Capítulo 2: Modelos de Sistemas

Modelos de Sistemas

- Ajudam a descrevem a estrutura de um sistema distribuído em termos de como seus componentes (processos, serviços, etc.) são organizados e como eles interagem, fornecendo uma visão geral da topologia
  - Identificar e corrigir problemas antes que eles afetem o sistema em produção → melhora a qualidade do projeto.
    - **Ex**: servidores, redes, serviço etc.

Tipos de Modelos em Sistemas Distribuídos

1.*Modelos Arquitetônicos:*

- Fornecem uma visão estrutural dos sistemas distribuídos, definindo como os componentes do sistema (nós, processos, dispositivos, etc.) são organizados e como eles se comunicam entre si
  - **Ex**: componentes, como servidores, clientes, serviços, e a interação entre eles.

1.1 Modelo em camadas

- Maneira de organizar de forma simplificada algo complexo em algo fácil de entender e usar
- Cada camada é responsável por uma parte específica do sistema e usa os serviços fornecidos pela camada abaixo dela.

- **Ex**: Sistema Web (contendo navegador, middleware - Apache -, SO, Roteadores e Switches)

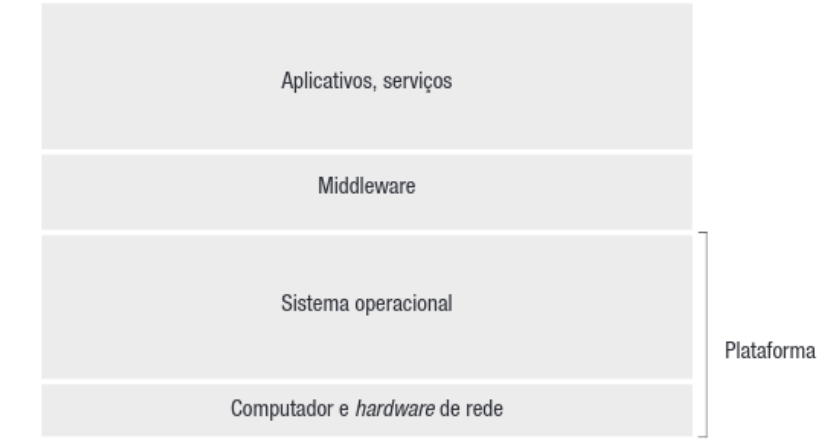


Figura 1: Ajuda a separar as responsabilidades e permitir que desenvolvedores se concentrem em aspectos específicos do sistema

2.*Modelos Fundamentais:*

- Focam em problemas e características de projeto de qualquer sistema distribuído.

2.1 Modelo de Interação

- Explicam como os componentes de um SD se comunicam entre si

Influenciado pela:

- **Latência:** Atraso entre o envio e a recepção de uma mensagem por meio do canal.Pode ser influenciada por:
  - **Transmissão:** Tempo para transmitir a mensagem.
    - **Acesso à Rede:** Tempo para acessar a rede e começar a transmissão.
    - **Propagação:** Tempo que a mensagem leva para percorrer a rede.

- **Largura de Banda:** Quantidade de dados que pode ser transmitida em uma unidade de tempo
- **Jitter:** Variação no atraso das mensagens

Desafios

1. **Ausência de Estado Global:**
    - **Tempo Não Sincronizado:** Em sistemas distribuídos cada processo pode ter seu próprio relógio, e sincronizá-los através da rede é um desafio.

Modelo Síncrono x Assíncrono

- Em relação a comunicação entre componentes em um sistema distribuído:

1. **Síncrono**:
    - A comunicação dos componentes interagem diretamente em tempo real.Quando um componente faz uma solicitação, ele aguarda até receber uma resposta antes de continuar sua execução.
        - ***Ex: Whatsapp:*** *Quando você envia uma mensagem pelo Whatsapp, o aplicativo aguarda a confirmação de entrega e leitura antes de mostrar o status de "enviado" ou "lido".*

2. **Assíncrono**:
    - mensagens sejam enviadas e recebidas em momentos diferentes.
        - ***Ex: MSM:*** *armazenadas em um servidor até que o destinatário esteja disponível para lê-la*

Diferença Entre Programa Simples e Sistema Distribuído

- **Programa Simples:**

    - Os componentes (ou partes) do programa estão todos no mesmo ambiente e podem interagir diretamente e rapidamente.
- **Sistema Distribuído:**
    - Os componentes estão espalhados por diferentes locais. Eles precisam se comunicar pela rede, o que pode introduzir atrasos e desafios adicionais.

Sincronização de Relógios

- Problema em sistemas distribuidos: cada computador ou nó na rede possui seu próprio relógio interno para marcar o tempo de eventos locais, como a execução de processos ou a recepção de mensagens
- Pode causar problemas na ordenação e coordenação de eventos

  Desafios:

    - **Defasagem Entre Relógios:** Devido a diferenças de hardware e condições operacionais, os relógios de diferentes computadores podem apresentar variações, conhecidas como defasagem ou desvio.
    - **Falta de Um Relógio Global:** Não existe um relógio único ou global que possa fornecer uma referência de tempo comum a todos os nós

Ordenação de Eventos em Sistemas Distribuídos

- Entender a sequência das ações e garantir a consistência dos dados
    - Devido a possíveis atrasos na rede, as mensagens podem ser recebidas fora de ordem

Tentativas de Solução para a Ordenação

1. ***Hora Local***
    - **Descrição**: Cada mensagem enviada inclui o carimbo de hora local do processo remetente.

- **Problemas**:
    1. **Desalinhamento de Relógios**: Relógios de diferentes sistemas não são perfeitamente sincronizados, o que pode resultar em carimbos de hora que não refletem a ordem real dos eventos.
    2. **Relógios Falhos**: Ajustes manuais ou falhas nos relógios podem introduzir inconsistências.
    3. **Latência Variável**: Variações na latência de rede podem levar à entrega de mensagens fora de ordem, independentemente dos carimbos de hora.

2. ***Tempo Lógico***
    - **Descrição**: Cada processo mantém um contador lógico, incrementado a cada evento (local ou de recepção de mensagem). O valor do contador é incluído em cada mensagem enviada.
    - **Vantagem**: Permite ordenar eventos em uma sequência lógica, sem a necessidade de relógios sincronizados.
    - **Problemas**:
        1. **Complexidade na Implementação**: Embora resolva o problema de ordem, a implementação pode ser complexa, especialmente em sistemas grandes.
        2. **Não Reflete Tempo Real**: Tempo lógico não proporciona uma noção de tempo real, apenas uma ordem relativa dos eventos.

Cenário Exemplo

Situação: Um grupo de quatro usuários (X, Y, Z, e A) troca mensagens sobre um encontro.

- **Usuário X envia mensagem m1**: Inicia a conversa.
- **Usuário Y responde com m2**: Recebe m1 e responde.
- **Usuário Z responde com m3**: Recebe m1 e m2 e também responde.

