Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

Simon Csaba
TMIT
2017

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)

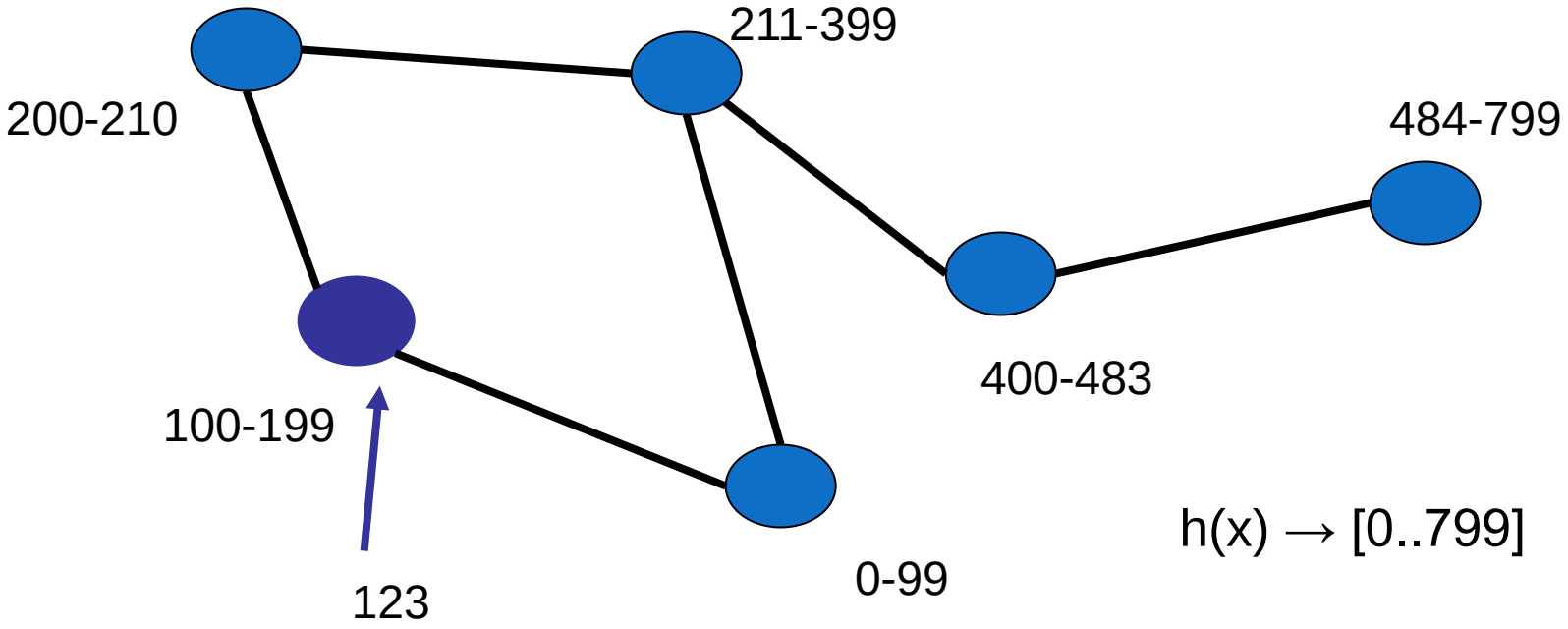
- Strukturált P2P rendszerek
  - Irányított keresés
    - Kijelölt peer-ek kijelölt fájlokat (vagy azok fele mutató pointereket) kell tároljanak
      - Mint egy információs pult
    - Ha valaki keresi a fájlt, tudja kit kérdezzen

- 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

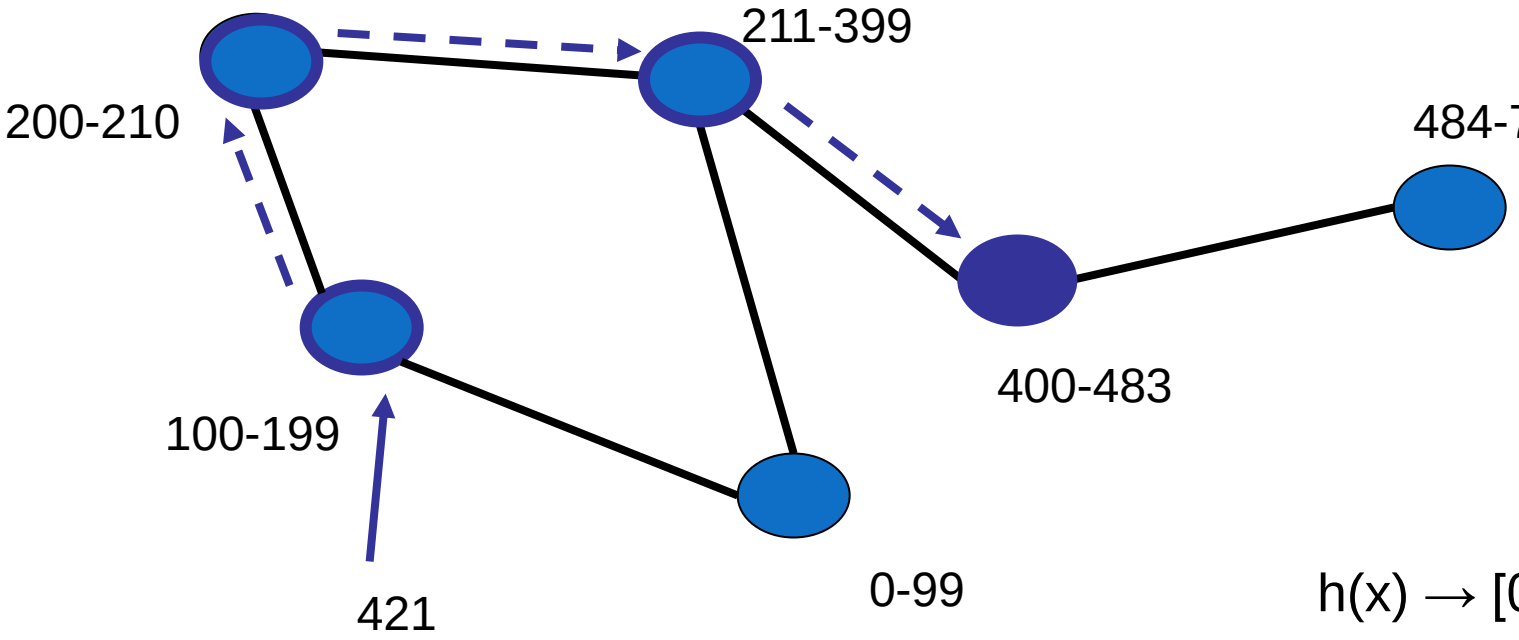
- Distributed Hash Tables
- Egy hash függvény a keresett fájlhoz egy egyéni azonosítót társít
  - Példa: h(„P2P_előadás”) -> 123
- A hash függvény értékkészletét elosztja a peer-ek között
- Egy peer-nek tudnia kell minden olyan fájlról, melynek a
- hash-elt értéke a saját értékkészletébe tartozik
  -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

DHT példa



h(x) → [0..799]

DHT példa



h(x) → [0..799]

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

CAN

- Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, Scott Shenker, "A Scalable Content-Addressable Network", Proceedings of the ACM SIGCOMM'01 Conference, San Diego, CA (August 2001).
  - http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.140.3129

- Sylvia Ratnasamy, „A Scalable Content-Addressable Network", Ph.D. Thesis, October 2002
  - http://berkeley.intel-research.net/sylvia/thesis.pdf

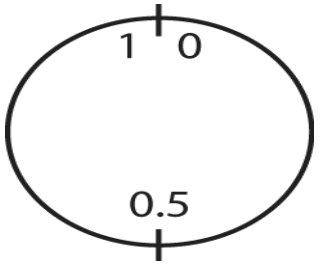
Content Addressable Network

- CAN = egy elosztott, Internet szinten skálázható hash tábla
  - Tároló (Insert), kereső (Lookup) és adattörlő (Delete) műveleteket biztosít egy (Kulcs, Érték) paraméter párra

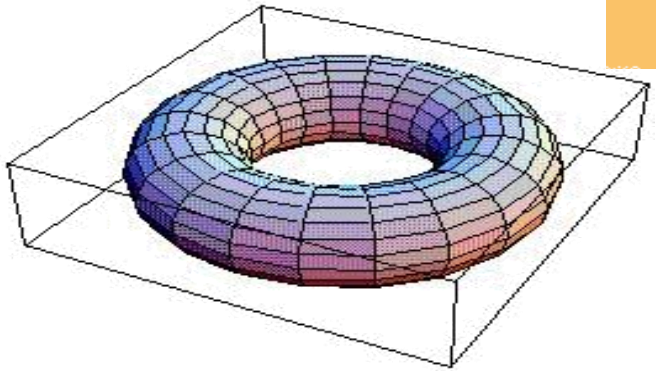
- CAN tulajdonságok
  - Teljesen elosztott rendszer
  - Skálázható: minden résztvevő csak korlátozott számú állapotot tárol, a résztvevők számától függetlenül
  - Hibatűrő: akkor is működőképes ha a rendszer egy része meghibásodott

CAN architektúra

- A hash-tábla egy d-dimenziós ortogonális (Descartes) koordináta rendszerben kialakított D-tóruszon működik
  - Ciklikus d-dimenziós tér
  - hash(K)=(x1, …, xd)



1-dimenziós ciklikus tér
0.5 + 0.7 = 0.2

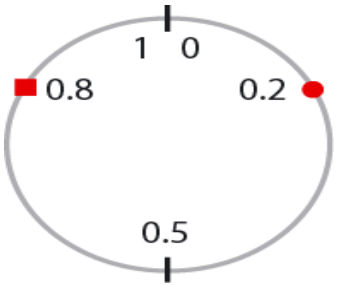


CAN architektúra

A hash-tábla egy d-dimenziós ortogonális (Descartes) koordináta rendszerben kialakított D-tóruszon működik

- **Ciklikus d-dimenziós tér**
- **hash(K)=(x1, …, xd)**

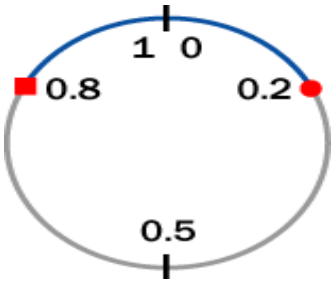
Ortogonális (Descartes) távolság



CAN architektúra

A hash-tábla egy d-dimenziós ortogonális (Descartes) koordináta rendszerben kialakított D-tóruszon működik

- Ciklikus d-dimenziós tér
- hash(K)=(x1, …, xd)



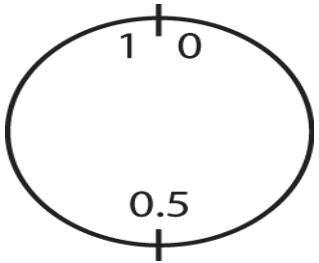
Ortogonális (Descartes) távolság

- p1 = 0.2; p2 = 0.8
- CartDist (p1, p2) = 0.4

CAN architektúra

A hash-tábla egy d-dimenziós ortogonális (Descartes) koordináta rendszerben kialakított D-tóruszon működik

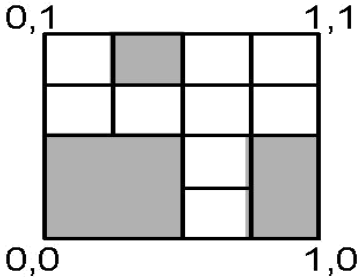
- Ciklikus d-dimenziós tér
- hash(K)=(x1, …, xd)



