Sisteme Peer-to-Peer

Doua caracteristici definitorii

1. Sisteme distribuite alcatuite din noduri cu capabilitati egale (peers) care interactioneaza direct, fara intermedierea unui server central, pentru a realiza
   - partajarea resurselor – continut, procesor, memorie, banda
   - conectarea la- si deconectarea de la vecini,
   - rutarea mesajelor,
   - cautarea si localizarea altor noduri,
   - memorarea in cache a continutului,
   - criptarea, publicarea, regasirea, decriptarea si verificarea continutului,
   - etc.
2. Au abilitatea de a trata instabilitatea si conectivitatea variabila
   - adaptarea la defectari in conexiuni si noduri
   - adaptarea la o populatie de noduri variabila

Definitie

- *Peer-to-peer systems are distributed systems consisting of interconnected nodes*
- *able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth,*
- *capable of adapting to failures and accommodating transient populations of nodes*
- *while maintaining acceptable connectivity and performance,*
- *without requiring the intermediation or support of a global centralized server or authority.*

STEPHANOS ANDROUTSELLIS-THEOTOKIS and DIOMIDIS SPINELLIS **A Survey of Peer-to-Peer Content Distribution Technologies** ACM Computing Surveys, Vol. 36, No. 4, December 2004

Utilizare Peer-to-Peer

- *Communicare si Colaborare*
  - Jabber.org pentru instant messaging
  - Internet Relay Chat (IRC)
- *Calcul Distribuit*
  - SETI@home, genome@home
- *Suport servicii Internet si sisteme distribuite*
  - sisteme multicast peer-to-peer
  - infrastructuri de indirectare in Internet
  - securitate, protectie impotriva DoS sau virusi
- *Sisteme de baze de date*
  - Local Relational Model (LRM) – baze de date locale cu translatare intre ele
  - PIER – motor de interogare distribuit, scalabil
  - Edutella – infrastructura de metadata si capabilitati de interogare
- *Content distribution*
  - Napster, Gnutella, Freenet, BitTorrent

Categorii P2P ptr distributie de continut

- Sisteme Peer-to-Peer pentru schimb de fisiere
  - stabilesc o retea de peers care cauta si interschimba fisiere
  - simple – nu trateaza securitatea, disponibilitatea, persistenta
  - exemple: Napster, Kazaa, Gnutella
- Sisteme P2P pentru stocare si publicare de continut.
  - creaza un mediu distribuit in care utilizatorii pot publica, stoca si distribui continut
  - au capabilitati suplimentare schimbului de fisiere
    - continutul este accesibil in mod controlat (cf. privilegiilor)
    - gestiunea continutului – actualizare, stergere, control versiuni
    - trateaza persistenta, disponibilitatea, securitatea
  - exemple: Napster, Gnutella, Freenet, BitTorrent, Groove

- Infrastructuri Peer-to-Peer - Ofera servicii peer-to-peer ca suport pentru distribuitie de continut
  - Rutare si localizare
    - Chord
    - CAN
    - Pastry
    - Tapestry
    - Kademlia.
  - Anonimitate
    - Anonymous remailer mixnet.
    - Onion Routing.
    - ZeroKnowledge Freedom.
    - Tarzan.
  - Management reputatie
    - Eigentrust.
    - PeerTrust.

Localizare si rutare P2P

Modelul Retelei Overlay

- ***Pur Descentralizat***
  - nu exista coordonare centrala
  - toate nodurile sunt servere si clienti (servents)
- ***Partial Centralizat***
  - Similar cu cel dinainte dar
  - unele noduri (numite supernoduri) centralizeaza indecsi pentru fisiere partajate de peers locali
  - supernodurile sunt asignate dinamic (nu sunt "single point of failures")
- ***Hibrid Descentralizat***
  - un server central faciliteaza cautarea si identificarea nodurilor care contin fisierele cautate (prin mentinerea unor directoare pentru metadate)
  - schimburile de date se fac direct intre peers

