

Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

Simon Csaba

TMIT

2019

Strukturált P2P

P2P Keresés

- Strukturálatlan P2P rendszerek
 - Gnutella, KaZaa, stb...
 - Véletlenszerű keresés
 - Nincs információ a fájl lehetséges tárolási helyéről
 - Nagy terhelés a rendszeren
 - Minél több helyen tárolva, annál kisebb a keresés által generált terhelés

P2P Keresés (II)

- Strukturált P2P rendszerek

- Irányított keresés

- Kijelölt peer-ek kijelölt fájlokat (vagy azok fele mutató pointereket) kell tároljanak
 - Mint egy információs pult
 - Ha valaki keresi a fájlt, tudja kit kérdezzen

- Elvárt tulajdonságok

- Elosztottság

- Felelősség elosztása a résztvevők között

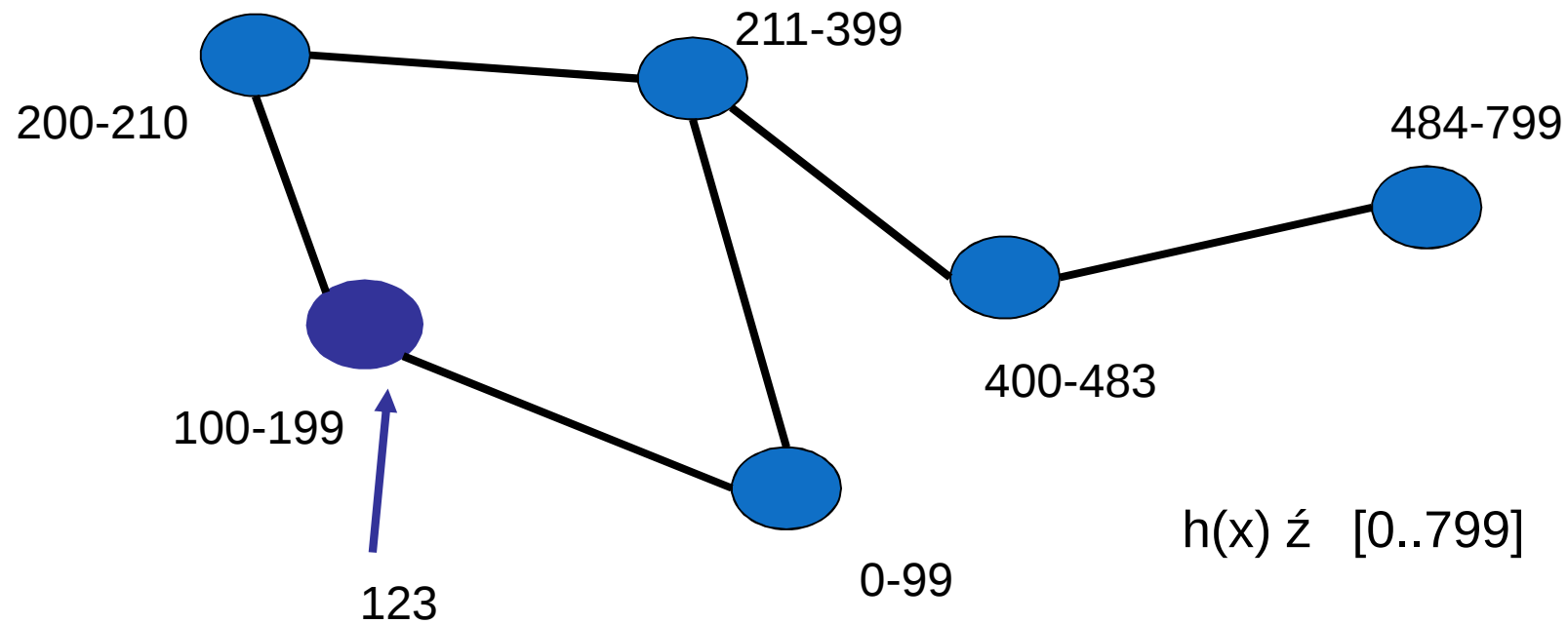
- Alkalmazkodás

- A peer-ek ki és bekapcsolódnak a rendszerbe
 - Az új peer-eknek feladatokat osztani
 - A kilépő peer-ek feladatai újraosztani

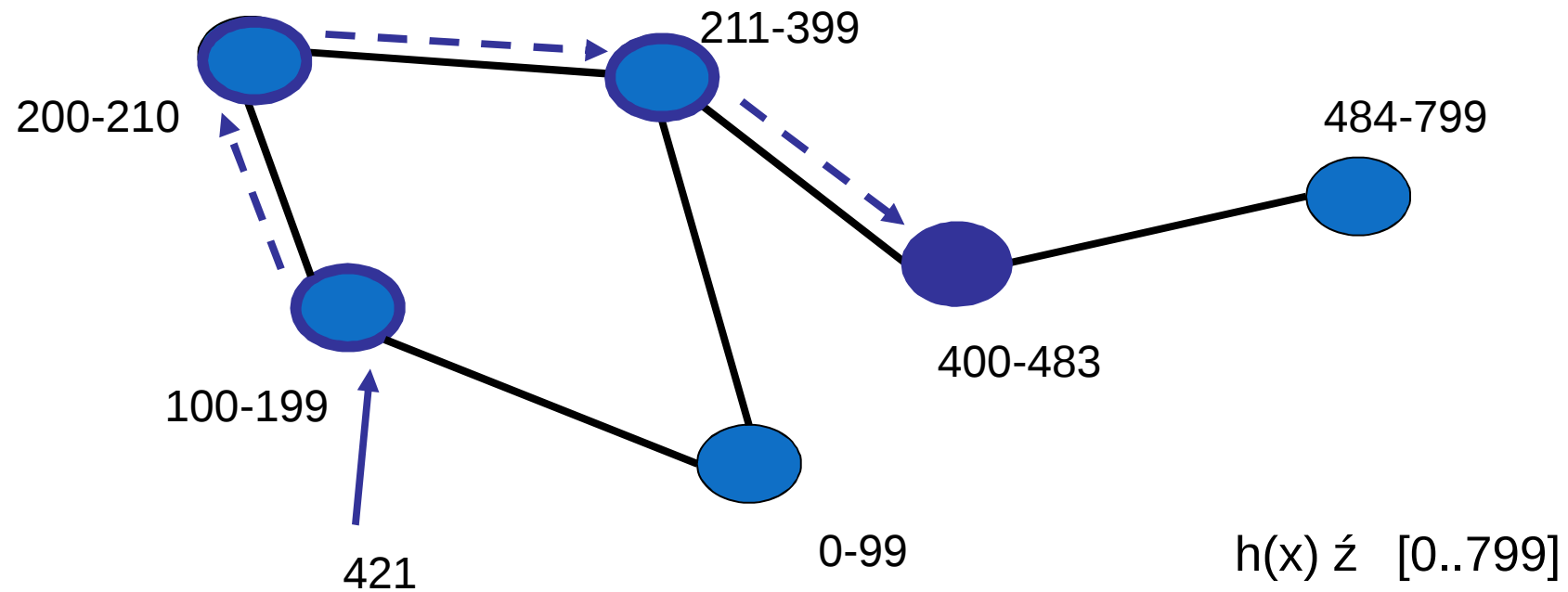
Elosztott hash táblák - DHT

- Distributed Hash Tables
- Egy hash függvény a keresett fájlhoz egy egyéni azonosítót társít
 - Példa: $h(„P2P_előadás”) \rightarrow 123$
- A hash függvény értékkészletét elosztja a peer-ek között
- Egy peer-nek tudnia kell minden olyan fájlról, melynek a hash-elt értéke a saját értékkészletébe tartozik
 - Vagy saját maga tárolja a fájlt....
 - Vagy tudja ki tárolja a fájlt.