1-dimenziós ciklikus tér
0.5 + 0.7 = 0.2

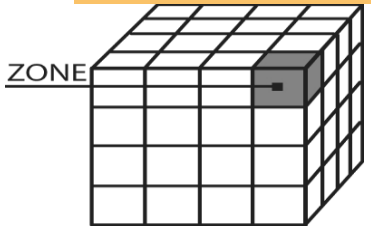
Ortogonális (Descartes) távolság

- p1 = 0.2; p2 = 0.8
- CartDist (p1, p2) = 0.4



Koordináta zóna

- Egy „szelet” a Descartes térből

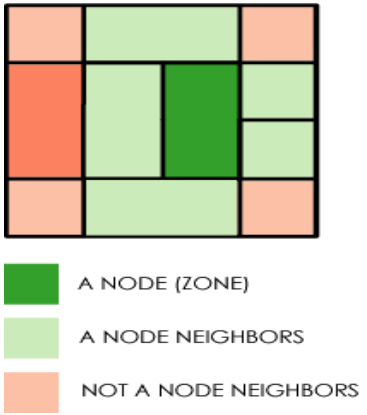


CAN architektúra (II)

- CAN csomópontok
  - Egy CAN csomópont = egy gép a rendszerben
  - Egy CAN csomópont ≠ peer
    - A peer-nek itt más jelentése van, lásd „Zónák túlterhelése” - később
  - Minden CAN csomópont a tér egy külön zónájáért (szeletéjért) felel
    - A csomópont információt tárol az összes olyan állományról mely a saját zónájába hash-elődik
  - A csomópontok egymás között lefedik a teljes teret

CAN szomszédok

- 2 CAN csomópont szomszéd ha a zónáik átfedik egymást d-1 dimenzió mentén és szomszédosak 1 dimenzió mentén
  - Egy csomópont ismeri az összes szomszédja IP címét
  - Egy csomópont ismeri az összes szomszédos zóna koordinátáit
  - A csomópontok csak a szomszédaikkal tudnak kommunikálni



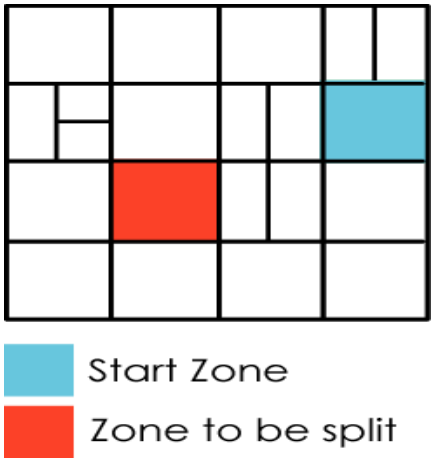
CAN csatlakozás

- Egy CAN-nek van egy DNS domain neve
  - Egy vagy több „Bootstrap Node” IP címére képződik le
- A Bootstrap Node tárolja néhány résztvevő csomópont IP címét
  - Gateway csomópontok
- Az új csomópont véletlenszerűen kiválaszt egyet, és rajta keresztül
- próbál csatlakozni
  - A hash tábla egy szeletét (zónát) az új csomópontnak kell adni
    - A kapcsolódó állományokat, illetve az azokra vonatkozó adatokat tárolni kell az új csomóponton
  - Az új csomópont be kell kapcsolódjon az útválasztásba

CAN csatlakozás (II)

Zóna keresése az új csomópont részére

1. **Véletlenszerűen választ egy P pontot**
2. A JOIN üzenetet a **P** pontért felelős csomóponthoz küldi
3. A kérést a CAN útválasztást használva továbbítja

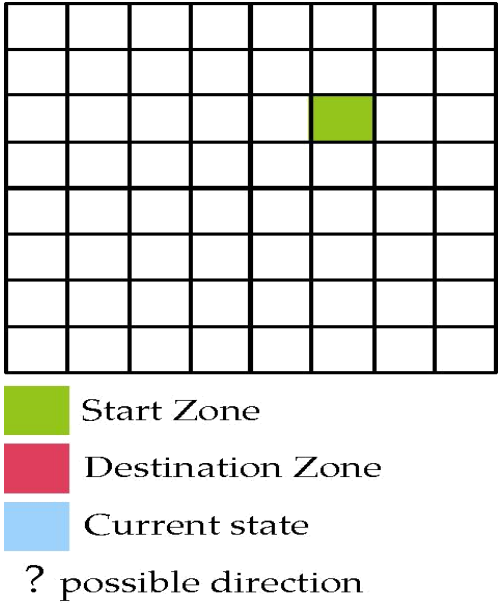


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:
   1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. HA IGEN

      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz



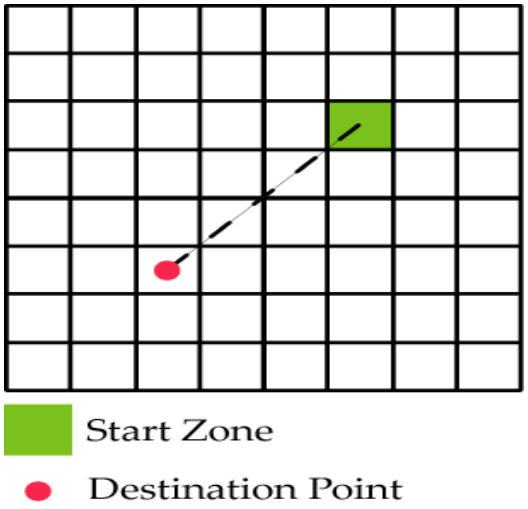
CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:

   1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. HA IGEN

      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

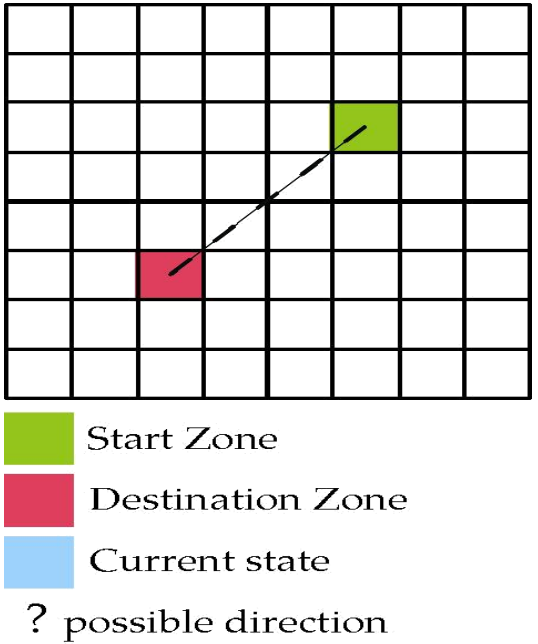


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:
   1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. HA IGEN

      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

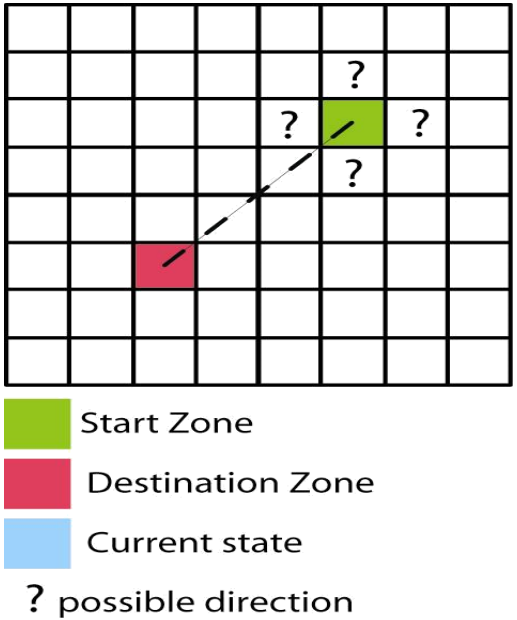


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:
   1. **Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot**
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. HA IGEN

      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

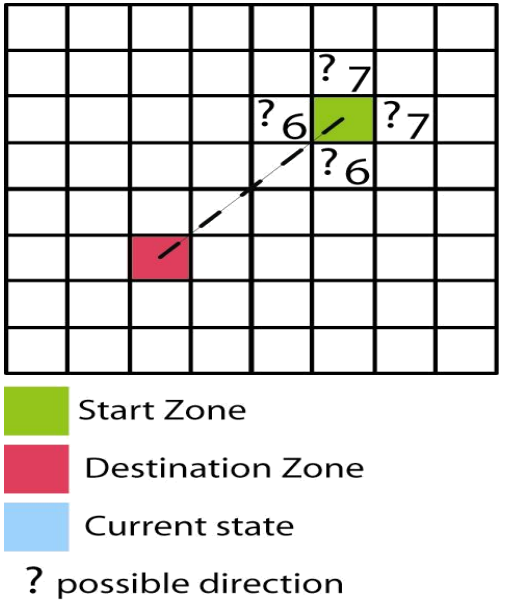


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

Az aktuális csomópont:

1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
2. **HA NEM**
   a. **Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján**
   b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
   c. Megismétli az 1. lépést
3. HA IGEN
   A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

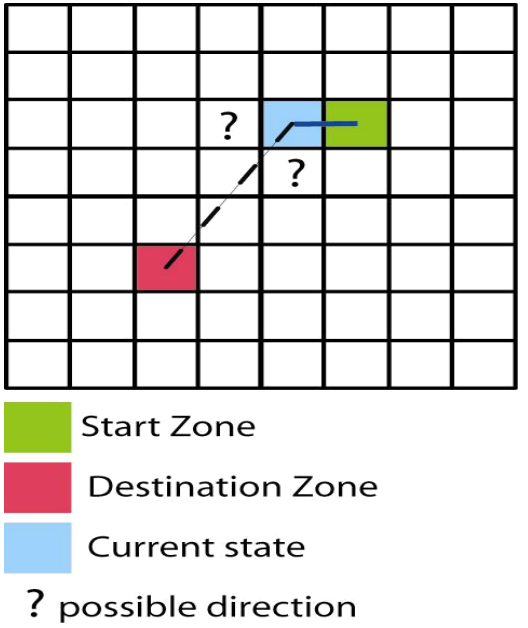


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

Az aktuális csomópont:

1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
2. **HA NEM**
   a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
   b. **Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi**
   c. Megismétli az 1. lépést
3. HA IGEN
   A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

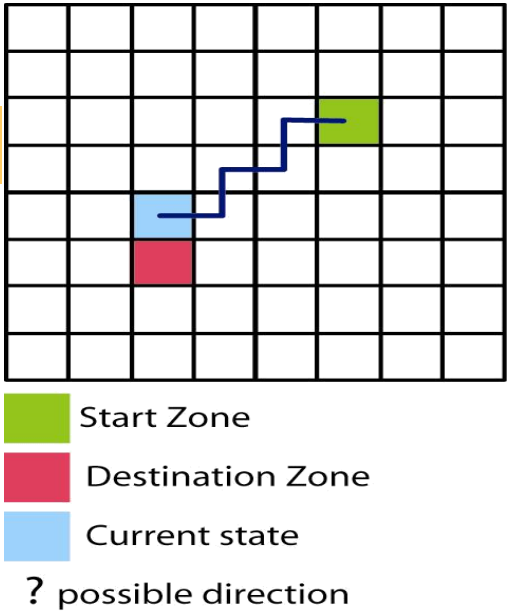


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:
   1. **Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot**
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. **HA IGEN**

      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz

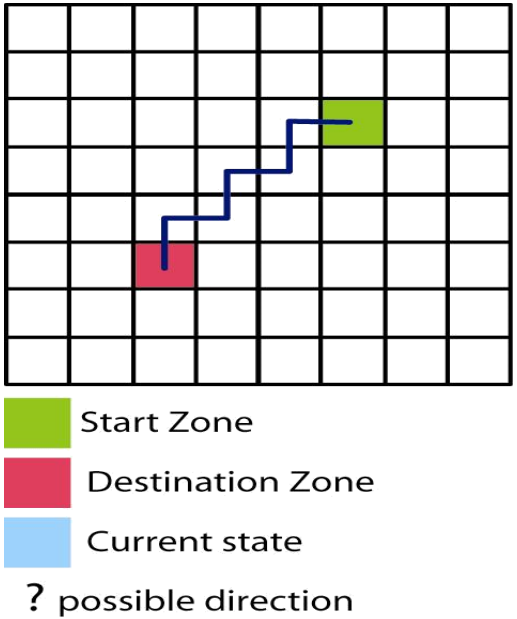


CAN routing

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding

   Az aktuális csomópont:
   1. Leellenőrzi hogy a saját vagy a szomszédai zónája tartalmazza-e a P pontot
   2. HA NEM
      a. Rangsorolja a szomszédait a P ponttal szembeni Descartes távolságuk alapján
      b. Az üzenetet a P ponthoz legközelebb álló szomszédhoz küldi
      c. Megismétli az 1. lépést
   3. **HA IGEN**

