Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Simulace vybrané metody DHT

Plzeň, 2014 Ondřej Kolman

# Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 25. 6. 2015

Ondřej Kolman

# Abstrakt

Práce se zabývá rešerší a analýzou strukturovaných Peer-to-Peer sítí. Především na systémy CAN, CHORD, PASTRY a KADEMLIA. Systém KADEMLIA je popsán detailněji společně s popisem implementací systému ve vlastním simulačním programu, který má za úkol simulovat základní charakteristiky tohoto systému.

# Abstract

The work deals with the research and analysis of the structured Peer-to-Peer network. Foremost system CAN, CHORD, PASTRY and KADEMLIA. System KADEMLIA is describe in detailed together with the description of the implementation of this system in own simulation program, which is designed to simulate the basic characteristics of the system.

# Obsah

# 1. Úvod

Peer-to-Peer sítě si získali špatné jméno především kvůli jejich využití k nelegálnímu sdílení obsahu na Internetu. Za posledních pár let se však našlo i několik způsobů jejich legálního využití. Ve většině případů se používají především jako podpora pro zrychlení síťových aplikací nebo pro snížení zátěže serverů. Firmám, které poskytují obsah tak umožnili ušetřit nemalé finanční prostředky za nákup serverového vybavení a proto se můžeme do budoucna těšit rozvoji všech forem těchto sítí.

Cílem této práce je nastudovat problematiku strukturovaných Peer-to-Peer sítí, které jsou alternativou pro klasické sítě typu klient-server a poté navrhnout vlastní simulátor který dokáže nasimulovat základní charakteristiky vybrané sítě pro tisíce až sta tisíce uzlů. Ze získaných dat poté udělat analýzu.

# 2. Rešerše peer to peer sítí založených na DHT

## 2.1. Peer to peer sítě

Peer to Peer (zkr. P2P) sítě jsou alternativou k architektuře server/klient. V P2P sítích jsou si všechny uzly rovny, jednotlivé uzly tak jsou klienty i servery zároveň. Libovolný uzel se může od sítě kdykoliv odpojit a zároveň se může k síti připojit nový uzel.

### 2.1.1. Strukturované P2P sítě

Jsou zvláštním případem Peer to Peer sítí. Spojení mezi uzly je vytvářeno podle přesných pravidel daného algoritmu[3]. Každý uzel má směrovací tabulku, podle které dokáže najít cestu k libovolnému klíči v síti. Aby každý uzel nemusel znát všechny uzly v síti, jsou směrovací tabulky rozděleny, tak aby každý uzel znal pouze část sítě. K rozdělení směrovacích tabulek se používají distribuované hashovací tabulky (zkr. DHT).

### 2.1.2. DHT – distribuované hashovací tabulky

Každý uzel v síti spravuje svoji část globální hashovací tabulky, čímž se snižuje počet zpráv nutných k udržení aktuálních záznamů. Uložení i vyhledání prvku znamená směrovat dotaz k uzlu, který spravuje oblast do níž prvek patří. Nalezení existujících dat je dokončeno vždy v konečném počtu kroků.

### 2.1.3. 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. Snahou je omezit počet skoků nutných k nalezení hledaného klíče, čehož lze dosáhnout zvětšením směrovací tabulky.

Ovšem čím je větší směrovací tabulka, tím větší jsou náklady na její údržbu – připojení nebo odpojení uzlu se musí zanést do všech směrovacích tabulek. Pro snížení nákladů na udržbu je snaha mít tabulku co nejmenší. Proto je nutné najít rovnováhu mezi

velikostí směrovací tabulky a náklady na její údržbu.

## 2.2. Přehled strukturovaných P2P sítí

Tato kapitola představuje přehled základních strukturovaných Peer-to-Peer sítí.

### 2.2.1. Content Addressable Network (CAN)

#### 2.2.1.1. Obecný popis

CAN je organizovaný jako *d*-dimenzionální toroid s kartézským souřadnicovým systémem. Mezi zdrojem a cílem existuje více možných cest, a při jejich hledání se využívá hladového algoritmu. Celý prostor je rozdělen na zóny a každou zónu vlastní a spravuje jeden uzel viz obr. 2.1.

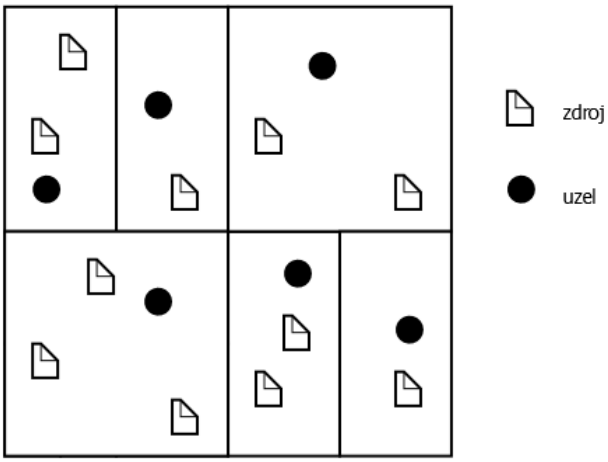
Uvažujeme-li *d*-dimenzionální CAN na *n* uzlech, v průměru má každý uzel *2d* sousedů a průměrná délku cesty je $\frac{d}{4} n^{\frac{1}{d}}$ . Přidání nového uzlu prodlouží délku cesty v průměru o $O(n^{\frac{1}{d}})$ .

