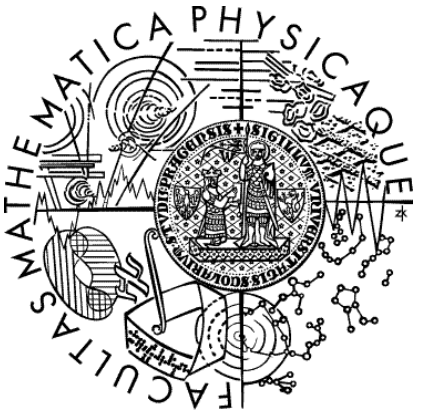
Peer-to-Peer sítě



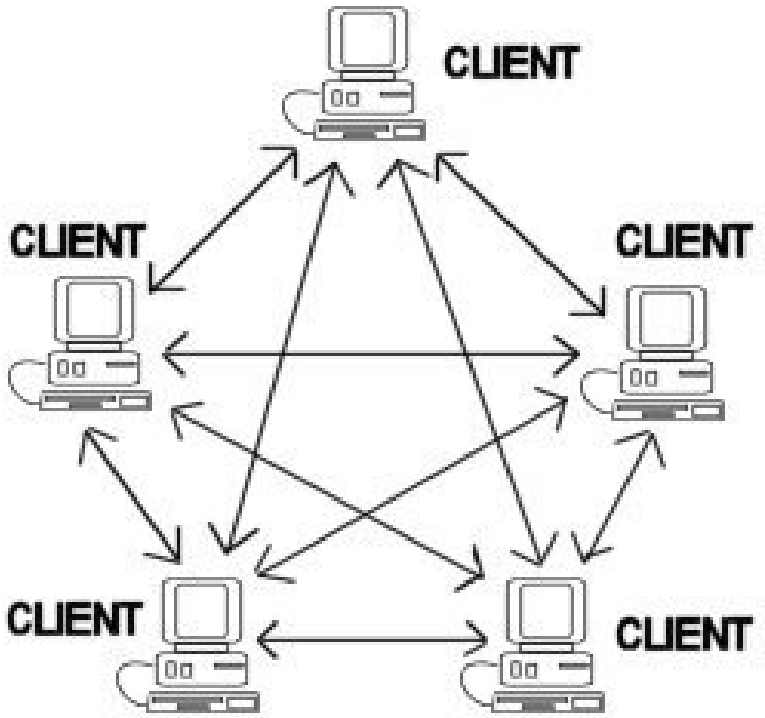
Mgr. Miroslav Novotný

Katedra Softwarového inženýrství

novotny@ksi.mff.cuni.cz

Peer-to-Peer sítě

- Označení architektury, ve které spolu přímo komunikují koncové uzly → Opak architektury klient-server.
- Všechny uzly mají stejnou funkci. Mohou vystupovat v roli klienta i serveru → Označení peer.
- Stávající uzel se může kdykoliv odpojit nebo se může připojit nový uzel → Dynamické změny v síti.

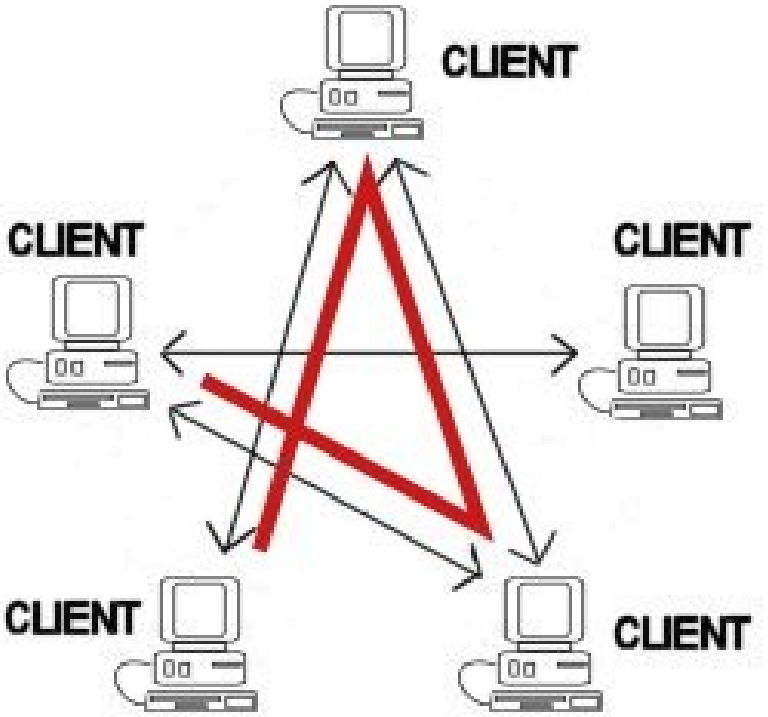


Motivace vzniku P2P

- Uzly sdílejí své prostředky
  - Čím více uzlů tím vetší je kapacita a výkonnost sítě.
  - V klient-server více klientů znamená snížení výkonnosti
- Eliminace single point of failure
  - Síť funguje i po výpadku libovolného počtu uzlů.
  - V klient-server výpadek serveru znamená nefunkčnost sítě.
- Minimalizace nákladů na poskytování služby
  - Poskytovatel nemusí investovat do budování serverové infrastruktury.

Dělení P2P sítí

- P2P systémy budují overlay networks nad stávající síťovou infrastrukturou.
  - Mezi některými uzly jsou virtuální spojení - linky.
  - Veškerá komunikace se odehrává po těchto linkách.
- Nestrukturované P2P sítě
  - Nemají pravidla pro vytváření linek mezi uzly.
- Strukturovaní P2P sítě
  - Linky mezi uzly jsou vytvářeny podle přesných pravidel.



Nestrukturované P2P sítě

- Linky mezi uzly jsou vytvářeny libovolně.

  ☺ Jednoduché připojení nového uzlu.

  ☹ Při vyhledávání se nelze opřít o žádnou strukturu.
- Vyhledávací algoritmy (z teorie grafů):
  - Breadth-first search (flooding)
  - *k*-Random Walk
  - Generic Adaptive Probabilistic Search

Žádný z algoritmů nezajistí, že existující data budou skutečně nalezena !

„Nečisté“ P2P Sítě

- Čisté P2P sítě
  - Podle úvodní definice.
- Hybridní P2P sítě
  - Některé uzly mají speciální funkci → supernode.
  - Soustřeďují na sebe traffic ostatních uzlů.
  - Poskytují extra služby pro některé další uzly.
- Centralizované P2P sítě
  - Centrální servery zajišťují některé funkce: bootstrap, indexování, vyhledávání.

File sharing networks

- Napster
  - První P2P síť.
  - Závislá na centrálním serveru – indexy.
- Gnutella, Gnutella2
  - Velké množství klientů.
  - Čistá P2P síť.
  - Později rozlišení uzlů na ultrapeers a leavers.