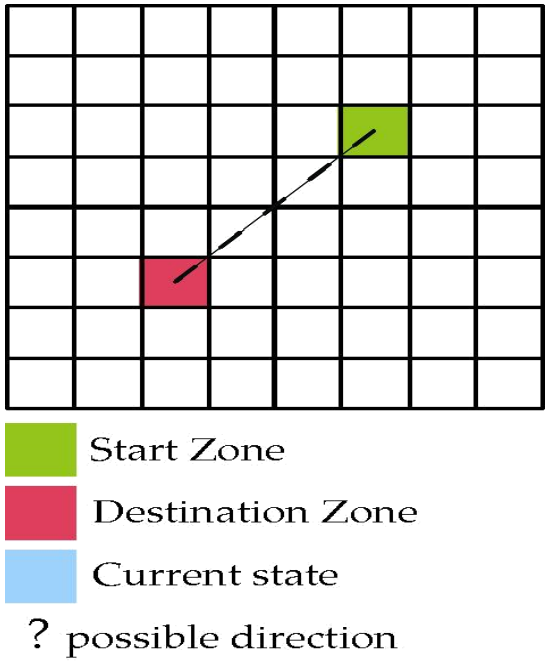
      A pozitív választ (a keresett állományt) visszaküldi a keresőhöz



CAN routing

Hibatűrő útválasztás

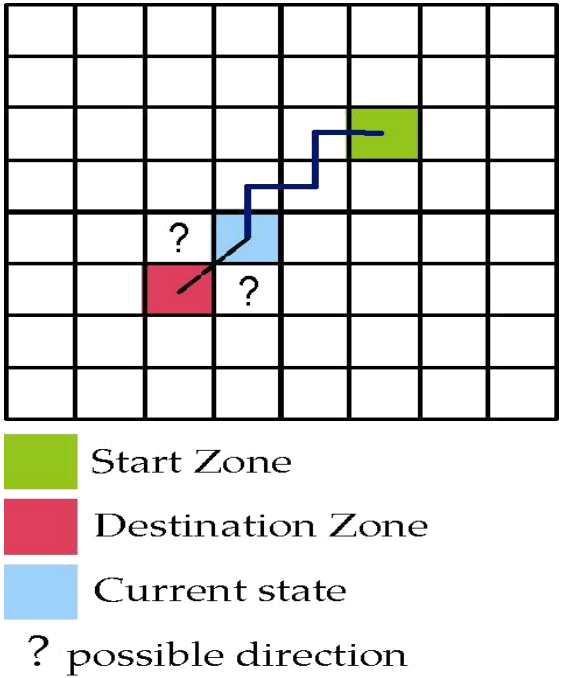
1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding
   a. Mielőtt továbbküldené az üzenetet, ellenőrzi a szomszéd készenléti állapotát
   b. Az üzenetet a rendelkezésre álló legjobb szomszédhoz küldi el



CAN routing

Hibatűrő útválasztás

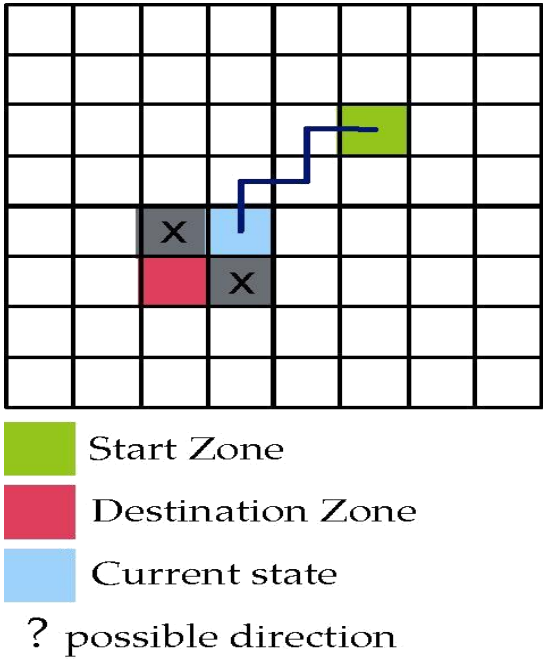
1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding
   a. Mielőtt továbbküldené az üzenetet, ellenőrzi a szomszéd készenléti állapotát
   b. Az üzenetet a rendelkezésre álló legjobb szomszédhoz küldi el



CAN routing

Hibatűrő útválasztás

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding
   a. **Mielőtt továbbküldené az üzenetet, ellenőrzi a szomszéd készenléti állapotát**
   b. Az üzenetet a rendelkezésre álló legjobb szomszédhoz küldi el

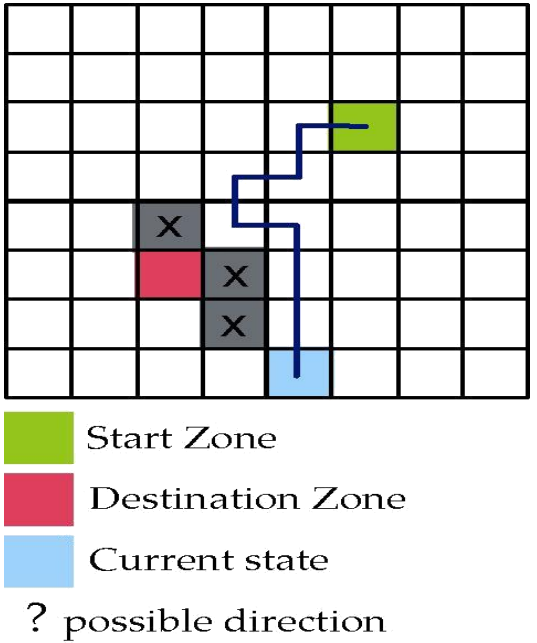


CAN routing

Hibatűrő útválasztás

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding
   a. Mielőtt továbbküldené az üzenetet, ellenőrzi a szomszéd készenléti állapotát

   b. Az üzenetet a rendelkezésre álló legjobb szomszédhoz küldi el

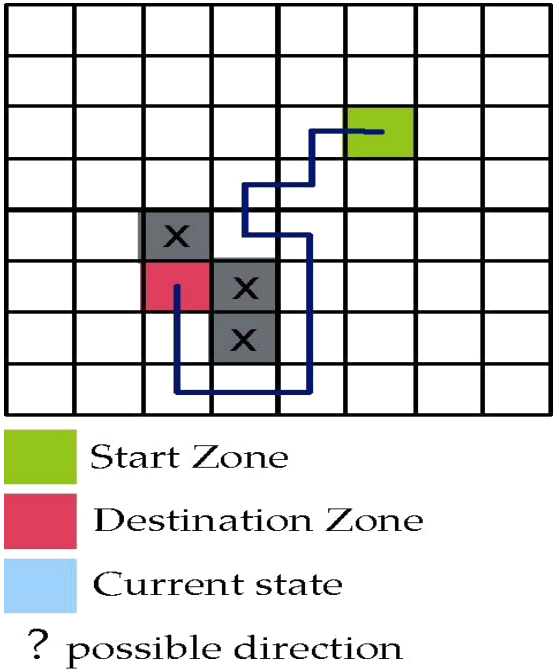


CAN routing

Hibatűrő útválasztás

1. Elindul egy tetszőleges pontból
2. P = a K kulcs hash-elt értéke
3. Greedy (mohó) forwarding
   a. Mielőtt továbbküldené az üzenetet, ellenőrzi a szomszéd készenléti állapotát
   b. Az üzenetet a rendelkezésre álló

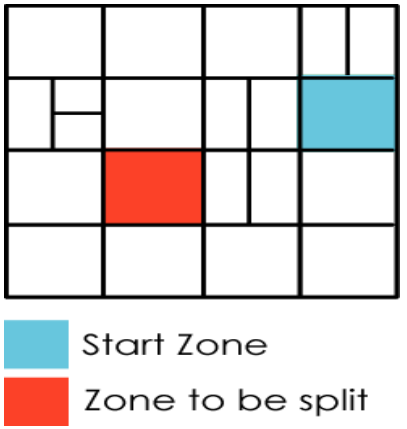
A célcsomópontot elérjük ha létezik legalább egy út



CAN csatlakozás (II)

Zóna keresése az új csomópont részére

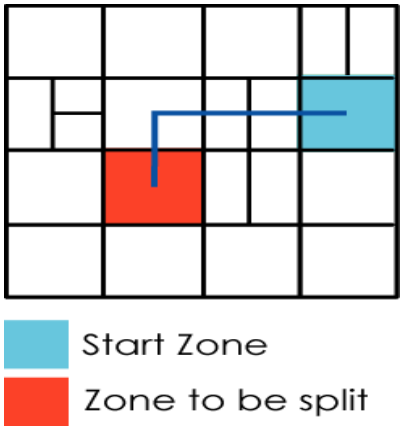
1. **Véletlenszerűen választ egy P pontot**
2. A JOIN üzenetet a **P** pontért felelős csomóponthoz küldi
3. A kérést a CAN útválasztást használva továbbítja
4. A **P** pontért felelős csomópont kettévágja zónáját
   - Az egyik fele az új csomóponté lesz
   - A másik fele megmarad a régi csomópontnál
5. Az elosztás a zóna legnagyobb oldalának megfelelő dimenzió mentén történik
   - Ha két oldal egyenlő, a kisebbik rangú dimenzió mentén osztódik
6. A hash tábla alapján az új csomópont zónájába eső állományokra vonatkozó adatokat átküldi



CAN csatlakozás (II)

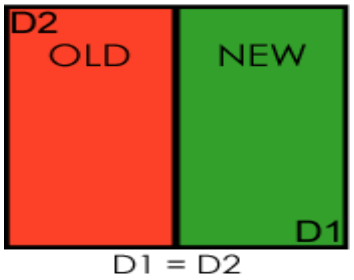
Zóna keresése az új csomópont részére

1. Véletlenszerűen választ egy **P** pontot
2. **A JOIN üzenetet a P pontért felelős csomóponthoz küldi**
3. **A kérést a CAN útválasztást használva továbbítja**
4. A **P** pontért felelős csomópont kettévágja zónáját
   - Az egyik fele az új csomóponté lesz
   - A másik fele megmarad a régi csomópontnál
5. Az elosztás a zóna legnagyobb oldalának megfelelő dimenzió mentén történik
   - Ha két oldal egyenlő, a kisebbik rangú dimenzió mentén osztódik
6. A hash tábla alapján az új csomópont zónájába eső állományokra vonatkozó adatokat átküldi



CAN csatlakozás (II)

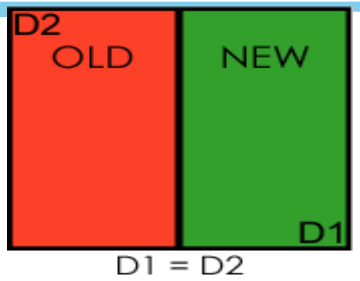
Zóna keresése az új csomópont részére



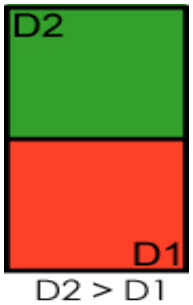
1. Véletlenszerűen választ egy **P** pontot
2. A JOIN üzenetet a **P** pontért felelős csomóponthoz küldi
3. A kérést a CAN útválasztást használva továbbítja
4. **A P pontért felelős csomópont kettévágja zónáját**
   - **Az egyik fele az új csomóponté lesz**
   - **A másik fele megmarad a régi csomópontnál**
5. Az elosztás a zóna legnagyobb oldalának megfelelő dimenzió mentén történik
   - Ha két oldal egyenlő, a kisebbik rangú dimenzió mentén osztódik
6. A hash tábla alapján az új csomópont zónájába eső állományokra vonatkozó adatokat átküldi

