Aplicações Distribuídas

Prof. Dr. Marcos A. Simplicio Jr. – mjunior@larc.usp.br

Sistemas Distribuídos: Histórico

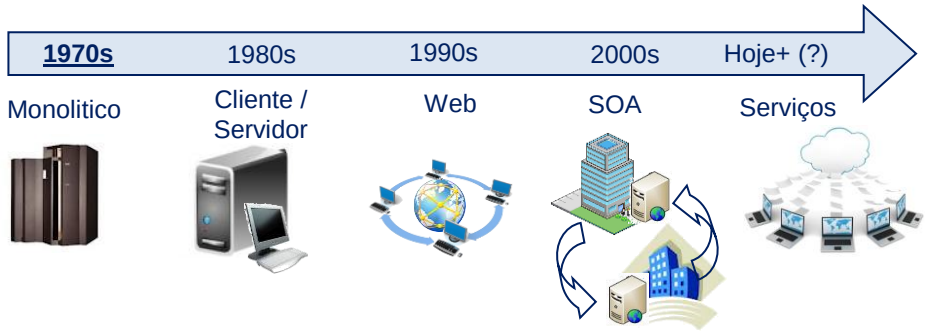
- Até início dos anos 80:
  - Computadores eram recursos **caros**
  - Disponíveis em pequeno número em empresas de grande porte e, em geral, eram **máquinas isoladas**.
- Em meados da década de 80, dois avanços tecnológicos mudaram esse cenário:

  - **Microprocessadores**: deram origem às máquinas de 8 bits e, logo a seguir 16, 32 e 64 bits, com poder de **processamento crescente** e **custo reduzido** em relação à sua potência.

  - **Redes rápidas**: taxas de transferência de 10 Mbps, dando origem a redes locais de computadores, interconectando dezenas ou centenas de máquinas

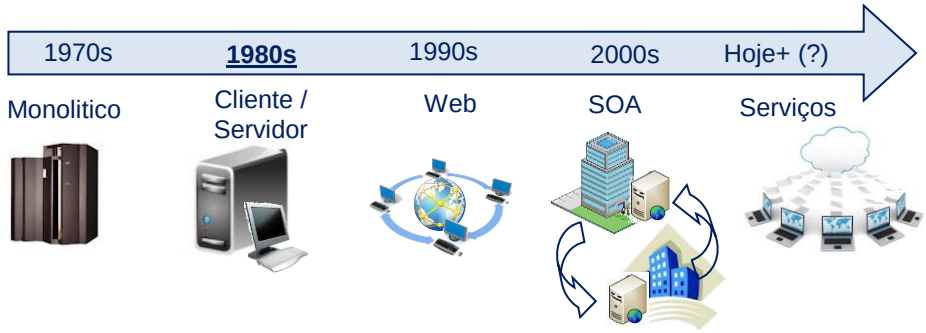
Sistemas Distribuídos: Histórico

- Terminais "burros"
- Processamento em mainframes (batch)



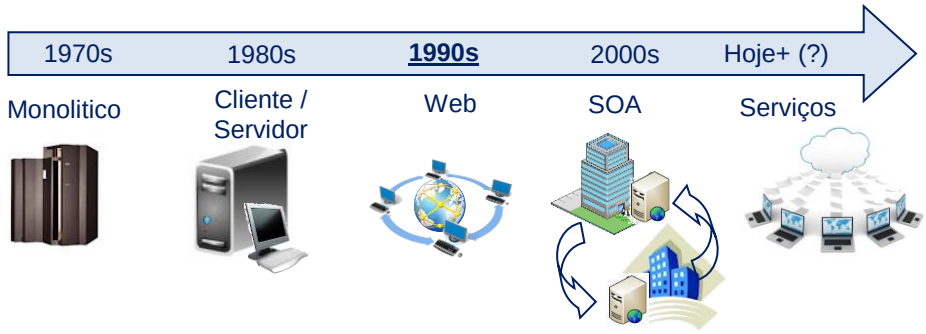
Sistemas Distribuídos: Histórico

- Terminais com microprocessadores: mais tarefas executadas localmente
  - Melhor relação poder computacional/custo
- Comunicação principalmente local (intranet)



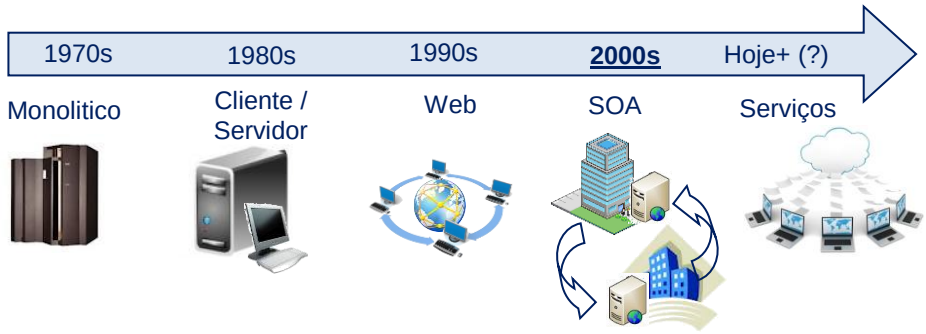
Sistemas Distribuídos: Histórico

- Explosão no número de computadores e serviços
- Consolidação da Internet (comunicação global)



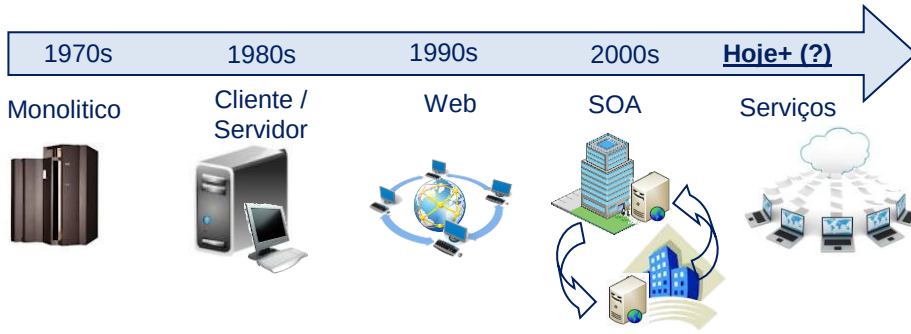
Sistemas Distribuídos: Histórico

- Service-Oriented Architecture (SOA): software passa a ser construído para **consumir serviços** de terceiros na Internet
  - Ex.: informações sobre previsão de tempo, ou hora certa
- Surgem interfaces padrão para facilitar construção e consumo de serviços (REST, SOAP, ...)



Sistemas Distribuídos: Histórico

- Serviços passam a incluir "poder computacional"
  - Ex.: processamento, armazenamento, etc.
- Graus variados de descentralização:
  - Ex.: nuvem < névoa < P2P



Sistemas Distribuídos: Definição

"Um sistema distribuído é aquele cujos componentes, localizados em computadores ligados em rede, se comunicam e coordenam suas ações por meio da passagem de mensagens"

- Coulouris et al.

- Foco desta apresentação: **P2P**
  - Blockchain encontra-se nessa categoria!

Sistemas P2P

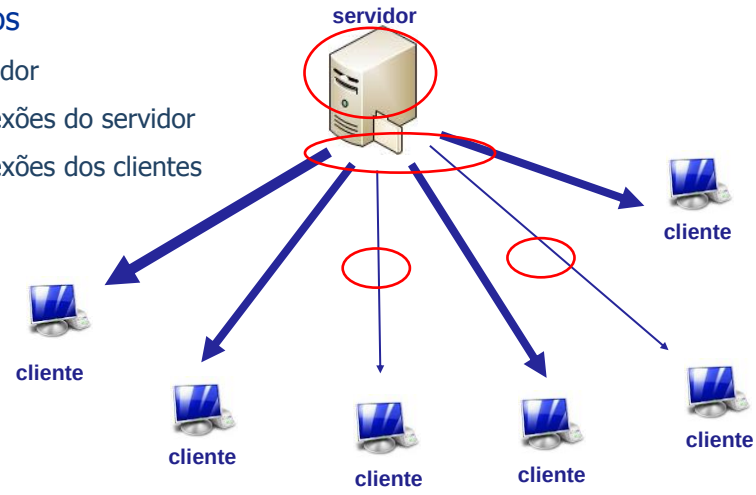
Conceitos básicos

Arquitetura Cliente-Servidor: Limitações

- Arquitetura cliente/servidor: cara de criar e manter
  - CAPEX do Google em centro de dados: ~U$10 bi/ano
    - http://www.datacenterknowledge.com/archives/2017/02/01/google-ramped-data-center-spend-2016
  - Cada centro usa de 50 a 100 MW de potência

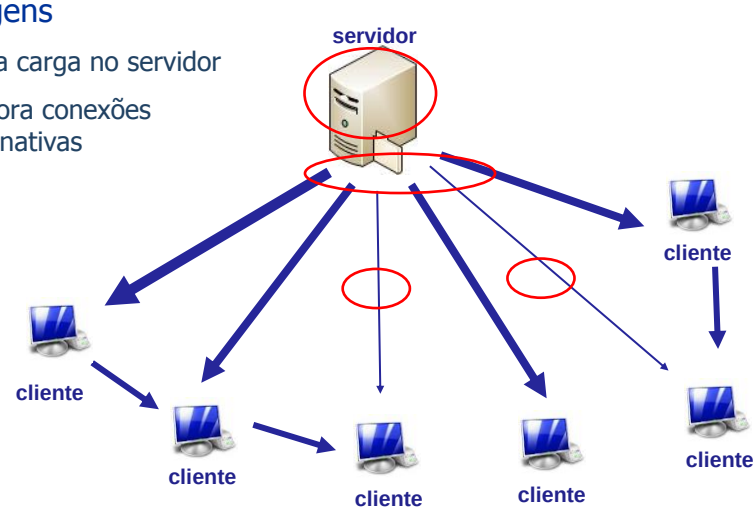
Arquitetura Cliente-Servidor: Limitações

- Escalabilidade limitada: gargalos
  - Servidor
  - Conexões do servidor
  - Conexões dos clientes

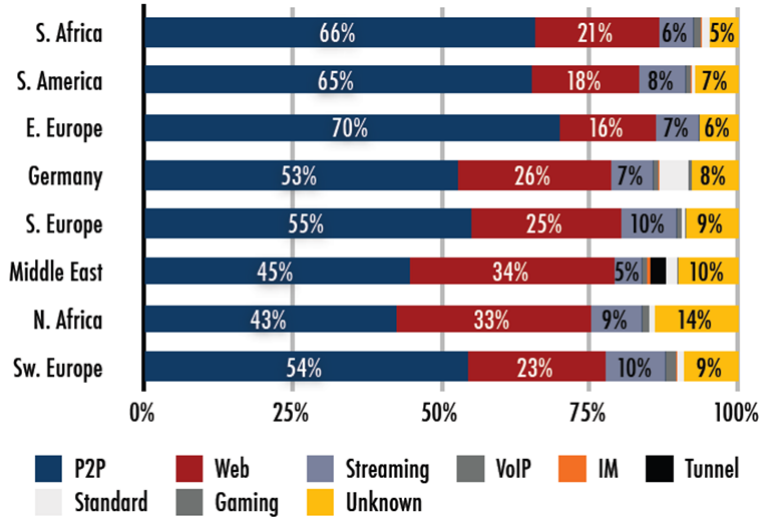


Arquitetura P2P: Colaboração

- Vantagens
  - Alivia carga no servidor
  - Explora conexões alternativas



Internet: Distribuição de tráfego 2008/2009

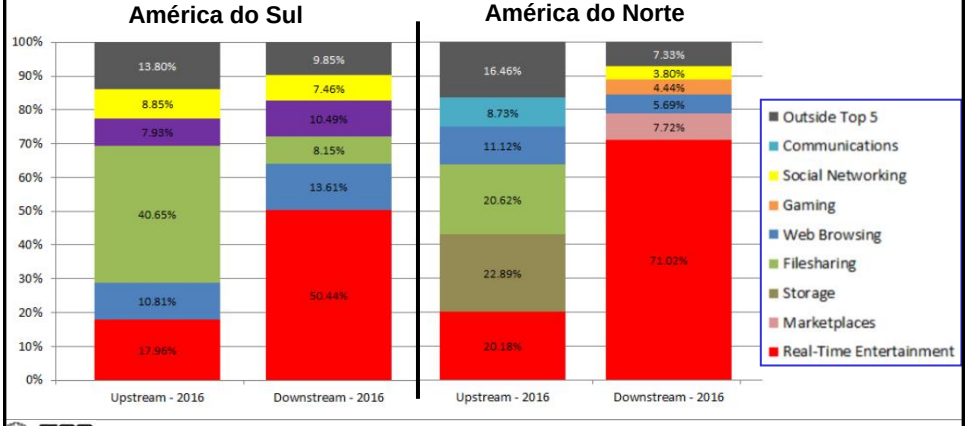


***Streaming**: crescimento mais expressivo desde então

Fonte: IPOQUE

Internet: Distribuição de tráfego 2016

- Tráfego em período de pico (redes fixas)
  - Fonte: Sandvine Global Internet Phenomena 2016



Internet: Distribuição de tráfego 2016

- Tráfego em período de pico (redes fixas) – Principais aplicações
  - Fonte: Sandvine Global Internet Phenomena 2016