Strukturované P2P sítě

- Pravidla pro vytváření linek mezi uzly.
- Každý uzel má směrovací tabulky podle níž dokáže najít cestu k libovolnému klíči v síti.
  - ☺ Efektivní vyhledávání v síti.
  - ☹ Připojení/odpojení uzlů → změna směrovacích tabulek.
- Využívají distribuované hašovací tabulky (DHT).

Existující data budou vždy nalezena, a to v konečném počtu kroků !

Shrnutí P2P

	Strukturované P2P	**Nestrukturované P2P**
Směrování	Na základě směrovací tabulky	Záplavově, náhodná procházka, ...
Možnosti vyhledávání	Pouze podle klíče	Možnost pokládat složitější dotazy
Existující záznam je vždy nalezen	Ano	Není zaručeno
Kritické místo	Připojení/odpojení uzlu	Vyhledávání/směrování

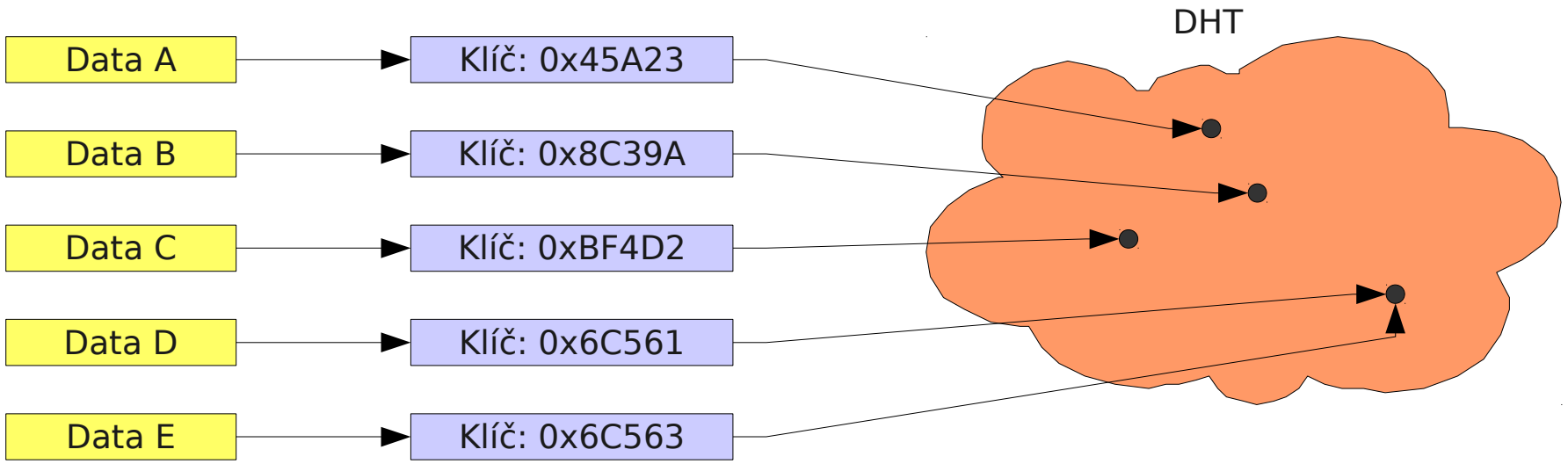
Distribuované hašovací tabulky

Klasické hašovací tabulky

- S – reprezentovaná množina ($S \subseteq U$)
- h – hašovací funkce ($h: U \rightarrow \{1,..,m\}$)
- A – pole velikosti m
- Prvek s uložíme do pole A na pozici $h(s)$
  - Kolize $h(s) = h(t)$ a $s \neq t$
  - Pokud nejsou kolize mluvíme o perfektním hašování.
  - Konsistentní hašování: pokud zvětšíme pole A pouze omezený počet prvků je nutné přemapovat.
- Vyhledání prvku s: položka $A(h(s))$

Distribuované hašovací tabulky

- Distribuovaná varianta konsistentního hašování.
- Každý uzel v sítí spravuje svoji část globální hašovací tabulky.
- Uložení i vyhledání prvku *s* znamená směrovat dotaz k uzlu, který spravuje oblast do které patří *h*(*s*).



Obecné vlastnosti DHT

- Hodnotu $h(s)$ budeme chápat jako klíč prvku s a značit k, k_s. Množina všech klíčů je K.
- Funkce $\delta: K^2 \rightarrow N^+$ je vzdálenost mezi dvěma klíči.
- Každý uzel má svojí identifikaci $id \in K$.

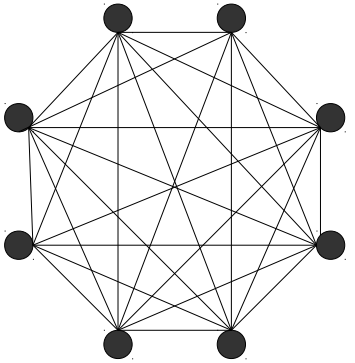
- Prvek s je přiřazen uzlu jehož id je nejblíže k_s ve smyslu funkce δ.
- Připojení nebo odpojení uzlu ovlivní pouze bezprostřední okolí dotčeného uzlu ve smyslu funkce δ.

Směrovací tabulky v DHT

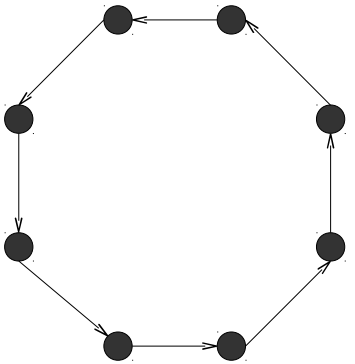
- Každý uzel je buď vlastníkem hledaného klíče, nebo má záznam ve směrovací tabulce ukazující na uzel, který je blíže hledanému klíči.
- Vyhledávání po konečném počtu přeskoků skončí u vlastníka hledaného klíče.
  - Snaha je mít počet přeskoků co nejmenší
    → rychlé hledání.
- Směrovací tabulky nemusí obsahovat údaje o všech uzlech.
  - Snaha je mít směrovací tabulky co nejmenší
    → méně práce při připojení/odpojení uzlu.

Naivní implementace

- Úplná informace (one-hop lookup)
  - Směrovací tabulky obsahují všechny uzly.
  - Počet přeskoků je vždy 0.
  - Velikost směrovacích tabulek je *n*.



- Minimální informace (n-hop lookup)
  - Směrovací tabulka obsahuje pouze jeden záznam na nejbližší větší uzel.
  - Počet přeskoků může být až *n*.
  - Velikost směrovacích tabulek je 1.



Vlastnosti implementace DHT

- Průměrný počet přeskoků pro nalezení klíče.
  - Ve většině implementací O(log(n)).
- Velikost stavových informací na každém uzlů.
  - Směrovací tabulky a pomocné struktury.
  - Většinou také O(log(n)).