DHT példa



DHT példa



DHT útválasztás

- Minden peer mely információt tárol egy fájlról elérhető
- kell legyen egy „rövid” úton
 - Korlátozott számú lépésen belül
- Korlátozott számú szomszéd
 - Skálázható a rendszer méretétől függetlenül
- A különböző DHT megoldások lényégében az
- útválasztásban térnek el
 - CAN, Chord, Pastry, Tapestry, Kademlia

CAN

- <https://www.tribler.org/Khashmir/>

Kademlia

Kademlia: A peer-to-peer information system based on the XOR metric

- Petar Maymounkov and David Mazieres
- New York University
- *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 53-65, March 2002.



- Kademlia does not have a main implementation called Kademlia, instead the Kademlia DHT has been implemented in different languages such as C (KadC), Python (Kashmir), and Java (Plan-x)

Kademlia

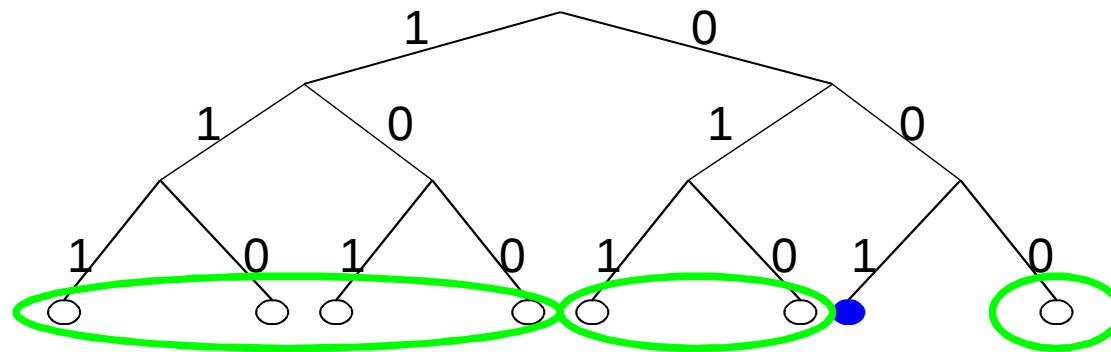
- P2P adattároló hálózat
 - Kulcs-érték párok tárolása
 - egyedi, 160 bites kulcsok
 - DHT
- Kialakítandó környezet (aka specifikáció)
 - Bármelyik node bármikor eltűnhessen
 - A node-ok legyenek egyenletesen terheltek
 - tárolási kapacitás
 - sávszélesség
- Cél
 - Gyors adatelérés
 - Minimális számú vezérlő üzenet

Kademlia

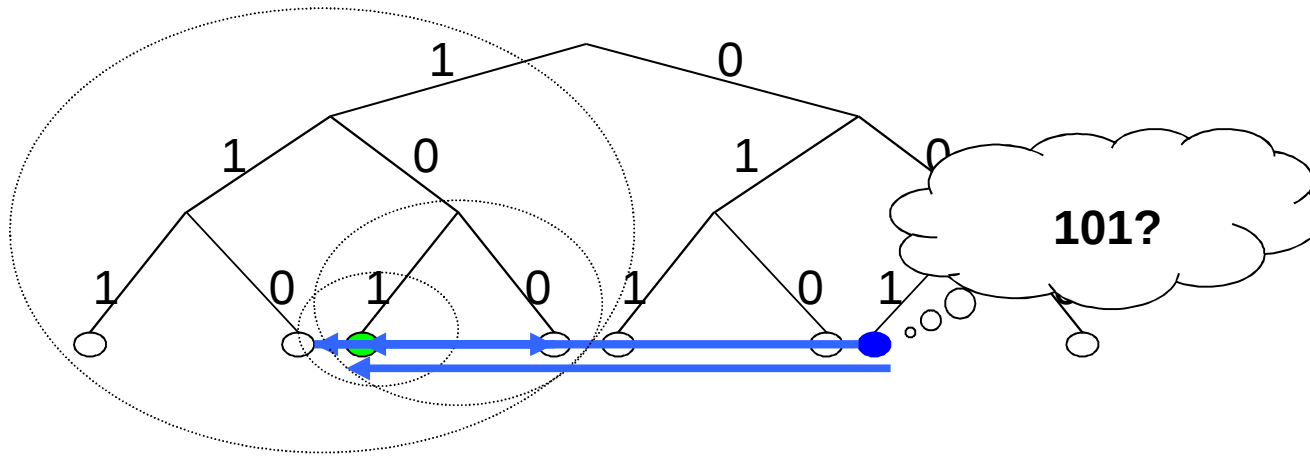
- A routing-tábla rugalmas
 - Közelségi alapú (proximity-based) routing
 - Laza követelmények
 - minimális fenntartás
- Bejövő és kimenő üzenetek eloszlása azonos
 - A hálózat önmegerősítő
- Csak $\log n$ node elérését kell ismerni

Routing tábla

- Rugalmas kialakítás
 - Mindegy, hogy kit ismerek az adott tartományban
- A kimenő és bejövő eloszlás azonos



Kademlia - routing



- Minden hop eggyel kisebb ágra visz a cél körül
 - Az adott ágon belül bármelyik node-hoz ugorhatunk

XOR - topológia

- Definíció: $d(x,y) = x \text{ XOR } y$
 - Szimmetrikus: $d(x,y) = d(y,x)$. Kevesebb vezérlő üzenet
 - Egyértelmű: minden $x \in \{0,1\}^{160}$ és $1 \leq N$ -hez pontosan egy olyan $y \in \{0,1\}^{160}$ létezik, hogy $d(x,y) = 1$.
- A nagyobb helyiértékű bitek eltérése nagyobb távolságot
- eredményez:
 - $x = \underline{0}10\underline{1}01$
 - $y = \underline{1}10\underline{0}01$
 - $x \text{ XOR } y = 100100$, azaz $32 + 4 = 36$
- Geometriai értelmezés:
 - A közös ágon lévő csomópont közelebb van, mint a többi

Alkalmazások

- A Kademliát használó néhány alkalmazás:

- Overnet háló: Overnet, eDonkey, mlDonkey
 - 2006-ban 645.000 párhuzamos felhasználó
 - A RIAA betiltatta 2006-ban
 - Az eredeti eD2k hálót helyettesítette



- Kad háló: eMule, mlDonkey, aMule
 - eMule fejlesztői az Overnet helyett saját Kademlia implementációt készítettek
- Directconnect: RevConnect
- Trackerless BitTorrent: Azureus, BitSpirit, eXeem

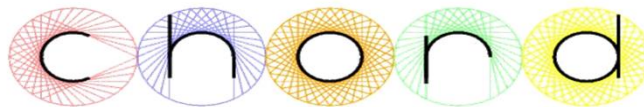


Összegzés - 1

- Strukturálatlan vs strukturált P2P
- Tervezhető, algoritmizálható rendszer
- Nincs központi entitás - hurrá
 - És ez nem tetszik a felhasználóknak - búúú
- CAN – szép elméleti példa
- Kademlia és Tapestry – használ(t/j)ák
- Chord to come...

Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

Ion Stoica, Robert Morris, David Karger,
M. Frans Kaashoek, Hari Balakrishnan - MIT



Web szerver terhelésének elosztása

- Mirroring

- A teljes tartalom duplikálva több helyen
 - Nagy forgalom, nagy adattárolási kényszer
 - Nagy terhelés minden mirror oldalon
 - **Az összes európai ugyanahhoz az európai mirror szerverhez fordul amikor az USA-ban alszanak, az amerikai mirror szerver pedig kihasználatlan**

- Caching

- A tartalmat részekre vágjuk, és elosztottan tároljuk
 - A népszerű tartalmakat mindenhol tárolják
 - Ha nincs nálam, tudom kitől kell kérni
- Hagyományos hash függvény
 - $C = \text{hash}(o) \bmod N$
C – cache, o – object, N – cache szerverek száma
- Probléma: Ha egy cache eltűnik, a teljes tartalmat újra kell osztani
 - $C' = \text{hash}(o) \bmod (N-1)$

Consistent hashing

- D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, R. Panigrahy,
- „Consistent hashing and random trees: distributed caching protocols for
- relieving hot spots on the World Wide Web”, Proceedings of ACM Symposium
- on Theory of Computing, El Paso, Texas, 1997.
- <http://theory.lcs.mit.edu/~karger/Papers/web.ps.gz>
- Tim-Berners Lee és T. Leighton eredeti ötlete

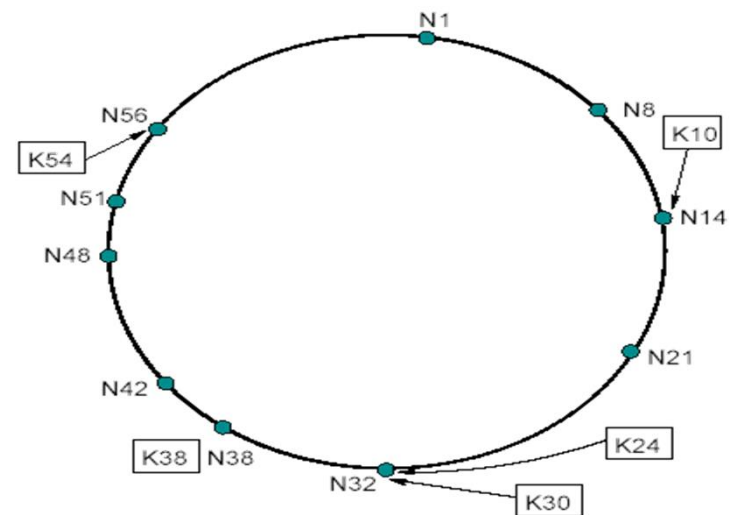


Consistent Hashing

- m bites azonosító tér a fájlloknak és a csomópontoknak
 - m tetszőleges szám, elég nagy, hogy az ütközés valószínűsége kicsi legyen
- Fájl azonosító = $H(\text{Kulcs})$
 - Kulcs = “Let_It_Be”, $H(\text{Kulcs}) = 60$
- Csomópont azonosító = $H(\text{IP cím})$
 - IP = “198.10.10.1”, $H(\text{IP cím}) = 123$
- Egyenletes eloszlással
- Hogyan lehet a fájl azonosítókat a csomópont azonosítókhoz rendelni?

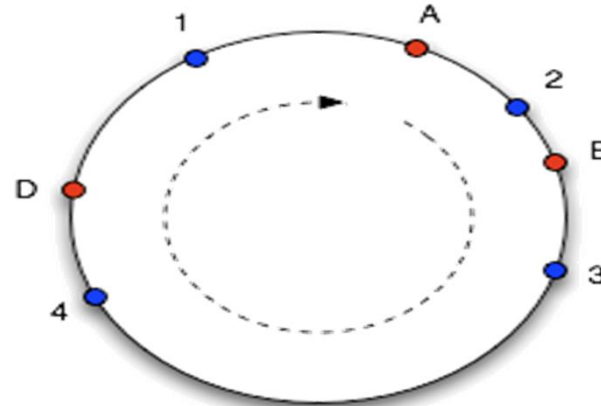
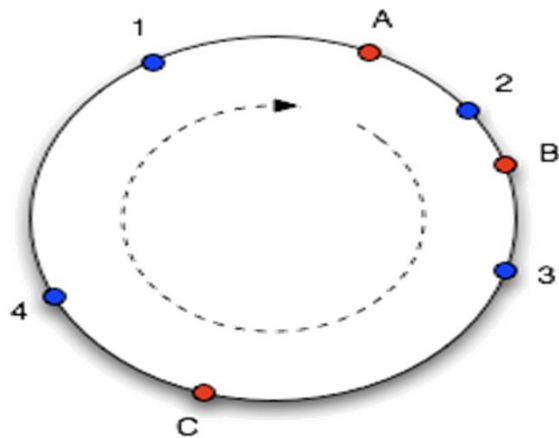
Consistent Hashing

- Azonosítók egy *azonosító gyűrű* mentén
- elhelyezve modulo 2^m
 - Példa: $m = 6$
- Minden K kulcs az őt követő legközelebbi
- N csomópontnál kerül tárolásra
 - $N = \text{successor}(k)$



Consistent Hashing

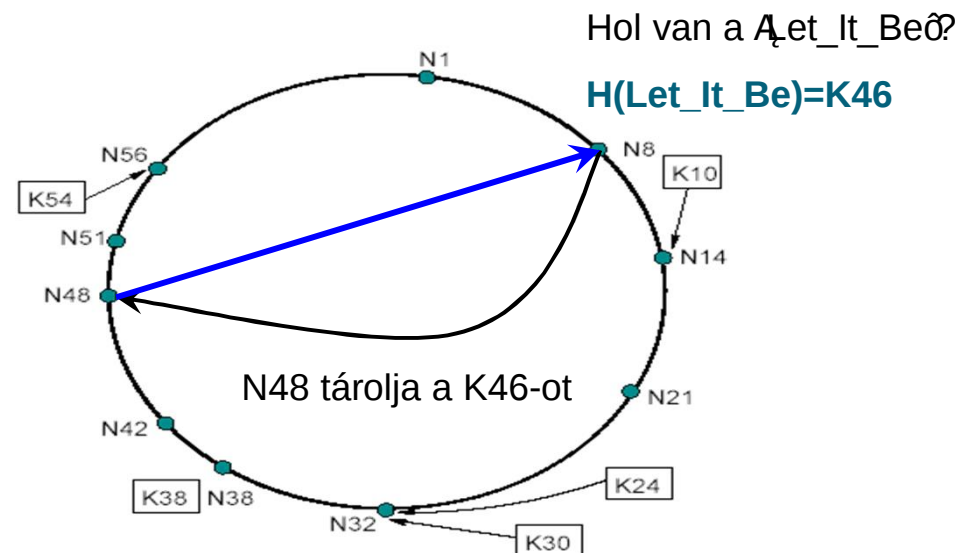
- A csomópont felelős az 1. és 4. kulcsokért
- Ha C kilép, A lesz felelős a 3. kulcsért is
- Ha D belép, átveszi a felelősséget a 3. és 4. kulcsért
 - A többi megfeleltetés változatlan