América do Norte

P2P

Upstream		Downstream		Aggregate	
BitTorrent	18.37%	Netflix	35.15%	Netflix	32.72%
YouTube	13.13%	YouTube	17.53%	YouTube	17.31%
Netflix	10.33%	Amazon Video	4.26%	HTTP - OTHER	4.14%
SSL - OTHER	8.55%	HTTP - OTHER	4.19%	Amazon Video	3.96%
Google Cloud	6.98%	iTunes	2.91%	SSL - OTHER	3.12%
iCloud	5.98%	Hulu	2.68%	BitTorrent	2.85%
HTTP - OTHER	3.70%	SSL - OTHER	2.53%	iTunes	2.67%
Facebook	3.04%	Xbox One Games	2.18%	Hulu	2.47%
FaceTime	2.50%	Facebook	1.89%	Xbox One Games	2.15%
Skype	1.75%	BitTorrent	1.73%	Facebook	2.01%
	69.32%		74.33%		72.72%

Internet: Distribuição de tráfego 2016

- Tráfego em período de pico (redes fixas) – Principais aplicações
  - Fonte: Sandvine Global Internet Phenomena 2016

América do Sul

P2P

Upstream		Downstream		Aggregate	
BitTorrent	30.03%	YouTube	28.48%	YouTube	25.91%
YouTube	9.30%	HTTP - OTHER	11.66%	HTTP - OTHER	11.12%
HTTP - OTHER	7.59%	SSL - OTHER	9.76%	BitTorrent	10.06%
Facebook	6.72%	Netflix	8.31%	SSL - OTHER	9.28%
SSL - OTHER	6.19%	BitTorrent	6.96%	Netflix	7.45%
Ares	5.27%	Facebook	5.10%	Facebook	5.32%
Skype	2.53%	MPEG - OTHER	2.28%	MPEG - OTHER	2.10%
Netflix	1.97%	RTMP	1.79%	RTMP	1.66%
Dropbox	1.16%	Google Market	1.69%	Google Market	1.52%
MPEG - OTHER	0.92%	Flash Video	1.60%	Flash Video	1.46%
	71.69%		77.63%		75.87%

Arquitetura P2P: características

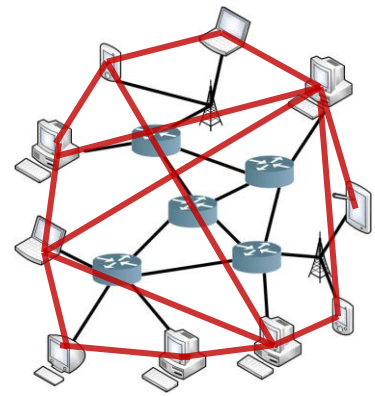
- Rede de nós com capacidades e responsabilidades simétricas

  - Nós se comunicam diretamente
  - Nós são tanto servidores como clientes: são "serventes"
- Características:
  - **Descentralização**: diversos computadores operando de forma dinâmica e (espera-se) colaborativa

  - **Agrupamentos por interesses** (ex.: usuários interessados em obter uma cópia de certo arquivo)
  - **Heterogeneidade**: capacidades de processamento, memória, banda, etc.

Arquitetura P2P: overlay

- Redes P2P são redes sobrepostas (ou "overlay")
  - Uma rede formada por **conexões lógicas** entre nós sobre um conjunto conexões físicas existentes
  - **Proximidade física** não é necessariamente levada em consideração
  - Manutenção do overlay pode ser problemática devido à **entrada e saída dinâmica** de nós



Arquitetura P2P: Benefícios

- Redução de custos: **recursos compartilhados**
- **Escalabilidade inerente**

  - Mais nós = mais recursos, não apenas mais demanda
- **Confiabilidade**
  - Eliminação de **pontos únicos de falha**
  - **Falhas independentes**: uma não afeta outra

- **Autonomia e anonimato**

  - Independente de servidores/provedores (mais difícil de aplicar **censura** ou restrições de **licenciamento**)

Arquitetura P2P: Desafios

- **Administração**
  - **Entrada e saída** dos nós dinamicamente
    - Solução: **redundância**; detecção & recuperação

  - Difícil garantir **qualidade de serviço**
    - Solução: **incentivo** para colaboração

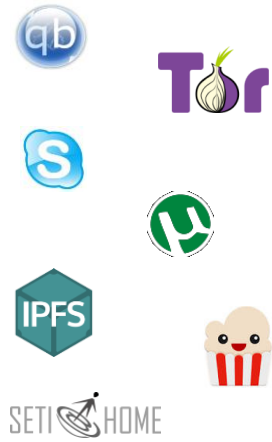
  - **Heterogeneidade** dos nós:
    - Solução: **middlewares** para abstração; padrões **abertos**

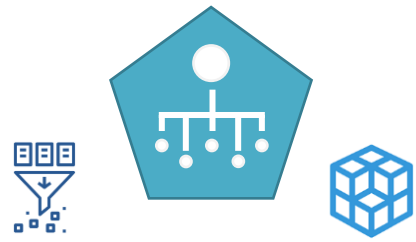
  - Acesso **concorrente**, sem relógio global: pode causar **conflitos**
    - Solução: semáforos vs. **consistência eventual**

- **Localização**
  - Nós precisam localizar **uns aos outros**
  - Nós precisam localizar **recursos** distribuídos pela rede

Aplicações P2P: Exemplos

- Diversos aplicativos, alguns deles extremamente populares
  - **Compartilhamento de arquivos**
  - **Navegação Web com privacidade**
  - **Sistemas de arquivos**
  - Comunicação instantânea
  - Jogos online
  - Computação distribuída
  - Transmissão de vídeo
  - *Moedas digitais*





Sistemas P2P

Organização e Mecanismos

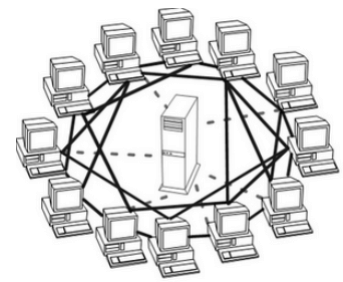
Aplicações P2P: Classificação

- Grau de descentralização
  - Parcialmente centralizado
  - Híbrido descentralizado
  - Puramente descentralizado

- Grau de estruturação da busca
  - Estruturada
  - Fracamente estruturada
  - Não estruturada

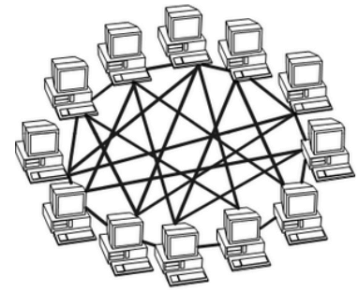
Organização: híbrido descentralizado

- Servidor central facilita interação entre peers.
  - Servidor faz buscas de conteúdo e identifica peers
- Ex.: Napster, BitTorrent com tracker
- Limitações (?): ponto único de falha, escalabidade



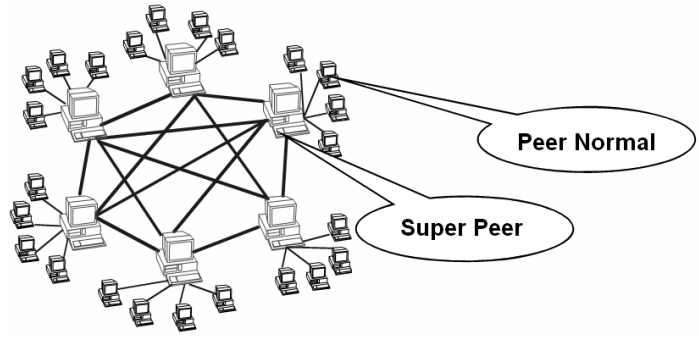
Organização: puramente descentralizado

- Nós de rede realizam todos as mesmas tarefas
  - Nenhuma coordenação central das atividades da rede
- Ex.: Gnutella, IPFS, Freenet, Bittorrent sem tracker
- Limitações (?): consistência dos dados, complexidade de gerenciamento, segurança, overhead de comunicação



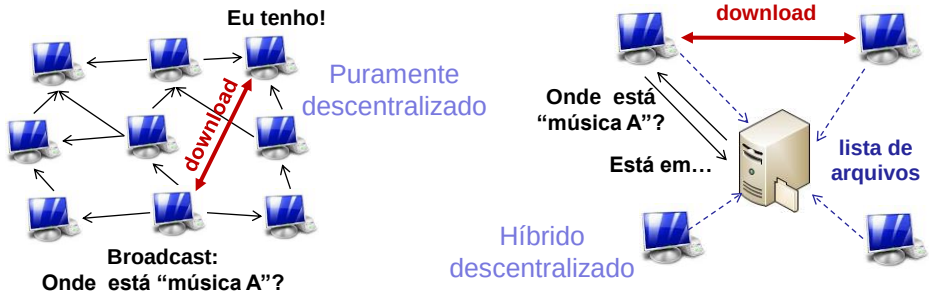
Organização: parcialmente centralizado

- Alguns dos nós assumem um papel mais importante do que outros no sistema
  - Supernós: agem como indexadores locais de nós/conteúdos
- Ex.: Skype, KaZaa, novo Gnutella



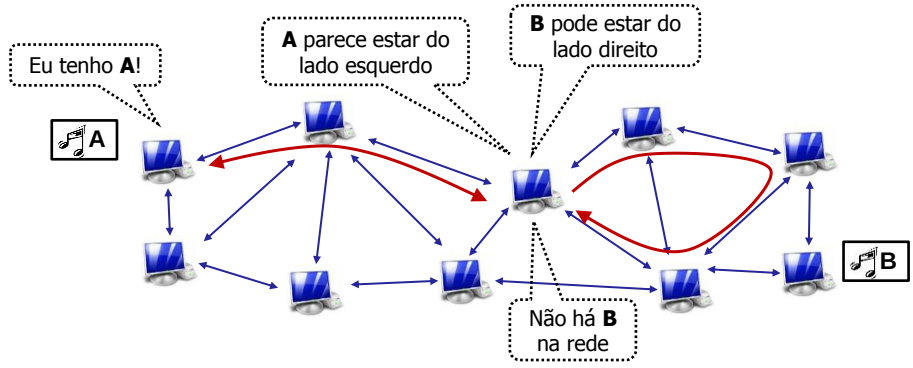
Busca: não-estruturada

- Dados distribuídos aleatoriamente entre os peers.
  - Localização dos dados não tem relação com a organização da rede.
- Ex.: Napster, Gnutella, KaZaa



Busca: fracamente estruturada

- Busca baseada em heurística
  - Localização dos arquivos afetada por "dicas" de roteamento, mas não totalmente especificada
- Ex.: Freenet



Busca: estruturada (DHT)

- Topologia da rede bastante controlada e dados (ou referências para os dados) são colocadas em locais específicos
  - Mapeamento entre ID do conteúdo e sua localização
- Ex.: Chord, Kademlia, Tapestry, Pastry.

Busca: estruturada (DHT)

- Hash Table
  - Estrutura de dados que mapeia "chaves" em "valores"
  - Interface:
    - put(chave, valor): insere valor na tabela
    - get(chave): recupera valor da tabela
- Distributed Hash Table (DHT)
  - Similar, mas espalhada pela Internet
  - Desafio: localizar conteúdo

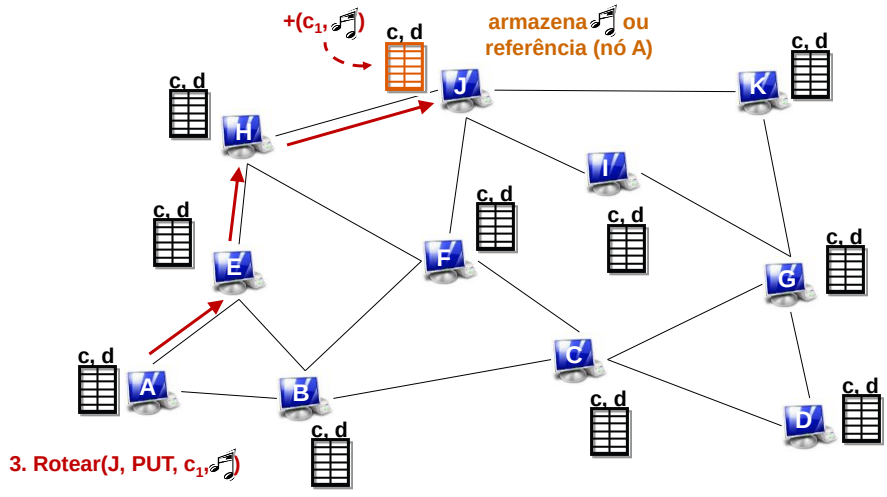
Busca: estruturada (DHT)

- Hash table em um único nó:
  - chave = hash (dado)
  - put(chave, dado)
  - get(chave): dado
- Distributed Hash Table (DHT):
  - chave = hash (dado)
  - Busca (chave) : IP_nó
  - Rotear (IP_nó, PUT, chave, dado)
  - Rotear (IP_nó, GET, chave) : dado
- Ideia:
  - Nó específico armazena (referência para) conteúdo específico
  - Todos os nós têm capacidade de roteamento: dada uma chave, eles roteiam mensagens para o nó que armazena a chave