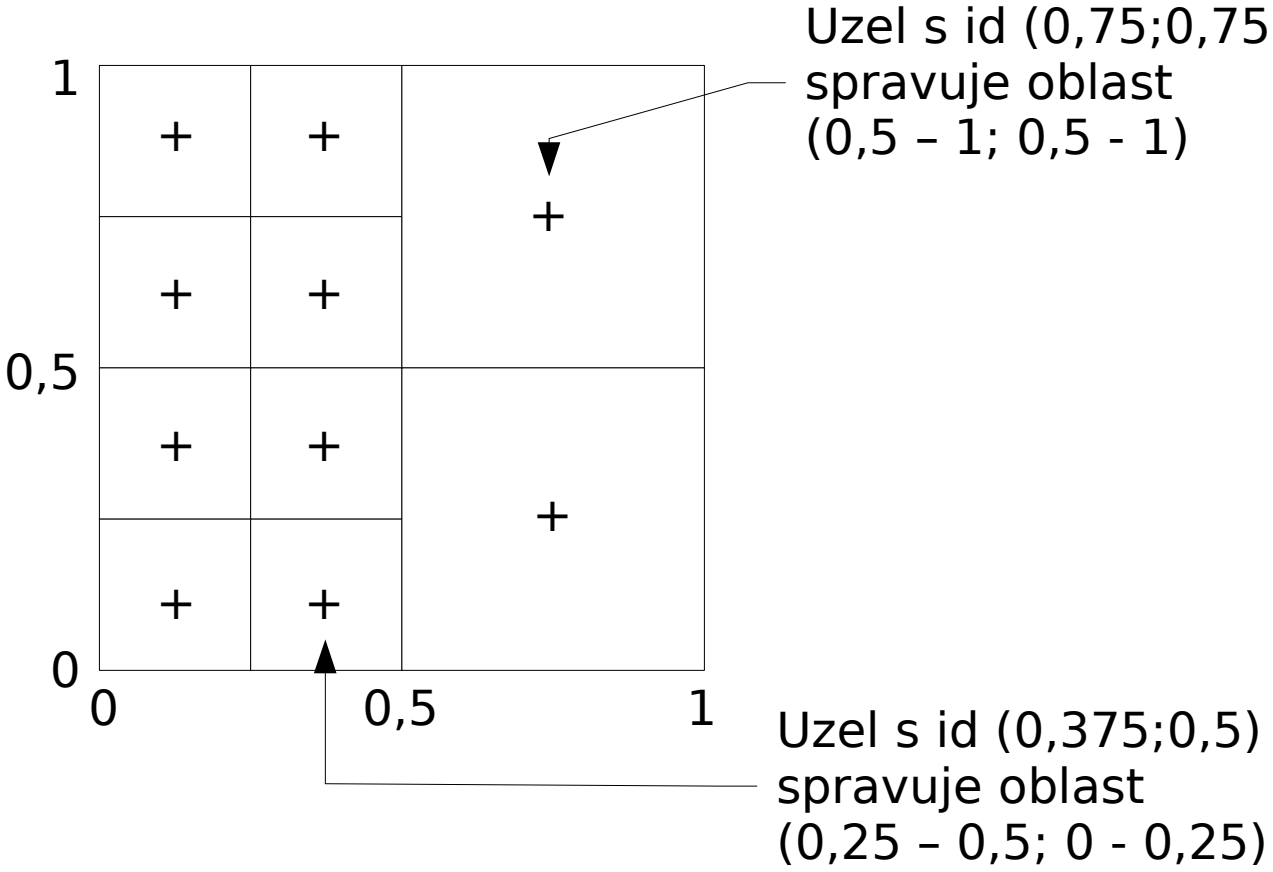
V čem se jednotlivé implementace liší

- Jak je realizována funkce δ ?
- Jak jsou spravovány směrovací tabulky ?
- Jaký je algoritmus směrování ?

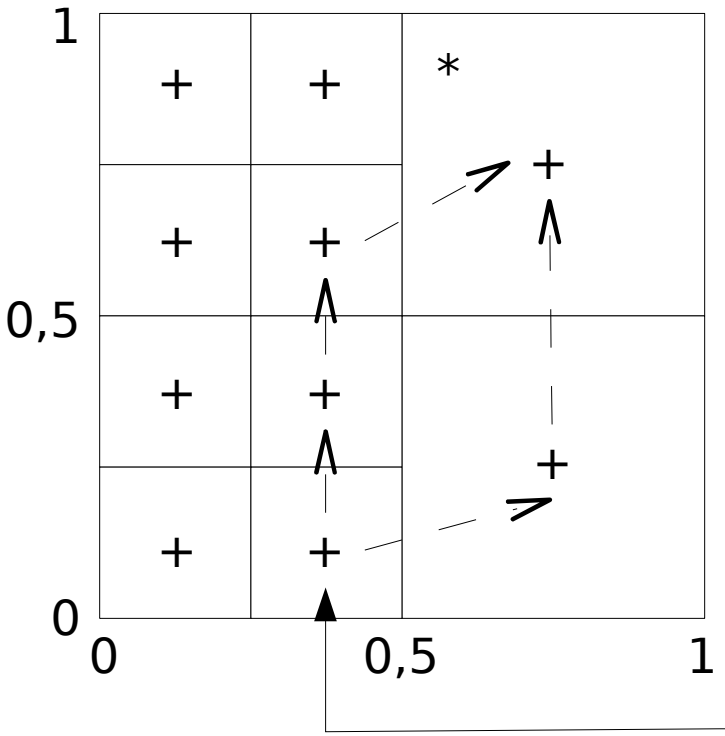
Content-Addressable Network

- Prostor klíčů je d-rozměrný kartézský souřadnicový systém.
  - Souřadnice bodů v prostoru představují klíče.
  - Přirozené chápání funkce δ jako vzdálenost v prostoru.
  - Všechny osy jsou uzavřené – jedná s d-torus.
- Prostor je rozdělen na zóny.
  - Každou zónu vlastní jeden uzel.
- Každý uzel si drží ukazatele na uzly sousední zóny.
  - Směrování probíhá podle hladového algoritmu.

Příklad v 2-rozměrném prostoru



Příklad v 2-rozměrném prostoru



Je možné více cest k dosažení cíle.

Hledá prvek s klíčem (0,6;0,8)

Připojení a odpojení uzlu v CAN

- Připojení:
  - Najít libovolný uzel, který je již připojen do sítě.
  - Identifikovat zónu, která může být rozdělena.
  - Dohodnout se s vlastníkem zóny na jejím rozdělení.
  - Upravit směrovací tabulky sousedních uzlů.
- Odpojení:
  - Identifikace souseda, který může převzít moji zónu.
  - Zóny jsou buď sloučeny do jedné, nebo jeden uzel dočasně vlastní dvě zóny.

Vlastnosti CAN

- Počet stavových informací, které si musí každý uzel držet je $O(d)$
- Průměrný počet přeskoků k nalezení klíče je $O(d * n^{1/d})$.
  - To není hodnota typická pro DHT. Pokud ale volíme $d = (log_2 n)/2$ dostane typickou hodnotu.
- Možné vylepšení
    - Směrování bere v úvahu RTT
      → rychlejší, efektivnější.
    - Jednu zónu vlastní více uzlů
      → odolnější proti výpadkům.