Consistent hashing - keresés

Minden csomópont ismeri *az összes többi* csomópontot

- Nagy „útválasztó” táblák - $O(N)$
- *Gyors* keresés $O(1)$

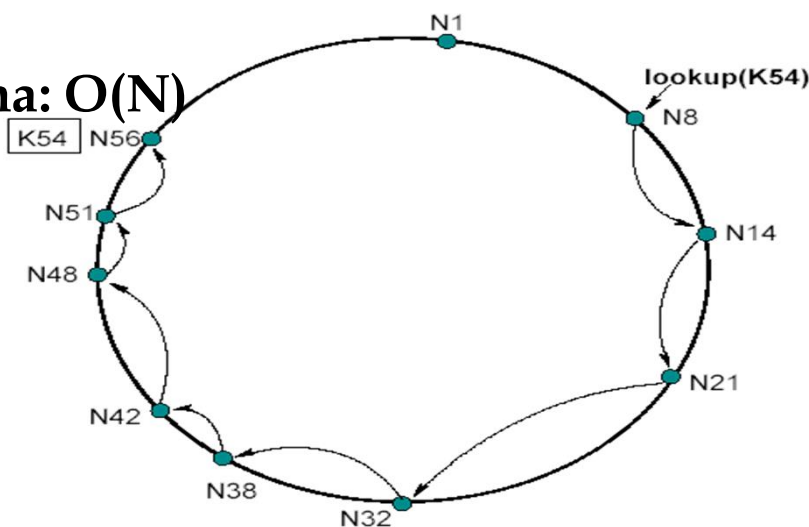


Chord: alap keresés

Minden csomópont ismeri az őt követőt a gyűrűn

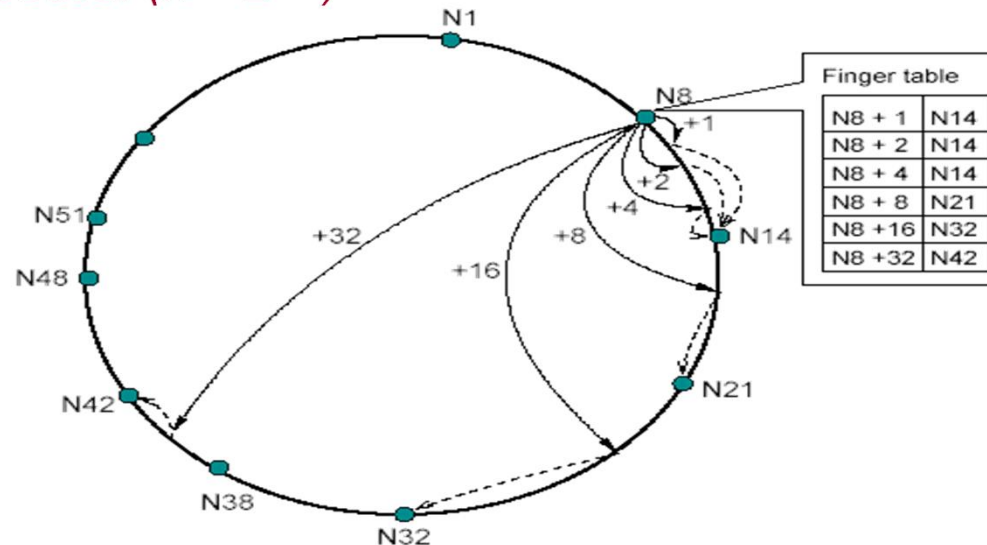
- Az őt megelőzőt is hasznos ismernie

Keresési idő ~ üzenetek száma: $O(N)$



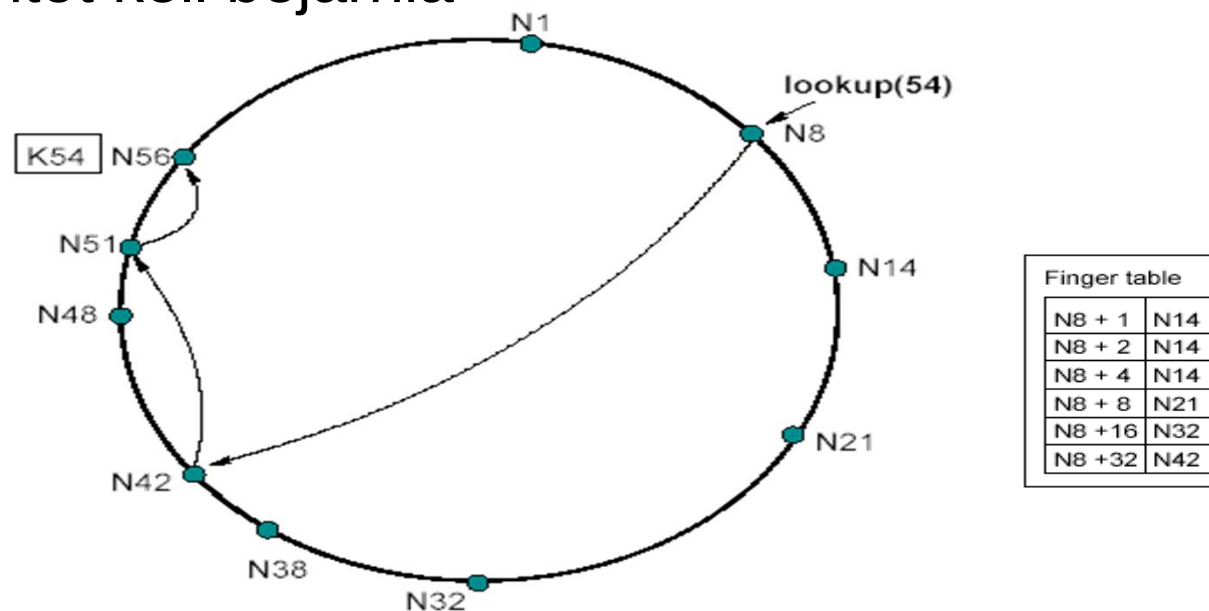
„Mutató táblák” (Finger tables)

- Minden egyes csomópont m számú további csomópontot tart nyilván
- Az előre mutató távolság exponenciálisan növekszik
 - $finger[i] = successor(n + 2^{i-1})$