CAN csatlakozás (II)

Zóna keresése az új csomópont részére



1. Véletlenszerűen választ egy **P** pontot
2. A JOIN üzenetet a **P** pontért felelős csomóponthoz küldi
3. A kérést a CAN útválasztást használva továbbítja
4. A **P** pontért felelős csomópont kettévágja zónáját
   - Az egyik fele az új csomóponté lesz
   - A másik fele megmarad a régi csomópontnál



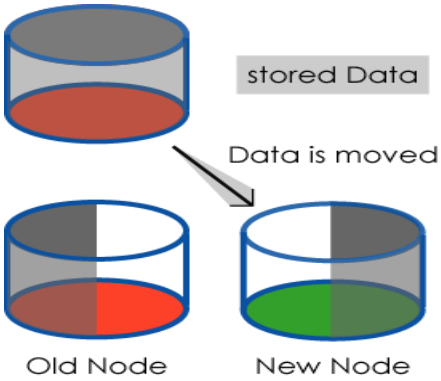
5. **Az elosztás a zóna legnagyobb oldalának megfelelő dimenzió mentén történik**
   - **Ha két oldal egyenlő, a kisebbik rangú dimenzió mentén osztódik**

6. A hash tábla alapján az új csomópont zónájába eső állományokra vonatkozó adatokat átküldi

CAN csatlakozás (II)

Zóna keresése az új csomópont részére

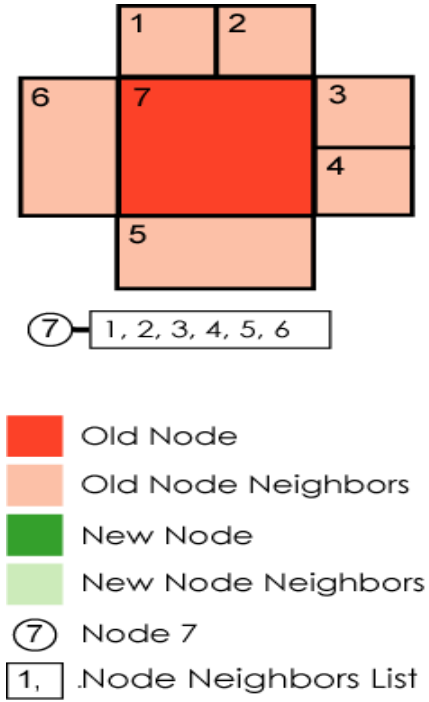
1. Véletlenszerűen választ egy **P** pontot
2. A JOIN üzenetet a **P** pontért felelős csomóponthoz küldi
3. A kérést a CAN útválasztást használva továbbítja
4. A **P** pontért felelős csomópont kettévágja zónáját
   - Az egyik fele az új csomóponté lesz
   - A másik fele megmarad a régi csomópontnál
5. Az elosztás a zóna legnagyobb oldalának megfelelő dimenzió mentén történik
   - Ha két oldal egyenlő, a kisebbik rangú dimenzió mentén osztódik
6. **A hash tábla alapján az új csomópont zónájába eső állományokra vonatkozó adatokat átküldi**



CAN csatlakozás (III)

Bekapcsolódás az útválasztásba

1. Az új csomópont megkapja a régi csomópont (kinek a zónáját osztottuk) szomszédainak listáját
2. A régi csomópont frissíti a szomszédait
   - Törli az „elvesztett” szomszédokat
   - Bejegyzi az új csomópontot
3. Minden szomszéd kap egy frissítő üzenetet, a saját szomszéd-táblája frissítéséhez
   - Törli a régi csomópontot
   - Bejegyzi az új csomópontot



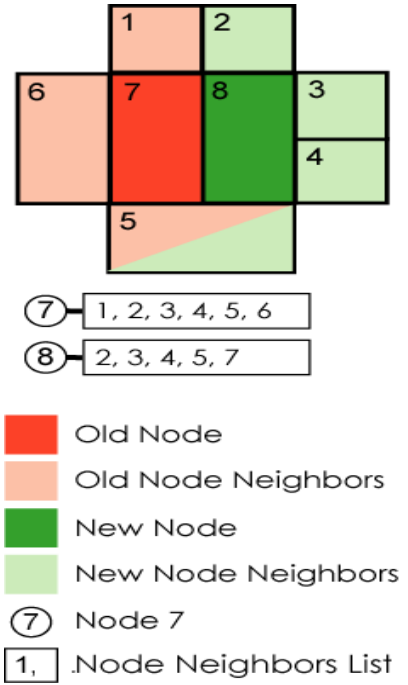
CAN csatlakozás (III)

Bekapcsolódás az útválasztásba

1. Az új csomópont megkapja a régi csomópont (kinek a zónáját osztottuk) szomszédainak listáját

2. A régi csomópont frissíti a szomszédait
- Törli az „elvesztett” szomszédokat
- Bejegyzi az új csomópontot

3. Minden szomszéd kap egy frissítő üzenetet, a saját szomszéd-táblája frissítéséhez
- Törli a régi csomópontot
- Bejegyzi az új csomópontot



CAN csatlakozás (III)

Bekapcsolódás az útválasztásba

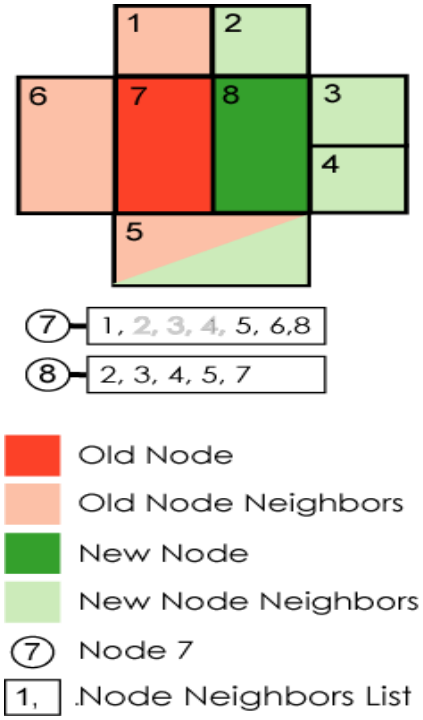
1. Az új csomópont megkapja a régi csomópont (kinek a zónáját osztottuk) szomszédainak listáját

2. A régi csomópont frissíti a szomszédait

- **Törli az „elvesztett” szomszédokat**
- **Bejegyzi az új csomópontot**

3. Minden szomszéd kap egy frissítő üzenetet, a saját szomszéd-táblája frissítéséhez

- Törli a régi csomópontot
- Bejegyzi az új csomópontot



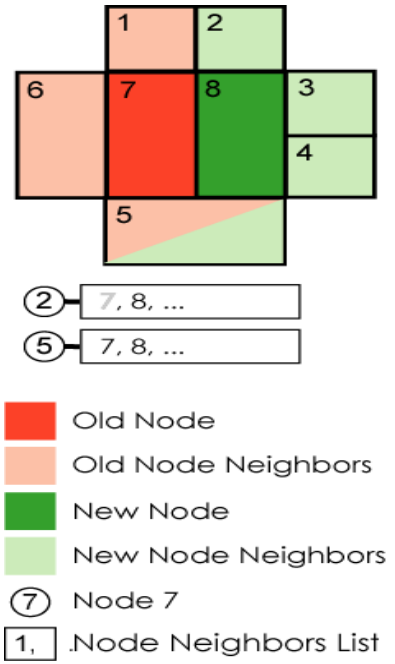
CAN csatlakozás (III)

Bekapcsolódás az útválasztásba

1. Az új csomópont megkapja a régi csomópont (kinek a zónáját osztottuk) szomszédainak listáját
2. A régi csomópont frissíti a szomszédait
   - Törli az „elvesztett” szomszédokat
   - Bejegyzi az új csomópontot

3. Minden szomszéd kap egy frissítő üzenetet, a saját szomszéd-táblája frissítéséhez

- **Törli a régi csomópontot**
- **Bejegyzi az új csomópontot**

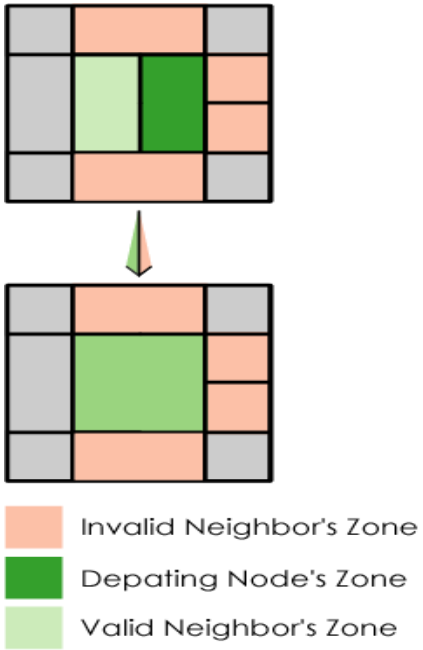


Csomópont távozása

Csomópont kilép a rendszerből

a. Ha az egyik szomszéd zónája egyesíthető a kilépő csomópont zónájával, akkor ez a szomszéd veszi át az egyesített zóna kezelését egyesíthető = „szabályos” konvex zóna keletkezik

b. Ha nem, akkor az egyik szomszéd két külön zónát kezel majd

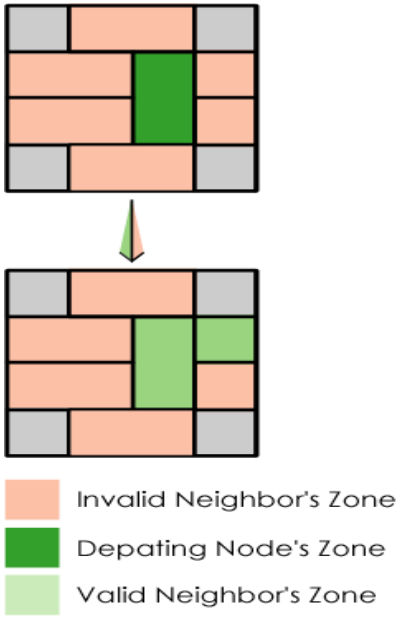


Csomópont távozása

Csomópont kilép a rendszerből

a. Ha az egyik szomszéd zónája egyesíthető a kilépő csomópont zónájával, akkor ez a szomszéd veszi át az egyesített zóna kezelését
egyesíthető = „szabályos" konvex zóna keletkezik

b. Ha nem, akkor az egyik szomszéd két külön zónát kezel majd

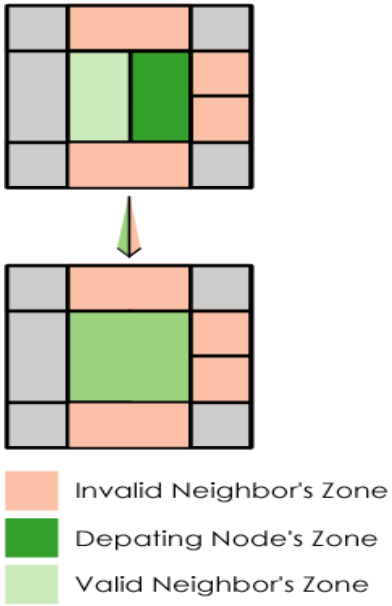


Csomópont távozása

Csomópont kilép a rendszerből

a. Ha az egyik szomszéd zónája egyesíthető a kilépő csomópont zónájával, akkor ez a szomszéd veszi át az egyesített zóna kezelését
egyesíthető = „szabályos” konvex zóna keletkezik

b. Ha nem, akkor az egyik szomszéd két külön zónát kezel majd



Mindkét esetben (a és b):