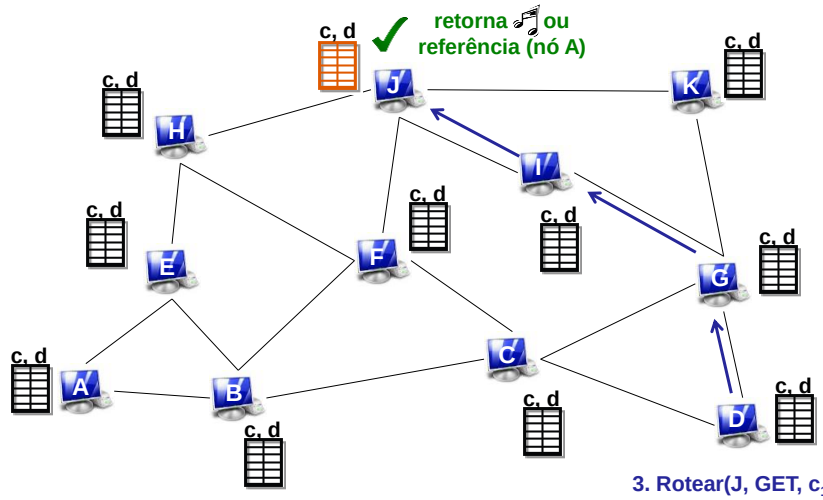
Hash(♫M) = 2442

chave	dado
0140	♫ E
1515	ZIP Z
2442	♫ M
3910	PDF P
4441	ZIP X
...	...
8731	ZIP Y

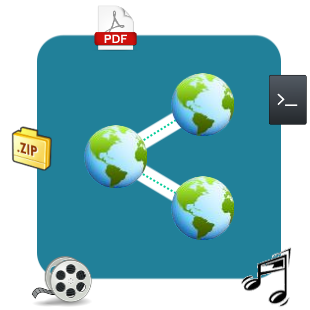
Busca: estruturada (DHT)



Busca: estruturada (DHT)



Obs.: busca pode ser iterativa ou recursiva



Sistemas P2P

Aplicações:

Compartilhamento de arquivos

BitTorrent

- Uso: distribuição de arquivos P2P
  - Exemplos: uTorrent, qBittorrent, Deluge

- Nova rede overlay criada para cada arquivo sendo distribuído
- Pode-se enviar "link" (arquivo .torrent) a um amigo
  - "Link" sempre se refere ao mesmo arquivo
    - Não é o caso de Napster, Gnutella, ou KaZaA: redes baseadas em buscas (difícil identificar arquivo específico)
  - Buscas não estão inclusas no protocolo, mas podem ser implementadas via sites web ou na interface de um aplicativo

BitTorrent

- Nomenclatura:

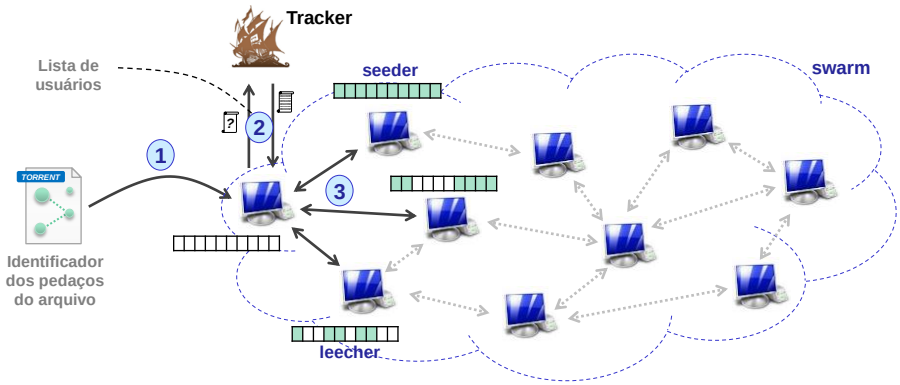
  - **Tracker** (Rastreador): mantém lista de peers interessados em certo conteúdo
  - **Piece** (Pedaço): Uma parte de um arquivo que está disponível na rede.

  - **Seeders** (Semeadores): peers que têm o arquivo completo e continuam compartilhando-o (comportamento autruísta)

  - **Leechers** (Sanguessugas): peers que têm apenas partes do arquivo e estão compartilhando e recebendo pedaços
  - **Arquivo .torrent**: metadados do arquivo
  - **Swarm** (enxame): conjunto de peers que participam na distribuição de um determinado conteúdo.

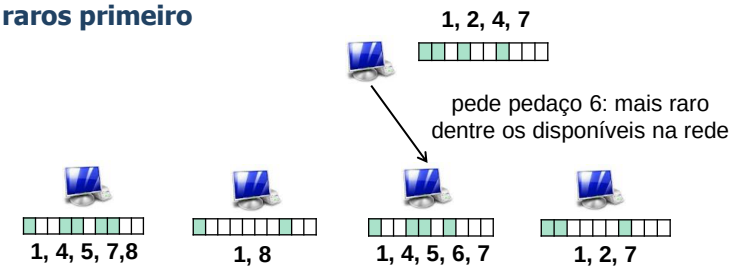
BitTorrent

- Funcionamento
  - Assumindo presença de tracker



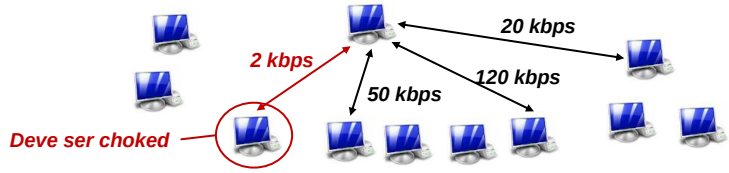
BitTorrent: mecanismos

- Arquivo dividido em pedaços (comum: 256 KiB)
- Recebendo pedaços:
  - Periodicamente, cada peer pede a cada vizinho a lista de pedaços que eles têm;
  - Ele então envia requisições para os pedaços que faltam, dando prioridade àqueles com menor disponibilidade: **mais raros primeiro**



BitTorrent: mecanismos (cont.)

- Enviando pedaços:
  - Um peer envia pedaços a *n* (comum: 4) vizinhos atuais, dando preferência àqueles que estão fornecendo pedaços na maior velocidade: "**tit-for-tat**", ou "**olho por olho**"
    - Diz-se que os peers neste grupo estão "unchoked"
  - Reavalia grupo unchoked a cada t (comum: 10) segundos
  - Seleciona um peer aleatoriamente a cada t_c (comum: 30) segundos, e o coloca no grupo unchoked no lugar do peer com menor velocidade: **optimistic unchoke**
    - Diz-se que o nó removido foi "choked".

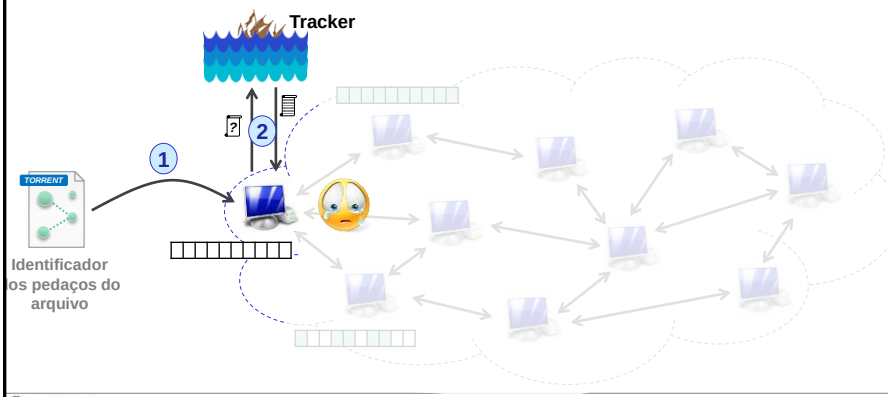


BitTorrent: mecanismos (cont.)

- Resultado das políticas do BitTorrent
  - Peers servem peers que os servem em retorno: peers com **capacidades semelhantes** (i.e., banda) tendem a interagir
  - Encoraja **cooperação**, desencoraja free-riding: free-riders são "choked" após algum tempo
  - "Rarest first" não apenas ajuda a **manter arquivo na rede**, mas também **incentiva colaboração** com novos peers: eles recebem primeiro os pedaços mais raros!
  - **Conexão a diversos peers** ao mesmo tempo pode levar a **"chokes" frequentes**: divisão da banda disponível entre os peers conectados
  - **Seeders** também têm política de "tit-for-tat", mas observam a **taxa de download** do leecher ao invés da taxa de upload.
    - Preferência por leecher baixando bastante

BitTorrent sem tracker: disponibilidade

- Tracker essencial para busca de peers...
  - E se tracker sair do ar...?



BitTorrent sem tracker: disponibilidade

- BitTorrent sem tracker:
  - Distributed Hash Table (**DHT**): peers se organizam de forma que um auxilia o outro na busca por arquivos
    - Nota: 1º peer obtido via tracker, cache local, ou servidor web

  - **Links magnéticos**: usa DHT para obter arquivo .torrent, antes de iniciar download do arquivo em si
  - Peer Exchange (**PEX**): peers conectados a um nó qualquer fornecem listas de nós aos quais eles estejam conectados
- Maior disponibilidade

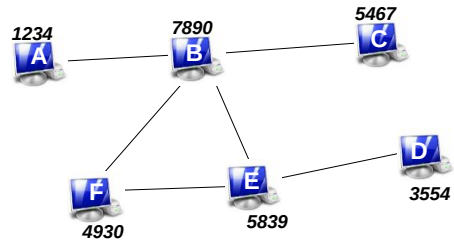
  - Resistência a censura/ações legais
  - Resistência a ataques de negação de serviço

Freenet

- Puramente descentralizado, fracamente estruturado
- Objetivo: fornecer método **anônimo** para armazenar e buscar dados (arquivos), que ficam cifrados no sistema.
  - Usuários fornecem parte de seu disco para armazenamento, e recebem "da rede" um armazenamento semelhante
  - Pode ser usado em modo "open" ou "restrito": conexão com todos os usuários ou com conjunto específico (e.g., web of trust)
- Índices (chaves) usadas para:
  - Agrupamento de arquivos com chaves semelhantes
    - Não há relação semântica entre índices (hashes pseudoaleatórios)
  - Ajuda no roteamento e otimização de buscas
- Cada nó mantém:
  - Arquivo local de dados: arquivos com chaves semelhantes
  - Tabela de roteamento: pares (endereço, identificador)

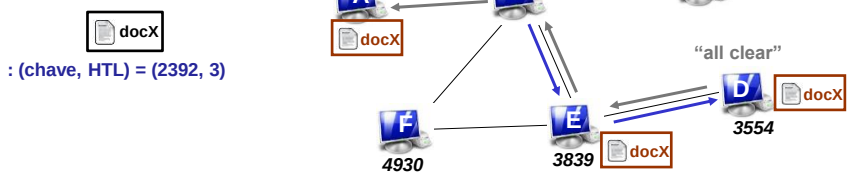
Freenet

- Juntando-se à rede:
  - Novo nó deve descobrir endereço de nó(s) existente(s)
    - Exemplos: site web ou árvore de confiança GnuPG
  - Novo nó "anuncia sua existência" e recebe um identificador aleatório pelo sistema
    - Identificador determina quais chaves ele irá armazenar
    - Caso seja detectado conflito, identificador é recalculado



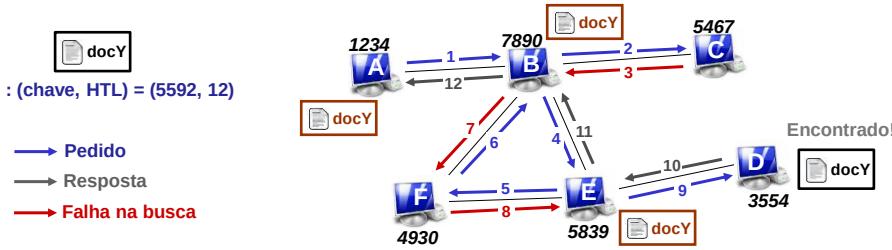
Freenet

- Inserção de arquivos:
  - Nó **A** calcula chave (hash) do arquivo e especifica número de saltos (HTL).
  - **A** Envia pedido de inserção a nó cujo identificador é "próximo" à chave
  - Nó que recebe pedido:
    - Se já tem a chave, retorna o arquivo pré-existente
    - Se não encontra a chave, propaga o pedido para nó seguinte (novamente, baseado em proximidade lexicográfica)
  - Quando (HTL) é atingido, mensagem "all clear" é enviada ao originador.
  - Originador envia arquivo, que é armazenado em todos os nós no caminho
  - Rotas criadas em todos os nós no caminho



Freenet

- Busca de arquivos:
  - Nó **A**: pedido da forma (chave, HTL).
  - Se não disponível localmente, envia pedido para nó cujo identificador é "próximo" à chave
  - Nó que recebe pedido:
    - Se encontrado, retorna arquivo e indica quem é a fonte
    - Se não encontra, propaga o pedido para nó seguinte (novamente, baseado em proximidade lexicográfica)
  - Arquivo retornado é armazenado em todos os nós no caminho
  - Rotas criadas em todos os nós no caminho



Aplicações P2P & Anonimato: Freenet

- Peculiaridades
  - Usuário tem **pouco controle** sobre o que armazena
  - **Resistente a censura**: buscar arquivo leva a sua replicação; apenas arquivos pouco buscados acabam sendo removidos
    - Política de remoção de arquivos "antigos" depende do nó
  - Atacantes tentando **sobrescrever** arquivos legítimos acabam **replicando** arquivos pré-existentes
    - **Hash sempre verificado**: respostas falsas facilmente filtradas
    - Arquivos podem ser **assinados** e **atualizados por dono**
  - Comumente: chave para **decifrar** o conteúdo é calculada a partir de texto com **descrição completa** do conteúdo
    - Descrição: publicada por usuário que gera arquivo
    - Logo, usuário pode negar saber que arquivo estava em seu nó, e não é possível provar o contrário ("**plausible deniability**")

Freenet (cont.)