Modelo de Falha em Sistemas Distribuídos

- Descreve como as falhas podem ocorrer em processos e canais de comunicação, ajudando a entender seus impactos no sistema.

Tipos de Falhas

1. Falhas por Omissão

- Ocorrem quando um processo ou canal de comunicação deixa de realizar uma ação esperada.
    - **Processo**: Um processo pode terminar inesperadamente. Quando isso acontece, outros processos podem detectar a falha com certeza, especialmente em sistemas síncronos, onde o uso de timeouts é comum. Esse tipo de falha é conhecido como *fail-stop*.
        - **Canal de Comunicação**: As mensagens enviadas através do canal podem não ser recebidas pelo destinatário.

2. Falhas Bizantinas (Arbitrárias)

- Falhas mais graves e complexas, onde qualquer tipo de erro pode ocorrer.
    - **Processo**: O processo pode omitir passos de um algoritmo ou executar ações não intencionadas.
    - **Canal de Comunicação**: Mensagens podem ser omitidas, alteradas, duplicadas, ou modificadas de outras formas inesperadas.
    - ***Não é possível detectar a falha***

3. Falhas de Temporização

- Falhas são específicas para sistemas síncronos, onde há expectativas de tempo para a execução de processos e entrega de mensagens.

- **Processo**: Um processo pode não completar suas tarefas dentro dos tempos estipulados.
- **Canal de Comunicação**: As mensagens podem não ser entregues no tempo esperado, violando as suposições sobre o tempo de comunicação no sistema.

Lidando com Falhas

1. Mascaramento de Falhas

- Garantir que um sistema continue funcionando mesmo quando parte dele falha.Se um quebrar, o outro continua trabalhando
  - **Exemplo**: Se você tem um site que usa vários servidores para armazenar dados, mesmo que um servidor quebre, os outros servidores ainda estarão funcionando, e seu site continua no ar.

2. Confiabilidade da Comunicação

- Garantir que as mensagens enviadas entre computadores sejam recebidas corretamente.
- ***Funcionamento:***
  - **Detecção de Mensagens com Erro**: Usa um código (chamado CRC) para verificar se a mensagem foi corrompida durante a transmissão. Se algo estiver errado, a mensagem é corrigida ou reenviada.
  - **Detecção de Mensagens Perdidas**: Se uma mensagem não chega ao destino, o sistema automaticamente a reenviará.
  - **Detecção de Mensagens Duplicadas**: Cada mensagem tem um identificador único para garantir que não seja processada mais de uma vez.

Proteção de Recursos

Direitos de Acesso

- Controla quem pode fazer o quê com os recursos do sistema.

- **Exemplo**: Você pode ter um arquivo que só algumas pessoas podem ler ou editar. O sistema verifica se a pessoa tentando acessar o arquivo tem permissão para isso.

Responsabilidade Compartilhada

- Responsabilidade compartilhada entre quem oferece o serviço e quem o usa.
  - Funcionamento:
    - **Servidores**: Antes de atender uma solicitação, um servidor verifica se o usuário tem permissão para realizar a operação solicitada.
    - **Clientes**: Quando um cliente se conecta a um servidor, ele também verifica se o servidor é o correto e confiável.

Modelo de Segurança

Medidas de Proteção

- **Objetos e Processos**:
  - **Direitos de Acesso**: Define quem pode acessar ou alterar cada parte do sistema.
  - **Identidade e Autenticação**: Verifica se as entidades (pessoas ou programas) são quem dizem ser

---

Segurança dos Processos e Interações

Principais Ameaças

- **Aos Canais de Comunicação**: Mensagens podem ser interceptadas, alteradas ou falsificadas enquanto viajam pela rede.
- **Negação de Serviço (DoS)**: Ataques que sobrecarregam o sistema com um volume excessivo de solicitações, impedindo que ele funcione corretamente.
- **Mobilidade de Código**: Código malicioso que se move de um lugar para outro dentro de um sistema, disfarçado como algo inofensivo (como um vírus em um arquivo).

Comunicação Segura

Mecanismos Principais

- **Criptografia**: Processo de codificar mensagens para que só pessoas autorizadas possam ler o conteúdo.
    - **Autenticação**: Método para verificar se um usuário ou processo é realmente quem diz ser, usando chaves secretas e criptografia.
    - **Canal Seguro**: Um caminho protegido para comunicação entre dois pontos, garantindo que as mensagens sejam privadas e não possam ser alteradas ou interceptadas.
        - **Exemplo: VPN e SSL.**

Middleware - Modelo em Camadas

- Camada de software que oferece serviços e abstrações que facilitam a comunicação e o gerenciamento de recursos entre diferentes componentes e sistemas
- Atua entre o SO e as aplicações em uma arquitetura distribuida

    - ***Serviços oferecidos pelo Middleware:***

        1. ***Métodos remotos:***
            - Processo em um PC execute métodos ou funções em outro PC
                - **Exemplo:** Se um cliente precisa solicitar informações de um servidor, o middleware facilita a comunicação entre o cliente e o servidor, executando a solicitação como se fosse local.