Ke snížení prodlevy při vyhledávání se používají různé způsoby. Jedním z nich je použití vícenásobné hashovací funkce k mapování jednoho klíče na více míst v prostoru. Z toho plyne zvýšení dostupnosti klíče.

Dalším způsobem je rozvržení zón tak, aby uzly v sousedních zónách byly i uzly ve fyzickém světě. Zamezí se tím, že jeden skok v CAN, může jít fyzicky přes mnoho IP skoků.

V neposlední řade je použití několika nezávislých souřadnic nazývaných reality. Uzel patřící do *r* realit má *r* nezávislých sad sousedů a *r* souřadnic ukazujících na oddělené zóny. Používání více realit, zvyšuje spolehlivost směrování. Použití jiné reality může být využito v případě selhání uzlu při odesílání. Navíc používání různých směrovacích tabulek pro různé reality zkracuje průměrnou délku cesty a vzdálené místa mohou být

dosaženy v rámci jednoho skoku.




*Obr. 2.1 Ukázka rozdělení prostoru v algoritmu CAN*

Zvyšování dimenze i zvyšování počtu realit vede k snižování průměrné délky cesty. Narůstající počet dimenzí také zvyšuje počet sousedů a tím vede ke zvýšení nákladů na údržbu směrovací tabulky.

#### 2.2.1.2. Připojení uzlu

Aby se do sítě CAN mohl připojit nové uzly, je nutné jim přidělovat vlastní zóny. Nová zóna je vytvořena rozpůlením jedné dimenze stávající zóny. Jedna polovina zóny zůstane původnímu uzlu a druhá polovina připadne uzlu novému. Proces připojení se skládá ze tří kroků:

1. Nalezení uzlu, který je již v síti CAN připojen - toho je docíleno buď broadcastem v lokální síti, multicastem v Internetu, nebo zadání adresy ručně z veřejného zdroje.

2. Nalezení uzlu, u kterého lze provést rozdělení zóny. Připojující se uzel vybere náhodný bod *P* a pošle požadavek *JOIN* do bodu *P*. Zpráva je poslána přes existující uzly v CAN pomocí běžného směrování do zóny, ve které leží bod *P*. Současný majitel zóny ji rozdělí na dvě části. Dvojice (klíč, hodnota) z nově rozdělené zóny budou přesunuty na nově se připojující uzel.

3. Aktualizace směrovacích tabulek všech sousedů kolem nově vzniklé zóny.

#### 2.2.1.3. Odpojení uzlu

Při odpojení uzlu ze sítě je zóna, spravovaná tímto uzlem, rozdělena mezi sousední uzly.

Pokud je zóna některého souseda vhodná ke sloučení, jsou tyto zóny spojeny do jedné větší. V případě, že žádný takový soused neexistuje bude daná zóna předána sousedovy s nejmenší spravovanou zónou. Tak vznikne stav ve kterém jeden uzel zároveň spravuje dvě zóny.

#### 2.2.1.4. Vyhledávání a vložení klíče

Pro vyhledání klíče spočítáme jeho pozici v prostoru a do příslušné zóny pošleme požadavek na jeho hodnotu.

Pro uložení dvojice *(klíč, hodnota)* spočítáme pozici klíče v prostoru a do příslušné zóny odešleme zprávu *INSERT(klíč,hodnota)*.

#### 2.2.1.5. Výpadek uzlu – rekonfigurace

CAN je zabezpečen proti nedostupnosti kteréhokoliv uzlu. V běžném stavu, posílá pravidelně každý uzel svým sousedům zprávy o souřadnicích své zóny a o souřadnicích svých sousedních zón.

Absence těchto zpráv je známkou výpadku. Jakmile se uzel rozhodne, že jeho soused je nedostupný, spustí odpočítávání. Každý soused chybového uzlu spouští odpočítávání nezávisle na ostatních. Čas odpočítávání je vyvozen z velikosti zóny, kterou uzel obsluhuje. Když odpočítávání skončí, uzel pošle do zón sousedících s chybovým uzlem zprávu *TAKEOVER* která obsahuje i velikost aktuálně obsluhované zóny.

Při přijetí zprávy *TAKEOVER* uzel buď zruší svoje odpočítávání, v případě, že je velikost zóny ve zprávě menší než jeho obsluhovaná zóna, nebo pošle vlastní zprávu *TAKEOVER*. Tak se najde žijících uzel s nejmenší zónou, který převezme zónu od nedostupného uzlu.

Při výpadcích a odpojení uzlů dochází k situaci, kdy jeden uzel obstarává více zón. Aby se předešlo vysoké fragmentaci, používá se algoritmus, který má za cíl udržet stav jeden uzel na jednu zónu a to díky slučování stejných sousedních zón.

### 2.2.2. CHORD

#### 2.2.2.1. Obecný popis

ChORD je jedním z prvních významných DHT algoritmů, který využívá kruhovou topologii. Identifikátory klíče i uzlu jsou 160-ti bitová čísla, generovaná pomocí SHA-1 algoritmu. Pro získání identifikátoru uzlu se hashuje jeho IP adresa, u klíče jsou to jeho atributy (nejčastěji název souboru).

V kruhu CHORD je predecessor první uzel proti směru chodu hodinových ručiček a successor je následující uzel ve směru chodu hodinových ručiček. Uzel je zodpovědný za objekty mezi ním a jeho predecesorem.

Každý uzel si udržuje tabulku ukazatelů. V *m*-bitovém prostoru má každý uzel až *m* ukazatelů. Řádek *i* v tabulce ukazatelů uzlu *n* odkazuje na uzel *y* který je successorem uzlu *n*. Neboli ukazatel[*i*] = successor($n + 2^{i-1}$) viz obr. 2.2.