Structura Retelei

- ***Nestructurata***

  Plasarea continutului nu are legatura cu topologia overlay

  – mecanisme de cautare posibile
    - inundare
    - cai alese aleator (random walks)
    - folosirea unor indici de rutare

  – potrivita pentru populatii foarte dinamice de noduri
    - dar, au probleme de scalabilitate

- ***Structurata***

  Ofera o mapare intre continut (e.g. identificator fisier) si locatie (e.g. adresa nodului), in forma unei tabele de rutare distribuite

  – se cunoaste identificatorul datelor (nu cautare dupa cuvinte cheie!)

  – interogarile pot fi dirijate eficient

  – dar, greu de pastrat structura pentru populatii dinamice

Structura retelei cu exemple

	Centralizare		
	Hibrid descentralizat	Partial centralizat	Pur descentralizat
Nestructurate	Napster, Publius	Kazaa, Morpheus, Gnutella, Edutella	Gnutella, FreeHaven
Infrastructuri Structurate			Chord, CAN, Tapestry, Pastry
Sisteme Structurate			OceanStore, Mnemosyne, Scan, PAST, Kademlia, Tarzan

Arhitecturi nestructurate - Hibrid descentralizate

Clientii pastreaza continutul

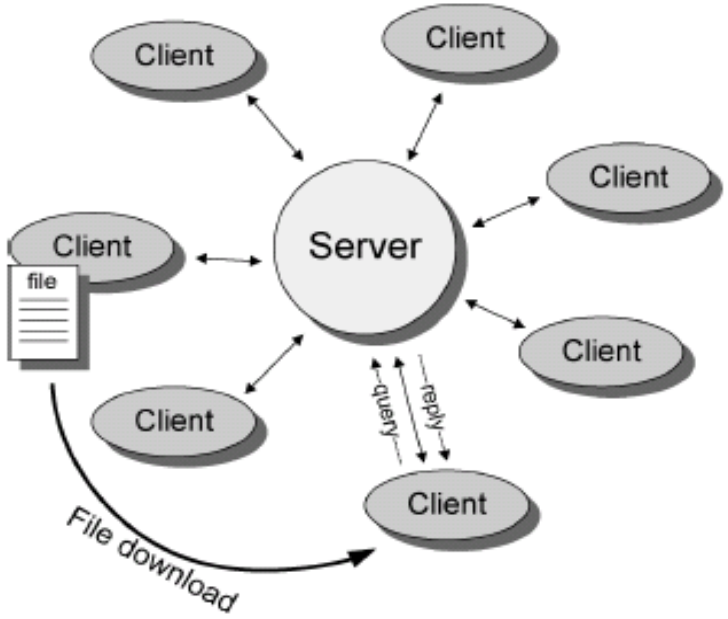
- fisierele pe care le partajeaza cu alti clienti

Serverul include meta-date despre fisiere

- un tabel cu **conexiunile** (adresa IP, largimea de banda a conexiunii etc.) clientilor inregistrati
- meta-date despre **fisierele** pe care clientii le partajeaza cu ceilalti (nume fisier, data crearii etc.)

Clientii contacteaza serverul in doua ipostaze

- pentru a se inregistra si a raporta fisiere pe care le detin
- pentru a afla informatii despre fisiere



- Pentru a descarca un fisier
  - clientul trimite o cerere serverului
  - serverul cauta in tabelul de fisiere si trimite lista clientilor care au fisierul
  - clientul descarca fisierul direct de la una din surse
- Avantaje
  - simplitatea
- Dezavantaje - un singur server detine controlul
  - sunt vulnerabile cenzurii, supravegherii, actiunilor legale, atacurilor malitioase si defectelor tehnice
  - sunt nescalabile
- Au utilizare (limitata la anumite faze) si in alte arhitecturi; ex.
  - pentru pornirea sistemului (bootstrap) - MojoNation
  - la alaturarea unui nou client furnizeaza o lista a clientilor cu care se pot conecta - Gnutella