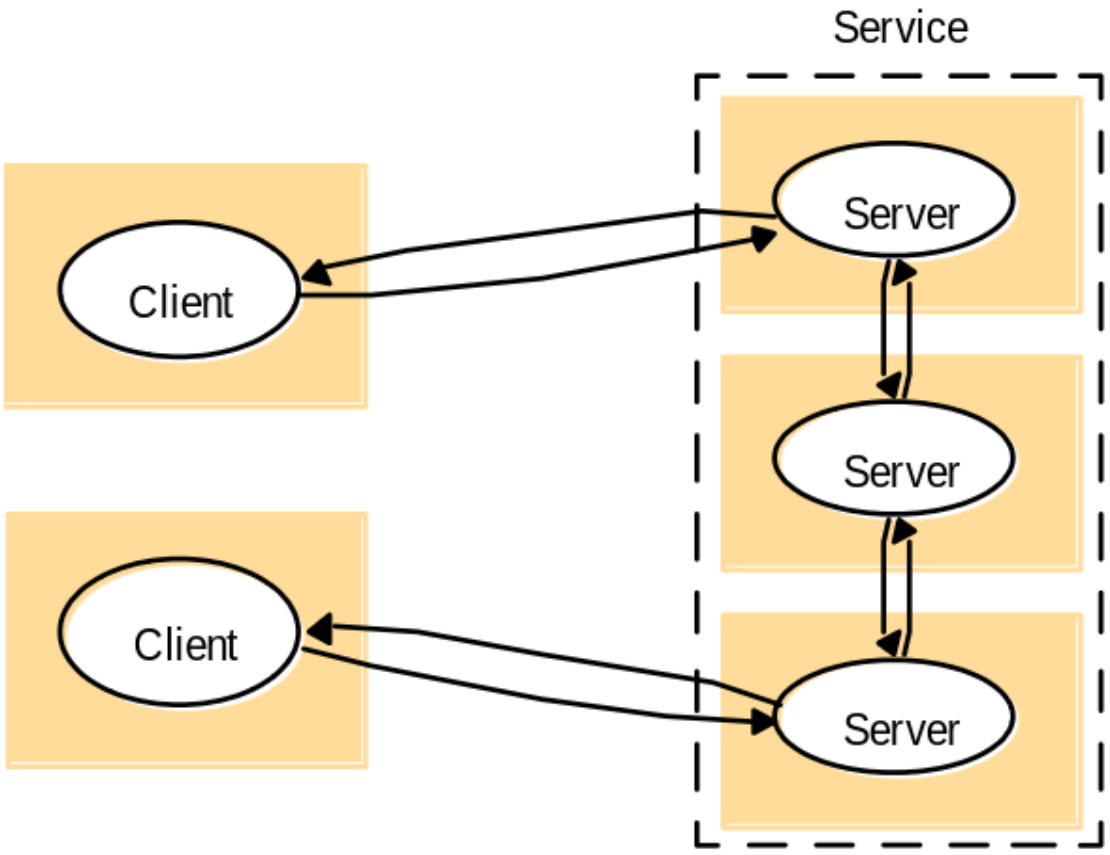
2. **Comunicação Entre um Grupo de Processos:**
    - Gerencia a troca de mensagens entre diferentes processos que podem estar em máquinas diferentes
        - **Exemplo:** Em um sistema de e-commerce, o middleware coordena a comunicação entre processos responsáveis por gestão de estoque, processamento de pagamentos e envio de confirmações.
3. **Transações:**
    - Garante que todas as partes de uma transação sejam concluídas
        - **Exemplo:** Em um sistema de reservas de voos, o middleware pode garantir que a reserva de um voo, o pagamento e a emissão do bilhete sejam processados como uma única transação, evitando problemas como a venda do mesmo assento para dois clientes.

Cliente-Servidor e suas variações/variantes

1. **Múltiplos Servidores por Serviço**

Descrição: Um serviço pode ser suportado por vários servidores para melhorar a escalabilidade e a disponibilidade.

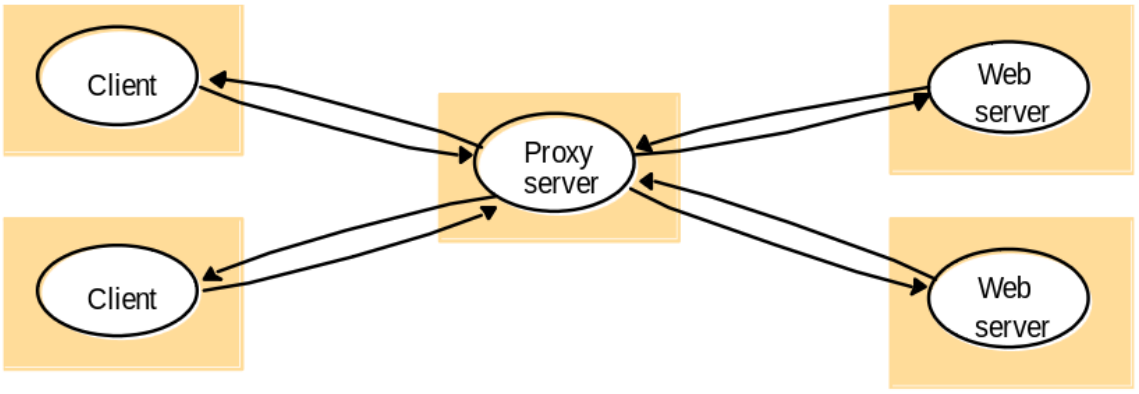
- **Exemplo:** Um serviço de web hosting pode ter múltiplos servidores que hospedam o mesmo site. Quando um usuário acessa o site, o tráfego é distribuído entre esses servidores para balancear a carga.



2. **Cache e Servidores Proxy**

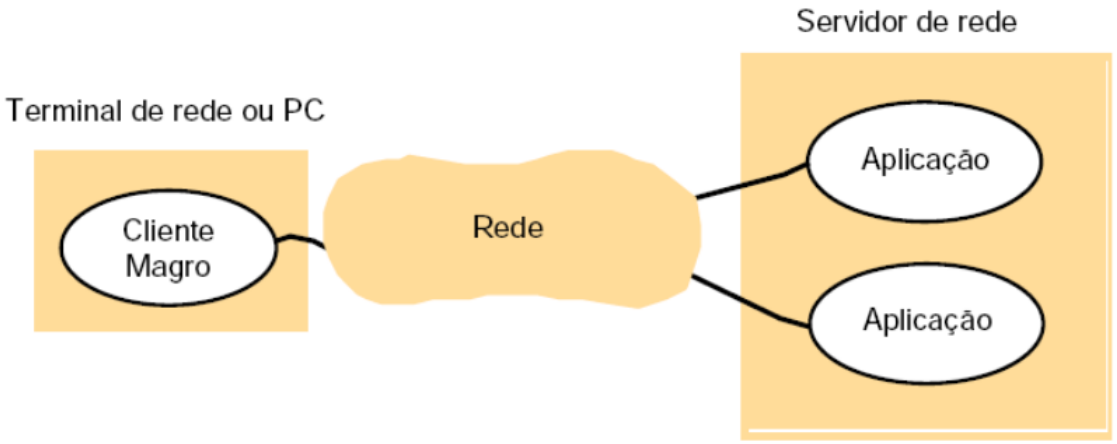
Descrição:

- **Cache** é uma forma de armazenar dados temporariamente.
- **Servidores Proxy** atuam como intermediários entre clientes e servidores, podendo também realizar caching.
- **Exemplo:** Um servidor proxy pode armazenar em cache páginas da web visitadas frequentemente. Quando um cliente solicita uma página, o proxy pode fornecer a versão em cache, reduzindo o tempo de resposta e a carga no servidor original.



3. **Clientes Magros**

- **Descrição**: O trabalho pesado é feitos por servidores remotos.O seu notebook apenas envia e recebe informações
    - **Exemplos:**
        - VNC - controlar remotamente um PC
        - LTSP (Linux Terminal Server Project): Utilizando uma combinação de DHCP, TFTP e NFS para rodar aplicativos instalados no servidor



4. Código Móvel

- **Descrição:**
    - Ao invés de o cliente simplesmente solicitar dados ou serviços do servidor, o próprio programa ou uma parte dele é transferido para o cliente para ser executado localmente.
        - Permite que o cliente interaja com a aplicação ou serviço de maneira mais rápida e direta, sem precisar sempre enviar pedidos ao servidor.
            - **Exemplo: Java applets**. Esses são pequenos programas que eram baixados pelo navegador e executados localmente para fornecer funcionalidades interativas, como jogos simples, gráficos dinâmicos, etc.