Funkce *delta* 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)*.

Návrh algoritmu CHORD má za cíl:

- Vyvážit zátěž související s distribucí klíče napříč uzly.

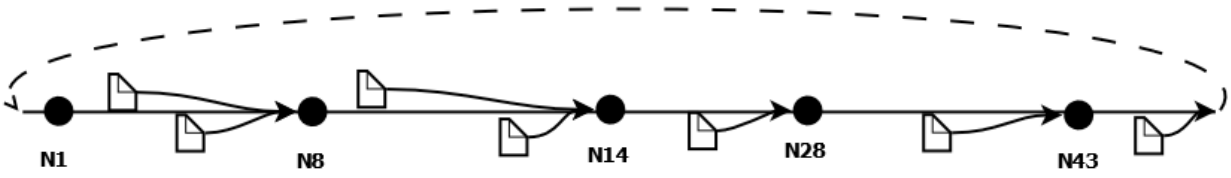
- Škálovatelnost - náklady na vyhledání klíče rostou pomaleji než počet uzlů. To umožňuje vytváření velkých systémů.

- Decentralizace – udržuje hlavní myšlenku P2P sítí, všechny uzly v CHORD jsou jsi rovny.

- Dostupnost – je zajištěna automatickou organizací uzlů v závislosti na připojování a

odpojování jednotlivých uzlů.

K zajištění konzistence sítě, každý uzel periodicky spouští udržovací protokol k obnovení successorů a záznamů ve směrovací tabulce.




Tabulka ukazatelů

| | |
|---|---|
| N1 + 1 | N8 |
| N1 + 2 | N8 |
| N1 + 4 | N8 |
| N1 + 8 | N14 |
| N1 + 16 | N28 |
| N1 + 32 | N43 |

*Obr. 2.2: ukázka rozdělení prostoru v algoritmu CHORD s tabulkou ukazatelů uzlu N1*

#### 2.2.2.2. Připojení uzlu

Pro připojení uzlu *n* je nutné:

1- inicializovat predeccesor a směrovací tabulku uzlu *n*

2- aktualizovat záznamy ve směrovací tabulce a predecessory existujících uzlů vzhledem k novému uzlu

3- informovat vyšší vrstvu softwaru o nutnosti přesouvat hodnoty sdružené s klíčem za které je zodpovědný uzel *n* na tento uzel.

#### 2.2.2.3. Odpojení uzlu

Při odpojení, uzel odešle zprávy o odpojení svému successoru, predecessoru a všem uzlům ve vzdálenosti ukazatelů. Poté přesune všechny objekty, za které je zodpovědný svému successoru.

#### 2.2.2.4. Vyhledávání klíče

Díky kruhové topologii lze pro vyhledání klíče procházet postupně všechny uzly, až do nalezení toho, který je zodpovědný za klíč. Tento způsob je zbytečně zdlouhavý a v průměru je potřeba projít polovinu kruhu uzlů, proto si každý uzel drží směrovací tabulku, aby tím zkrátil dobu hledání. K nalezení klíče je potřeba v průměru kontaktovat $O\log(n)$ uzlů. Záznamy ve směrovací tabulce obsahují identifikátor uzlu, jeho IP adresu a port.

Vyhledávání může být implementováno rekurzivním nebo iterativním směrováním. Při rekurzivním přístupu se každý nový uzel ve vyhledávací cestě ptá na další skok.

Při iterativním přístupu uzel, který započal vyhledávání dostává odpovědi od mezilehlých uzlů a sám vykonává jednotlivé skoky.

Rekurzivní přístup snižuje počet nutných zpráv k nalezení klíče. Iterativní přístup je na druhou stranu robustnější v dynamické síti.

#### 2.2.2.5. Výpadek uzlu – rekonfigurace

Při výpadku uzlu *n* musí uzly, které mají ve své směrovací tabulce odkaz na tento uzel najít jeho successor. Aby bylo nalezení jednoduší každý uzel má k tabulce ukazatelů ještě tabulku successorů.

#### 2.2.2.6. Použití

Chord File System (CFS),

### 2.2.3. Pastry

#### 2.2.3.1. Obecný popis

Pastry je založena na tzv. PRR stromové struktuře. Původní návrh PRR stromu byl zamýšlen jako statická síťová struktura bez možnosti nasazení v reálném světě. Podobně jako u sítě CHORD je Pastry postavena do kruhu.

Každý uzel má unikátní 128-bitový identifikátor. Sada existujících identifikátorů uzlů je rovnoměrně rozložena. Síť typu Pastry sestávající z *N* uzlů dokáže najít cestu do libovolného uzlu v průměru na $\log_{2^b} N$ skoků (*b* je konfigurační parametr, typická hodnota 4).

Tabulky v každém uzlu mají pouze $2^{b-1}\log_{2^b} N + 2l$ záznamů, kde každý záznam obsahuje identifikátor uzlu přiřazený k IP adrese uzlu. Navíc po selhání uzlu nebo připojení nového uzlu, mohou být všechny záznamy v zasažených tabulkách obnoveny výměnou O( $\log_{2^b} N$ ) zpráv. Dvojice (klíč, hodnota) jsou obstarávány numericky nejbližším uzlem.

### 2.2.3.2. Směrovací tabulka:

Obsahuje $\log(N)$ řádků s $2^b - 1$ záznamy. Záznamy na řádku *n* mají stejných *n* číslic s uzlem. Číslo řádku udává délku společného prefixu a číslo sloupce pak možné pokračování.