1. A kilépő csomópont adatai átkerülnek a zónát átvevő csomóponthoz
2. A zónát átvevő csomópont frissíti a szomszédainak listáját
3. Az összes szomszédot értesíti a cseréről, azok pedig frissítik a saját szomszéd-listájukat

Csomópont távozása (II)

- Ha egy csomópont „angolosan” távozik (ungraceful departure)…
  - Szabályos időközönként a csomópontok frissítő üzeneteket küldenek a szomszédaiknak
    - A saját zonájuk koordinátái, a szomszédaik listája, illetve azok zónáinak koordinátái
  - Ha egy csomópont egy adott *t* időn belül nem kapott üzenetet az egyik szomszédjától, elindítja a TAKEOVER procedúrát
    - Megpróbálja átvenni a „halottnak” hitt csomópont zónáját
    - Egy TAKEOVER üzenetet küld a halott csomópont minden szomszédjának
  - Ha egy csomópont kap egy TAKEOVER üzenetet, összehasonlítja a saját zónáját a küldő zónájával
    - Ha optimálisabb, hogy ő vegye át a zónát, saját maga küld egy új TAKEOVER üzenetet a halott csomópont szomszédainak
  - Az a szomszéd veszi át a zónát, aki nem kap választ a TAKEOVER üzenetére *t* időn belül
- A halott csomóponton tárolt adatok nem elérhetőek mindaddig
- ameddig a forrás nem publikálja újból őket (elküldi az új csomópontnak)

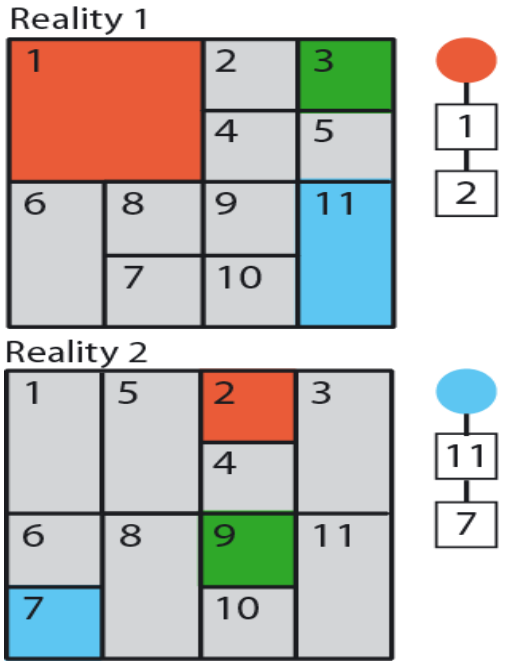
CAN Optimalizálás

- Optimalizációs lehetőségek
  - Rövidebb utak: kevesebb ugrás a forrás és a cél között
  - Rövidebb ugrások: két szomszéd közötti ugrás a fizikai topológián
  - Optimalizált zónafelosztás

CAN valóságok

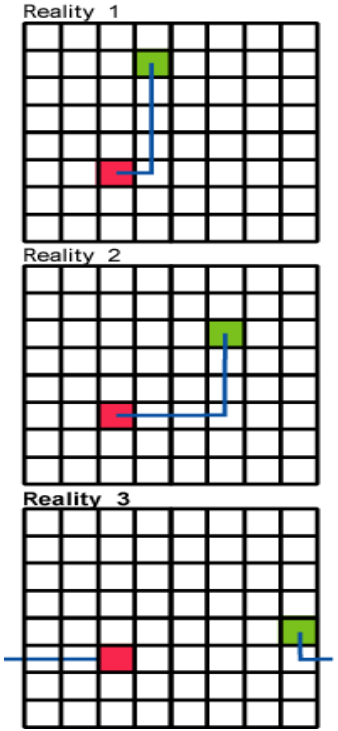
CAN valóságok (Reality):
- többszörös koordináta rendszerek

- Minden csomópont több koordináta rendszer (valóság) része
- Minden valóságban ...
  - Egyenlő a zónák száma
  - Ugyanazok a tárolt adatok
  - Ugyanaz a hash függvény
- Egy csomópontnak különböző zónai vannak a különböző valóságokban
  - A zónák kiosztásakor a P pontot véletlenszerűen választjuk
- A hash tábla tartalma minden valóságban elérhető – nagyobb rendelkezésre állás



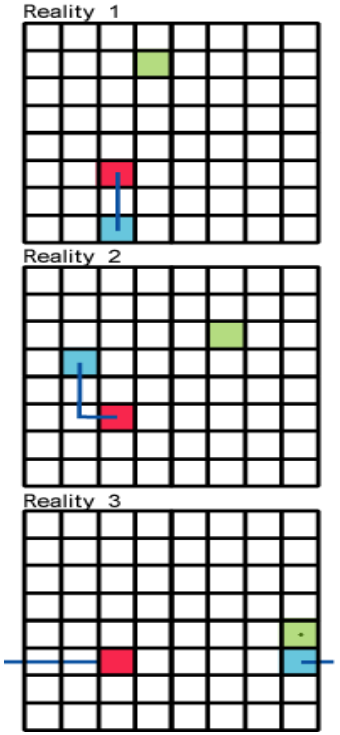
Útválasztás a valóságokban

1. A cél minden valóságban ugyanabba a H pontba hash-elődik
   - Minden valóságban más-más csomópont zónájába tartozhat a H pont
2. A hagyományos útválasztás a következő módon egészítjük ki:
   a. Minden csomópont az útvonalon ellenőrzi, hogy melyik valóságban a legkisebb a távolsága a célhoz
   b. Az üzenetet a legjobb valóságban küldi tovább



Útválasztás a valóságokban

1. A cél minden valóságban ugyanabba a H pontba hash-elődik
   - Minden valóságban más-más csomópont zónájába tartozhat a H pont
2. A hagyományos útválasztás a következő módon egészítjük ki:
   a. Minden csomópont az útvonalon ellenőrzi, hogy melyik valóságban a legkisebb a távolsága a célhoz
   b. Az üzenetet a legjobb valóságban küldi tovább



Többdimenziós koordináta rendszer

A routing hatékonysága függ a koordináták számától

- Az útvonal átlagos hossza: $O(d*n^{1/d})$

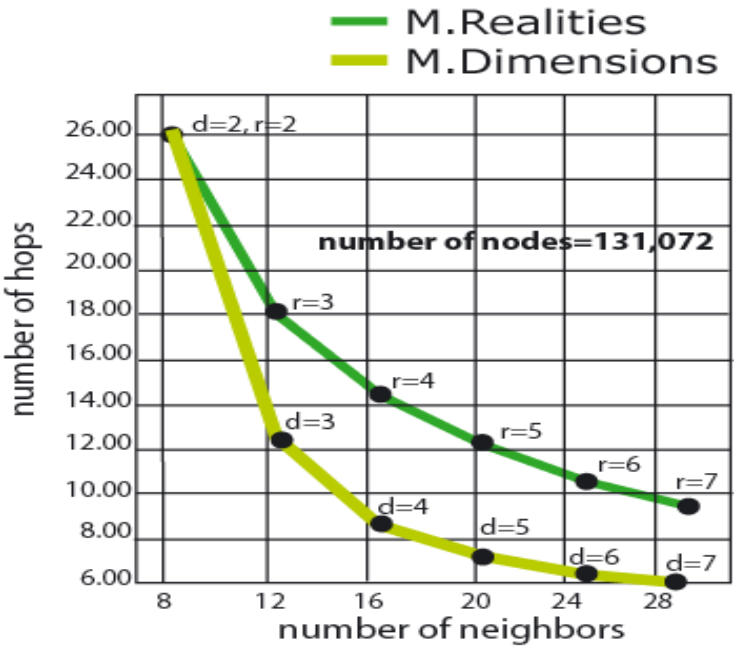
- Ha a dimenziók száma (***d***) nő, az út hossza csökken

n = 1000, egyenlő zónák	
D	**Út átlagos hossza**
2	**15**
3	**7.5**
5	**5**
10	**4.95**

Több valóság vagy több dimenzió?

A dimenziók növelése hatékonyabb az útvonal optimalizálásában mint a valóságok növelése

- **A valóságok nagyobb hibatűrést és rendelkezésre állást biztosítanak**



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

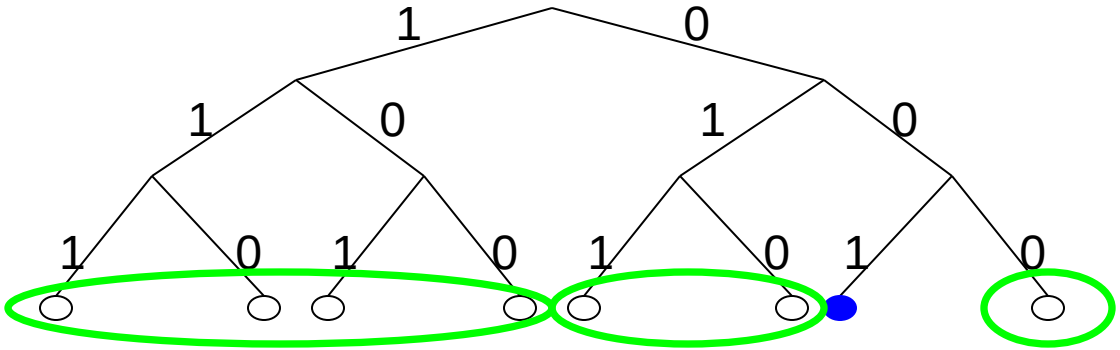
- P2P adattároló hálózat
  - Kulcs-érték párok tárolása
    - egyedi, 160 bites kulcsok
  - DHT
- Kialakítandó környezet
  - Bármelyik node bármikor eltűnhessen
  - A node-ok legyenek egyenletesen terheltek
    - tárolási kapacitás
    - sávszélesség
- Cél
  - Gyors adatelérés
  - Minimális számú vezérlő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

Kademlia

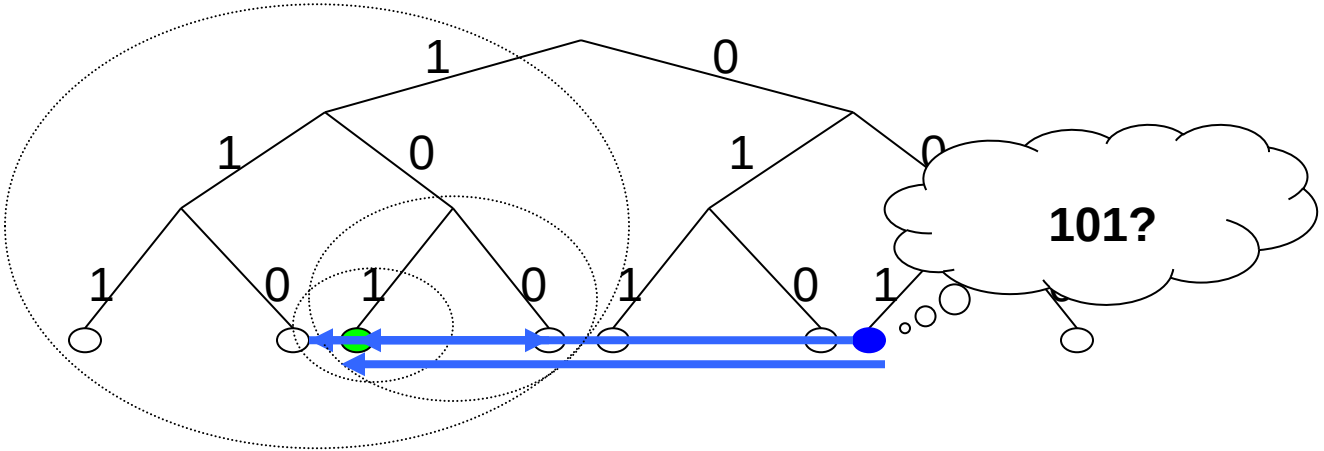
- A routing-tábla rugalmas
  - Közelségi alapú (proximity-based) routing
  - Laza követelmények
    - minimális fenntartás
- Bejövő és kimenő üzenetek eloszlása azonos
  - A hálózat önmegerősítő
- Csak *log n* node elérését kell ismerni

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 XOR y
  - Szimmetrikus: d(x,y) = d (y,x). Kevesebb vezérlő üzenet
  - Egyértelmű: minden $x \in \{0,1\}^{160}$ és $l \in N$-hez pontosan egy olyan $y \in \{0,1\}^{160}$ létezik, hogy d(x,y) = l.