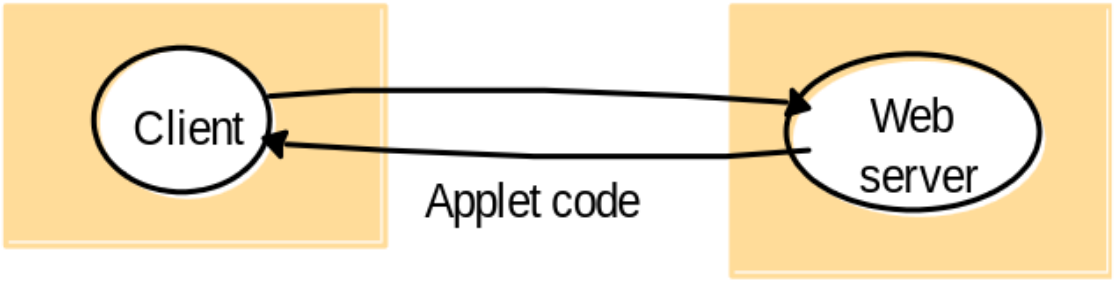
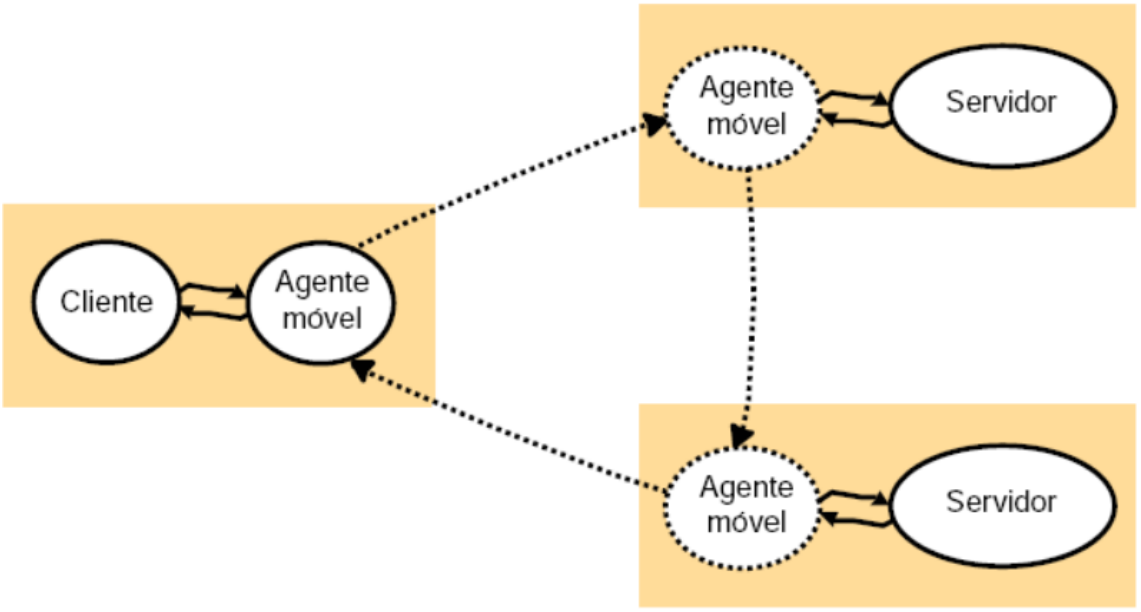


Figura 1: O cliente, que pode ser um navegador da web ou outro tipo de aplicação, faz uma solicitação a um servidor para acessar um serviço específico.

Figura 2: Em resposta à solicitação, o servidor envia o código necessário para o cliente. Esse código pode ser um applet Java, um script, ou outro tipo de módulo executável. Esse processo é conhecido como "download" do código.

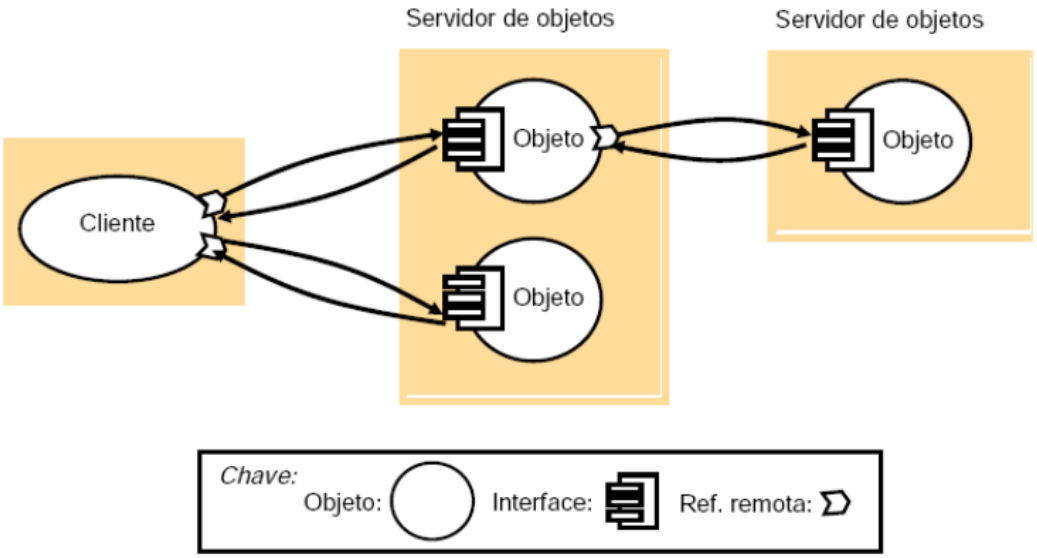
5. Agentes Móveis

- **Descrição:** Programas viaja para diferentes computadores (servidores) na rede.Quando o agente chega a um computador, ele executa suas tarefas lá mesmo, usando os recursos locais desse servidor.
    - **Exemplo:** Imagine que você quer encontrar o melhor preço para um produto. Um agente móvel pode ser enviado para diferentes sites de lojas online, coletar os preços e retornar com os dados. Ele faz isso diretamente nos servidores das lojas, em vez de você ter que acessar cada site separadamente.