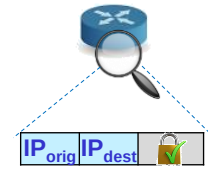
- Vantagens:
  - Anonimato
  - Robustez a falhas de nós aleatórios
  - Baixo overhead de comunicação
  - Boa escalabilidade
  - Rede auto-organizada
- Desvantagens:
  - Baixa precisão no roteamento (heurístico...)
  - Negação de serviço via roteamento incorreto

Aplicações P2P

Roteamento com privacidade

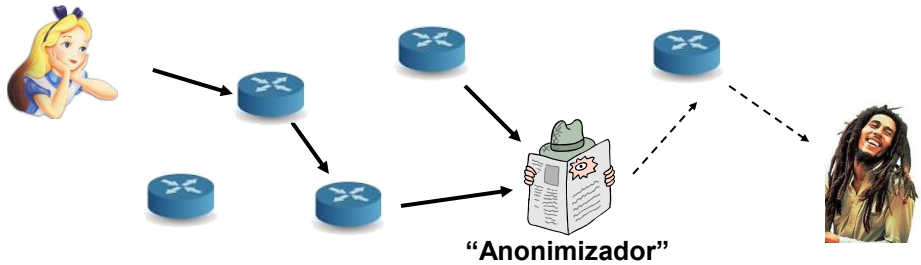
Roteamento com privacidade

- Nenhuma medida de segurança
  - Tráfego não é cifrado ou autenticado
  - Além da origem e do destino, todos os roteadores intermediários têm acesso ao tráfego
- Solução simples: **s**egurança nas camadas superiores
  - Ex.: HTTP**S**, TL**S**, **S**SH, **S**FTP/**S**CP, etc.
  - Dão confidencialidade, mas não **privacidade**: roteadores intermediários (talvez desonestos) ainda sabem **quem são os nós comunicantes**



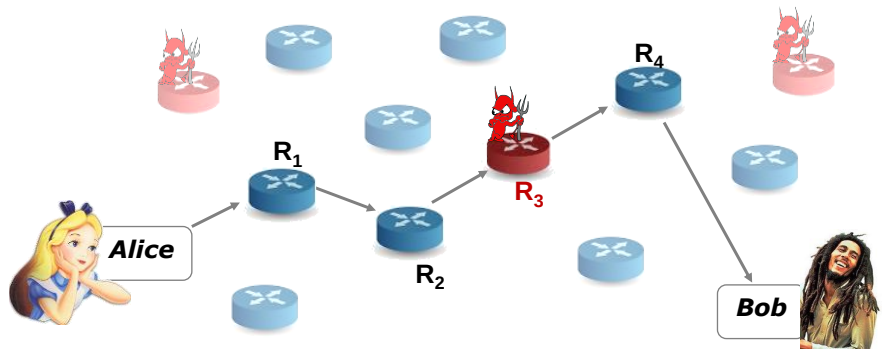
- Roteamento com privacidade?
  - "Como disfarçar a origem e o destino dos dados?"

Roteamento com privacidade



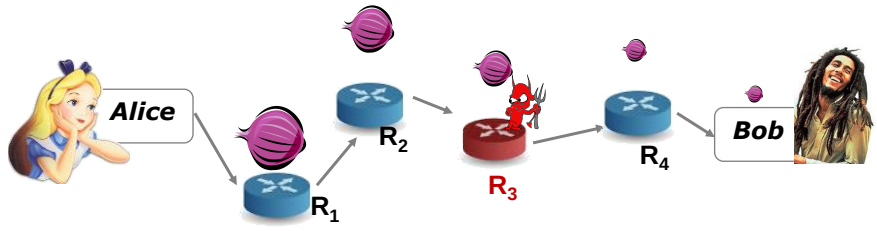
- Roteamento aleatorizado: disfarça origem das mensagens fazendo roteamento aleatório
  - Técnicas populares: Crowds, Freenet, Onion routing
  - Roteadores não têm certeza se a origem aparente de uma mensagem é de fato seu originador ou outro roteador

Roteamento com privacidade

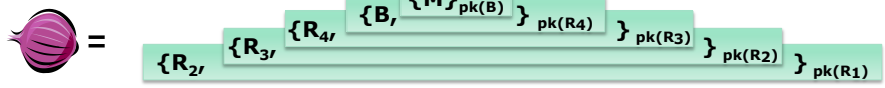


- Onion Routing: origem escolhe uma sequência aleatória de roteadores
  - Alguns roteadores honestos, outros controlados por atacantes
  - Origem controla a comprimento do caminho

Roteamento com privacidade: Onion Routing



cria



Info de roteamento de cada link é cifrada com a chave pública (pk) do roteador

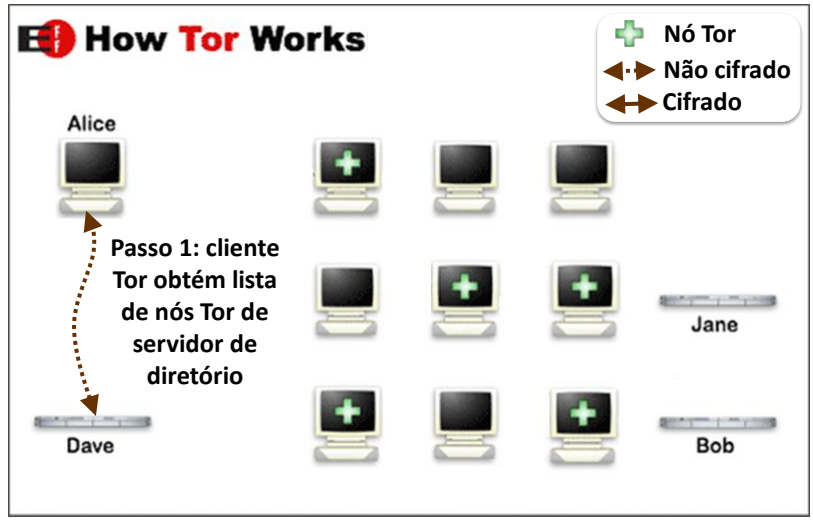
Cada roteador descobre apenas a identidade do próximo roteador

Apenas destinatário acessa mensagem M

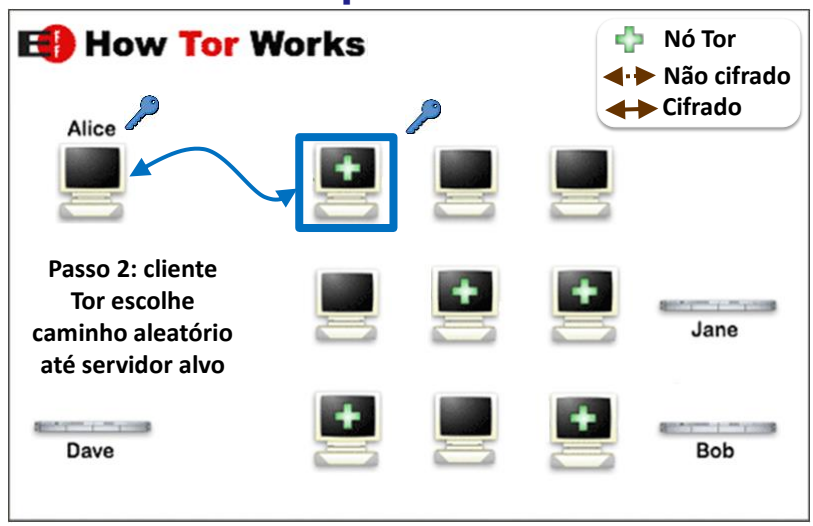
Roteamento com privacidade: Tor

- Segunda geração do onion routing
  - http://tor.eff.org
  - Desenvolvido by Roger Dingledine, Nick Mathewson and Paul Syverson
  - Projetado especificamente para comunicações na Internet que requerem baixa latência
- Ativo desde Outubro de 2003
  - Diversos nós espalhados pelo mundo todo
  - Milhares de usuários
  - Clientes de "fácil uso" (plugins, Tor Browser)
  - Navegação anônima e gratuita

Roteamento com privacidade: Tor

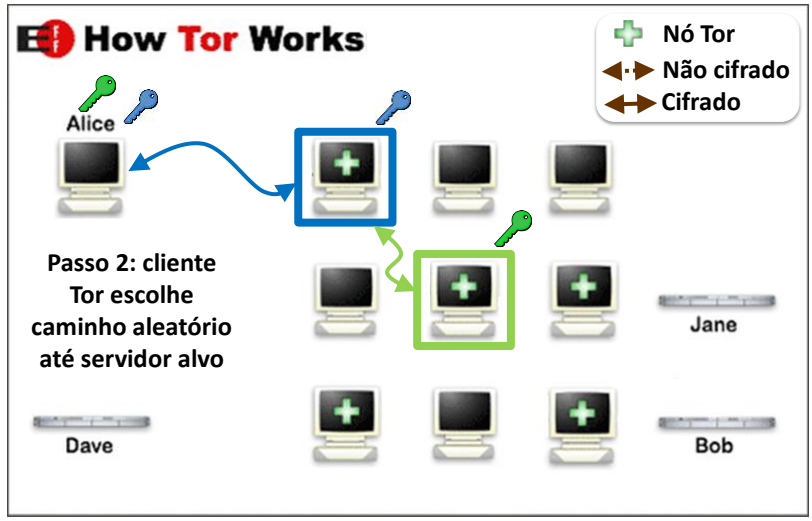


Roteamento com privacidade: Tor



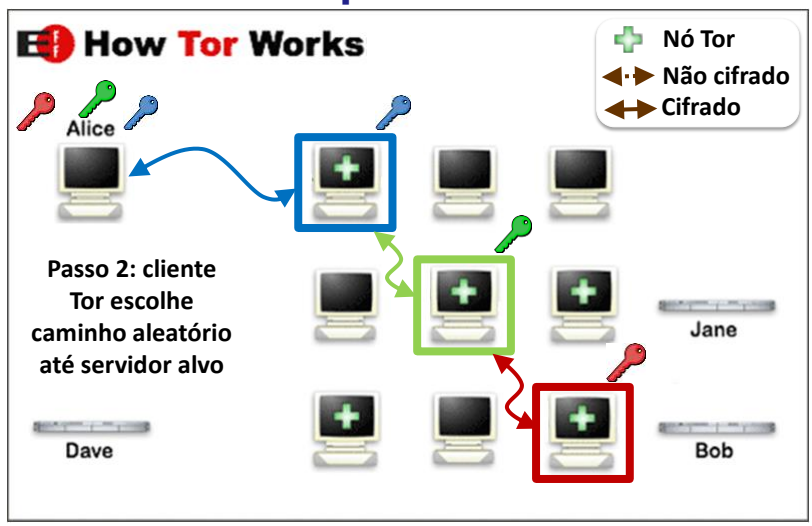
Chave simétrica de sessão estabelecida entre cliente Onion Router #1: circuito inicial

Roteamento com privacidade: Tor



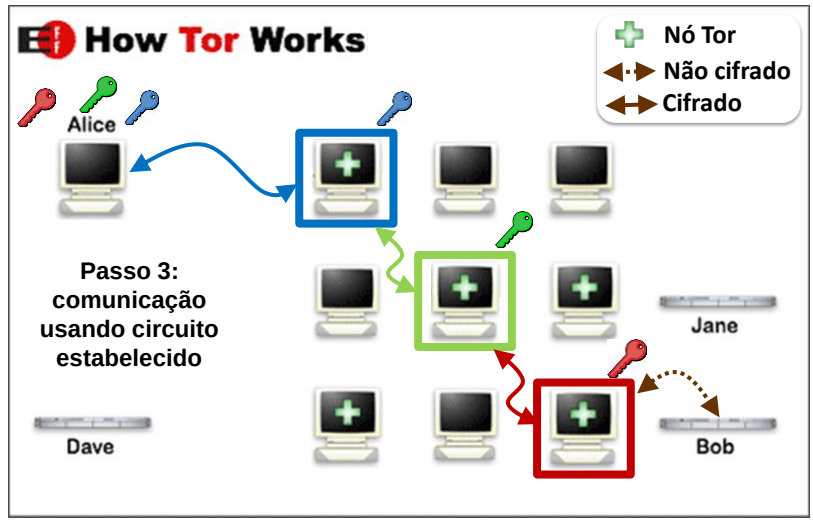
Cliente estende circuito, estabelecendo nova **chave simétrica** de sessão com **Onion Router #2** (tunelamento via onion router **#1**)