- A nagyobb helyiértékű bitek eltérése nagyobb távolságot
- eredményez:
  - x = 0101**0**1
  - y = **1**1000**1**
  - x XOR y = 100100, azaz 32 + 4 = 36

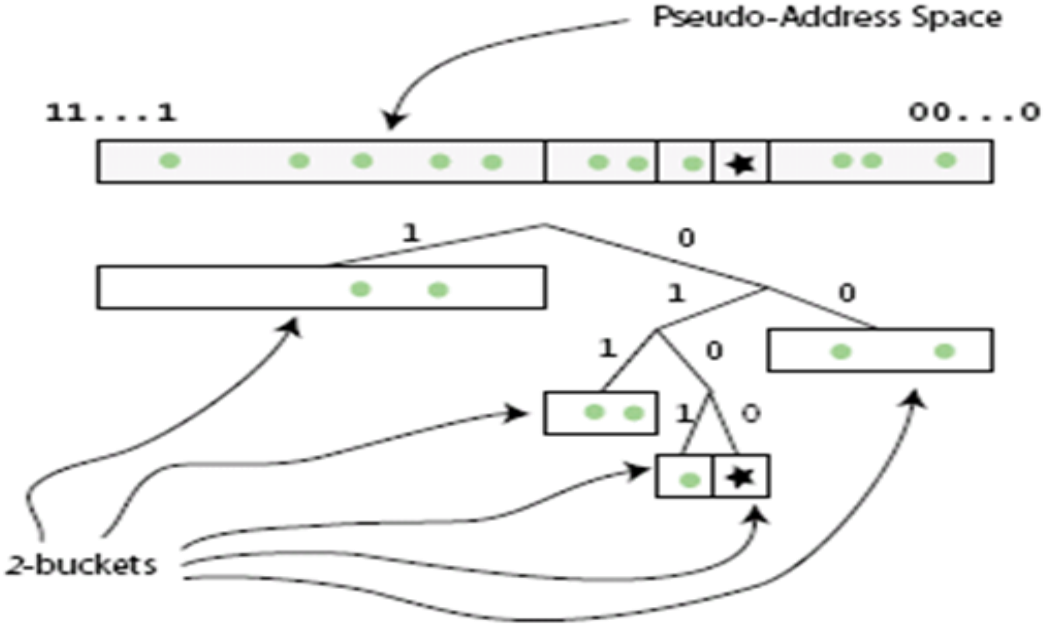
- Geometriai értelmezés:
  - A közös ágon lévő csomópont közelebb van, mint a többi

Adatstruktúrák

- Kapcsolatok
  - (nodeID, IP:UDP_port) párok
- k-vödör
  - tárolóegység maximum k db. kapcsolathoz
    - alapértelmezésben k=20
    - Egy A állományra (ID_A) vonatkozó infókat a (XOR metrika szerint) hozzá legközelebb álló k db. csomóponton fogunk tárolni
      - Annak a valószínűsége, hogy mind a k csomópont egyszerre kilépjen kellően kicsi legyen
- Routing tábla
  - Műveletek: kapcsolat létesítése, törlése
  - Egy k-vödrökből álló fa
    - Minden vödör a fa egy tartományáért felelős

Útválasztó tábla

- A 0100 csomópont útválasztó táblája



Keresés

- Cél: A T célhoz legközelebbi k node megtalálása, ahol $T \in \{0, 1\}^{160}$
- $find\_node_n(T)$: visszaadja azt a k darab címet az n csomópont
- útválasztó táblájából, amelyik T-hez legközelebb áll
  - Egy vagy több k-vödörből
  - Ha nincs k node az összes vödrében, akkor kevesebbet ad vissza
  - A válaszok minden lépésben visszamennek a keresést indító peer-hez
- Keresés:
  - n_0: a keresést végző node
  - $N_0 := \{\emptyset\}$
  - $N_1 = min_k\{find\_node_{n_0}(T), N_0\}$
  - $N_2 = min_k\{find\_node_{n_1}(T), N_1\}$
  - …
  - $N_i = min_k\{find\_node_{n_{i-1}}(T), N_{i-1}\}$
    - Ahol n_i $(i > 0)$ az N_i halmaz tetszőleges címe
- A keresés akkor ér véget, ha N_i csak olyan címeket tartalmaz, amelyeket n_0
- már megkérdezett
  - Az egymás utáni $find\_node_n(T)$ hívások egyre kisebb tartományhoz tartozó k-vödörből adnak választ

Párhuzamos keresés

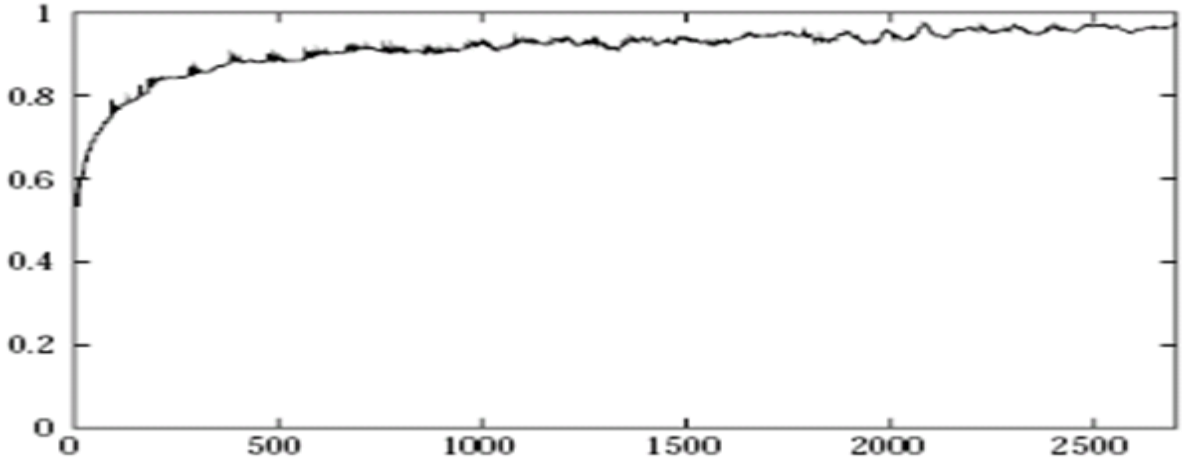
- Nagyobb forgalomért cserébe gyorsabb keresés
- Cél
  - A keresés menjen a gyorsabb / közelebbi gépeken keresztül
  - Kerüljük el a kilépett node-ok miatti timeout-okat
- Megoldás:
  - egyszerre $\alpha > 1$ párhuzamos $find\_node_n(T)$

Kapcsolatok fenntartása

- Ha egy node nem válaszol, töröljük a routing táblából
- Minden egyes node amelyiktől „hívást” kapunk bekerül a routing
- táblánkba ha…
  - Megnézzük, hogy a k-vödör ahova kerülne tele van-e
    - Ha nem, betesszük
    - Ha igen, megpingeljük azt a node-ot, melyről legrégebb hallottunk
      - Ha válaszol, az új node-ot nem tesszük be a táblába
        - Ha a régi él, akkor valószínű hogy később is élni fog, kár elveszteni
        - Mérések alapján a Gnutella hálón
        - DoS támadások ellen is véd
      - Ha nem válaszol, töröljük, és felveszi az új node-ot

- Az XOR topológia szimmetriája miatt a minket hívó node-ok eloszlása éppen
- a routing táblánkéval egyező eloszlású lesz
  - Formálisan: annak valószínűsége, hogy egy $[2^i, 2^{i+1}]$ távolságra levő node-tól kapunk üzenetet konstans, azaz i-től független

Node-ok elérhetősége a Gnutella hálón



- x-tengely: az elérhetőség időtartama percben
- y-tengely: azon node-oknak, amelyek x percen át elérhetőek, mekkora hányada lesz x+60 perc múlva is elérhető

Érkezés, távozás, frissítés

- Node érkezése:
  - Egy már bentlévő node-tól néhány címet elkér
  - ‘Megkeresi’ önmagát
    - Megkapja a saját ID-jéhez legközelebb álló k node címét
    - Ezektől a node-októl megkapja a tárolandó infókat
  - Bekerül mások k-vödreibe, elkezdi feltölteni saját k-vödreit
- Node távozása
  - Nincs semmi tennivaló
- Periodikus frissítés
  - Ha egy csomópont tárol egy értéket, periódikus időközönként megkeresi a k-1 másik csomópontot amelyik legközelebb áll az adott értékhez
  - Tárolja az értéket ezeken a csomópontokon

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 fejleszői az Overnet helyett saját Kademlia implementációt készítettek
  - Directconnect: RevConnect
  - Trackerless BitTorrent: Azureus, BitSpirit, eXeem

Tapestry

Tapestry: A Resilient Global-scale Overlay for Service Deployment

- Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz

- *IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1.*

Tapestry

- Egy *elosztott, hibatűrő, adaptív* lokalizáló és útválasztó infrastruktúra
- Utótag (*suffix*) alapú útválasztás
- A Plaxton algoritmus ötletére alapoz
  - C.G. Plaxton, R. Rajaraman and A.W. Richa, *Accessing Nearby Copies of Replicated Objects in a Distributed Environment*., 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97), pp 311-320, Newport, RI, USA, 1997 http://citeseer.ist.psu.edu/plaxton97accessing.html

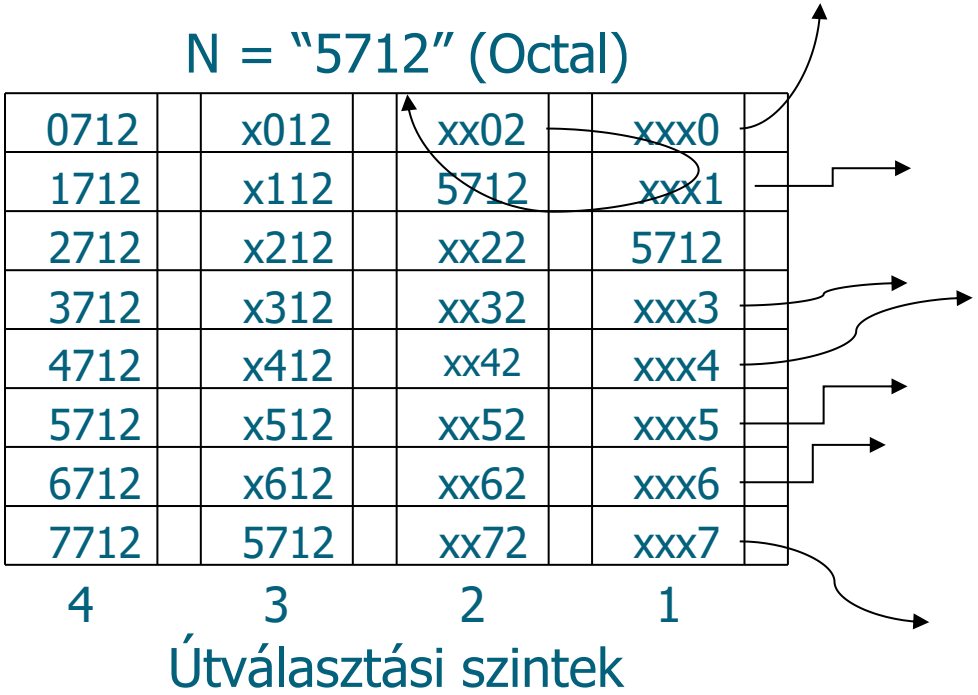
Plaxton/Tapestry címzés

- Bármely csomópont lehet:
  - Szerver – állományokat tárol
  - Router – csomagokat továbbít
  - Kliens – kéréseket kezdeményez
- Név (cím-) tartomány
  - Csomópontok és állományok egyaránt
  - Megfelelően nagy az ütközések elkerüléséhez
    - 160 bit, 40 hexa számjegy, $16^{40}=2^{160}$ cím
- ~ Kiegyensúlyozott eloszlás a tartományon belül
  - Hash algoritmus