6. **Objetos Distribuídos**

- **Descrição:** Os objetos distribuídos são "guardados" em servidores. Eles têm métodos (funções) que podem ser chamados por clientes para realizar operações.
- **Exemplo:** Um sistema de reservas de voo em que os dados sobre voos, reservas e disponibilidade são gerenciados por objetos distribuídos (um servidor pode gerenciar informações sobre voos, enquanto outro gerencia reservas)



7. **Dispositivos Móveis**

- **Descrição:** Uso de dispositivos móveis (smartphones, tablets) que rodam em dispositivos como PDAs, laptops, e celulares, e que acessam servidores através de uma rede sem fio
- **Exemplo:** Aplicativo de Navegação GPS usando smartphone que se conecta a servidores na nuvem

P2P (Rede Overlay - Nível de aplicação)

Conceito:

- Todos os computadores (ou "nós") são iguais e podem atuar tanto como clientes quanto como servidores.
  - Cada computador na rede pode compartilhar recursos, como arquivos, com outros computadores, sem precisar de um servidor central que coordene tudo.
  - É também conhecida como Rede Overlay (Nível de aplicação), pois é uma rede virtual criada sobre uma rede já existente que só para aqueles clientes que estão rodando o software

- ***Ex***: BitTorrent, Skype, Napster, Freenet, Gnutella, Chord, CAN

Características

1. **Arquitetura Totalmente Distribuída**
    - **Sem Controle Centralizado:** Em uma rede P2P, não há um servidor central que controla e coordena tudo.

2. **Troca Direta de Recursos**
    - **Interação Direta:** Os computadores na rede podem compartilhar recursos diretamente entre si

3. **Autonomia dos Nós**
    - **Entrar e Sair Livremente:** Cada computador na rede pode se conectar ou desconectar a qualquer momento, sem precisar de permissão de um servidor central

4. **Comunicação em Grupo**
    - **Difusão Coletiva:** A rede pode enviar informações para múltiplos computadores ao mesmo tempo (ex: nova versão de um arquivo)

5. **Problemas de Segurança e Disponibilidade**
    - **Segurança:** Garantir que nenhum computador na rede contenha código malicioso que possa comprometer a rede ou os dados.
    - **Disponibilidade:** se o nó sair da rede, o recurso pode se tornar inacessível.

OBS:

- Tem **softwares P2P que fazem a distribuição dos recursos sem os usuários participarem**
    - *Ex: tem pessoas que não escutam uma musica, mas outras procuram por ela)* → Armazena na sua máquina sem nem você

querer

Modelo de Comunicação

- Em relação a comunicação entre componentes em um sistema distribuído:

1. **Síncrono**:

    - A comunicação dos componentes interagem diretamente em tempo real.Quando um componente faz uma solicitação, ele aguarda até receber uma resposta antes de continuar sua execução.

        - ***Ex: Whatsapp:*** *Quando você envia uma mensagem pelo Whatsapp, o aplicativo aguarda a confirmação de entrega e leitura antes de mostrar o status de "enviado" ou "lido".*

2. **Assíncrono**:

    - mensagens sejam enviadas e recebidas em momentos diferentes.

        - ***Ex: MSM:*** *armazenadas em um servidor até que o destinatário esteja disponível para lê-la*

Tipos de Sistemas Distribuídos P2P

1. Não Estruturados (Aleatórios)

- Os dados podem estar em qualquer nó da rede, sem um esquema fixo de distribuição.

Característica	Descrição
Restrição de Localização dos Dados	Sem restrição. Os dados podem ser armazenados em qualquer nó da rede.
Método de Busca	Busca por palavra-chave. Utiliza técnicas de difusão para localizar recursos.
Disponibilidade de Arquivos	Alta disponibilidade devido à replicação dos arquivos em vários nós.

Entrada e Saída dos Nós	Os nós podem entrar e sair da rede a qualquer momento.
Exemplo	**BitTorrent:** Um sistema de compartilhamento de arquivos são distribuídos entre muitos usuários e a busca por partes dos arquivos é feita de forma distribuída.

2. Estruturados

- A localização dos dados é organizada de acordo com um esquema específico (tabela de hash distribuída (DHT)

Característica	Descrição
Escalabilidade	Alta escalabilidade devido a métodos de busca eficientes.
Disponibilidade de Arquivos:	estratégias específicas para replicação e recuperação para garantir alta disponibilidade.
Entrada e Saída dos Nós:	atualização da tabela de distribuição quando nós entram ou saem.

Arquitetura P2P Estruturada - Tabela Hash Distribuída (DHT)

Mapeamento de Recursos:

- A DHT é uma tabela que distribui recursos (informações, arquivos, etc.) pelos nós da rede.
- Cada recurso tem um identificador único gerado por um cálculo chamado **hash**.

Busca:

- Quando um nó deseja procurar um recurso, ele calcula o hash do recurso que deseja encontrar.
- Esse hash diz em qual nó o recurso deve estar.
- O nó então pergunta diretamente ao nó responsável por aquele intervalo de hash e recupera o recurso.

- ***Lookup***: Para encontrar um valor, você realiza uma operação de lookup, onde a chave é usada para procurar o valor correspondente na tabela.

Exemplo:

- **Você quer encontrar "documento.txt".**
- **Você calcula o hash de "documento.txt", que novamente é 12345.**
- **A DHT diz que o nó responsável pelo intervalo que inclui 12345 é o "Nó A".**
- **Você pergunta ao "Nó A" e ele te dá o "documento.txt".**

FUNCIONAMENTO

- Tem vários componentes na rede, e cada componente vai ser responsável por algumas chaves da minha tabela hash.
  - Se um nó receber uma requisição e tiver a informação (chave) eu respondo com o valor que ele está buscando.
  - Se eu não tiver, eu passo a requisição para os vizinhos (para o nó que esteja proximo da chave que está sendo buscada)

CARACTERÍSTICAS

- **Mapear chave de forma balanceada:** Se eu tenho 10000 milhão de nós na rede, eu não quero que alguns tenham muita carga para gerenciar e outros não.
- ***Função de distância entre a chave e o ID de um nó:*** Cada nó participante tem um ID, e eu consigo calcular a distância de um nó para a chave que está sendo buscada (dessa tabela)
- **Tabela de roteamento:** cada nó tem uma tabela de roteamento para que possa mandar requisição e garantir que quem sair ou entrar terá tabela e que , assim, as chaves sejam encontradas
- **Chaves e Valores**: Cada dado armazenado na rede é identificado por uma chave única, que é usada para localizar o nó que armazena o valor

correspondente.

***OBS*: Tem que ficar toda hora monitorando para ver se um nó saindo ou entrou para consultar e atualizar o sistema e sua tabela**

***OBS*: Para diminuir o problema de entrada e saida, tem que ter uma redundância na rede, como por exemplo, vários nós com o mesmo ID OU balancear a carga se ela for muito demandada**