Roteamento com privacidade: Tor



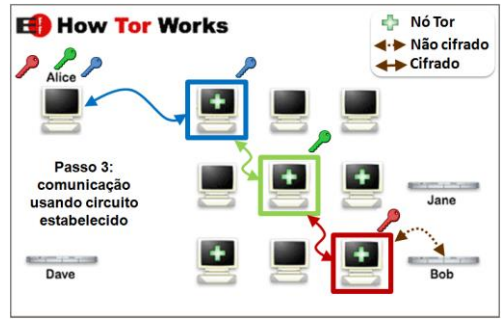
Cliente estende circuito, estabelecendo nova **chave simétrica** de sessão com **Onion Router #3** (tunelamento via onion routers **#1** e **#2**)

Roteamento com privacidade: Tor

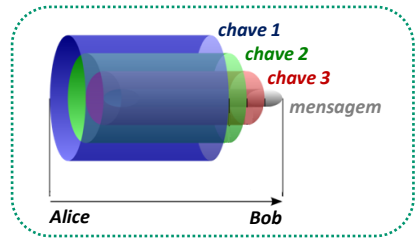


Cliente acessa destino final via circuito estabelecido: destino vê apenas Onion Router #3

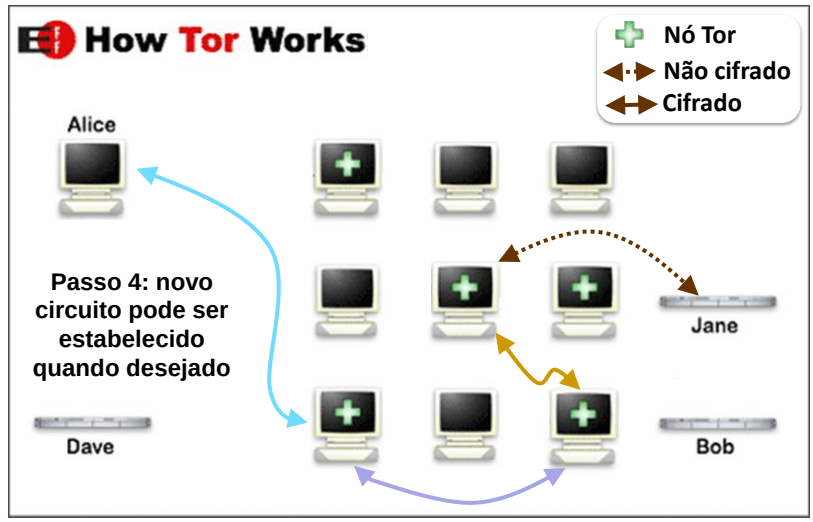
Roteamento com privacidade: Tor



Formato das mensagens Tor saindo da origem:



Roteamento com privacidade: Tor



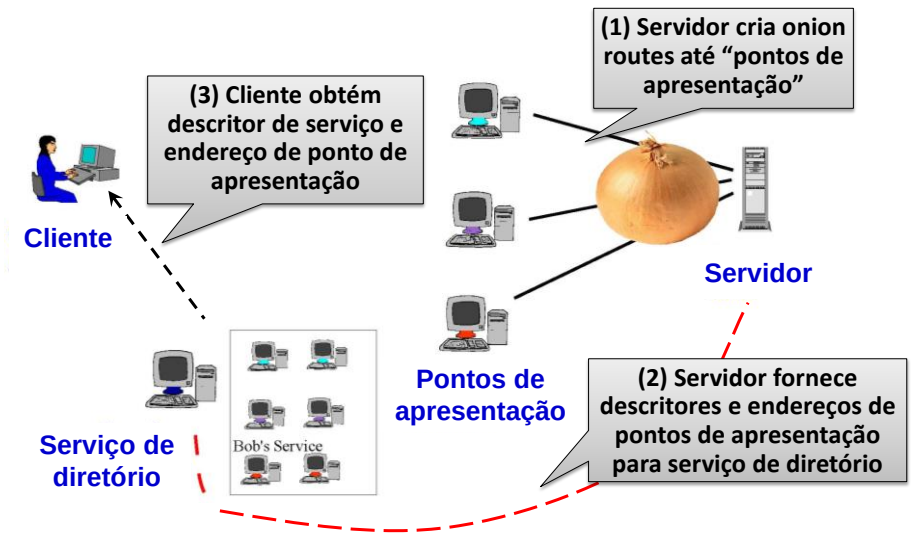
Tor: características

- "Perfect Forward secrecy"
  - No onion routing, um nó poderia gravar mensagens e depois comprometer a chave privada de todos os nós até o destino
  - No Tor, são criadas chaves de sessão, que são removidas após uso e, assim, não podem ser comprometidas
- Alguns serviços:
  - Diversos fluxos TCP podem **compartilhar** um mesmo circuito
  - Verificação de **integridade dos dados** antes da saída da rede
  - Nós confiáveis atuam como **servidores de diretório**: listas de roteadores conhecidos assinadas digitalmente
  - Pontos de encontro (rendezvous) e **serviços escondidos**: anonimato dos servidores

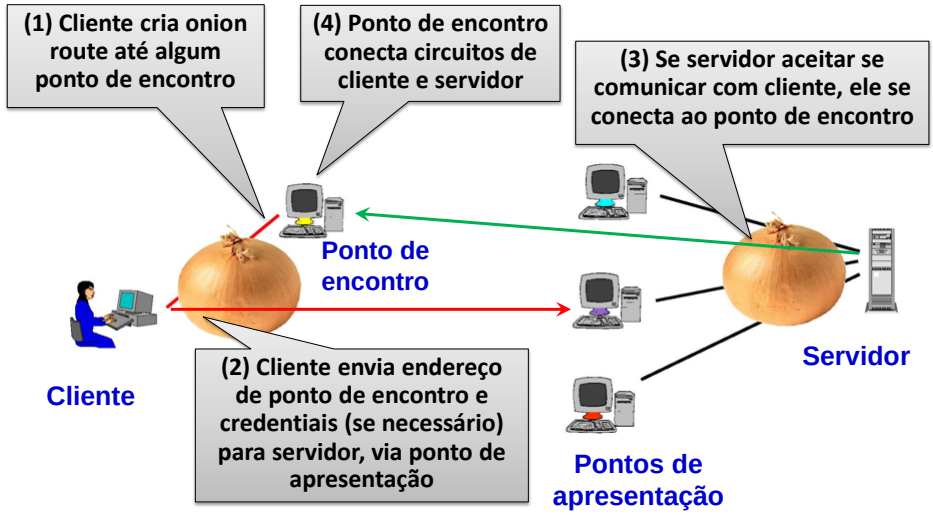
Tor: Servidores com localização oculta

- Objetivo: servidor na Internet com as seguintes características:
  - **Disponibilidade**: acessível por qualquer pessoa de qualquer lugar
  - **Privacidade**: pessoas que acessam não sabem onde servidor está ou quem o controla
- Resultado: servidor resistente a censura
  - Capaz de resistir a ataques de negação de serviço: serviço distribuído
  - Resistente a captura física: não se sabe onde está o servidor físico!

Tor: criando servidores com localização oculta



Tor: usando servidores com localização oculta



Ataques à rede Tor

- Ataques passivos
  - Não é tão difícil saber se um nó está executando protocolo Tor: o difícil é saber com quem ele está se comunicando
- Ataques ativos
  - DDoS, controle de um nó da rede Tor
- Ataques aos diretórios
  - Destruição ou subversão de servidores de diretório
- Pontos de encontro
  - Ataque a pontos de encontro ou pontos de apresentação

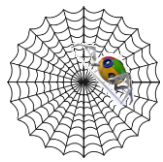
Tor, Web e "Deep Web": Indexação

- **Web Crawling**: **indexação** automática de páginas Web
  - Usado, por exemplo, para construir a base de dados de **sites de busca**
- **Web crawlers** (ou Web spiders): programas de computador que **automatizam indexação**
  - Visitam páginas e indexam texto visível e metadados
  - Seguem hiperlinks encontrados, continuando "navegação" pela Web e descobrindo novos sites
  - Podem executar indefinidamente, identificando modificações em páginas já visitadas.

Tor, Web e "Deep Web": Indexação

- "Deep web": conteúdo web não indexado por máquinas de busca tradicionais
  - Conteúdo **gerado dinamicamente** (ex.: via Javascript)
  - Conteúdo para o qual **não existem links** em sites já indexados
  - Conteúdo em **sites privados ou de acesso restrito**
    - Exigem login ou acesso via canal específico (ex.: redes Tor ou Freenet)
  - Texto embutido em **arquivos multimídia**

"surface web"

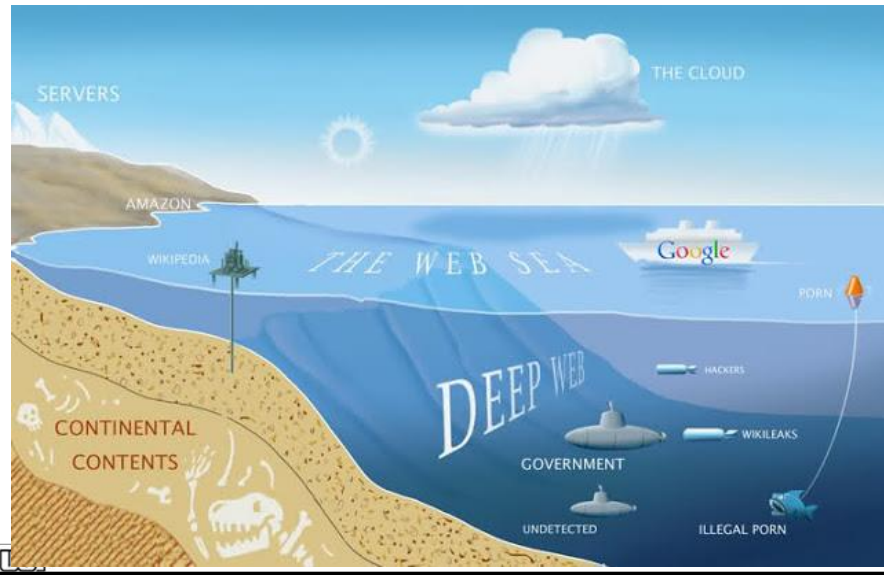


(parte da) "deep web"

não é lido por web crawlers

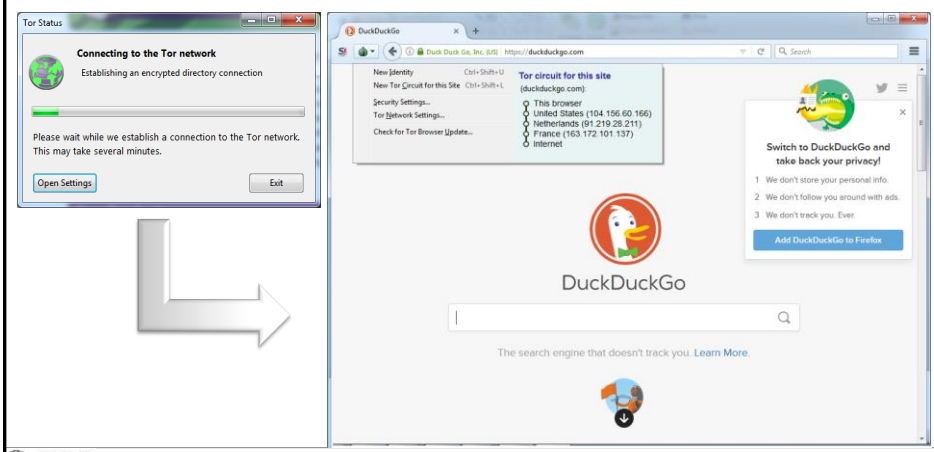
Tor, Web e "Deep Web": Indexação

Termo às vezes usado (de modo simplista) como "conteúdo ilegal"



Tor Browser: teste você mesm@!