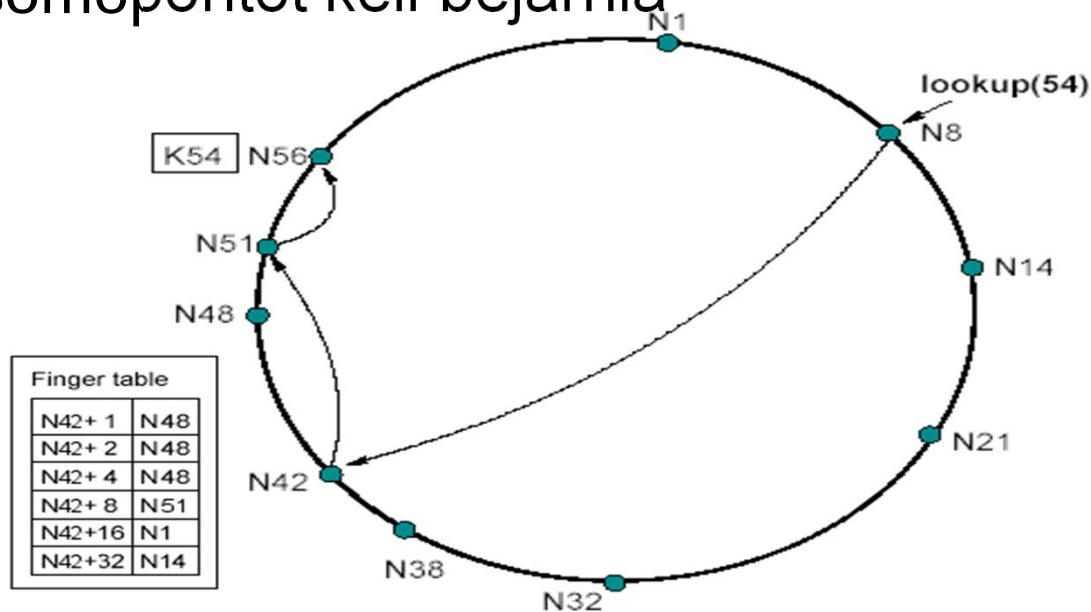
Chord: gyors/skálázódó keresés

- A mutató táblák segítségével a keresésnek $O(\log N)$ csomópontot kell bejárnia



Chord: gyors/skálázódó keresés

- A mutató táblák segítségével a keresésnek $O(\log N)$ csomópontot kell bejárnia



Chord: gyors/skálázódó keresés

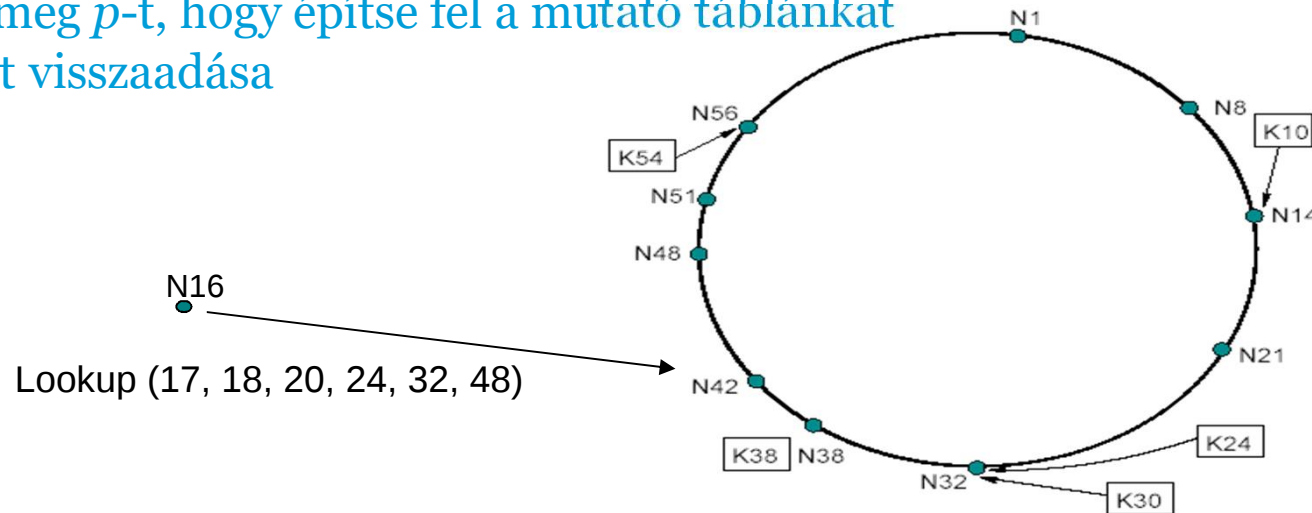
- Minden csomópont m további bejegyzést tartalmaz
- Minél közelebbi a kulcs, annál részletesebb
- információval rendelkezik róla a csomópont
- Általában nem biztosítja az azonnali célba jutást

Új érkező

- Három lépésben (alap működés)
 - Újonnan érkező *mutató táblájának* feltöltése
 - Gyűrű csomópontok *mutató táblájának* frissítése
 - Kulcsok cseréje
- „Lusta” vagy kevésbé agresszív működés
 - Csak a *követő* csomópont beállítása
 - Periodikus *követő (successor)*, *megelőző (predecessor)* ellenőrzés
 - Periodikus *mutató tábla* frissítés

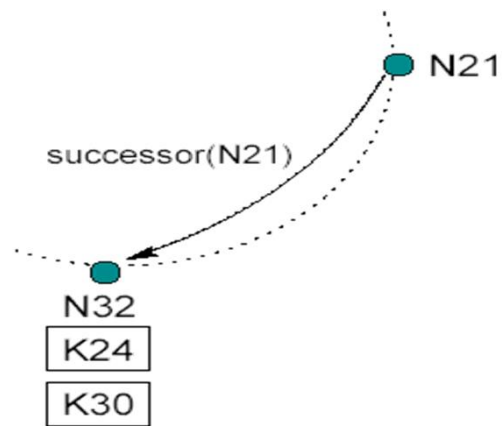
Új érkező (mutató táblák)

- Kiindulás: bármely p ismert csomópontból
 - Kérjük meg p -t, hogy építse fel a mutató táblánkat
 - Táblázat visszaadása



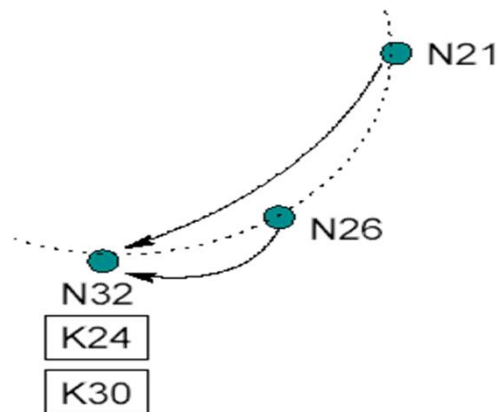
Új érkező

A gyűrű csomópontok mutató tábláinak frissítése



- új érkező a *frissítés* funkciót kelti életre a szomszédos csomópontokban
- csomópontok rekurzívan frissítetik a további csomópontok mutató tábláit