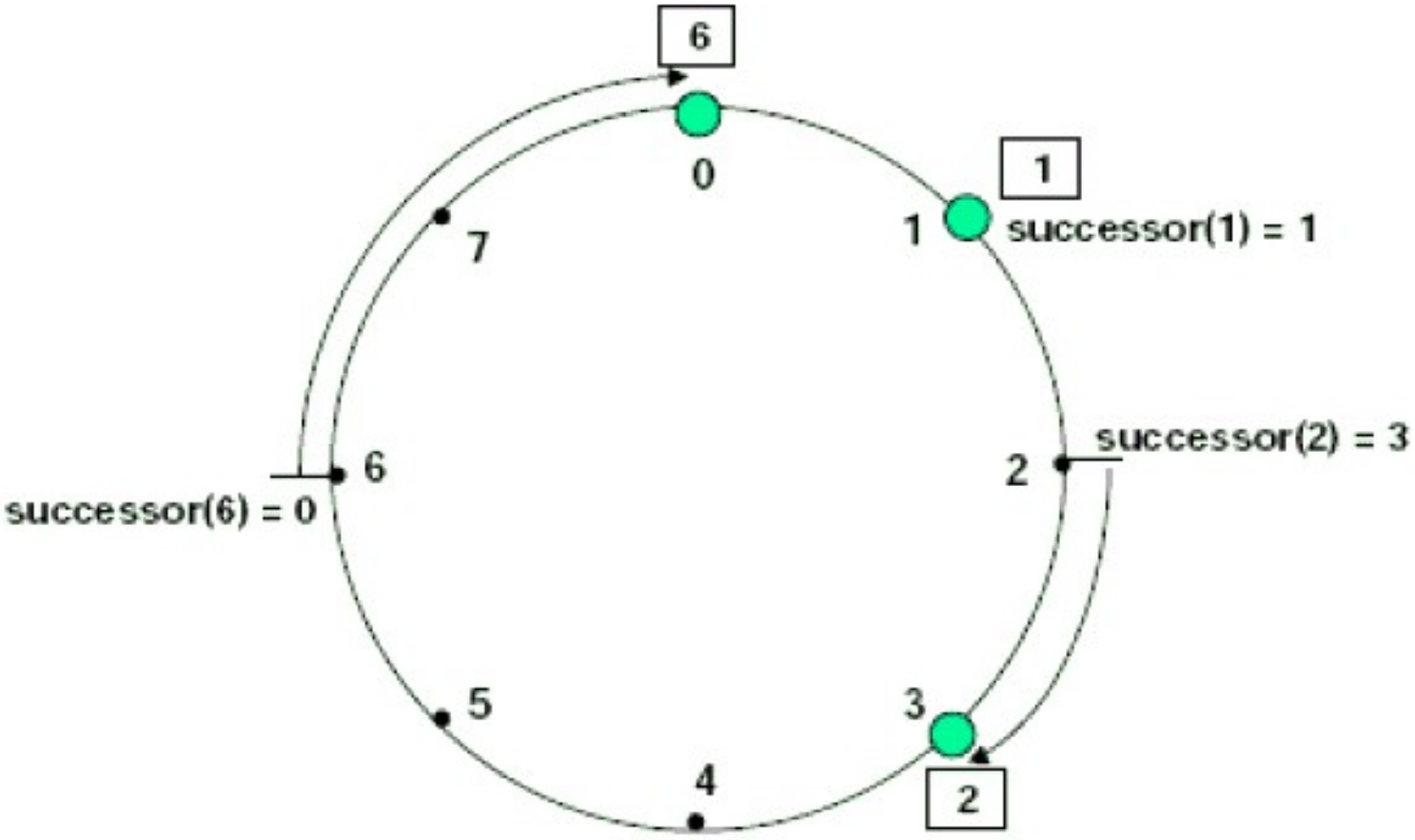
Jak najít první uzel ?

- Za pomoci centrálního serveru.
  - Už to není plně decentralizovaná síť.
- Seznam stabilních uzlů.
  - Distribuované s aplikací.
  - Postupně upravované během běhu aplikace.
- Použití jiného systému.
  - Například DNS.
- Aktivní hledání.
  - Broadcast na lokální sítí, multicast na Internetu.
- Je to necháno na uživateli.
  - Pozvánka od jiného uzlu.

Chord

- Klíče jsou *m*-bitová čísla uspořádaná do kružnice.
- Funkce δ je vzdálenost dvou klíčů na kružnici ve směru hodinových ručiček.
- Funkce *successor*(*k*) vrací identifikátor uzlu, který je rovný nebo větší *k*.
- Klíč *k* je uložen na uzlu *successor*(*k*).