- Página oficial: https://www.torproject.org/
  - Ex. de "hidden wiki": https://thehiddenwiki.org/



Aplicações P2P

Sistemas de Arquivos

Problemas com o HTTP

- Centralizada em servidores, que podem ser desligados
  - **Conteúdo** acaba sendo **perdido**, proposital ou acidentalmente
- Cria dependência de alguns serviços essenciais
  - Buscas: Google pode controlar o que usuários encontram
  - Hosting: Facebook/Amazon pode controlar o que usuários armazenam/veem
  - Resultado: censura, espionagem, direcionamento de opiniões, possibilidade de interrupção de serviços…
- Ineficiente: servidor de conteúdo popular torna-se gargalo
  - Escolha entre lentidão ou contratação serviços de caching…
- Voltado a locais, não a conteúdos:
  - Links quebram se local for alterado (404 Not Found)

IPFS: InterPlanetary File System

- Projeto criado em 2015, por Juan Benet
  - Um sistema de arquivos versionado, distribuído globalmente
  - "Uma web permanente e distribuída": similar a um enorme swarm bittorrent para troca de objetos versionados
  - Permite a construção de sites sem um servidor correspondente: "servidor" distribuído na rede!
- Links:
  - Página oficial: https://ipfs.io/
  - White paper: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf
  - Um pequeno experimento: https://linuxroot1.github.io/IPFS/

IPFS: Arquitetura

- Combina diferentes tecnologias

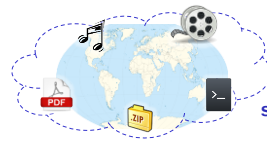
Fonte: https://github.com/ipfs/specs/tree/master/architecture

IPFS: Arquitetura (roteamento)

- Roteamento descentralizado, via **DHT**
  - Se dados pequenos (<1KB): dados armazenados na DHT;
  - Caso contrário, DHT armazena referência para dados (IPs de nós que podem fornecer dados)
- Algoritmos:
  - **S/Kademlia**: estrutura em árvore para busca eficiente
    - Vide **apêndice**
  - **Coral**: considera **localidade** dos dados para melhorar eficiência das buscas
    - Clusters organizados por região e tamanho
    - "Busca(chave)" retorna subconjunto de IPs que têm o conteúdo em vez de lista completa ("Sloppy DHT")

IPFS: Arquitetura (troca de arquivos)

- Baseado no **BitSwap**
  - similar a **BitTorrent**, mas com **um único swarm** com todos os conteúdos
- Mecanismos de **incentivo**:
  - **Tit-for-tat**: pode obrigar nó A a buscar pedaços que nó B deseja para que então possa receber pedaços vindos de B
  - Nó A nantém lista de "**credores**" (nós que mandaram mais dados para A do que receberam de A) e tenta pagar débito
  - Também pode usar ***distributed ledger*** para gerenciar créditos



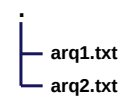
IPFS: Arquitetura (versionamento – GIT)

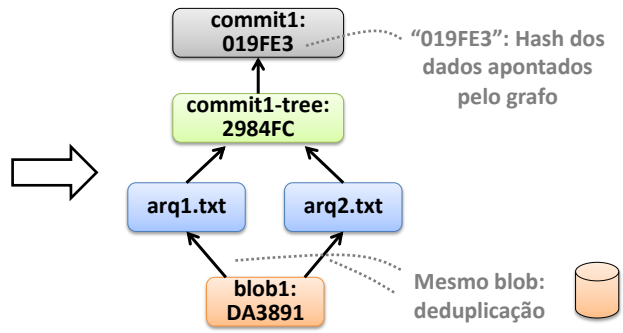
- Usa Merkle ~~tree~~ DAG (Grafo Direcionado Acíclico)

IPFS: Arquitetura (versionamento – GIT)

- Usa Merkle DAG (Grafo Direcionado Acíclico)
  - Commit: autor, mensagem, ponteiro (hash) para uma árvore
  - Tree: ponteiro para árvores e arquivos (estrutura de pastas)
  - Blob: dados

Ex.: dois arquivos idênticos na raíz

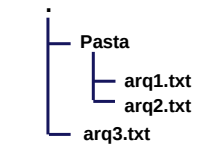


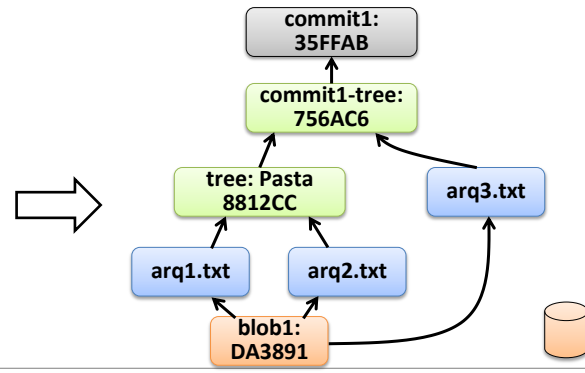


IPFS: Arquitetura (versionamento – GIT)

- Usa Merkle DAG (Grafo Direcionado Acíclico)
  - Commit: autor, mensagem, ponteiro (hash) para uma árvore
  - Tree: ponteiro para árvores e arquivos (estrutura de pastas)
  - Blob: dados

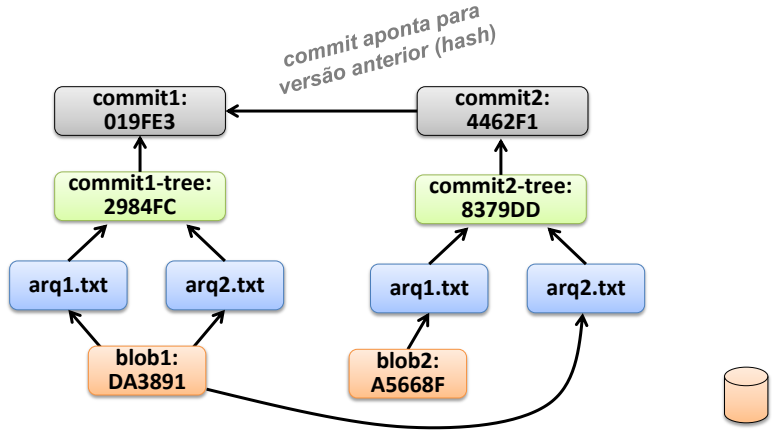
Ex.: três arquivos idênticos, em pastas





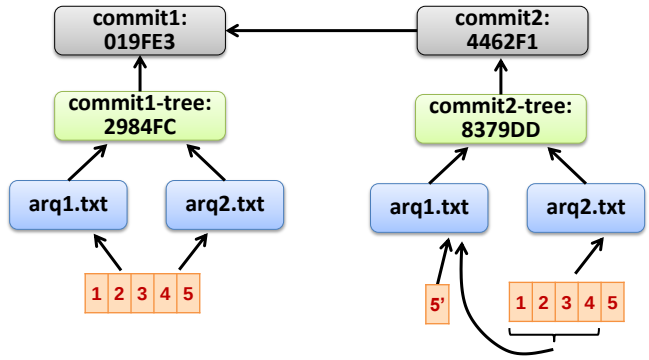
IPFS: Arquitetura (versionamento – GIT)

- Controle de versão via hashes
  - Permite acompanhar o caminho das alterações dos arquivos



IPFS: Arquitetura (versionamento – GIT)

- Controle de versão via hashes
  - Permite acompanhar o caminho das alterações dos arquivos
  - Se blobs quebrados em blocos: **deduplicação** mais efetiva



IPFS: Arquitetura (versionamento – GIT)

- **Objetos** no IPFS: todos identificados pelo seu hash, tanto se forem **arquivos** ou **links**

  - **Multihash** para suporte a diferentes algoritmos
    - Formato: `<Algoritmo><Tamanho_Hash><Bytes_Hash>`
  - Navegação entre links de um domínio = navegação no Merkle DAG, usando o hash de cada link

Formato: `/ipfs/<hash-of-object>/<name-path-to-object>`

Ex.: `/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/arq.txt`

  - Para acessar o arquivo "fig.png" localizado no caminho "<dominio>/pasta/fig.png, pode-se usar qualquer das opções:

1) Domínio: `/ipfs/<hash-de-dominio>/pasta/fig.png`

2) Pasta: `/ipfs/<hash-de-pasta>/fig.png`

3) Arquivo: `/ipfs/<hash-de-fig.png>`

IPFS: Arquitetura (versionamento – GIT)

- Qualquer usuário pode publicar objetos na rede
  - Basta incluir hash do objeto na DHT, se declarar como um peer para objeto e publicar caminho para objeto
- Uso de Merkle DAGs: hash não pode ser alterado, logo **objetos são permanentes**!
  - Redução do consumo de banda: **caching**
  - Conteúdo servido por nós **sem confiança**: análogo ao que ocorre no Bittorrent
  - **Links são permanentes**: sem "links quebrados" desde que alguém tenha arquivo
  - Usuários podem escolher fazer backups de **dados específicos** para garantir sua **longevidade**

IPFS: Arquitetura (sistema de nomes)

- SFS (Self Certified Filesystem)

  - Identificadores dos nós (NodeId) correspondem ao hash de suas chaves públicas
    - NodeID usado para roteamento de buscas
    - Dificulta "escolha do ID" para eventuais ataques
  - Totalmente descentralizado: basta servidor provar posse da chave privada para verificar corretude
    - Permite troca de chaves para estabelecer canais seguros
    - Similar a HTTPS, mas sem depender de certificados digitais emitidos por Autoridade Certificadora

Formato: `/ipns/<NodeID>`

Ex.: `/ipns/hj17rsy89MnOo`

IPFS: Arquitetura (sistema de nomes)

- IPNS (Name Space): **mutabilidade** de objetos
  - Nós pode associar seu domínio /ipns/<NodeId> a um objeto
  - Busca na DHT por domínio retorna versão atual do objeto (seu hash), que pode ser modificado quando desejado
    - Assinatura do servidor sobre objeto garante autenticidade
    - Objeto pode ser commit de página web completa: carrega histórico de versões!

	NodeID	Valor	
mesma chave	**/ipns/**hj17rsy89MnOo	98sdfHjyu87q	conteúdos distintos
	/ipns/hj17rsy89MnOo	3dfvJ3KdgYr	

IPFS: Arquitetura (sistema de nomes)

- IPNS (Name Space): mutabilidade de objetos
  - Facilita nomes amigáveis a humanos, no lugar de hashes
  - Ex.: criar registro DNS (e.g., blockchaindev.blog) p/ nodeID
    - blockchaindev.blog: `"dnslink= /ipns/hj17rsy89MnOo"`

Serviço centralizado ou distribuído (Namecoin, Blockstack, …)

DNS	NodeID	Valor
/ipns/blockchaindev.blog	**/ipns/**hj17rsy89MnOo	98sdfHjyu87q
/ipns/blockchaindev.blog	**/ipns/**hj17rsy89MnOo	3dfvJ3KdgYr

Leitura: https://decentralized.blog/ten-terrible-attempts-to-make-ipfs-human-friendly.html

IPFS: Teste você mesmo!

- Solução ainda razoavelmente experimental
  - Mas "mão na massa" ajuda a entender funcionamento!

https://ipfs.io/docs/getting-started/

Apêndice

P2P: Questões de segurança

Segurança em Sistemas P2P: ameaças

- Necessidade de proteção contra diversas ameaças.
- Ataques de roteamento
  - Redirecionamento de buscas na direção errada, ou para nós que não existem
  - Atualizações enganosas
  - Particionamento da rede
- Ataques de armazenamento/recuperação de dados
  - Nó responsável por armazenamento de um certo dado não o armazena ou não o fornece conforme deveria
- Comportamento inconsistente
  - Os nós às vezes se comportam corretamente, outras vezes não

Segurança em Sistemas P2P: ameaças (cont.)

- Falsificação de identidades na rede
  - Nó afirma ter uma identidade que na realidade pertence a outro nó
  - Nó fornece conteúdo falso para a rede fingindo ser outro nó
- Ataques de (D)DoS
  - Sobrecarga de um nó com operações (buscas, envio de dados, etc.)
  - Solução costuma envolver mecanismos de autenticação
- Entradas/saídas rápidas na rede
  - Afetam disponibilidade do sistema
- Problemas de segurança surgem em razão da presença de nós maliciosos
  - Necessidade de avaliar e gerenciar confiança dos nós

Segurança em Sistemas P2P: Mecanismos

- É necessário prover um nível adequado de disponibilidade, privacidade, confidencialidade, integridade e autenticidade.
- Desafios:
  - Armazenamento seguro
  - Roteamento seguro
  - Controle de acesso, autenticação e gerenciamento de identidades
  - Anonimato
  - Gerenciamento de reputação e confiança

Segurança em Sistemas P2P: Mecanismos

- **Armazenamento seguro**: algoritmos e protocolos criptográficos para garantir a segurança do conteúdo da rede
  - Uso de funções de hash para verificação de integridade
  - Redundância e dispersão de informações (disponibilidade)
- **Roteamento seguro**: depende de três fatores
  - Segurança na atribuição de IDs aos nós
  - Segurança na manutenção das tabelas de roteamento
  - Segurança no encaminhamento de mensagens

Segurança em Sistemas P2P: Mecanismos

- **Controle de acesso, autenticação e gerenciamento de identidades**: comumente ignorado, mas essencial para prevenir ataques de roubo de identidades, (D)DoS, etc.
  - Autoridade central de autenticação/identificação
  - Chaves de acesso e listas de controle de acesso
  - Respeito a direitos de propriedade intelectual

- **Anonimato**: alguém pode identificar o autor, o leitor, ou o responsável pela publicação de um documento, ou mesmo o próprio documento?
  - Mecanismos de cifração, ofuscação do roteamento, etc.