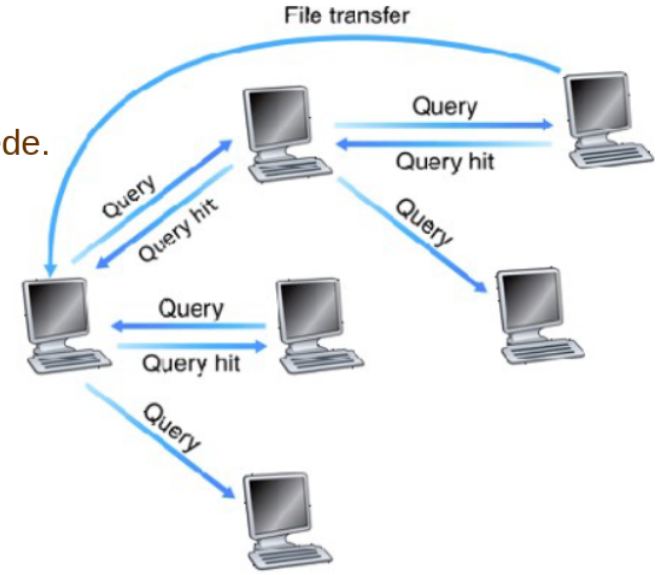
TOPOLOGIAS DE REDES P2P

1. PURA

- **Descrição**: **Inexistente de um servidor**:
  - Não há servidores centralizados ou hierarquia; todos os nós atuam simultaneamente como clientes e servidores
    - ***Vantagens***:
      - Alta resiliência e robustez devido à ausência de um ponto único de falha.
      - Balanceamento de carga distribuído.
    - ***Desvantagem***:
      - Eficiência de busca pode ser baixa, pois a pesquisa de dados pode envolver muitos nós.
      - Maior consumo de largura de banda devido à necessidade de comunicação entre muitos nós.
- **Exemplos:** Gnutella e Freenet.

Gnutella

- ping
  - Descobre servidores na rede.
- pong
  - Resposta ao ping.
- query
  - Procura por um arquivo.
- query hit
  - Resposta ao query.
- push
  - Solicitação de download para um nó protegido por firewall.



2. *HÍBRIDA*

- **Descrição**:
  - Existrencia de **um ou mais servidores centrais.**
  - Esses servidores não armazenam os dados, mas ajudam na localização dos nós que possuem os dados.

    Funcionamento: Os clientes se registram nos servidores centrais e informam quais serviços ou recursos estão disponíveis. Quando um cliente precisa de um recurso, ele consulta o servidor central para descobrir quais nós o possuem.

    - **Exemplo**: Napster, onde o servidor central era consultado para identificar onde a música estava armazenada entre os clientes.
      - Ex: Pergunta para o servidor Napster para identificar onde é que está a música

    - ***Vantagens***:

- Alta resiliência e robustez devido à ausência de um ponto único de falha.
- Balanceamento de carga distribuído.

- ***Desvantagem***:
  - Eficiência de busca pode ser baixa, pois a pesquisa de dados pode envolver muitos nós.
  - Maior consumo de largura de banda devido à necessidade de comunicação entre muitos nós.

1. ***SUPERPURA (Super Peer)***

- **Descrição**: alguns nós assumem um papel de "super-peers" **(intermediários) para outros nós comuns para ajudar um nó comum a encontrar alguma informação ou recurso**.
- **Vantagens**: Alta disponibilidade de dados e recursos, com redundância e eficiência de comunicação.
- **Desafios**: Maior complexidade de gerenciamento de conexões e recursos.

BUSCA POR RECURSOS

- Se refere a como buscar recursos pela rede distribuída

1. **INUNDAÇÃO**

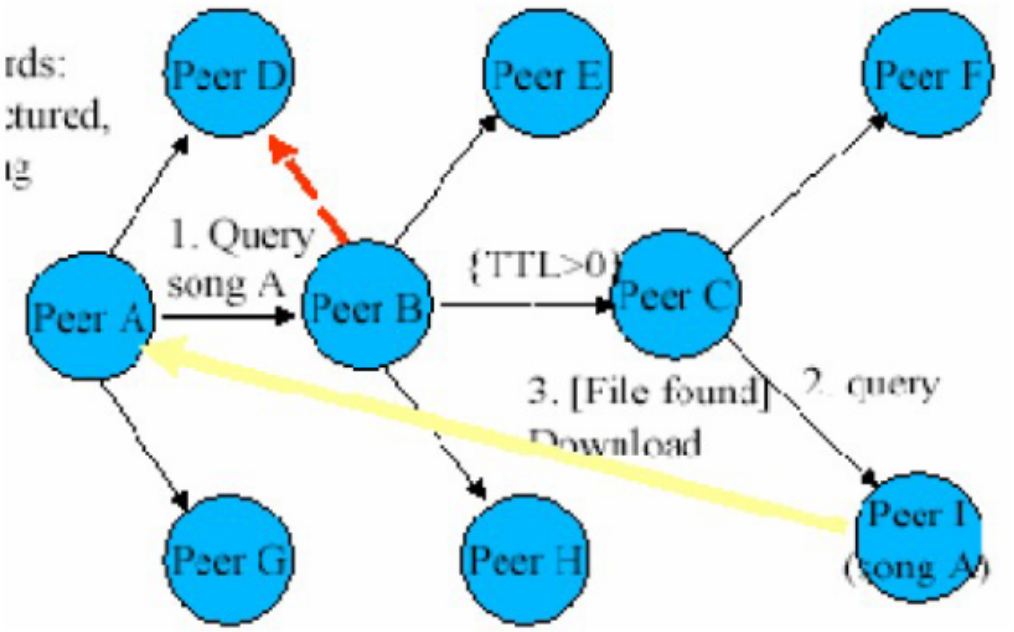
- Maneira de encontrar recursos sem a necessidade da Tabela Hash Distribuída (DHT)

Funcionamento:

1. **Solicitação de Consulta:** Quando um nó (um computador ou dispositivo) precisa encontrar algo, como um arquivo ou uma informação, ele

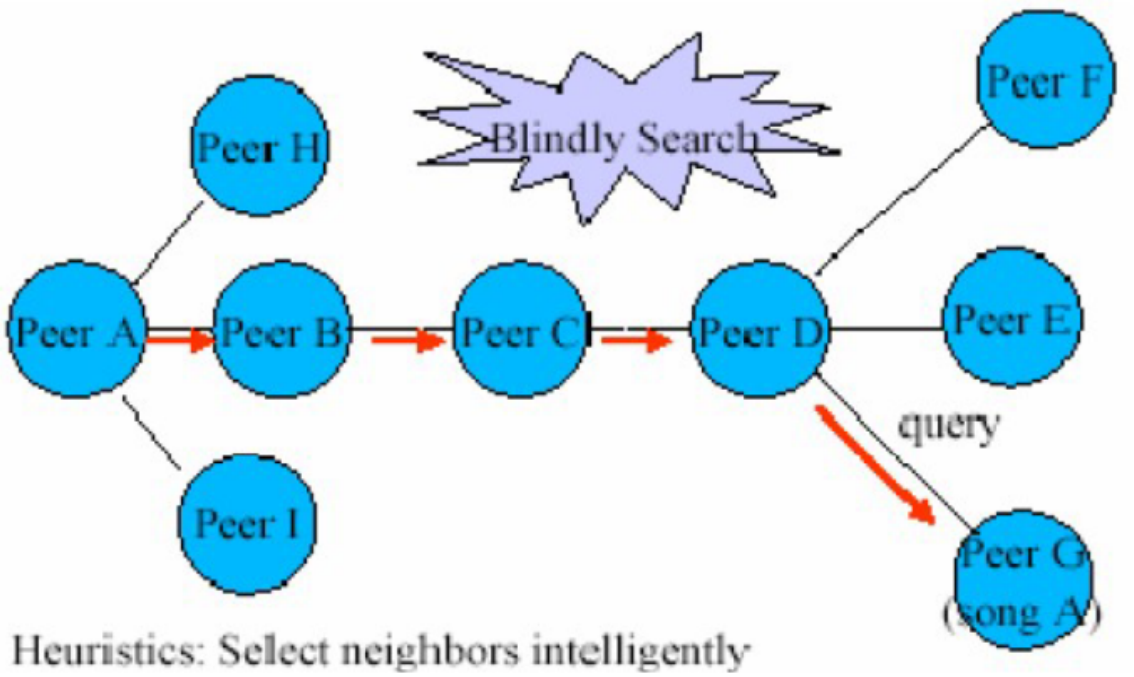
começa a busca enviando uma solicitação para todos os seus nós vizinhos diretamente conectados.