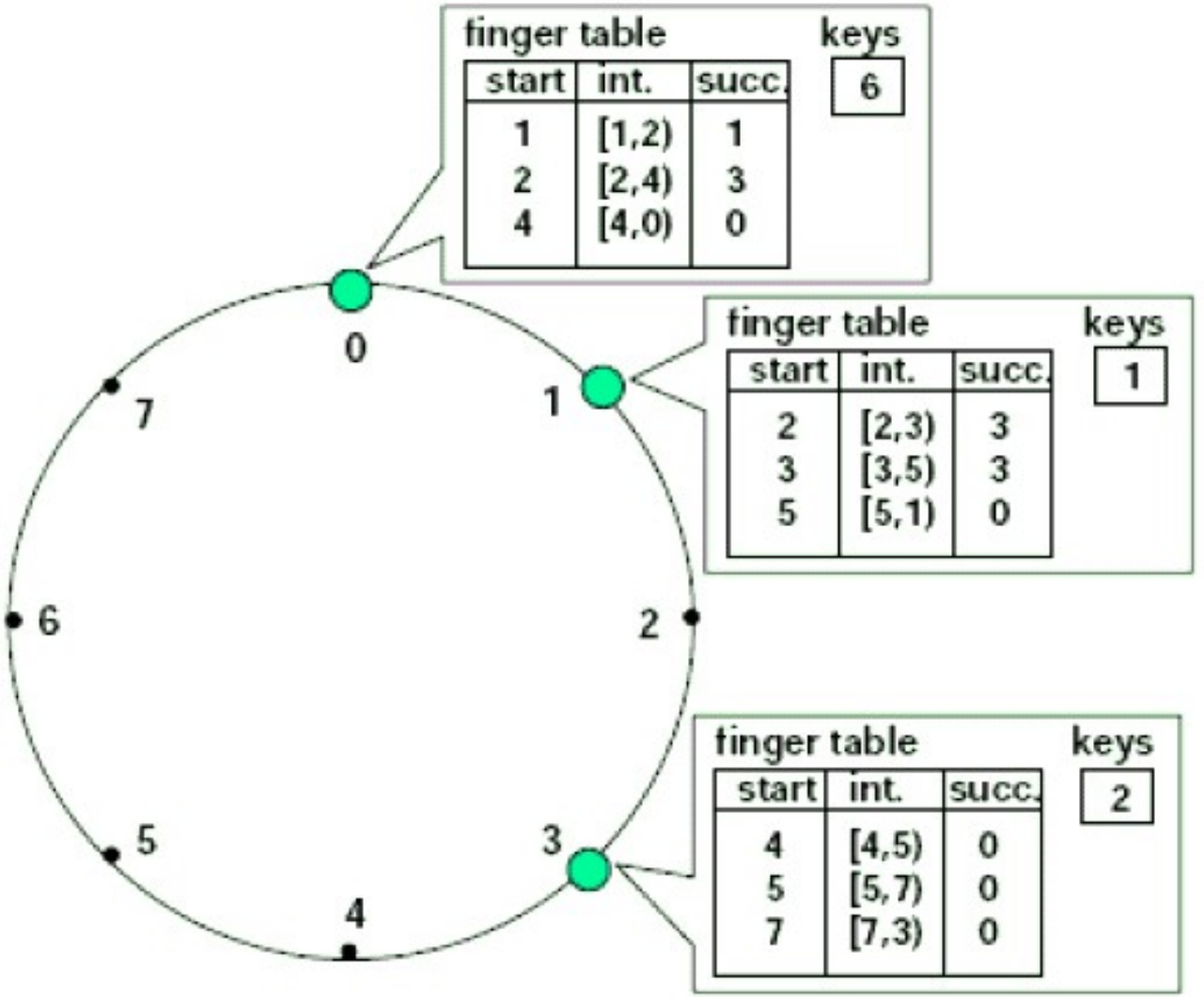
Ukázka topologie Chord



Směrování v Chord

- Směrovací tabulka každého uzlu má velikost m.
- Položka i ve směrovací tabulce uzlu id pokrývá oblast $[id+2^{i-1} ; id+2^{i})$
  - Položka 1 pokrývá oblast velikosti 2.
  - Položka m pokrývá oblast velikosti 2^{m-1} .
- Položka i ve směrovací tabulce uzlu id obsahuje ukazatel na uzel s = $successor(id + 2^{i-1})$
- Počet přeskoků k nalezení klíče je $O(\log n)$.
- K uložení stavových informací je potřeba $O(\log n)$.

Směrování v Chord



Připojení nového uzlu - naivně

- Každý uzel si také drží ukazatel na svého předchůdce.
- Připojení nového uzlu*:*
  - Od nějakého připojeného uzlu získat směrovací tabulky.
  - Informovat uzly, které si musí změnit směrovací tabulky.
  - Přesun dat na nově připojený uzel.
- Problém se souběžném připojení více uzlů.
  - Tabulky se mohou dostat do nekonzistentního stavu.
  - Během připojování může selhat vyhledávání.

Připojení nového uzlu - lepší

- Pro korektní vyhledávání je nutný správný ukazatel na bezprostředního následníka.
  - Vyhledávání může být pomalejší, ale je vždy korektní.
- Připojení nového uzlu
  - Při připojení upravíme pouze ukazatel na následníka a předchůdce.
  - Jednou za čas spustíme stabilizační proceduru, která opraví směrovací tabulky.
- Opojení uzlu
  - Pouze ukazatel na následníka a předchůdce a pak stabilizační procedury.

Pastry

- Klíče jsou m-bitová čísla rozdělená na sekvenci číslic o základu 2^b.
  - Pro b = 4 se na klíče můžeme dívat jako na posloupnost hexadecimální číslic.
- Funkce *share*(k_1,k_2) udává počet číslic společného prefixu klíčů k_1 a k_2.