**Tabulka sousedů (Neighborhood set):**

Identifikátory uzlů a IP adresy |M| nejbližších uzlů (|M| ~ 2 * $2^b$ )

**Tabulka listů L (Leaf set L)**

Sada |L|/2 numericky nejbližších větších/menších identifikátorů uzlů, vztaženo relativně k současnému uzlu (|L| ~ $2^b$ ).

**Směrování**

Směrování v síti pracuje inkrementálně tzn., že v každém kroku se délka společné části identifikátoru mezi hledaným klíčem a aktuálním uzlem zvětší alespoň o jedna.

### 2.2.3.3. Připojení uzlu

Připojující se uzel s nově vybraným identifikátorem *X* inicializuje svůj status kontaktováním nejbližšího uzlu *A* a zeptá se uzlu *A* na cestu speciální zprávou použitím *X* jako klíče. Zpráva je adresována do existujícího uzlu *Z*, který je svým identifikátorem

nejblíže *X*.

### 2.2.3.4. Odpojení uzlu

Uzly se mohou odpojit kdykoliv bez jakéhokoliv varování. Uzel je považován za odpojený ve chvíli kdy přestane komunikovat se svými sousedy ve svém prostoru. K nahrazení uzlu *n* ve větvi stromu, se jeho soused spojí s žijícím uzlem s největším indexem na straně chybné větve a požádá ho o tabulku listů.

Chybný uzel ve směrovací tabulce je objeven ve chvíli kdy je kontaktován a nedává žádnou odpověď. To pozastaví doručování zpráv do doby než je nalezen jiný uzel.

K opravení záznamu ve směrovací tabulce uzel kontaktuje první uzel ze stejné řady , aby se zeptal na jeho záznamy.

### 2.2.3.5. Vyhledávání klíče

Při vyhledávání klíče se vypočítá jeho identifikátor a v každém kroku se délka společného prefixu mezi hledaným klíčem a aktuálním uzlem zvětší alespoň o 1.

### 2.2.3.6. Výpadek uzlu – rekonfigurace

I při výpadku více uzlů je stále garantováno doručení zprávy, ledaže by $l/2$ uzlů se sousedními identifikátory selhalo naráz ( $l$ je konfigurační parametr s typickou hodnotou 16).

K ošetření výpadku všechny uzly ve svém okolí posílají pravidelně tzv. „keep-alive" zprávy. Pokud uzel neodpovídá po určitou dobu *T* je to považováno za chybu uzlu. Všichni členové listu s chybovým uzlem jsou uvědoměni a začnou aktualizovat svojí sadu listů tak, aby obnovili funkční stav sítě.

### 2.2.3.7. Použití

**PAST**

Služba k trvalému uchování dat. Kopie souborů jsou uložené na uzlu *k* který má identifikátor číselně nejbližší k identifikátoru souboru.

**SCRIBE**

Udržuje velké množství témat. Pokud je téma změněno jsou informování všichni čtenáři tohoto tématu.

## 2.3. Závěr

Po prozkoumání všech známých DHT algoritmů jsem se rozhodl pro simulaci algoritmu KADEMLIA. Tento algoritmus je používán v sítích BitTorrent a na jeho základech pracují i některé další programy.

# 3. Simulace algoritmu Kademlia

## 3.1. Obecný popis

Kademlia byla představena v roce 2002. Má nižší nároky na množství zpráv potřebných k udržení směrovací tabulky díky schopnosti uzlů učit se o ostatních uzlech během procesu vyhledávání.

Používá 160-bitové identifikátory pro klíče i pro uzly seřazené do binárního stromu. Dvojice klíč/hodnota jsou uchovávány v uzlu s dostatečně blízkým identifikátorem. Pro určení vzdálenosti se využívá metrika XOR, díky tomu lze poslat více dotazů na uzly v blízkosti cílového uzlu.

### 3.1.1. Datová struktura

**Kontakt** je trojice tvořená identifikátorem uzlu, IP adresou a číslem portu.

***Přihrádka (k*-bucket)** je zásobník pro kontakty. Kontakty jsou v seznamu řazeny podle toho, kdy se daný uzel ozval naposledy. Uzel, který se ozval nejpozději je na vrcholu seznamu. Pokud uzel dlouho neodpovídá, propadne se na konec seznamu. V případě, že je přihrádka plná jsou nejstarší uzly odstraňovány, aby se do ní vešli nové kontakty.

**Směrovací tabulka** obsahuje pro každý bit z identifikátoru jednu přihrádku, celkem

tedy obsahuje 160 přihrádek. Číslo přihrádky určuje vzdálenost mezi uzlem a uzly uložených v přihrádce.

Kademlia používá čtyři druhy zpráv:

**PING** k ověření že vzdálený uzel je stále aktivní.

**STORE({klíč->hodnota})** k uložení hodnoty na volaný uzel.

**FIND_NODE(identifikátor uzlu)** pro hledání uzlu, příjemce této zprávy vrátí až *k* uzlů, kteří jsou nejblíže k hledanému uzlu. Pomocí opakování zprávy lze nalézt požadovaný uzel.

**FIND_VALUE(klíč)** pro vyhledání hodnoty, je tento příkaz v podstatě stejný jako FIND_NODE. Ovšem vlastní-li uzel danou hodnotu nevrací bližší uzly, ale danou hodnotu.

Odesílatel zprávy přidává ke každé z nich náhodně generovaný 160-ti bitový identifikátor. Uzel, který na zprávu odpovídá, odesílá tento identifikátor zpět. Tím je zaručená odpověď na dotaz a je také sníženo riziko padělání zpráv.