2. **Propagação**: Se não o tiver, o nó encaminha a solicitação para todos os seus próprios nós vizinhos, exceto para o nó de onde a consulta veio.
3. ***Time to Live:*** Cada vez que o pacote passa para um novo vizinho , esse TTL vai decrementando



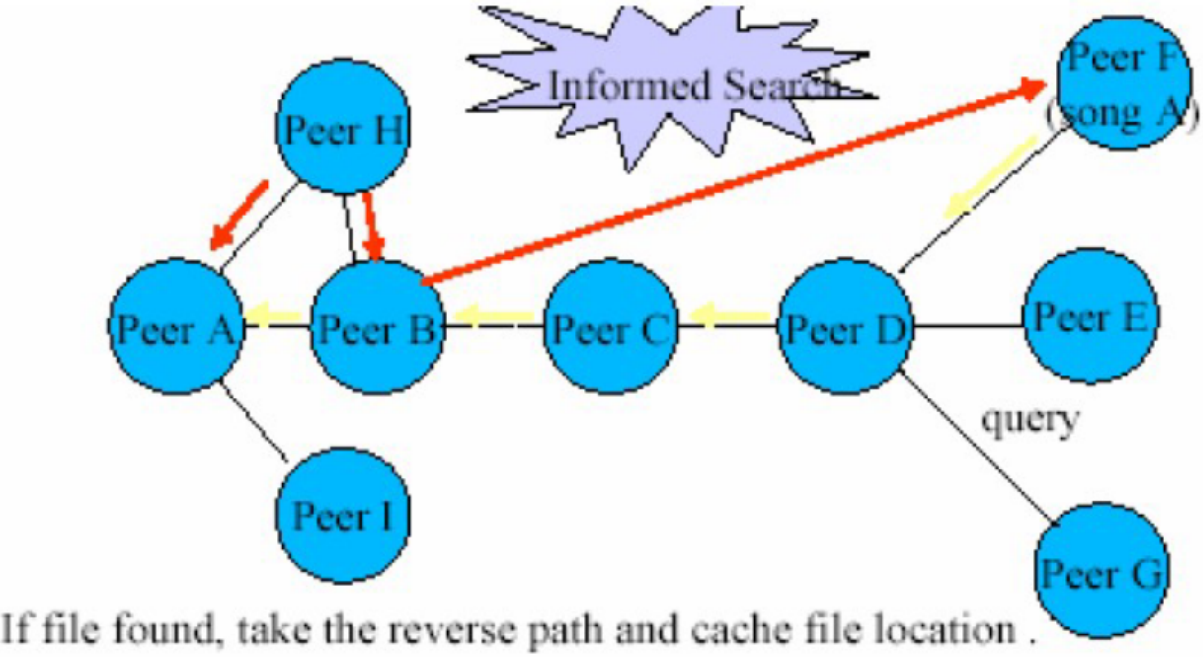
BUSCA CEGA

- Ao inves de enviar para todos os vizinhos, envia apenas para o vizinho que geralmente acerta quando eu procuro ele
  - Se não encontrar, ele manda para outro



BUSCA INFORMADA

- Nó não avisa direto ao emissor, e sim ele passa por todas os nós que participaram e que deu certo anteriormente (com base em histórico) → SEM REPLICAÇÃO

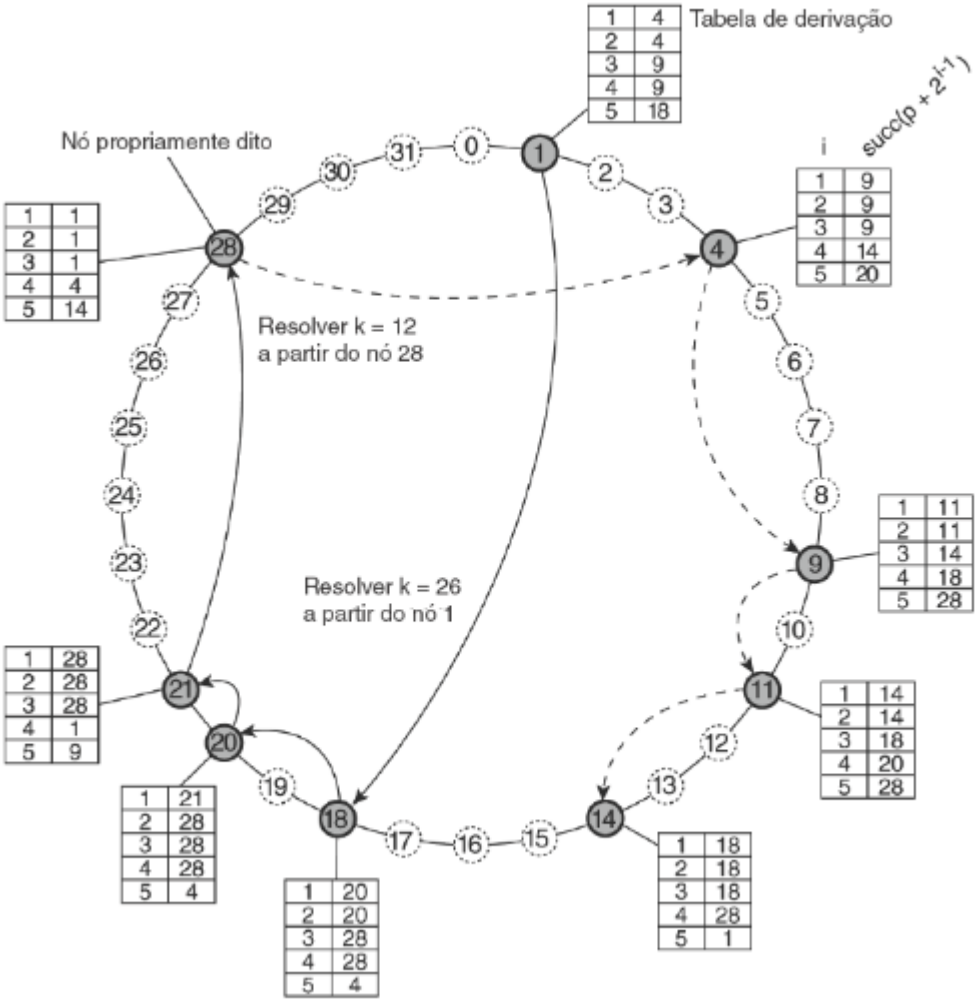


BUSCA INFORMADA COM REPLICAÇÃO

- próprios recursos são armazenados nos nós ao longo do caminho da consulta (registram a localização do recurso, mas também armazenam o próprio recurso em seus caches)