Új érkező



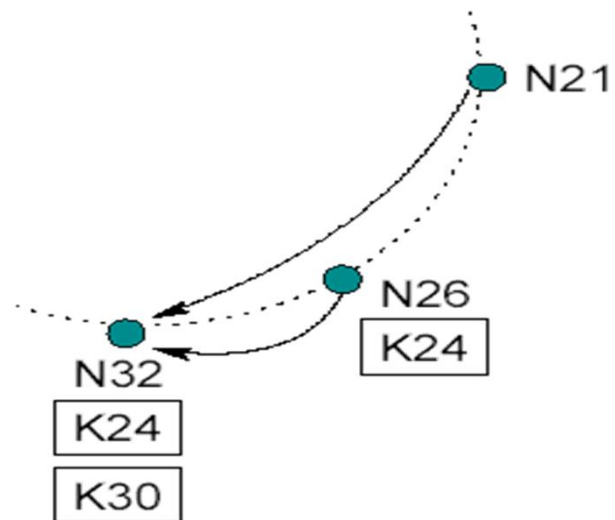
ÇN26 belép a rendszerbe

ÇN26.successor = N32

ÇN26 értesíti N32-t

ÇN32.predecessor = N26

Új érkező

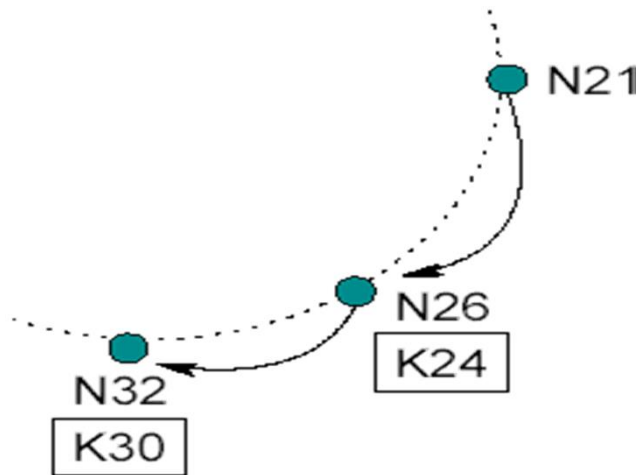


ÇN26 átmásolja a ráeső kulcsokat

ÇN21.frissítés:

Çlekéri N32-től a predecessor-t,
aki N26

Új érkező



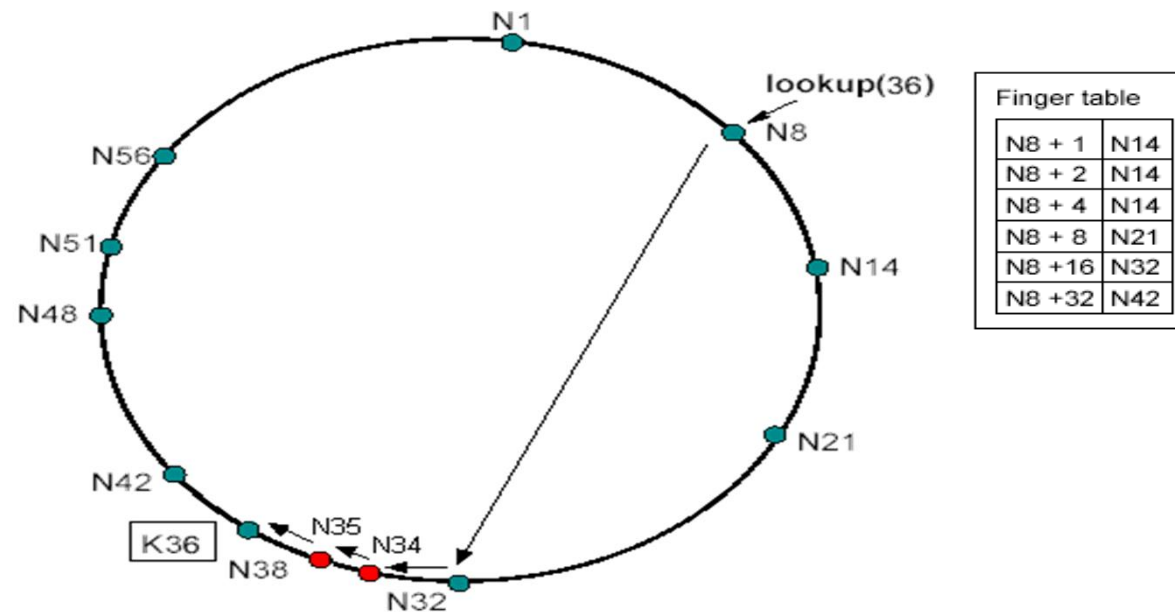
ÇN21.successor = N26

ÇN21 értesíti N26-ot a létezéséről

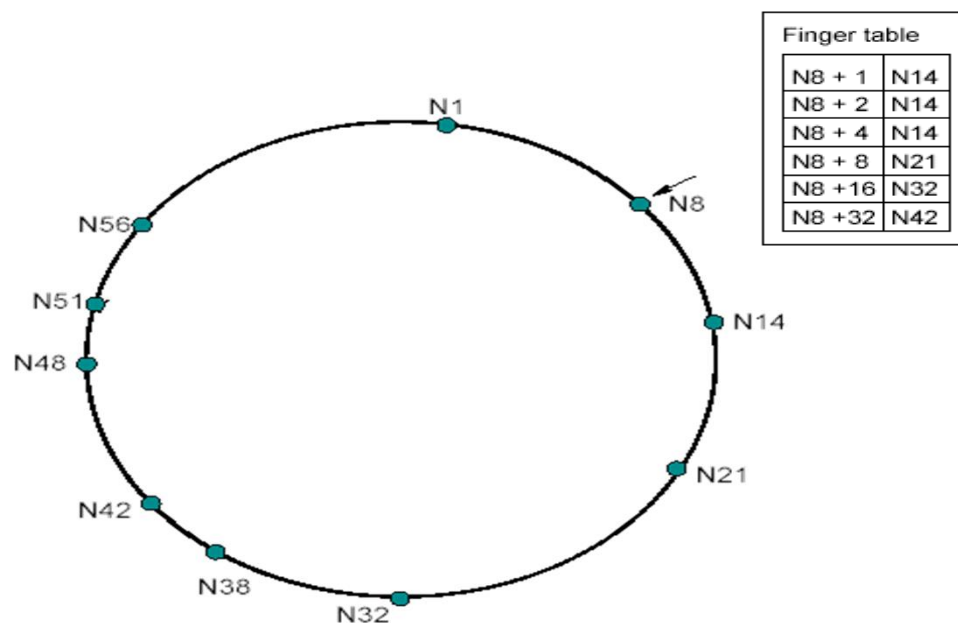
ÇN26.predecessor = N21

Új érkező: keresés

- Korrekt *mutató táblák* esetén $O(\log N)$
- Ha csak a *követő* lánc helyes, akkor is korrekt, de lassabb működés

