponential.

Efforts on developing models focus on random graphs to model random phenomena, small worlds to interpolate between random graphs and structured clustered lattices, and scale-free graphs to study network dynamics and network evolutions.

## INTERNET GRAPHS

### Basic laws and their definitions

Based on properties of Internet, Examples pertaining to the Web are:

- the number of links to a page,
- the number of pages within a Web location, and

- the number of accesses to a Web page.

We begin by taking the example of the popularity of Websites to illustrate the definitions of three related observed laws: **Zipf's law, the Pareto law, and the Power law.**

**Power law** $P[X = x] \sim x^{-a}$

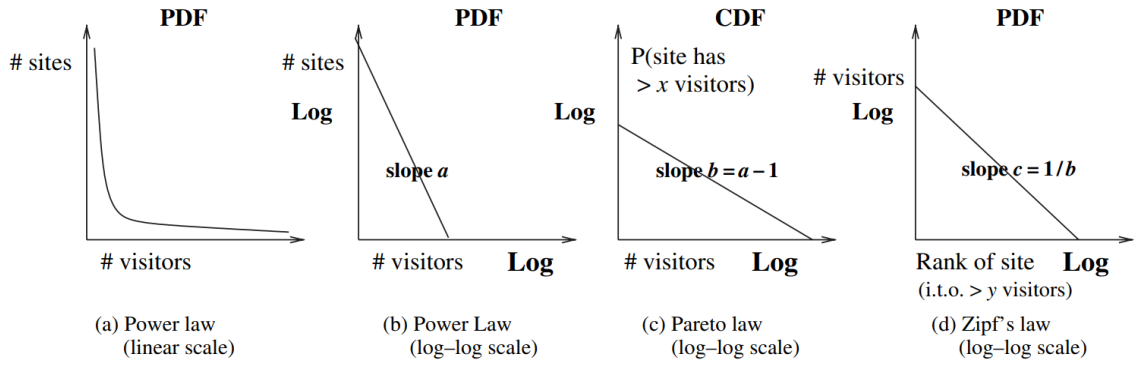
This law is stated as a probability distribution function (PDF). It says that the number of occurrences of events that equal x is an inverse power of x.

**Pareto law** $P[X \geq x] \sim x^{-b} = x^{-(a-1)}$

This law is stated as a cumulative distribution function (CDF). The number of occurrences larger than x is an inverse power of x. The CDF can be obtained by integrating the PDF. The exponents a and b of the Pareto (CDF) and Power laws (PDF) are related as b + 1 = a.

**Zipf's law** $n \sim r^{-c}$

This law states the count n (i.e., the number) of the occurrences of an event, as a function of the event's rank r. It says that the count of the $r^{th}$ largest occurrence is an inverse power of the rank r.




(a) Power law (linear scale) (b) Power Law (log–log scale) (c) Pareto law (log–log scale) (d) Zipf's law (log–log scale)

Based on the popularity of Websites., the above figure demonstrates (a) Power law showing the PDF using a linear scale. (b) Power law showing the PDF using a log–log scale. (c) Pareto law showing the CDF using a log–log scale. (d) Zipf's law using a log–log scale respectively.

## GENERALIZED RANDOM GRAPH NETWORKS

- Random graphs cannot capture the scale-free nature of real networks, which states that the node degree distribution follows a power law.
- The generalized random graph model uses the degree distribution as an input, but is random in all other respects. Thus, the constraint that the degree distribution must obey a power law is superimposed on an otherwise random selection of nodes to be connected by edges.
- These semi-random graphs can be analyzed for various properties of interest.
- Although a simple formal model for the clustering coefficient is not known, it has been observed that generalized random graphs have a random distribution of edges similar to the ER model, and hence the clustering coefficient will likely tend to zero as N increases.