share(0xABCDE,0xABC11) = 3

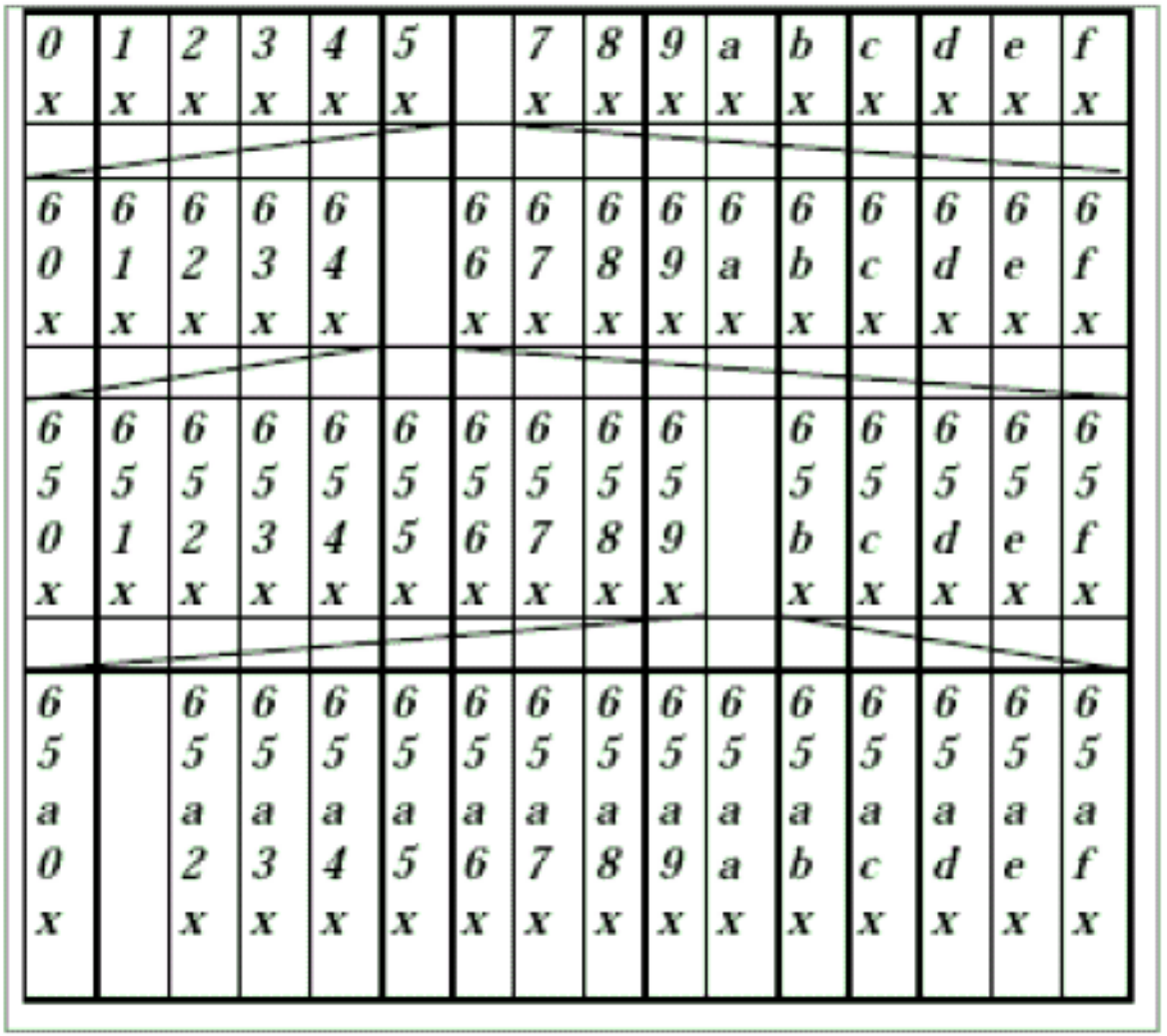
share(0xABCDE,0xAB111) = 2

share(0xABCDE,0xA1111) = 1

Pastry - Směrovaní

- V každém kroku se délka společného prefixu mezi hledaným klíčem a aktuálním uzlem zvětší alespoň o 1.
  - Maximálně $\lceil \log_{2^b}(N) \rceil$ kroků.
- Směrovací tabulka R obsahuje $\lceil \log_{2^b}(N) \rceil$ řádků každý po $(2^b - 1)$ záznamech.
  - Číslo řádku = délka společného prefixu.
  - Číslo sloupce = možné pokračování.

Směrovací tabulka v DHT Pastry



0 x	1 x	2 x	3 x	4 x	5 x		7 x	8 x	9 x	a x	b x	c x	d x	e x	f x
6 0 x	6 1 x	6 2 x	6 3 x	6 4 x		6 6 x	6 7 x	6 8 x	6 9 x	6 a x	6 b x	6 c x	6 d x	6 e x	6 f x
6 5 0 x	6 5 1 x	6 5 2 x	6 5 3 x	6 5 4 x	6 5 5 x	6 5 6 x	6 5 7 x	6 5 8 x	6 5 9 x		6 5 b x	6 5 c x	6 5 d x	6 5 e x	6 5 f x
6 5 a 0 x		6 5 a 2 x	6 5 a 3 x	6 5 a 4 x	6 5 a 5 x	6 5 a 6 x	6 5 a 7 x	6 5 a 8 x	6 5 a 9 x	6 5 a a x	6 5 a b x	6 5 a c x	6 5 a d x	6 5 a e x	6 5 a f x

Směrovací tabulka na uzlu: 65a1

Znak ***x*** v každé buňce reprezentuje směrovací informaci na uzel s daným prefixem.

Bílé místa odpovídají prefixům, které jsou schodné s aktuálním uzlem.

Směrování v DHT Pastry

- Každý uzel má také leaf set L.
  - Obsahuje množinu uzlů, které jsou numericky nejblíže.
  - Její velikost je typicky 2^b nebo 2^{b+1} .
  - Polovina z této množiny obsahuje klíče větší než je identifikátor daného uzlu, druhá polovina zase klíče menší než identifikátor uzlu.
- Směrování
  - Pokud je uzel v leaf set směruje se podle leaf set.
  - Pokud není v leaf set hledá se údaj ve směrovací tabulce.
  - Pokud není záznam ani ve směrovací tabulce pošle se na uzel z leaf set, který je nejblíže cílovému uzlu.

```
// D – Klíč zprávy, která má být směrována.
// A – Identifikátor aktuálního uzlu.
// R_l^i – Záznam ve směrovací tabulce na řádku l, ve sloupci i.
// L_i – i-tý nejbližší uzel v leaf set.
// D_l  - l-tá číslice v klíčí D.
// share(A,B) – délka sdíleného prefixu mezi klíči A a B.

if ( L_-l/2 ≤ D ≤ L_l/2 ) {  // použít leaf set
      přepošli na L_i takové, že | D – L_i | je minimální.
} else {    // použij routing table.
      len = share(D,A);
      dig = D_len;
      if ( R_len^dig ≠ nil) {
            přepošli na R_len^dig;
      } else {    // pokud ve směrovací tabulce chybí záznam.
            přepošli na T takové, že:
                  T ∈ ( L ∪ R );
                  share(T,D) ≥ len;
                  | T – D | < | A – D |
      }
}
```

Pastry

- Pro každý záznam ve směrovací tabulce existuje více možných uzlů, na které může ukazovat.
  - Lze optimalizovat a vybrat nejlepší.
- Řeší případy kdy uzel havaruje.
  - Havarovaný uzel může být nahrazen z leaf setu nějakého blízkého uzlu.

Připojení uzlu v DHT pastry

- Připojení uzlu s identifikátorem *X*.
  - Kontaktování libovolného uzlu *A* a předání zprávy *join*(*X*).
  - Uzel *A* směruje zprávu *join*(*X*) na uzel *Z*, který je nejblíže klíči *X*.
  - Uzel X získá leaf set z uzlu Z a *i*-tý řádek své směrovací tabulky z *i*-tého uzlu na cestě z *A* do *Z*.
  - Uzel X informuje ty uzly, které by si měli zařadit X do svých tabulek.
- Odpojení uzlu
  - Předání dat sousednímu uzlu.
  - Směrovací tabulky se časem upraví sami.

Kademlia

- Klíče jsou *m* bitová čísla.
- Funkce $\delta(x,y) = x \oplus y$ (XOR).
- Využívají paralelní asynchronní dotazy.
  - Výrazně snižují latenci dotazu.
  - Zajišťují korektní chování i ve velmi dynamické síti.
  - Zvyšují zátěž sítě.

Směrovací tabulky - Kademlia

- Každý uzel si drží *m* seznamů. Takzvaných k-buckets.
- Seznam *i* uzlu *id* obsahuje ukazatele na takové uzly *node* pro které

$$\delta(id,node) \in [2^i,2^{i+1})$$

- Každý k-bucket má maximálně *k* položek.

Směrovací tabulky - Kademlia

- Každý uzel si drží *m* seznamů. Takzvaných k-buckets.
- Seznam *i* uzlu *id* obsahuje ukazatele na takové uzly *node* pro které

$$\delta(id,node) \in [2^i,2^{i+1})$$

- Každý k-bucket má maximálně *k* položek.

Stejné jako chord - jen jiná funkce δ a každá položka má *k* záznamů.

Směrování v DHT Kademlia

- Směrovací algoritmus pracuje rekurzivně:
  - Uzel vybere α uzlů ze svých tabulek, které jsou nejblíže hledanému klíči a odešle jim dotaz na klíč.
  - Z odpovědí, které přijal pošle stejný dotaz nově objeveným uzlům.
  - Výsledkem je *k* uzlů, které jsou nejblíže hledanému klíči.
- Parametry systému:
    - m – počet bitů na reprezentaci klíče.
    - k – velikost k-bucketu.
    - α – stupeň paralelismu.

Úprava směrovací tabulek

- Při každé komunikaci s jiným uzlem si uzel upravuje své směrovací tabulky:
  - Pokud daný kontakt už v příslušném k-bucketu existuje, přesune se na konec seznamu.
  - Pokud daný kontakt ještě v příslušném k-bucketu není a seznam má méně než *k* položek, přidá se na konec seznam.
- Pokud má seznam již *k* položek tak je proveden ping na uzel v hlavičce seznamu.
  - Pokud uzel na ping neodpoví je ze seznamu vyjmut a nový kontakt je vložen na konec seznamu.
  - V opačném případě se nový kontakt zahodí.