**Směrování** - každým přeskokem se vzdálenost k cílovému uzlu půlí – složitost směrování je tedy O(log2 N) přeskoků.

### 3.1.2. XOR metrika

Kademlia je založena na vnímání vzdálenosti mezi uzly počítané metrikou XOR. Tato vzdálenost mezi dvěma uzly je definována jako XOR operace jejich identifikátorů:

$$\delta(x, y) = x \oplus y$$

$$\delta(x, x) = 0$$

$$x, y : \delta(x, y) = \delta(y, x)$$

$$\forall x, y, x \neq y : \delta(x, y) > 0$$

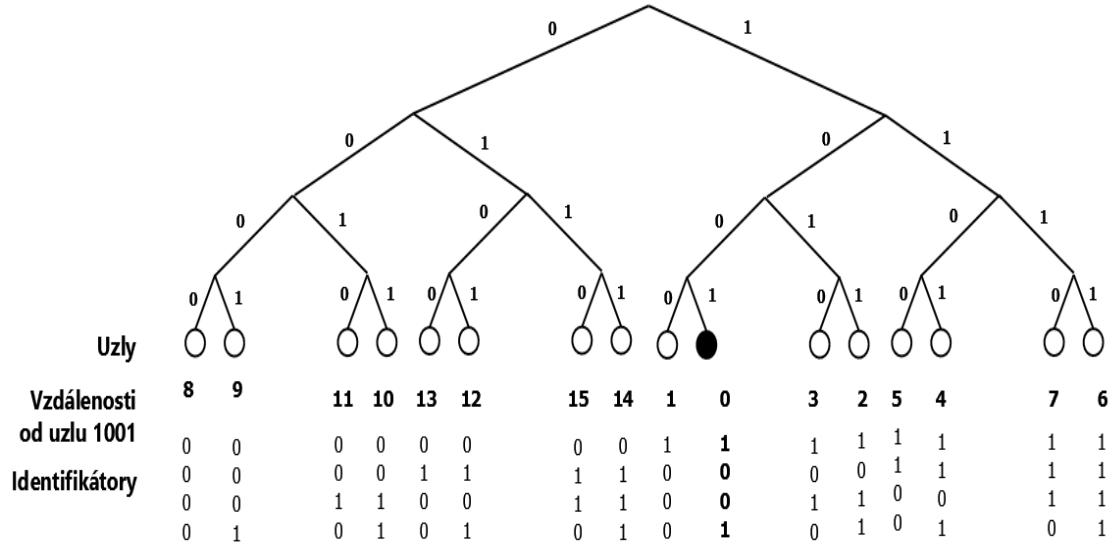
Platí trojúhelníková nerovnost:

$$\delta(x, y) \oplus \delta(y, z) = \delta(x, z)$$

$$\forall a \geq 0, b \geq 0, a \neq b : a + b \geq a \oplus b$$

$$\delta(x, y) + \delta(y, z) \geq \delta(x, z)$$

Pro libovolný uzel *x* a vzdálenost $\Delta > 0$, existuje právě jeden uzel *y* takový, pro který platí $\delta(x, y) = \Delta$. Pro grafické znázornění zobrazení vzdálenosti uzlů mezi sebou v prostoru identifikátorů uzlů lze použít úplný binární strom. Listy stromu představují potencionální identifikátor uzlů. Kademlia používá prostor identifikátorů velikosti $2^{160}$ [1]. Na obrázcích v tomto dokumentu je pro jednoduchost zobrazen prostor klíčů jen o velikosti $2^4$ viz obr. 3.1




*Obr. 3.1: Ukázka 4-bitového prostoru identifikátorů. Vzdálenosti jsou počítány metrikou XOR*

### 3.1.3. Připojení uzlu

Pro připojení do sítě musí uzel *U* znát alespoň jeden uzel, který je již v síti připojen. Pro získání takového uzlu může použít několik možností:

**Seznam stabilních uzlů** distribuovaných s aplikací.

**Použití jiného systému** například DNS nebo IRC.

**Aktivní hledání jako broadcas**t na lokální síti nebo **multicast** na Internetu.

**Zadáním adresy uživatelem** jako pozvánka o jiného uživatele.

Příklad připojení uzlu do sítě:

Uzel *U* se chce připojit do sítě, uzel *W* je již v síti připojen. *U* si vygeneruje pseudonáhodný 160-ti bitový identifikační klíč. V první fázi se snaží navázat spojení alespoň s jedním uzlem připojeným do sítě. Pošle tedy uzlu *W* zprávu FIND_NODE(*U*). Díky tomu obdrží od *W k* nejbližších uzlů, kterým znovu posílá FIND_NODE(*U*). Tímto o sobě dává vědět dalším uzlům, kteří si ho přidávají do svého seznamu. Plné připojení do sítě provede uzel v O(log2 N) krocích. Díky vyhledávání svého identifikátoru, zná uzel více bližších uzlů než vzdálenějších.

### 3.1.4. Odpojení uzlu

Uzly neoznamují svůj odchod ze sítě. Odpojení probíhá automaticky vyřazením z přihrádky při dlouhé nečinnosti. Některé implementace algoritmu KADEMLIA periodicky posílají zprávu PING a poté vyřazují „mrtvé" uzly.