Arhitecturi nestructurate pur descentralizate

Localizarea unui fisier

Localizarea se bazeaza pe cautare nedeterminista

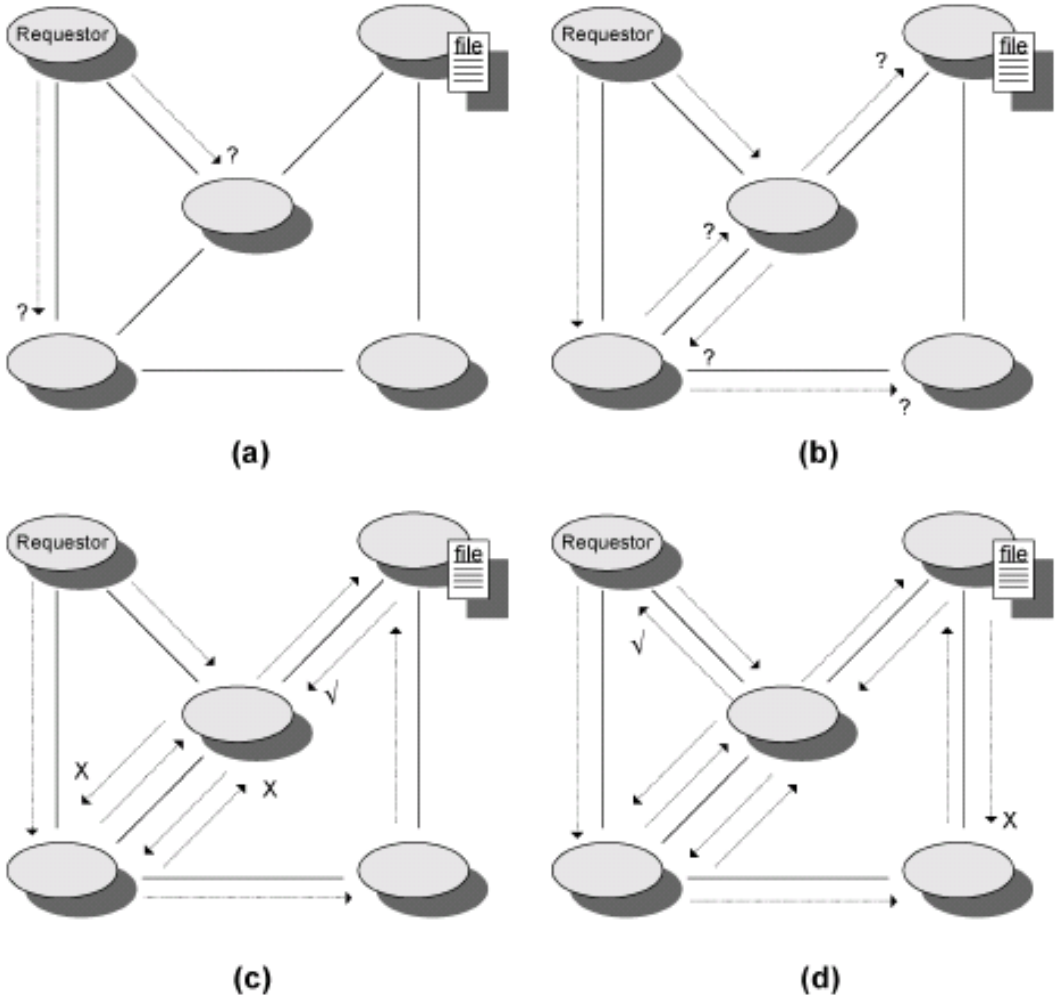
Solutia originala Gnutella

- transmitere mesaje de cerere prin inundare
- fiecare mesaj are un identificator unic
- un nod pastreaza o tabela de dirijare a raspunsurilor

  cu perechi <identificator mesaj, calea de retur>
  - elimina cererile duplicatele
  - transmite raspunsul (cu acelasi identificator) pe calea inversa
- limitare trafic prin includere in antet a unui camp Time-To-Live
  - **problema**: mesajele unui utilizator nu pot trece de un orizont limitat → segmenteaza reteaua in sub-retele