Kademlia

- Snadný postup po připojení uzlu:
  - Novému uzlu stačí zanést do svých tabulek pouze jednoho člena sítě. Ostatní položky se doplní během komunikace.
- Žádný postup na odpojení uzlu:
  - Uzel se prostě odpojí a síť si s tím poradí.

Vhodné pro velice dynamické sítě !

Shrnutí DHT

	CAN	**Chord**	**Pastry**	**Kademlia**
Routing performance	$O(d * N^{1/d})$	$O(\log N)$	$O(\log_B N)$	$O(\log N)$
Routing state	2 d	$\log N$	$B * \log_B N + B$	$k * \log N$
Peers join/leave	2 d	$(\log N)^2$	$\log_B N$	-

$B = 2^b$

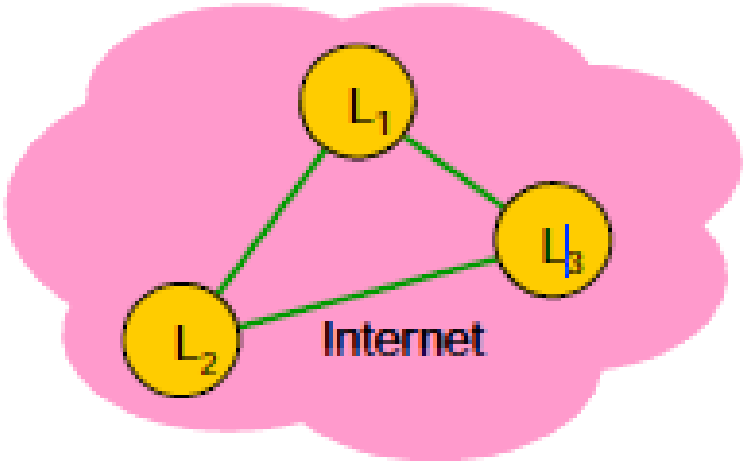
Další problémy spojené s DHT

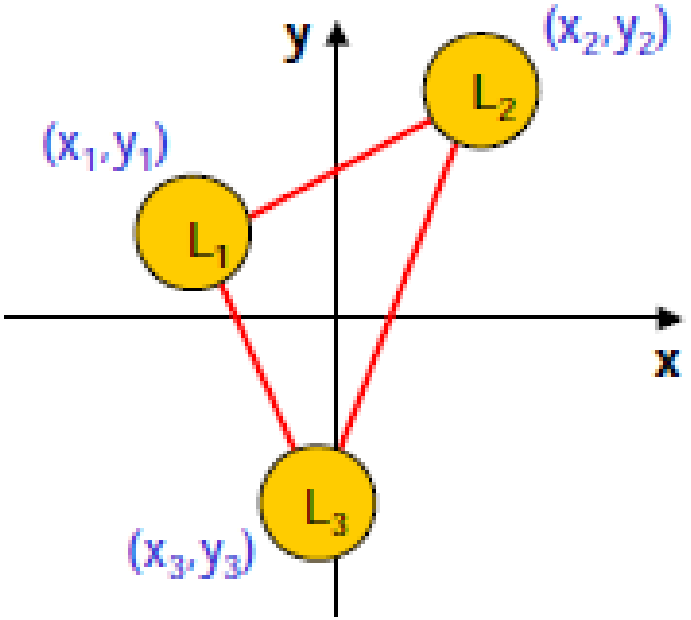
- Optimalizace směrování
  - Virtuální struktura co nejvíce schodná s fyzickou.
- Uzel může havarovat, nekorektně se odpojit.
  - Nutno opravit směrovací tabulky.
  - Nesmí se ztratit data → replikace.
- Bezpečnostní problémy
  - Není žádná centrální důvěryhodná autorita.
  - Uzly jsou anonymní.

Optimalizace směrování

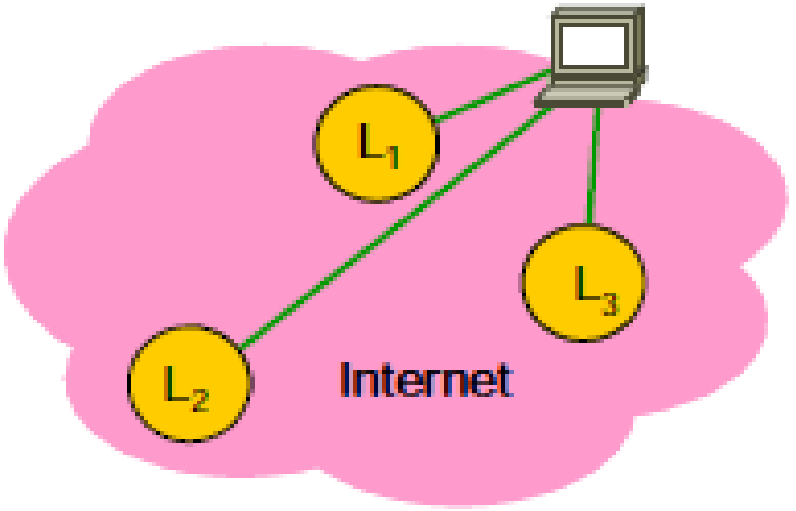
- Jaký použít další přeskok při směrování zprávy?
- K dispozici je několik možný uzlů.
  - S žádným uzlem se ještě nekomunikovalo.
  - Měřit round-trip time by trvalo příliš dlouho.
- Global Network Positioning (GNP)
  - Množina N uzlů, které měří RTT mezi sebou (Landmarks).
  - Podle změřených RTT jsou uzly umístěny do N-1 rozměrného prostoru.
  - Běžné uzly měří RTT směrem k Landmarks a aproximují svojí pozici v prostoru.