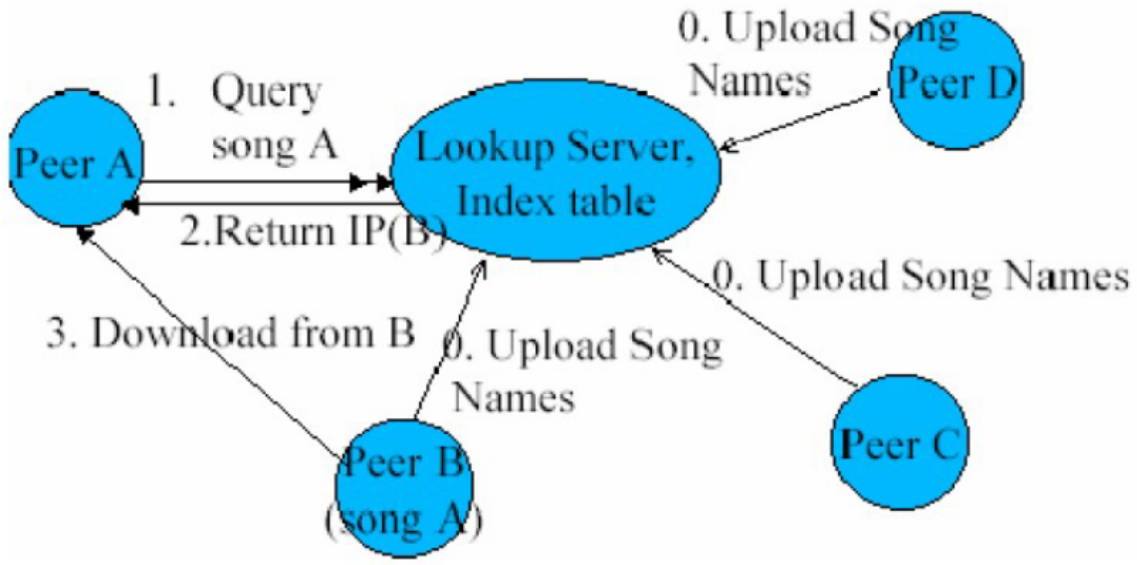
BUSCA VIA MAPEAMENTO DE IDs

- Técnica para encontrar recursos usando identificadores únicos (IDs - criado usando a função hash) evitando a necessidade de procurar por todos os nós na rede e torna a busca mais rápida.
    - Cada recurso é associado a um ID único, e os nós (computadores) na rede sabem onde encontrar esses recursos com base nesses IDs.
    - Quando você quer encontrar um recurso, você fornece o ID desse recurso. O sistema usa esse ID para descobrir qual nó deve ter o recurso.
    - Tem a replicação dos dados



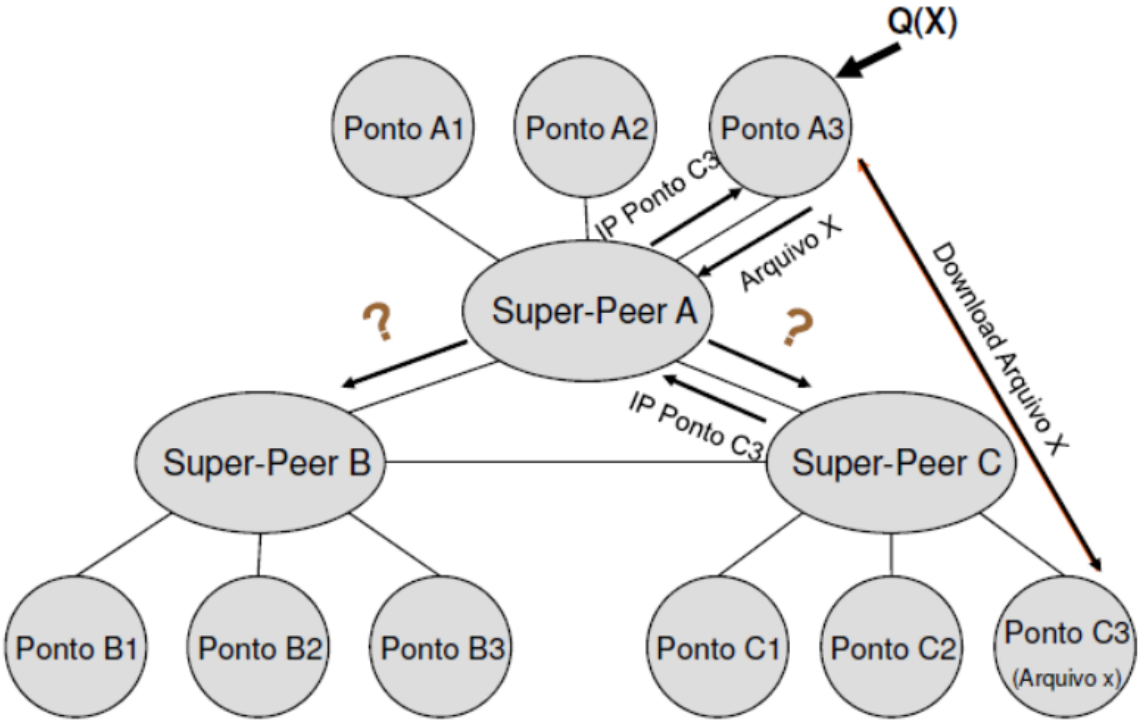
BUSCA EM CATÁLOGO CENTRALIZADO

- Servidor central responsável por manter uma lista de todos os recursos disponíveis na rede
    - lugar onde todos os nós da rede vão para encontrar informações sobre os recursos.



BUSCA VIA SUPER NÓS

- Quando um nó deseja localizar um recurso, ele consulta seu super nó. Se o super nó não tiver a informação desejada, ele pode encaminhar a consulta para outros super nós.



Perguntas

> “Como organizar/procurar os serviços no usuário para saber em que usuário está?”
>
> - Cada nó na rede é responsável por uma parte do espaço de chave e pode ajudar a localizar recursos ou serviços associados a essas chaves.