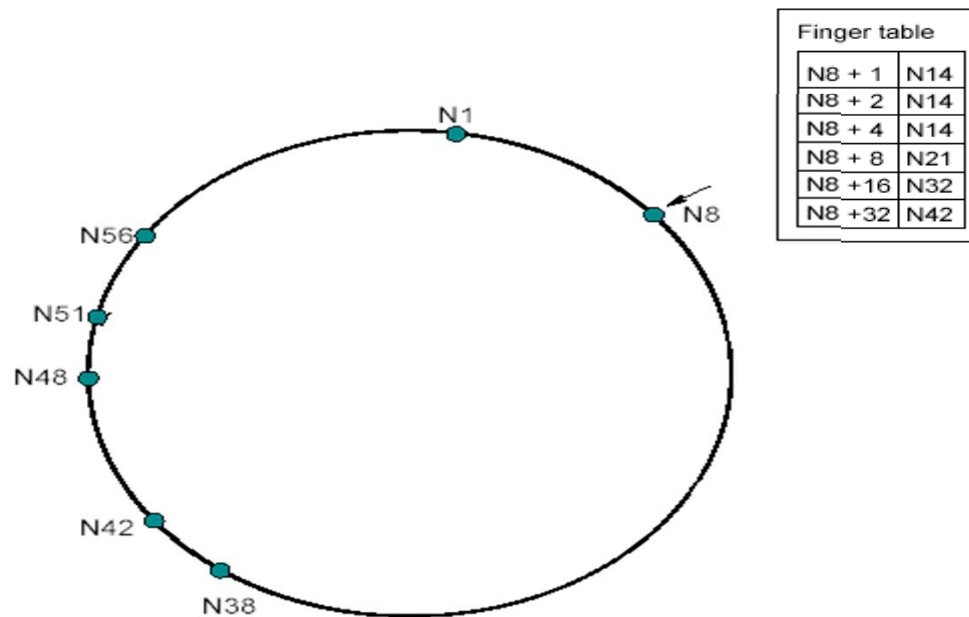


Hibák kezelése



- Csomópontok kiesése (hiba) helytelen keresést eredményezhet
- Mi van ha az N14, N21 és N32 egyszerre meghibásodik, vagy kilép?
- Hogyan tud az N8 tudomást szerezni az N38-ról?

Hibák kezelése



- Csomópontok kiesése (hiba) helytelen keresést eredményezhet
- Mi van ha az N14, N21 és N32 egyszerre meghibásodik, vagy kilép?
- Hogyan tud az N8 tudomást szerezni az N38-ról?

Hibák kezelése (II)

- Követő *lista*
 - Az egyetlen *követő* helyett r soron követő csomópont regisztrációja
 - Hiba esetén ismeri a soron következő (élő) csomópontot
 - helyes keresés
- Valószínűségi garancia
 - r megválasztása, hogy a keresési hiba valószínűsége megfelelően alacsony legyen
 - $r \sim O(\log N)$

Chord előnyök

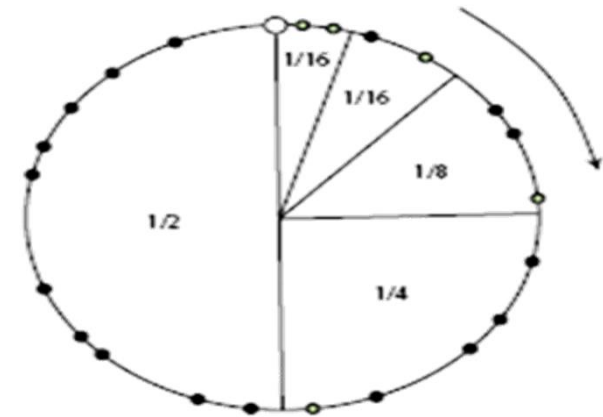
- **Hatékony:** $O(\log N)$ üzenet keresésenként
 - ahol N a kiszolgálók (csomópontok) száma
- Alacsony szórás a keresési időben
 - A CAN-ben nagy a szórás
- **Skálázódik:** $O(\log N)$ állapot csomópontonként
- **Robosztus:** megbirkózik jelentős résztvevő változással
- Állítások bizonyításai [tech_report]
 - <http://www.pdos.lcs.mit.edu/chord/papers/chord-tn.ps>
- Feltételezés: nincs rosszakarátú résztvevő

Chord előnyök

- Minden csomópont K/N kulcsot kezel (N - „node”, K - „key”)
- Be/kilépés egy N-tagú hálózatba: csak $O(K/N)$ kulcsot kell átmozgatni
 - Csak a dinamikus csomópont számára
- Egy keresés $O(\log N)$ üzenetet generál
- Be/kilépés után a mutató táblák frissítéséhez $O(\log^2 N)$ üzenet szükséges

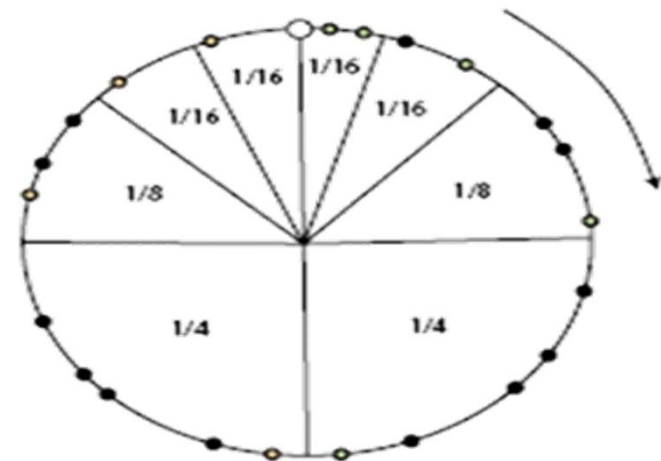
Hátrányok

- Merev finger (routing) tábla!
 - Nehezíti a tábla helyreállítását node-ok kiesése után
 - Lehetetlenné teszi a közelségi információ felhasználását
- A bejövő és kimenő irányú üzenetek eloszlása éppen
- ellentétes
 - Nem lehet a bejövő forgalmat a routing tábla frissítésére használni



Bidirectionnal Chord

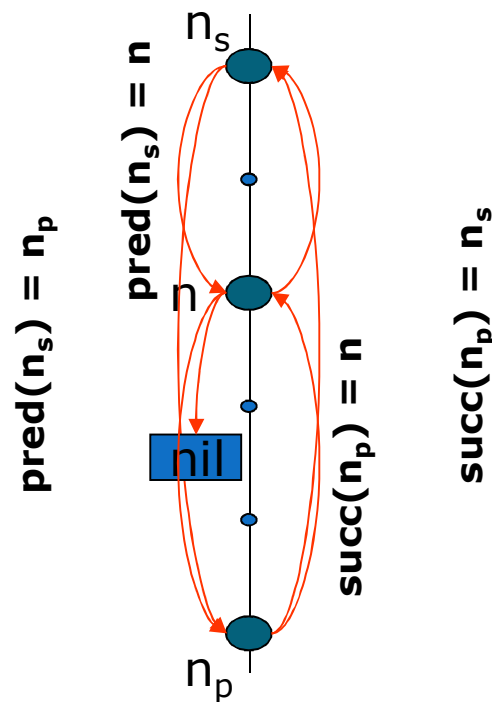
- Kétirányú routing tábla
 - A tábla kétszer akkora nő
 - Kétszer annyi vezérlő üzenetre van szükség