Segurança em Sistemas P2P: Mecanismos

- **Gerenciamento de reputação e confiança**: prevenção e punição de comportamento malicioso
  - "**Free riding**": peers que usam os recursos da rede sem fornecer recursos para ela
  - **Conluio** entre peers: tentativa de aumentar a reputação uns dos outros e possivelmente reduzir a de peers honestos
  - **Poluição de conteúdo**: inserção de conteúdos falsos na rede, que podem acabar sendo espalhados por peers honestos que não verificam imediatamente o conteúdo

    - KaZaa: reportou poluição em 80% de seus arquivos (2005)
    - Excesso de arquivos poluídos pode inutilizar o sistema
    - Usado por detentores de direitos autorais como tática anti-pirataria

Apêndice

Tabelas de Hash Distribuídas (DHTs): Chord, (S/)Kademlia

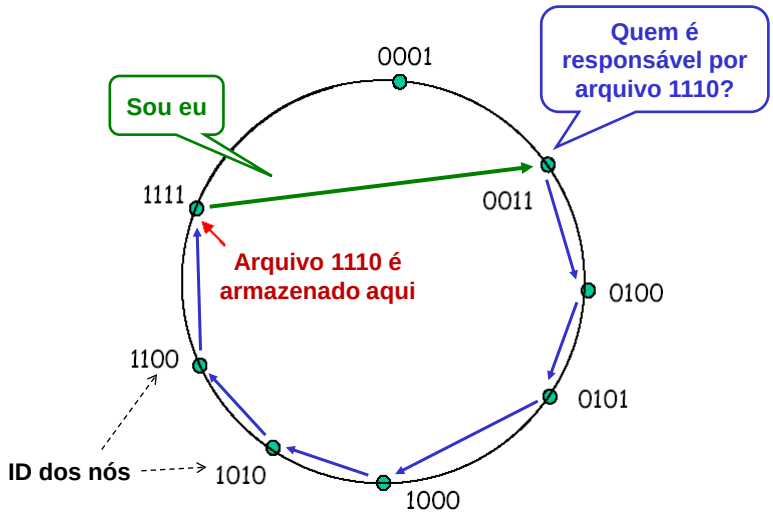
Abordagens de DHT

- Hashing consistente
- Chord
- CAN (Content Addressable Network)
- Pastry
- Tapestry
- Kademlia

Hashing consistente (cont.)

- Problema: associar arquivo a nó na rede
- Ideia: calcular hash **h** do arquivo (chave) e associá-lo ao nó "mais próximo" de **h**, dada uma certa métrica de proximidade
- Uma abordagem possível: rede overlay é um círculo
  - Cada nó tem um ID escolhido aleatoriamente: hash (end. IP)
  - Chaves no mesmo espaço de IDs
  - Sucessor de nó A: nó cujo ID seja imediatamente maior que ID_A
  - Cada nó sabe IP de seu sucessor

Hashing consistente (cont.)



Hashing consistente (cont.)

- Funcionamento básico

Saída de um nó	Entrada de um nó
• Se o sucessor de A deixa a rede, A precisa escolher próximo sucessor • **Logo:** cada nó mantém referência para **s** ≥ 2 sucessores • Quando o sucessor de A deixa a rede, A pede a seu novo sucessor pela sua lista de sucessores; A então atualiza sua própria lista de **s** sucessores	• Você é um novo nó, e seu ID é **k** •Peça a qualquer nó **N** para encontrar o nó **N'** que é sucessor de **k** • Obtenha sua lista de sucessores de **N'** •Diga a seus predecessores que atualizem suas listas de sucessores • **Logo:** cada nó deve saber quem é seu predecessor

- #vizinhos = s+1 = O(1)
  - Tabela de roteamento: (ID_vizinho, IP_vizinho)
- Número médio de mensagens para encontrar chave: O(n)

Podemos fazer melhor?!

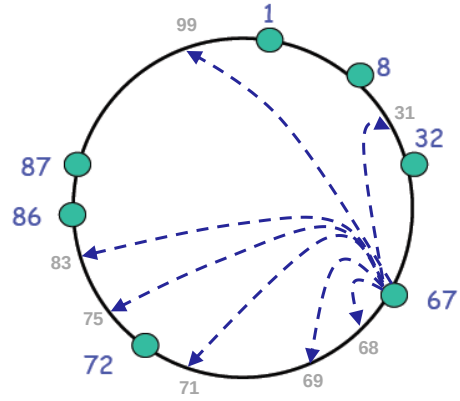
Chord

- Baseado em hashing consistente
  - Overlay circular
  - ID aleatório unidimensional no espaço de hashes
  - Domínio:] ID_anterior , ID_próprio] (mod |espaço de IDs|)
- Tabela de derivação (*finger table*)
  - Conjunto de vizinhos conhecidos
  - O i-ésimo vizinho (sentido horário) do nó de ID **n** tem o ID mais próximo de (e é maior que) $\mathbf{n+2^i}$ (mod |espaço de IDs|), i ≥ 0
- Roteamento
  - Para alcançar o nó responsável pelo ID **n'**, envie a mensagem para o vizinho número $\mathbf{\log_2(n' - n)}$

Chord (cont.)

- Finger table para nó 67 ➔ n=67
- Espaço de IDs de [0, 99] ➔ N = 100

i	$(n+2^i) \bmod N$	Finger table
0	68	
1	69	
2	71	
3	75	
4	83	
5	99	
6	31	
~~7~~	~~95~~	x



Chord (cont.)

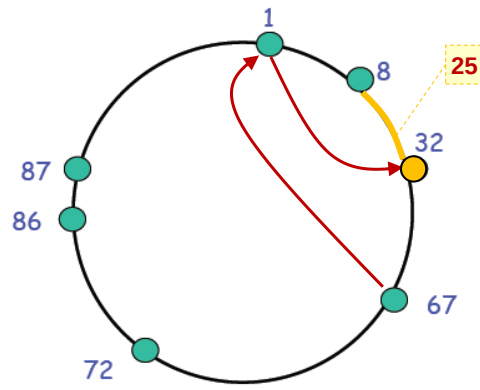
- Roteamento: busca de K = 25
  - **Nó 67**: destino = $\log_2((25 - 67) \bmod 100) = \log_2(58) \sim 5$
  - **Nó 1**: destino = $\log_2((25 - 1) \bmod 100) = \log_2(24) \sim 4$

n = 67

i	Finger table
0	72
1	72
2	72
3	86
4	86
5	1
6	32

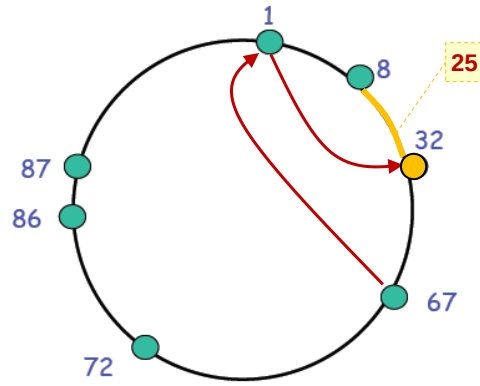
n = 1

i	Finger table
0	8
1	8
2	8
3	32
4	32
5	67
6	67



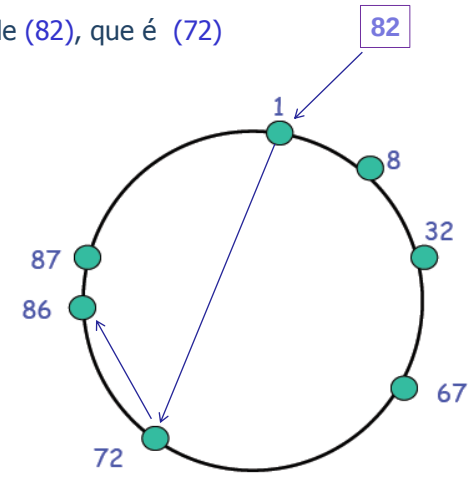
Chord (cont.)

- Roteamento: qualquer nó pode ser alcançado a partir de qualquer outro nó em até log2(N)
  - N: número máximo de nós da rede



Chord (cont.)

- Entrando na rede:
  - Nó (82) conhece nó (1), obtido fora de banda
  - (1) encontra predecessor de (82), que é (72)

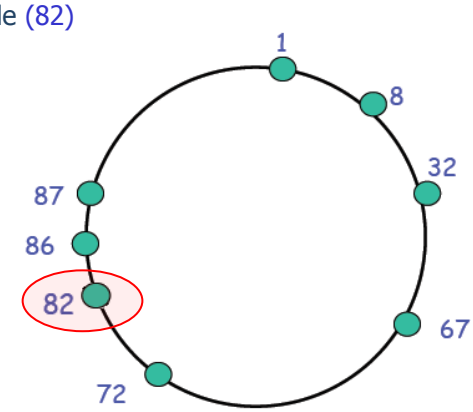


Chord (cont.)

- Entrando na rede:
  - Nó (82) conhece nó (1), obtido fora de banda
  - (1) encontra predecessor de (82), que é (72)
  - (72) constrói finger table de (82)

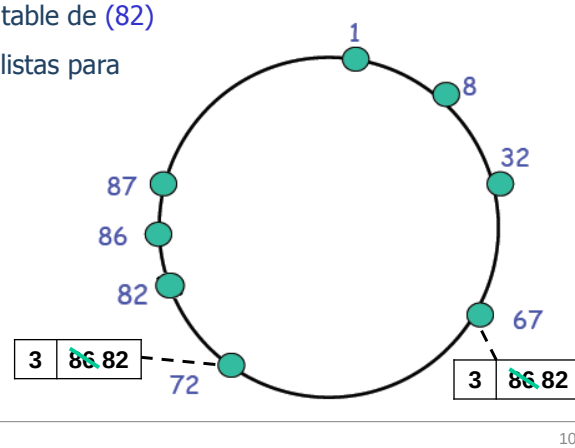
n = 82

i	$(n+2^i) \bmod N$	Finger table
0	68	72
1	69	72
2	71	72
3	75	86
4	83	86
5	99	1
6	3	8



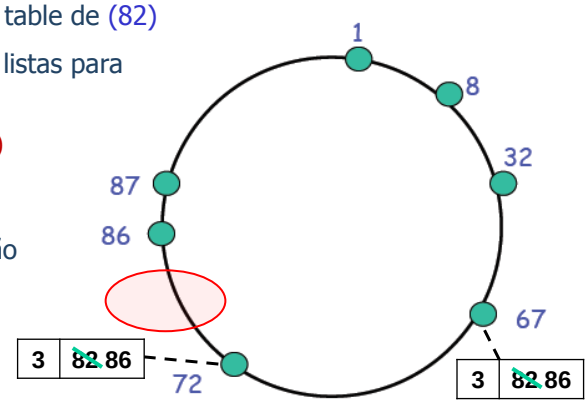
Chord (cont.)

- Entrando na rede:
  - Nó (82) conhece nó (1), obtido fora de banda
  - (1) encontra predecessor de (82), que é (72)
  - (72) constrói finger table de (82)
  - Nós atualizam suas listas para considerar (82)
  - Atualização: $\log_2(N)$



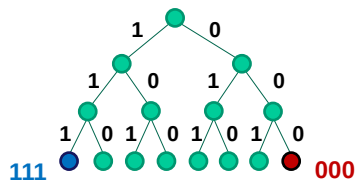
Chord (cont.)

- Entrando na rede:
  - Nó (82) conhece nó (1), obtido fora de banda
  - (1) encontra predecessor de (82), que é (72)
  - (72) constrói finger table de (82)
  - Nós atualizam suas listas para considerar (82)
  - Atualização: $\log_2(N)$
- Saindo da rede:
  - Atualização após não receber resposta a mensagem de "keep-alive"



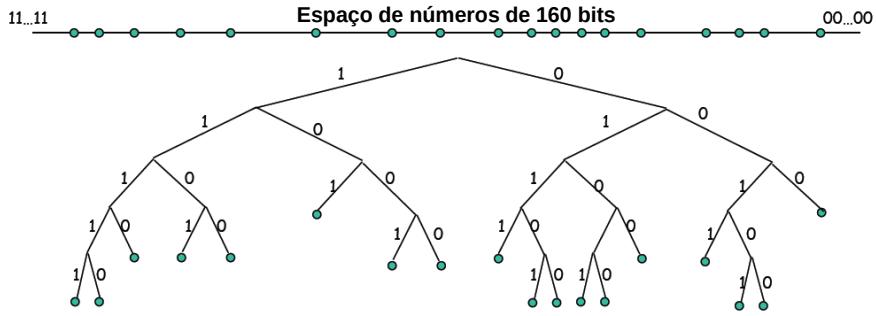
Kademlia

- Algoritmo de DHT criado em 2002
  - E atualmente um dos algoritmos de DHT mais utilizados por algoritmos P2P, como links magnéticos
- Estrutura baseada em uma árvore binária
  - Organização dos nós e chaves dos arquivos



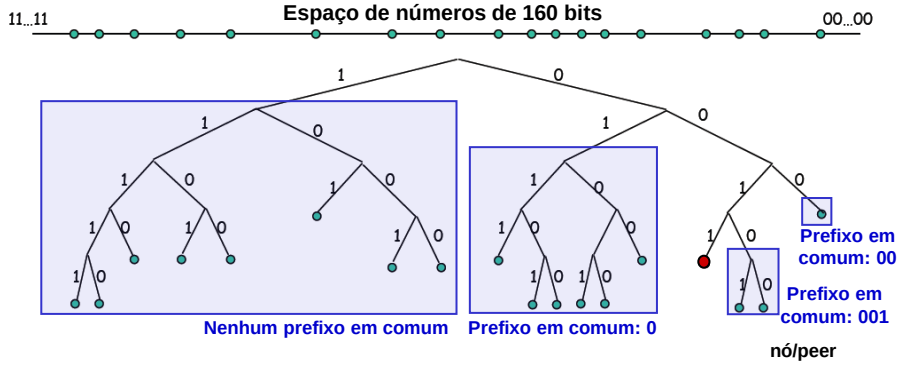
- S/Kademlia: ID é a chave pública do nó