Neighbour Map

- Legyen N egy csomópont (IP cím, ID)
  - utótag(N, k) = az utolsó k számjegy az ID-ből
- Minden csomópontban *szomszédossági térkép*
- (neighbour map)
  - Annyi szint, ahány számjegy az ID-ben
  - Minden szinten annyi bejegyzés, ahányas számrendszerben címzünk
  - A j. szint (j-1) hosszúságú utótagoknak felel meg
  - Az i. bejegyzés a j. szinten – a fizikailag legközelebb álló olyan csomópont IP címe, mely ID-je [„i” + utótag(N, j-1)]-re végződik
    - Példa: a 2. bejegyzés a 5712 csomópont térképének 3. szintjén az a 212-re végződő ID-jű csomópont IP címe, mely fizikailag legközelebb áll az 5712 ID-jű ponthoz

Neighbour Map



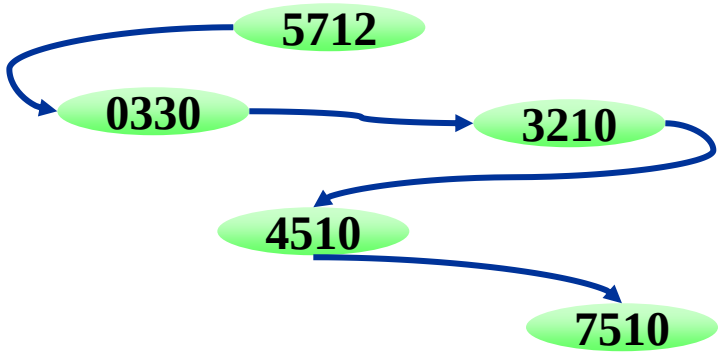
Utótag alapú útválasztás

- Pontról pontra való továbbítás, számjegyenként
  - **** → ***0 → **10 → *510 → 7510
- Hasonlít a *longest prefix match* alapú IP útválasztásra
- Mindegy melyik irányból közelítünk
  - Az eredeti Tapestry javaslat – utótag alapú
  - Jelenleg – előtag alapú

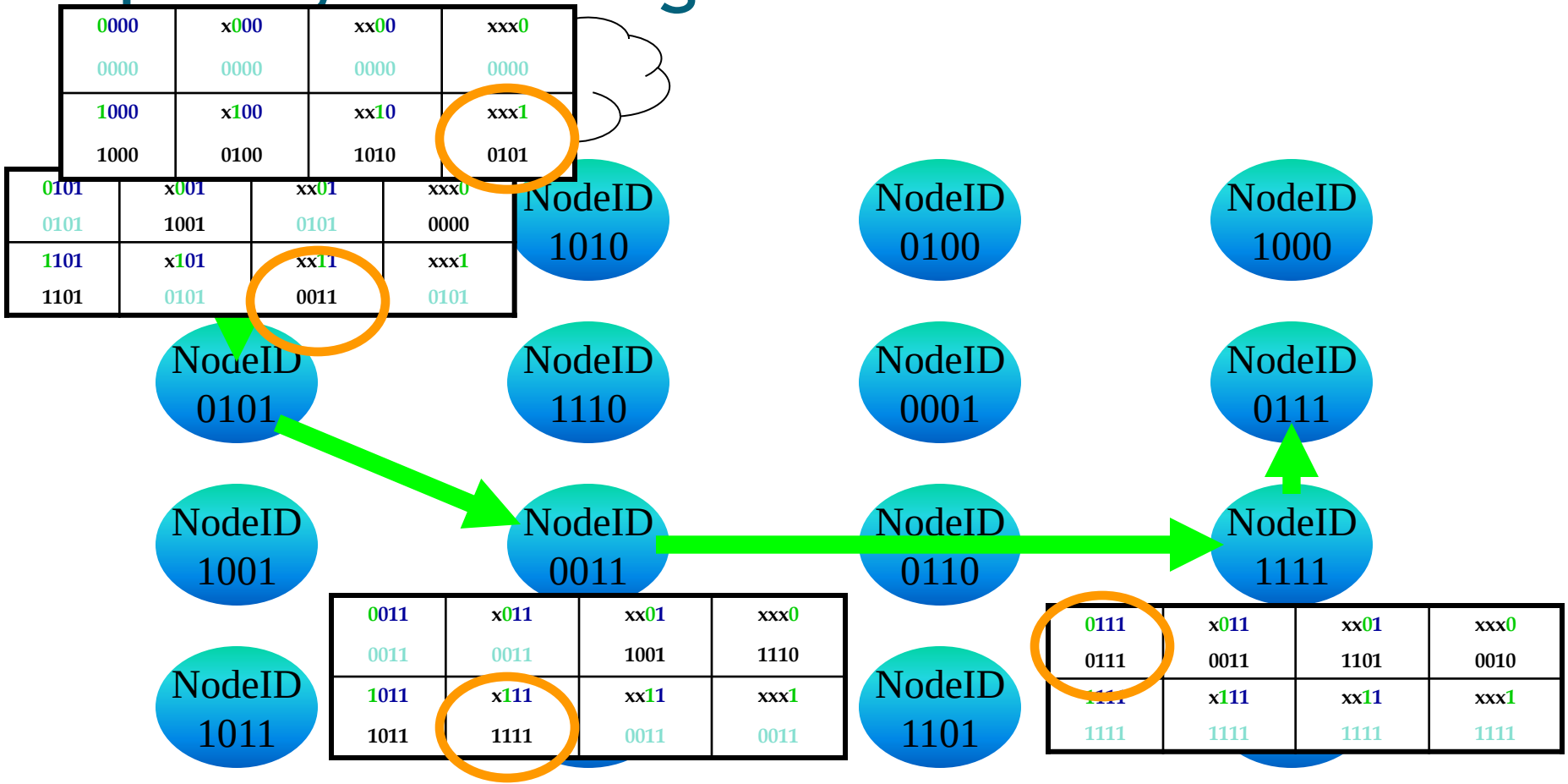
Példa

5712 → 7510

5712	0	1	2	3	4	5	6	7
0330	0	1	2	3	4	5	6	7
3210	0	1	2	3	4	5	6	7
4510	0	1	2	3	4	5	6	7
7510	0	1	2	3	4	5	6	7



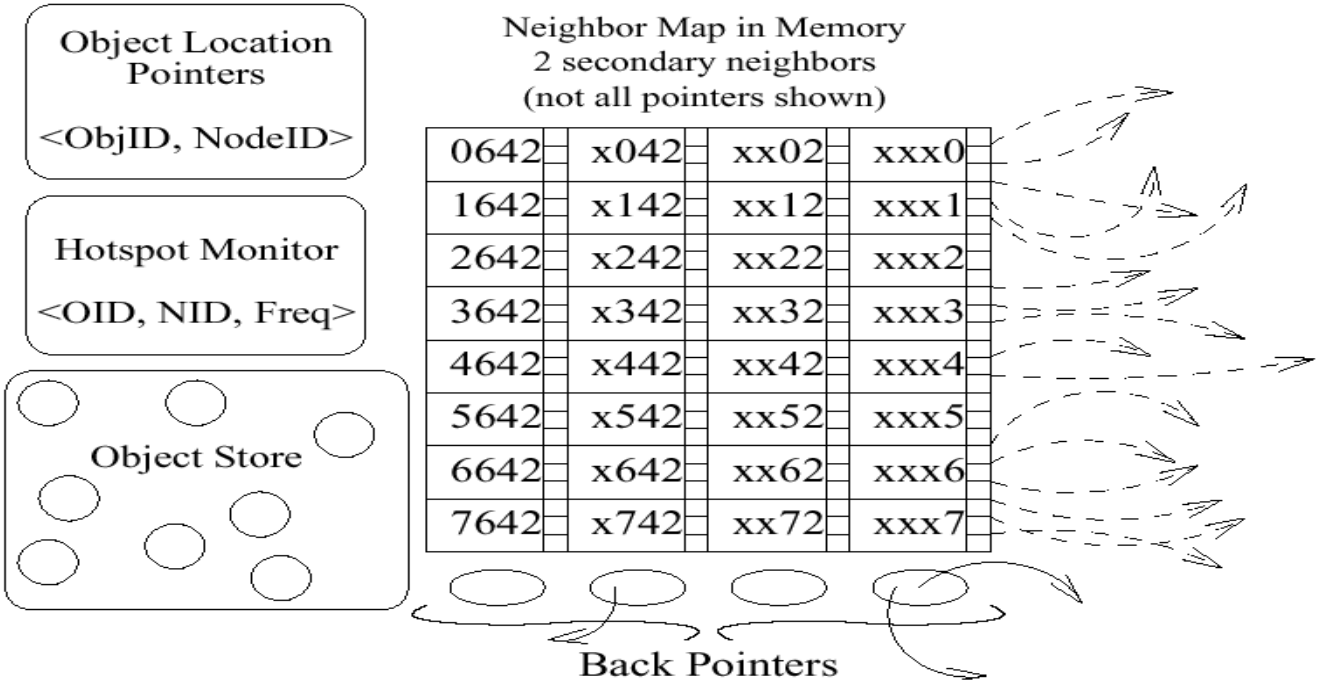
Tapestry – routing



Tapestry csomópont N

- Neighbour Map
- Object Store
  - A helyileg tárolt állományok
- Object Location Pointers
  - Információk bizonyos állományok tárolási helyéről
  - <ObjectID, NodeID>
- Back Pointers
  - Azokra a pontokra mutatnak, melyek szomszéduknak tekintik N-t
- Hotspot Monitor
  - <ObjectID, NodeID, Frequency> - segítenek a cache kezelésében

Tapestry csomópont



Root Node (Plaxton)

- Adott egy A állomány (ID_A)