### 3.1.5. Vyhledávání klíče

Vyhledání klíče probíhá identicky jako vyhledání uzlu. Uzel vybere ze své směrovací tabulky *a* nejbližších uzlů, které zná a pošle jim zprávu s hledaným identifikátorem. Uzly se podívají do svých směrovacích tabulek a jako odpověď pošlou uzly s bližším identifikátorem k hledanému klíči. V případě, že daný uzel vlastní hledaný klíč neposílá zpět bližší uzly, ale hodnotu k danému klíči (typicky soubor).

### 3.1.6. Použití

Používá se v síti Kad network, eDonkey, BitTorrent, Osiris sps, Gnutella DHT.

V síti BitTorrent je Kademlia využívána pro hledání ostatních peerů nezávisle na trackerech.

# 4. Simulace pomocí vlastního simulačního programu

V této kapitole bude popsána implementace algoritmu Kademlia ve vlastním simulačním programu.

Simulační program je napsaný v jazyce JAVA verze 8.

## 4.1. Implementace algoritmu Kademlia

### 4.1.1. Směrovací tabulka

Směrovací tabulka obsahuje pole 160-ti přihrádek (k-bucket), každá přihrádka obsahuje pole *k* ukazatelů na uzel. Každý ukazatel na uzel obsahuje identifikátor uzlu a adresu uzlu v rámci simulátoru.

Při obdržení jakékoli zprávy, si uzel aktualizuje směrovací tabulku. Uzel je přidán do příslušné přihrádky, pokud je již v dané přihrádce je přesunut na první místo přihrádky.

Číslo přihrádky udává společnou délku identifikátoru. Pro výpočet dané přihrádky je použita metrika XOR jak již bylo zmíněno výše.

### 4.1.2. Počítání vzdálenosti

V simulaci se nepočítá přímo vzdálenost mezi uzly kvůli nemožnosti uložit 160-ti bitové číslo do jedné proměnné. Pro uložení uzlu do směrovací tabulky se počítá rovnou číslo přihrádky. Pro nalezení vhodných uzlů k uložení souboru se počítá, který ze dvou daných uzlů je blíže.

### 4.1.3. Uzel

Každému uzlu je při jeho vzniku vygenerován pseudonáhodný 160-ti bitový identifikátor, který je uložen do pole typu byte o velikosti 20. Identifikátor je vytvořen pomocí SHA-1 hashe za pomoci standardní knihovny java.security.MessageDigest.

Běh každého uzlu je simulován vlastním běžícím vláknem. Adresa uzlu je dána pozicí v poli uzlů, čímž se usnadňuje adresování zpráv.

Přijímání zpráv je realizováno spojovým seznamem, kdy nově příchozí zpráva se řadí na konec tohoto seznamu. Uzel obsluhuje zprávy v pořadí v jakém mu byly doručeny a páruje je s odpovídajícími požadavky.

### 4.1.4. Zpráva

Zpráva obsahuje adresu příjemce i odesílatele, identifikační číslo odesílatele, druh zprávy a obsah zprávy samotné. Ke každé zprávě se ještě připojuje číslo požadavku.

#### 4.1.4.1. Zpracování zpráv

Obsluhuje se vždy nejstarší doručená zpráva a zpracování zpráv probíhá ve smyčce. Pokud uzel již nemá žádnou přijatou zprávu, kterou by mohl zpracovat uspí vlákno, aby zbytečně nevytěžoval procesor.

### 4.1.5. Požadavek

Požadavek slouží ke správnému zpracování přijaté zprávy a pro statistické údaje, ke každému požadavku se počítá počet odeslaných a přijatých zpráv. Požadavek může být dvojího druhu, buďto JOIN nebo ULOZ_SOUBOR.

Požadavek JOIN slouží pro připojení uzlu do sítě a je ukončen, když uzel zná všechny své nejbližší uzly.

Požadavek ULOZ_SOUBOR je vytvořen ve chvíli, kdy obsluha programu zadá příkaz simulaci pro uložení souboru do sítě. Pro každý ukládaný soubor se vytváří nový požadavek. V požadavku se ukládají informace o tom, které uzly jsem již kontaktoval a zda existuje ještě nějaký bližší uzel pro uložení souboru. Požadavek je ukončen v momentě nalezení nejbližších *k* uzlů k identifikátoru souboru.

### 4.1.6. Statistiky

Třída statistiky se stará o vytvoření složek a souborů, do kterých se vypisují statistiky z

jednotlivých uzlů. Také se stará o převod identifikačního čísla uzlu do čitelné podoby.

## 4.2. Běh simulace

Před simulací je potřeba vytvořit scénář, podle kterého bude simulace probíhat. Scénář je jakýkoliv textový soubor s danou vnitřní strukturou.

Díky uspání vláken bez příchozí zprávy, lze z vytížení procesoru poznat, zda již všechny zprávy byli úspěšně zpracovány.

### 4.2.1. Struktura scénáře simulace

Scénář je rozdělen na dvě části, inicializační a simulační část. Tyto části jsou od sebe odděleny klíčovým slovem *start*. Pokud nejsou inicializační data nastavené simulátor použije implicitní hodnoty. Scénář musí obsahovat klíčové slova *start* a *end* ostatní klíčová slova jsou nepovinná.

V inicializační části se nastavuje cesta ke složce, do které se budou ukládat výpisy a statistiky. Dále počáteční hodnota seedu pro generování pseudonáhodných identifikačních čísel a konstanta *k*.

V simulační části se do simulace přidávají uzly a soubory, tato část je zakončena klíčovým slovem *end*.

Před vložením souborů nebo uzlů do simulace a před jejím ukončením, je uživatel vyzván k potvrzení daného příkazu. Uživatel má tak jistotu, že simulace proběhne v pořádku a nebude například ukončena před úplným zpracováním všech zpráv.