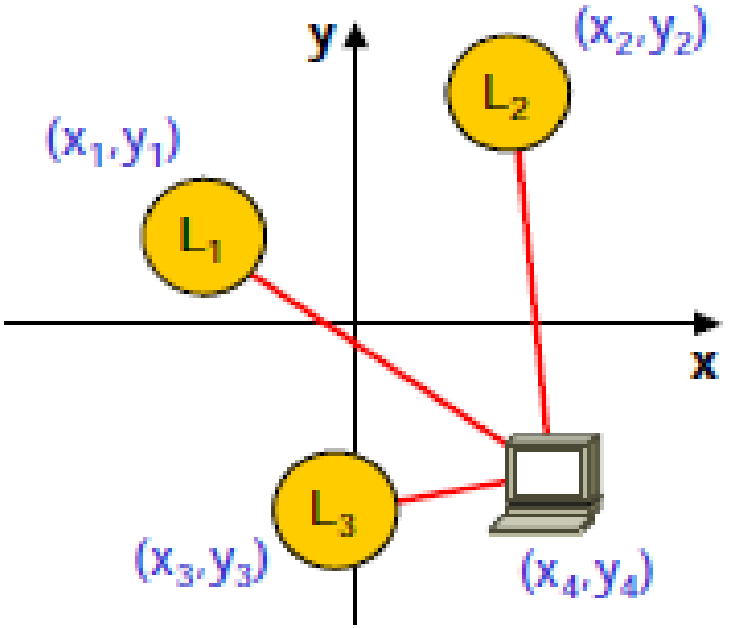
Global Network Positioning





Global Network Positioning





Vivaldi algoritmus

- Stejně jako GNP se snaží umístit uzly do n-rozměrného prostoru, který co nejvíce odpovídá fyzickým vzdálenostem.
  - Nepoužívá landmark.
  - RTT se měří k ostatním uzlům.
- V sítí nemusí platit trojúhelníková nerovnost.
  - Výsledky jsou vždy přibližné.
- Snaha minimalizovat chybu:

$$E = \sum_i \sum_j \left(L_{i,j} - \| \vec{x}_i - \vec{x}_j \| \right)^2$$

Replikace

- Na jaké uzly replikovat?
- Jak udržet repliky synchronizované?
  - Aktivní monitoring.
  - Periodické obnovování.
- Replikace
  - Více realit – několik nezávislých DHT nad stejnými uzly.
  - Více hašovacích funkcí – data jsou uložena podle více klíčů.
  - V každé oblasti více uzlů – data na více sousedních uzlech.
  - Cache, složitější datové struktury.

Bezpečnostní problémy

- Co když nějaký uzel chce záměrně ohrozit síť?
  - Pozměňování dat.
  - Vkládání falešných nebo nebezpečných dat.
  - Neukládání dat, které patří do jeho oblasti.
  - Neposkytování žádných dat – pouze využívat data jiných.
- Reputation-based trust management
  - Hodnocení každé transakce s jiným uzlem.
  - Vyměňování zkušeností s ostatními uzly.
  - Každý uzel si buduje reputaci.

Bezpečnostní problémy

- Útočník se snaží bránit tomu aby byl odhalen:
  - Poskytování falešných informací - zkušeností.
  - Vytváření velkého množství virtuálních identit.
  - Malicious collectives.

Bezpečnostní problémy

- Útočník se snaží bránit tomu aby byl odhalen:
  - Poskytování falešných informací - zkušeností.
  - Vytváření velkého množství virtuálních identit.
  - Malicious collectives.

Díky své otevřenosti a anonymnosti jsou P2P sítě potenciálně velmi nebezpečné prostředí.

Zajisti důvěryhodnost v takovém prostředí je velice těžké.

Eigentrust

- Local trust value – $s_{i,j} = sat(i,j) - unsat(i,j)$
- Normalized trust value - $c_{i,j} = \frac{max(s_{i,j}, 0)}{\sum_j max(s_{i,j}, 0)}$
- Global trust value – $t_{i,k} = \sum_j c_{i,j} c_{j,k}$
- Matrix notation - $\vec{t}_i = C^T \vec{c}_i$

$$\vec{t}_i = (C^T)^n \vec{c}_i$$

Eigentrust

$\vec{t}^{(0)} = \vec{e};$

repeat

$\quad \vec{t}^{(k+1)} = C^T \vec{t}^{(k)};$

$\quad \delta = ||t^{(k+1)} - t^k||;$

until $\delta < \epsilon;$

$\vec{t}^{(0)} = \vec{p};$

repeat

$\quad \vec{t}^{(k+1)} = C^T \vec{t}^{(k)};$

$\quad \vec{t}^{(k+1)} = (1-a)\vec{t}^{(k+1)} + a\vec{p};$

$\quad \delta = ||t^{(k+1)} - t^{(k)}||;$

until $\delta < \epsilon;$

Distribuovaný Eigentrust

Algorithm:
Each peer i do $\{$
Query all peers $j \in A_i$ for $t_j^{(0)} = p_j$;
repeat
Compute $t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + c_{2i}t_2^{(k)} + \ldots + c_{ni}t_n^{(k)}) + ap_i$;
Send $c_{ij}t_i^{(k+1)}$ to all peers $j \in B_i$;
Compute $\delta = |t_i^{(k+1)} - t_i^{(k)}|$;
Wait for all peers $j \in A_i$ to return $c_{ji}t_j^{(k+1)}$;
until $\delta < \epsilon$.;
$\}$

- A_i množina uzlů, které využívají služby uzlu *i*.
- B_i množina uzlů využívající služeb z *i*.

Zabezpečený Eigentrust

- V původní verzi každý uzel počítá svojí vlastní reputaci.
- V zabezpečené verzi se používá score manager:
  - Každý uzel má několik score managerů.
  - Vyhledání score managerů přes DHT.
  - Využívá se majority vote.

Další trust managementy

- TrustMe
  - Důraz na anonymitu.
  - Využívá centrální certifikační autoritu.
- Alliatrust
  - Pro nestrukturované sítě.
- Debit & Credit
  - Za správně poskytnuté služby získá uzel credit.
  - Za čerpání služeb platí creditem.
- … a další.

Kde se používají DHT

- File sharing networks
  - LimeWire - Kademlia
  - Overnet - Kademlia
  - eDonkey, eMule – Kademlia (Kad Network)
  - The circle – Chord
  - Bittorent – Peer Exchange (Vuze, µTorrent, ...)
- Sdílení diskové kapacity (www proxy)
  - Coral Content Distribution Network
  - CoDeeN – CoDNS
  - Dijjer

Kde se používají DHT

- Distributed search engine and web crawler
  - YaCy
  - Faroo
- Anonymizační sítě
  - využívá se principu key based routing.
  - I2P (pseudonymous overlay network).

Knihovny a implementace

- Chimera – v jazyce C, založena na Pastrech.
- Mojito – Java, Kademlia – LimeWire project.
- JXTA – Sada protokolů pro budování overlay networks.
  - Gisp - Global Information Sharing Protocol – implementace DHT.
- JDHT – Simple java DHT library.
- FreePastry – Java, Pastry.
- ...

Chimera: Basic interface

....

```
/* Initiates the chimera overlay */
state = chimera_init(port);

/* Set local key */
chimera_setkey(state, hash(hostname));

/* Registers an integers message type to be routed by the chimera
routing layer */
chimera_register(state, MSG_TYPE);

/* Join chimera network */
chimera_join(state, bootstrap);

...
/* Send a message */
chimera_send(state, peer_key, MSG_TYPE, len + 1, buf);
```

Chimera: Up-call interface

```
static void call_deliver(Key *key, Message *msg) {

    if (strcmp(get_key_string(key),my_key) == 0) {
            ...
    }


}

/* This up-call occurs when the current node receives a message msg
destined for a key that is responsible for */

chimera_deliver(state,call_deliver);

chimera_forward(state,call_forward);

chimera_update(state,call_update);
```

Distribuované hašovací tabulky

Konec