Kademlia



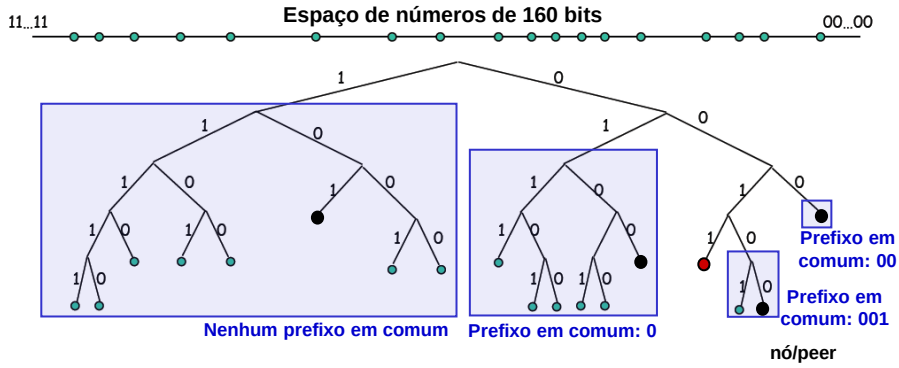
- Nós: folhas em uma árvore binária
  - Posição determinada pelo menor prefixo exclusivo de seu ID
  - Nó responsável por dados com IDs mais "próximos" a ele
  - Distância entre ID x e y: d(x,y) = x ⊕ y ➔ d(0101, 0011) = 0110_2 = 6_{10}
  - Nós/IDs na mesma sub-árbore têm maior prefixo em comum ➔ estão mais próximos

Kademlia: distribuição de IDs



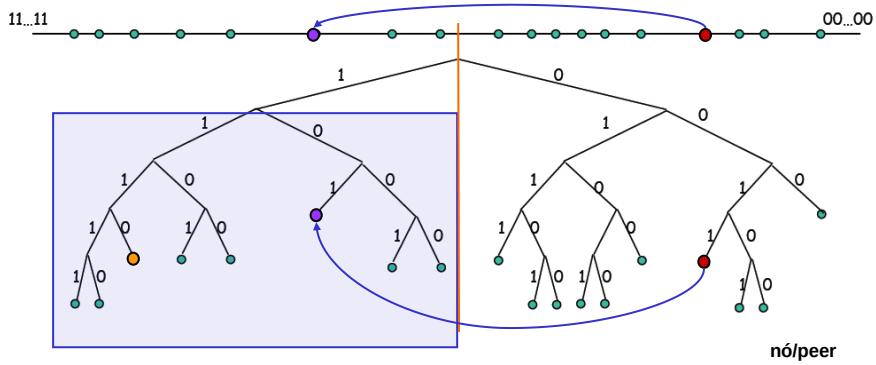
- Para um nó qualquer (ex.: nó 0011):
  - A árvore binária é dividida em sub-árvores de tamanho máximo que não contêm o nó

Kademlia: distribuição de IDs



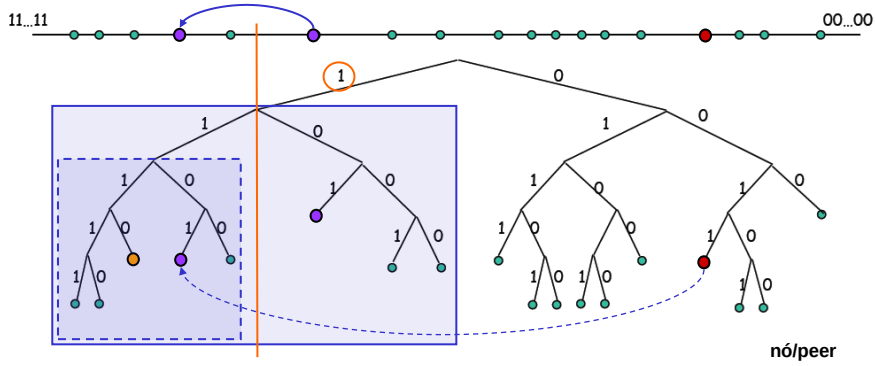
- Para um nó qualquer (ex.: nó 0011):
  - Um nó deve conhecer pelo menos um outro nó em cada uma destas sub-árvores (ex.: nó 0011 conhece nós em **preto**).
  - S/Kademlia: escolha de nós maximiza conectividade mesmo na presença de vários nós maliciosos

Kademlia: roteamento de busca



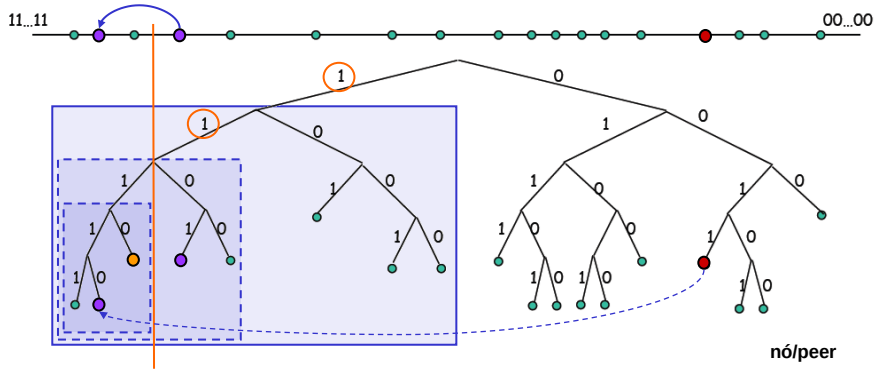
- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
  - Nenhum prefixo em comum: ele contacta nó na árvore correspondente (com prefixo 1: **101**...) que roteia a busca

Kademlia: roteamento de busca (cont.)



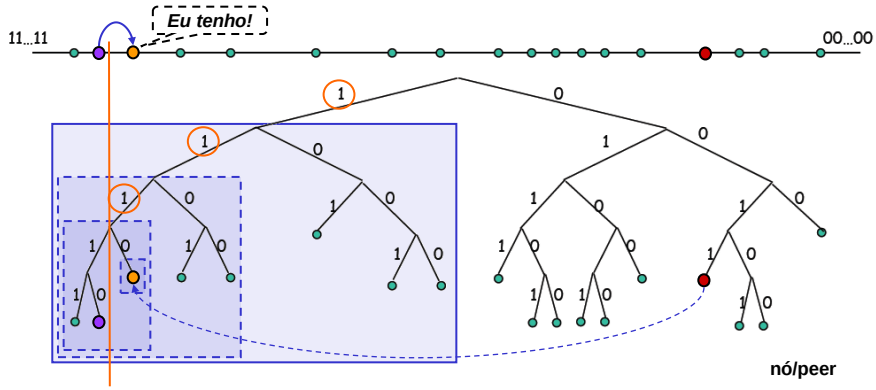
- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
  - Processo se repete até nó correto ser encontrado: nó **1101**...

Kademlia: roteamento de busca (cont.)



- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
  - Processo se repete até nó correto ser encontrado: nó **11110**...

Kademlia: roteamento de busca (cont.)



- Considere que o nó **0011100...** está procurando um dado cujo ID é **111010...**
  - Processo se repete até nó correto ser encontrado: nó **1110**...

Apêndice

Google File System & Hadoop

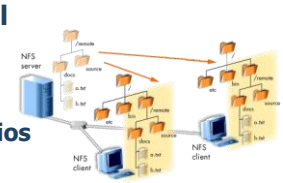
"Big Data": Google File System (GFS)

- **MapReduce** requer **sistema de arquivos** adequado
  - Distribuição de dados, tolerância a falhas, redundância …
- Para atacar problema, o Google desenvolveu o **GFS**
  - Projetado para necessidades e cargas de trabalho do Google
  - Linguagens de programação principais: C++ (base) ou Sawzall (otimizada para MapReduce)
    - Sawzall: código MapReduce ~10x menor que equivalente em C++
  - **Gerenciamento** automático de **tarefas MapReduce** por ferramenta Workqueue

"Big Data": Google File System (GFS)

- Por que não usar sistemas de arquivos existentes?
  - **"Big Data": problemas** neste cenário **são diferentes** dos encontrados no compartilhamento de arquivos por usuários
  - Logo, bom **desempenho** requer algumas "personalizações"

- Por exemplo: Network File System (**NFS**)
  - Sistema de arquivos de **propósito geral**
  - Cenário de uso comum: compartilhar arquivos entre vários usuários
    - **Leituras/escritas em pontos aleatórios**
    - **Muitos arquivos, em geral pequenos**
    - **Sem suporte a redundância** (provida externamente, se preciso)



"Big Data": Cenário do Google/GFS

- Alta taxa de **falhas**

  - Grande número de componentes de hardware baratos (ex.: discos magnéticos), que comumente acabam falhando
- **Número modesto** de **arquivos gigantescos**

  - Alguns milhões de arquivos tendo 100 MB ou mais (arquivos de GB são comuns)
- **Acesso** aos arquivos é "**atípico**"

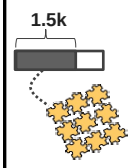
  - Múltiplas **leituras sequenciais**, seguidas de operações de **anexar** ao final do arquivo (ex.: MapReduce!)
  - **Leituras curtas e aleatórias** ocorrem, mas **são raras**

- **Alta vazão** é mais importante do que baixa latência

"Big Data": Projeto do GFS

- Arquivos armazenados em **"chunks" grandes**

  - Tamanho fixo: **64 MB**

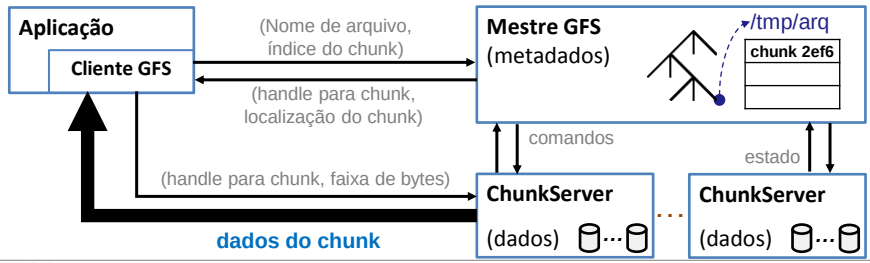


    - 64TB de dados ➔ 1 milhão de chunks
    - Isso **facilita leitura sequencial**
  - No NFS: o tamanho típico é **2k-8k**
    - **Tabela de arquivos grande**, armazenada em disco: menor desempenho para ler grandes volumes de dados
- **Sem caching** no cliente

  - Traria poucas vantagens dado o tamanho dos dados e leitura sequencial
  - No NFS: sincronização entre caches locais nos clientes aumenta congestionamento na rede.

"Big Data": Projeto do GFS (cont.)

- Confiabilidade de dados via **replicação de chunks**
  - Cada chunk armazenado em **3 ou mais ChunkServers**
  - Um único **mestre mantém metadados com localização dos chunks**: controla acesso e permite visão global da rede
  - **Tabela de metadados é pequena** (poucos chunks): pode ficar na **memória principal**, acelerando acesso



"Big Data": Projeto do GFS (cont.)

- Confiabilidade de dados via replicação de chunks (cont.)
  - Mestre deve garantir número de chunks replicados:

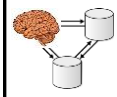
    - Distribuir **nova réplica** quando um ChunkServer **falha**
    - **Remover réplicas** excedentes quando ChunkServer é restabelecido
  - Mestre define ChunkServer "primário" para interagir com clientes

    - Em caso de **alterações** em chunks: cliente contacta diretamente os ChunkServers com as réplicas, mas **primário controla quando alterações são aplicadas**.

"Big Data": Projeto do GFS (cont.)

- Um único mestre: simples, mas

  - **Ponto único de falha**
  - **Gargalo** para escalabilidade do sistema
- Soluções do GFS:



  - **Replicação** remota de log de operações do mestre ("shadows"): **restabelecimento rápido em caso de falha**
  - **Envolvimento mínimo do mestre**: após obter metadados, cliente não acessa mestre enquanto lê chunk (grande)
- Obs.: o Google vem trabalhando em solução com **mestre distribuído ("Colossus")**

  - Poucas informações públicas, exceto pela estrutura da base de dados "Spanner": hierarquia por zonas
    (http://research.google.com/archive/spanner-osdi2012.pdf)

"Big Data": Uso do GFS

- Basicamente todos os sistemas do Google…
  - Mais de 50 clusters, gerenciando petabytes de dados
- Dentre eles: **BigTable**

Google BigTable

  - Armazenamento distribuído de dados da ordem de terabytes
  - **Dados não estruturados**: armazenamento na forma de mapa indexado por <linha, coluna, timestamp>
  - Ordenação das chaves por linhas
  - Cada célula da tabela pode conter diversas instâncias de um mesmo arquivo, diferenciadas pelo timestamp
  - Admite uso de locks: serviço provido por ferramenta "Chubby"

"Big Data": Hadoop

- **GFS** é sistema proprietário, de **código fechado**
- **Hadoop** Distributed File System (HDFS): versão de **código aberto** do GFS
  - Obtido por **engenharia reversa** do GFS, feita pelo Yahoo!
  - Distribuído pela Apache
  - Linguagens de programação principais: **Java** (base) ou **Pig** (otimizada para **MapReduce**)
    - Pig: código MapReduce 20x menor, requer 16x menos tempo para programar e executa pouco mais lentamente do que Java.
  - Usa JobTracker para agendar tarefas de MapReduce e TaskTracker para executá-las