# 5. Analýza naměřených výsledků

Každá simulace byla spuštěna dvakrát a dané výsledky byli zprůměrovány. Doba simulací se pohybovala od 5 minut u 5.000 uzlů, až po 35 minut u simulace 50.000 uzlů. Bohužel kvůli nedostatečné operační paměti při simulaci docházelo k pádu operačního systému při vyšším počtu uzlů (~ >60.000 uzlů) nebo vyšší konstantě *k (~ >15)*, proto nejsou tyto hodnoty uvedeny v analýze.

Pro výpočet průměrného počtu zpráv nutných k uložení souboru se simulovalo uložení 100 souborů. Stejně tak bylo použito i 100 uzlů pro výpočet průměrného počtu zpráv nutných k připojení k síti.

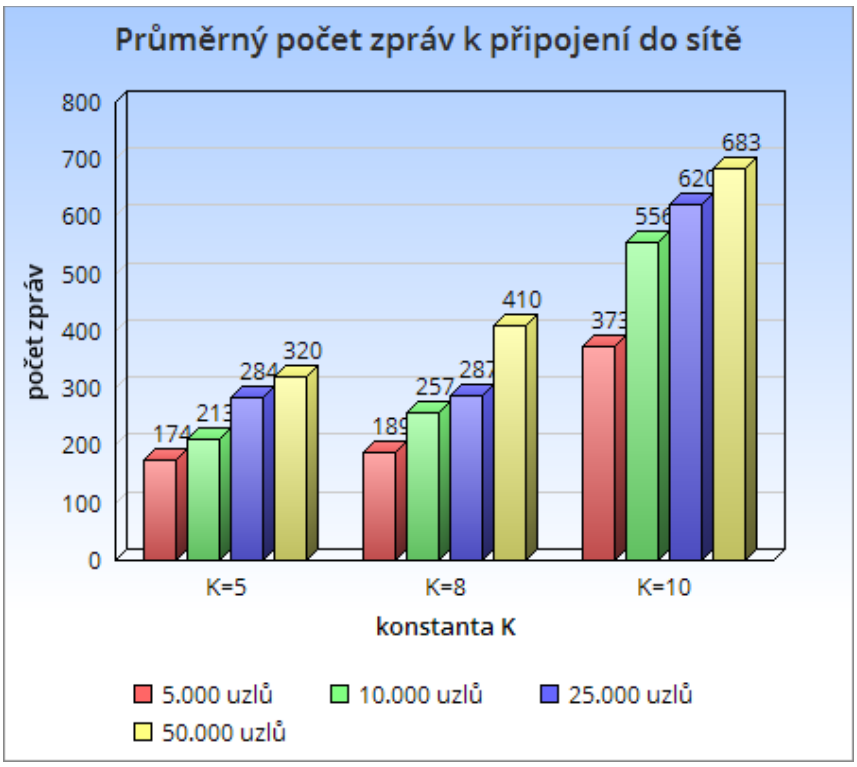
Veškeré simulace probíhali na počítači s parametry: 10GB RAM DDR3 , procesor AMD Phenom II X4 945 3,0GHz a operačním systémem Microsoft Windows 8.1 x64.

## 5.1. Vliv změny konstanty K na počet zpráv

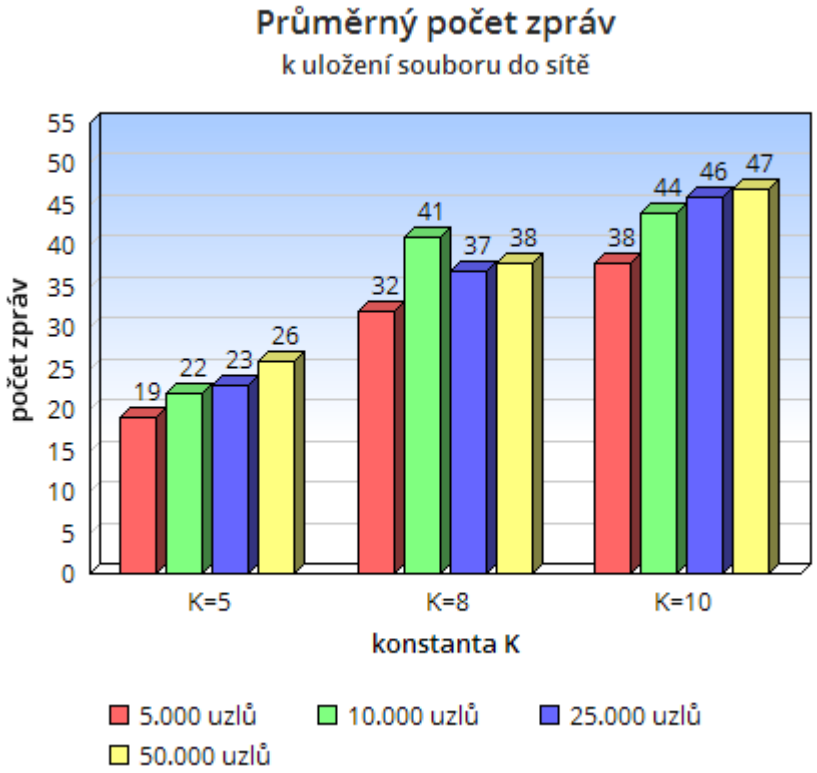
Čím je konstanta *k* vyšší tím je síť odolnější proti výpadkům uzlů. Zároveň, ale vzrůstají nároky na počet uzlů, se kterými je třeba udržovat kontakt a mít na ně uložený odkaz. Pro spolehlivé systémy se používá hodnota *k*=20. Síť BitTorrent, která používá upravenou verzi DHT kademlia používá *k=3*.