Chord implementáció

- 3000 soros C++ kód
- Library amely tetszőleges alkalmazáshoz linkelhető
- **UsenetDHT**
 - A teljes replikálás helyett elosztott tárolás
- **OverCite**
 - A CiteSeer elosztott változata

stabilize()



- **New n joins the DHT**
 - predecessor = nil
 - n acquires n_s as successor via some n'
 - n notifies n_s being the new predecessor
 - n_s acquires n as its predecessor
- **n_p runs stabilize**
 - n_p asks n_s for its predecessor (now n)
 - n_p acquires n as its successor
 - n_p notifies n
 - n will acquire n_p as its predecessor
- **all predecessor and successor pointers are now correct**
- **fingers still need to be fixed, but old fingers will still work**

fix_finger()

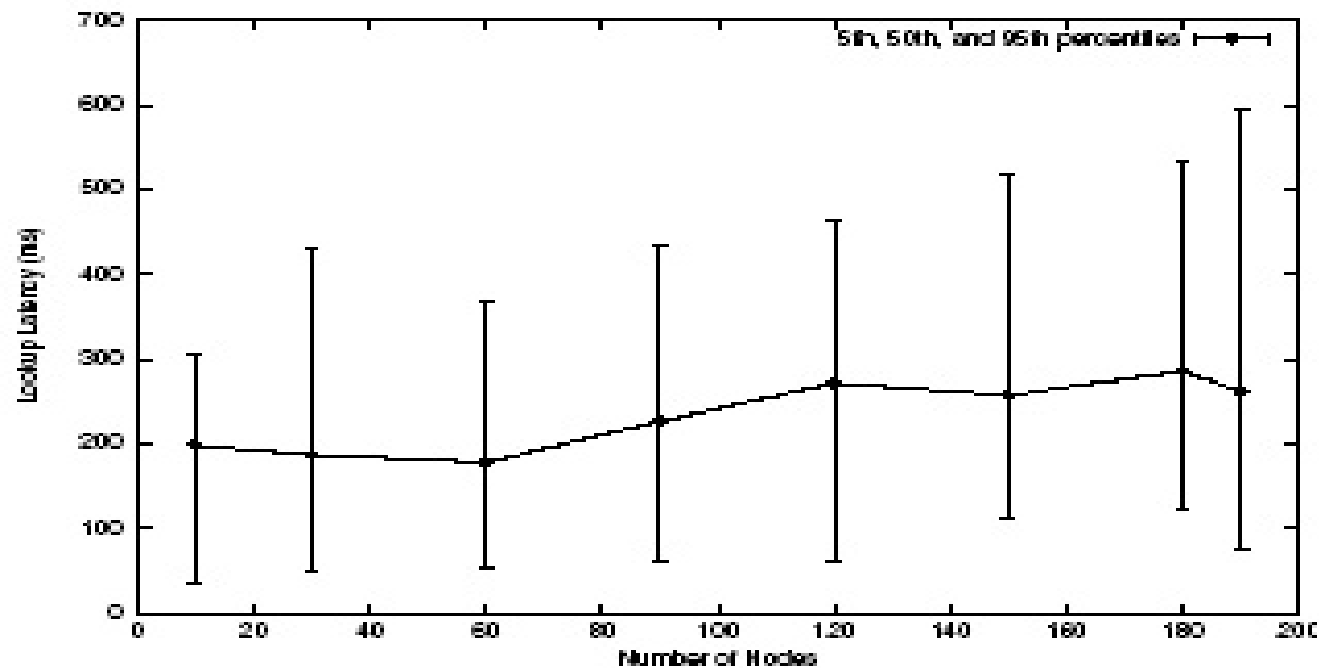
- Why do we have to fix fingers?
 - If we are operating the Chord by the numbers?

fix_finger()

- Why do we have to fix fingers?
 - If we are operating the Chord by the numbers?
- Who notifies the finger table-owners about a leaving member?
 - Predecessors + stabilize() are not handling this issue
 - „take the french leave”
 - Non-transitive connectivity.
 - Bidirectional Chord is not enough (stochastic)
- Refresh the entries of a finger table
 - Otherwise the time of a search increases (greedy solutions for failover)

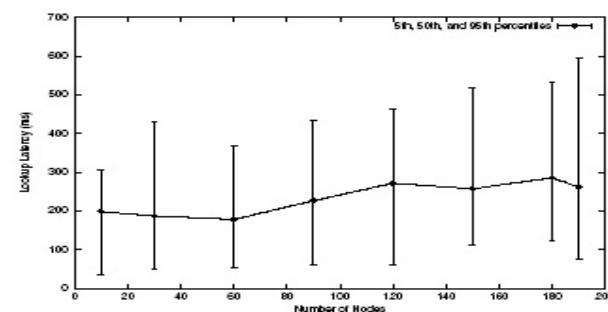
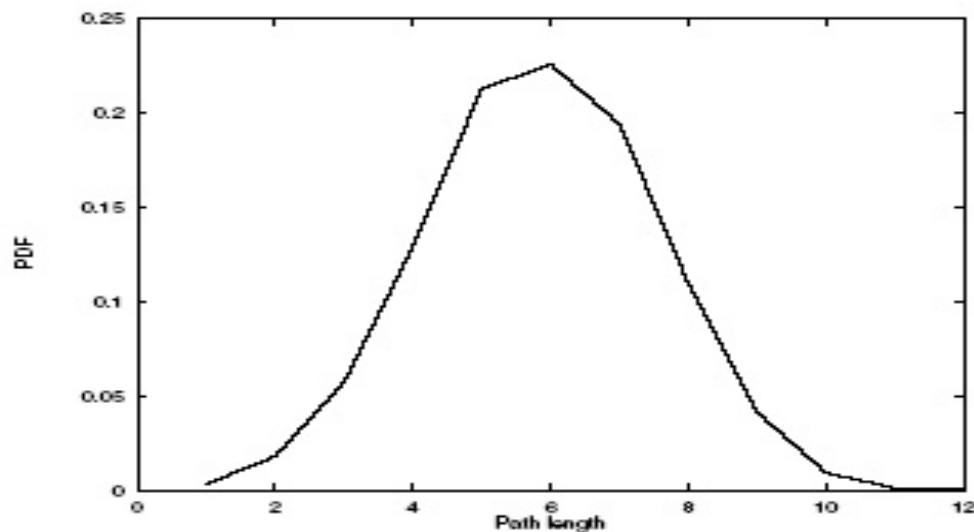
Mérési eredmények

- A késleltetés csak lassan növekszik, ha növeljük a csomópontokat



Mérési eredmények

- A késleltetés csak lassan növekszik, ha növeljük a csomópontokat
- Átlagos úthossz: $\frac{1}{2} \log_2 N$
- A chord egy robosztus protokoll



Irodealom

- I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications," ACM Sigcomm 2001.
- <http://www.acm.org/sigcomm/sigcomm2001/p12.html>
- The Chord Project
- <http://pdos.csail.mit.edu/chord/>