- Az A állomány gyökere (root node) az az R csomópont,
- melyre igaz a következő:

$$utótag(ID_A, k) = utótag(ID_R, k)$$

  - és nincs olyan más csomópont X melyre igaz lenne, hogy

$$utótag(ID_A, k+1) = utótag(ID_X, k+1)$$

- Ha több ilyen pont van, a legnagyobb címmel rendelkező
- lesz a root node

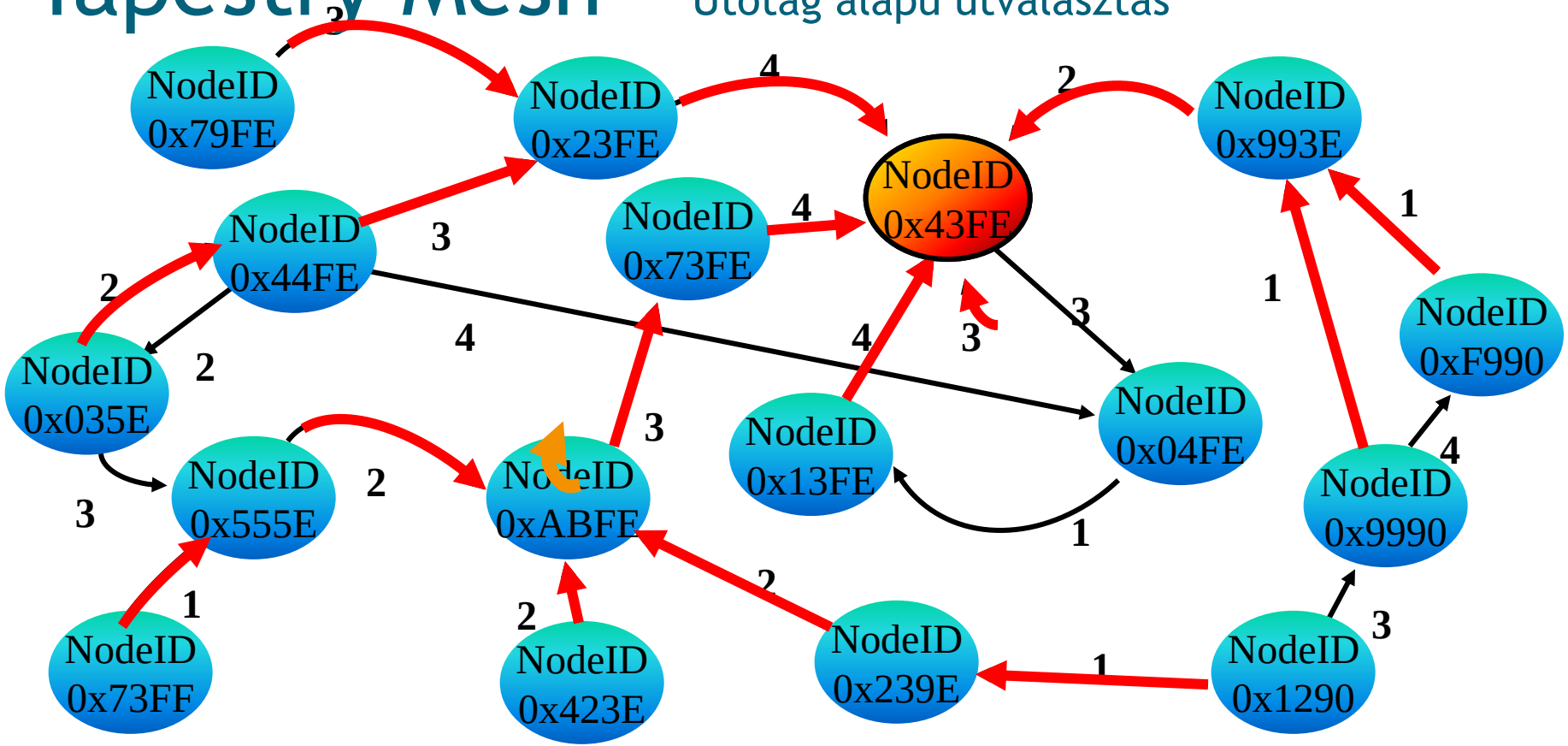
A feszítőfa

- Root(A) az a pont, ahova mindeki fordul ha A-ra kíváncsi
  - Minden A állományhoz egy Root(A) gyökerű feszítőfa tartozik

- A hálózat bármely pontjáról, véges számú lépés alatt eljutunk a
- feszítőfa gyökeréhez
  - Számjegyenként egyre közelebb kerülünk, ameddig egy üres szomszéd bejegyzéshez érünk
  - Egy utolsó ugrásként egy shortcut vezet a root-hoz
  - Információt szerzünk az A állományról

- Statikus megoldás, a hálózat teljes ismerete szükséges
  - Az összes shortcut-ot előre ki kell számolni

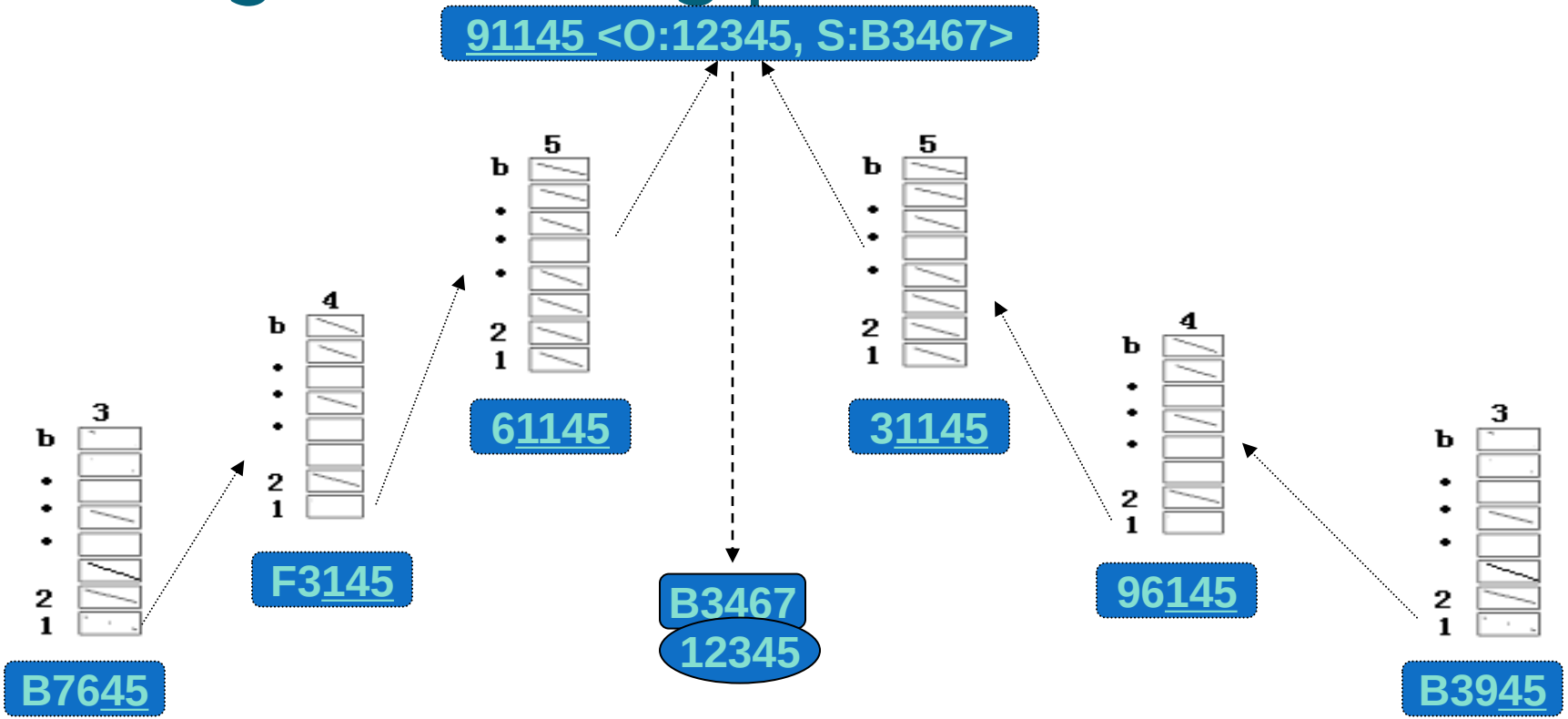
Tapestry Mesh - Utótag alapú útválasztás



Root node (Tapestry)

- Surrogate (helyettesítő) routing
  - Elosztott megoldás a root kiszámolására
  - Ha egy üres szomszéd bejegyzésre bukkan, kiválasztja az azon a szinten levő következő nem üres bejegyzést
  - Ha egy szint után egyetlen bejegyzés sincs saját magán kívül, megáll
  - Ez a pont lesz a surrogate (root)

Surrogate Routing példa

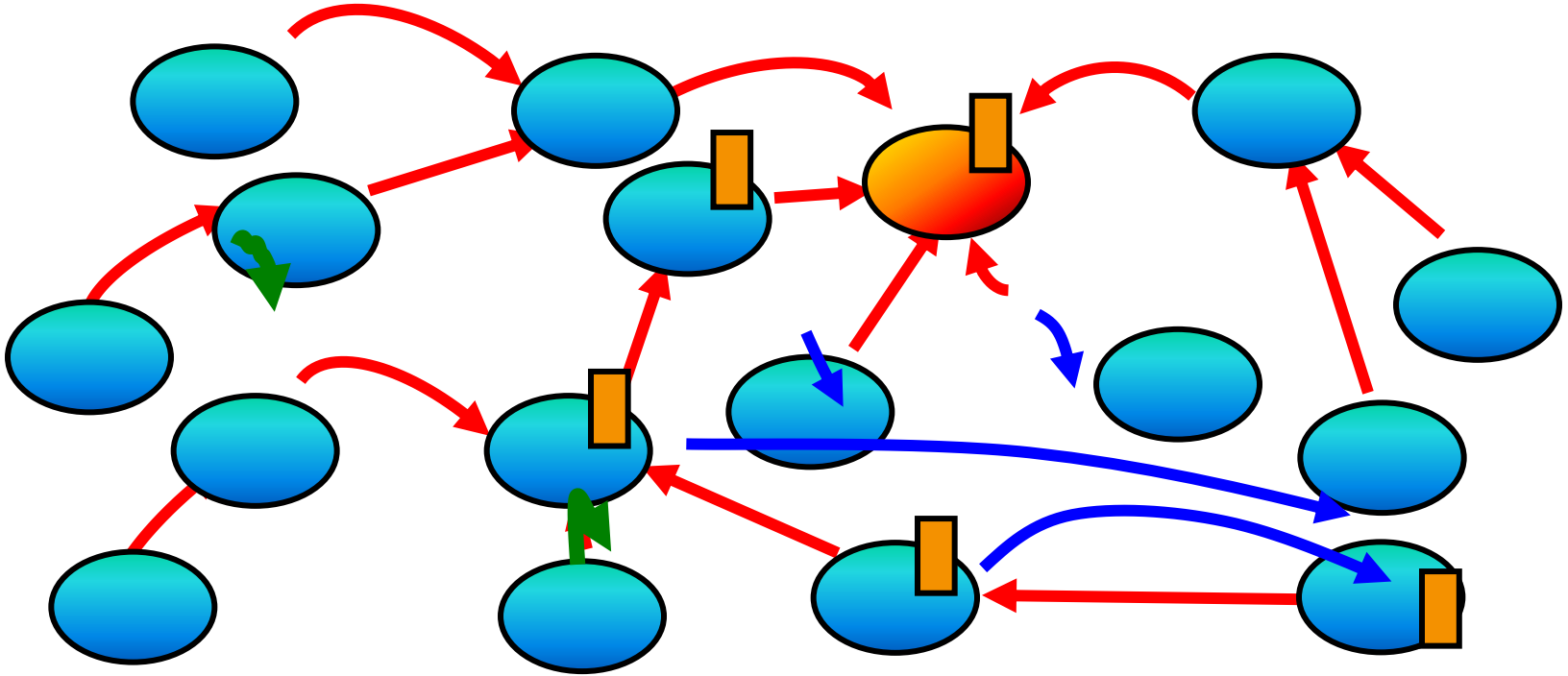


Lokalizáció

- Egy szerver S bejelenti hogy rendelkezik az A
- állománnyal
  - S elküld egy Publish (ObjectID(A), ServerID(S)) üzenetet a Root(A) felé
  - Minden közbeeső router tárolja a linket (A→S)

- Query(A) → Root (A) felé
  - Ha útközben valaki tárolta a linket, a kérést azonnal továbbküldi a megfelelő helyre

Lokalizáció



Tapestry alkalmazások

- Mnemosyne
  - Biztonságos elosztott fájlrendszer
    - A Freenet-hez hasonló szolgáltatás
  - http://www.cs.rice.edu/Conferences/IPTPS02/107.pdf
- Bayeux
  - Alkalmazás rétegbeli multicast megoldás
    - Későbbi előadásban részletesen
- Spamwatch
  - Elosztott spam szűrő
  - A felhasználók címkézik a spam leveleket
  - Az e-mail szerverek egy Tapestry hálóba kötve
  - http://www.zhoufeng.net/eng/spamwatch/
- OceanStore
  - Elosztott tárolás
  - http://oceanstore.cs.berkeley.edu/

Összefoglalás

- 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…