Na získaných datech ze simulátoru lze vidět, že zdvojnásobením konstanty *k* z hodnoty 5 na 10 se průměrný počet zpráv potřebných k připojení do sítě více než zdvojnásobil. Stejně tak došlo i ke zdvojnásobení průměrného počtu zpráv pro uložení souboru do sítě část.

Nárůst počtu průměrného počtu zpráv je vidět na obr. 5.1 a obr. 5.2.




*Obr. 5.1: Graf průměrného počtu zpráv potřebných k připojení k síti*




*Obr. 5.2: Graf průměrného počtu zpráv potřebných k připojení k síti*

# 6. Závěr

Při realizaci vlastního simulátoru jsem několikrát narazil na vážnou chybu v návrhu aplikace což mě vždy stálo spoustu času a energie, abych vše napravil. Jednou jsem dokonce zahodil téměř celý napsaný zdrojový kód a začal psát aplikaci od začátku. Původní představa, že budu moci simulovat až sta tisíce uzlů byla velmi přehnaná nicméně se mi podařilo vytvořit funkční simulátor pro základní charakteristiky algoritmu KADEMLIA.

Dalším možným pokračováním je optimalizace kódu, pro zmenšení nároků na operační paměť počítače při simulaci velkého počtu uzlů.

# Přehled zkratek

P2P – Peer-to-Peer volně přeloženo jako rovný s rovným

DHT – distribuované hashovací tabulky

# Použitá literatura

[1] TVRDÍK, P. *Implementace BitTorrent discovery protokolu do Clondike*. Praha, 2014. Diplomová práce na České vysoké učení technické v Praze. Fakulta informačních technologií. Vedoucí práce Josef GATTERMAYER.

[2] NOVOTNÝ, Miroslav. *Peer-to-Peer sítě* [online]. 2009. [citováno 10. listopadu 2015] Dostupné z http://ulita.ms.mff.cuni.cz/pub/predn/PDS/DHT.pdf

[3] LEDVINA, Jiří. *Strukturované a nestrukturované P2P sítě, DHT* [online]. 2008. [citováno 12. prosince 2014] Dostupné z http://www.kiv.zcu.cz/~ledvina/Prednasky-DS-2008/DS-12-P2Pa.pdf

[4] RATNASAMY, Sylvia a FRANCIS, Paul a HANDLEY, Mark a KARP, Richard a SHENKER, Scott. *A Scalable Content-Addressable Network* [online]. 2001. [citováno 10. listopadu 2014] Dostupné z http://www.eecs.berkeley.edu/~sylvia/papers/cans.pdf

[5] SARMADY, Siamak. *A Survey on Peer-to-Peer and DHT* [online]. [citováno 12. prosince 2014]. Dostupné z http://arxiv.org/ftp/arxiv/papers/1006/1006.4708.pdf

[6] GHOSI, Ali. *Distributed k-ary System: Algorithms for Distributed Hash Tables* [online]. 2006. [citováno 12. prosince 2014] Dostupné z https://www.sics.se/~ali/thesis/dks.pdf

[7] STOICA, Ion a MORRIS, Robert a KARGER, David a KAASHOEK, M. Frans a BALAKRISHNAN, Hari. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications* [online]. 2001. [citováno 18. prosince 2014] Dostupné z

http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[8] STOICA, Ion a MORRIS, Robert a KARGER, David a KAASHOEK, M. Frans a BALAKRISHNAN, Hari. *LOOKING UP DATA IN P2P SYSTEMS* [online]. [citováno 18. prosince 2014] Dostupné z http://www.cs.berkeley.edu/~istoica/papers/2003/cacm03.pdf

[9] PITA, Isabel. *A formal specification of the Kademlia distributed hash table* [online]. [citováno 18. prosince] Dostupné z http://maude.sip.ucm.es/kademlia/files/pita_kademlia.pdf

[10] SPORI, Bruno. *Implementation of the Kademlia Distributed Hash Table [online].* 2006.[Citováno 10. ledna 2015] Dostupné z ftp://ftp.tik.ee.ethz.ch/pub/students/2006-So/SA-2006-19.pdf

[11] MAYMOUNKOV, Petar a MAZIÉRES, David. *Kademlia: A Peer-to-peer Information System Based on the XOR Metric [online]. [citováno 10. ledna 2015] Dostupné z http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf*

# Přílohy

**Příloha A. - struktura scénáře simulace**

-- komentar

-- jeden radek jeden prikaz

-- oddelovacem mezi prikazem a parametrem je mezera

-- INICIALIZACNI CAST

-- nastavi cestu ke slozce do ktere se maji ukladat statistiky

stat_path D:\Sync_ondra\ZCU_Bakalarka\Simulator\VYSLEDKY\

-- nastaveni seedu pro generovani nahodnych cisel  {0}

seed 33

-- konstantaK {20}

konstK 10

--  SIMULACNI CAST

-- start simulace a ukonceni nastaveni konfigurace

start

-- pripojit do site # uzlu

peer 50000

-- připojí počet uzlů, kteří budou vypisovat statistiky (hlavně změny ve směrovací tabulce)

node_stat 1

-- pridat do site soubor, file #pocetSouboru #adresaUzlu

file 100 5

-- cekani v sekundach

wait 10

-- konec simulace

end