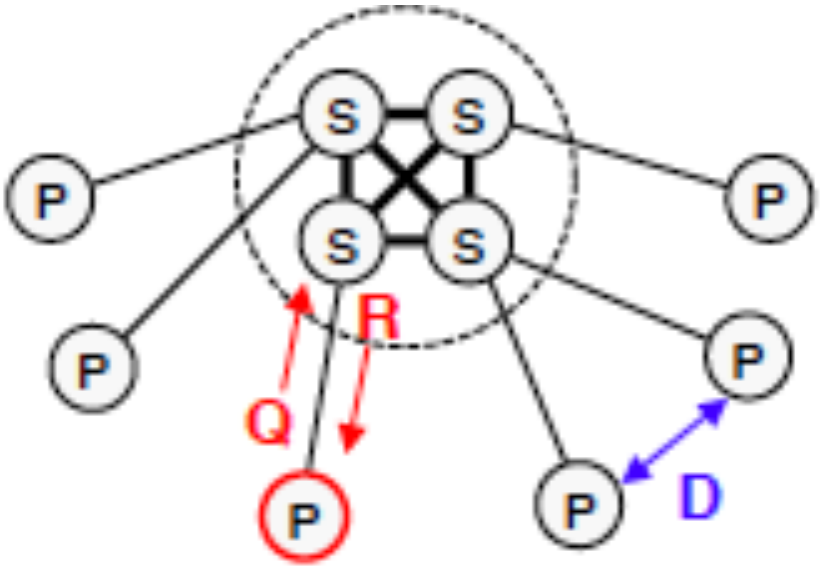
Un exemplu de cautare in Gnutella

- Cautarea incepe la Requestor
- mesajele Request (?) sunt trimise tuturor vecinilor si propagate din nod in nod
- la primirea aceleiasi cereri, nodul raspunde cu "failure" (X) pentru a evita ciclurile si a minimiza traficul
- cand este identificat fisierul, nodul raspunde cu un mesaj de succes (√).



Arhitecturi nestructurate Partial Centralizate

- *supernoduri* - asignate dinamic sa serveasca o mica parte a retelei de peers
  - indexeaza (si memoreaza in cache) fisierele partajate de peers conectate la ele
  - intermediaza (proxy) cererile de cautare pe seama acestor peers
    - cererile de cautare sunt transmise initial la supernoduri.
- supernodurile sunt alese automat dintre peers cu largime de banda si putere de procesare suficiente



P – Peer
S - Superpeer
Q – Querry
R – Response
D – P2P Download

Arhitecturi nestructurate Partial Centralizate (2)

- Avantaje
  - timp de descoperire redus in raport cu arhitecturi centralizate
  - nu exista "single point of failure"
  - exploateaza inerent heterogeneitatea peers
    - noduri mai puternice (supernodurile) sunt mai incarcate, altele cu resurse reduse sunt mai putin incarcate
- Exemple:
  - Kazaa, Edutella,
  - mai recent Gnutella = o interconectare de superpeers si clienti

Solutii noi pentru Arhitecturi nestructurate

- inlocuirea inundarii cu alegere aleatoare mai multe cai paralele
  - fiecare nod alege aleator vecini carora le paseaza cererea
  - sau alege cai dirijate spre noduri de mare capacitate

- utilizare indecsi locali plus replicare proactiva:
  - fiecare nod mentine un index al datelor stocate la noduri localizate pana la o anumita distanta
  - plus replicarea proactiva – numarul de replici este proportional cu rata interogarilor obiectului replicat

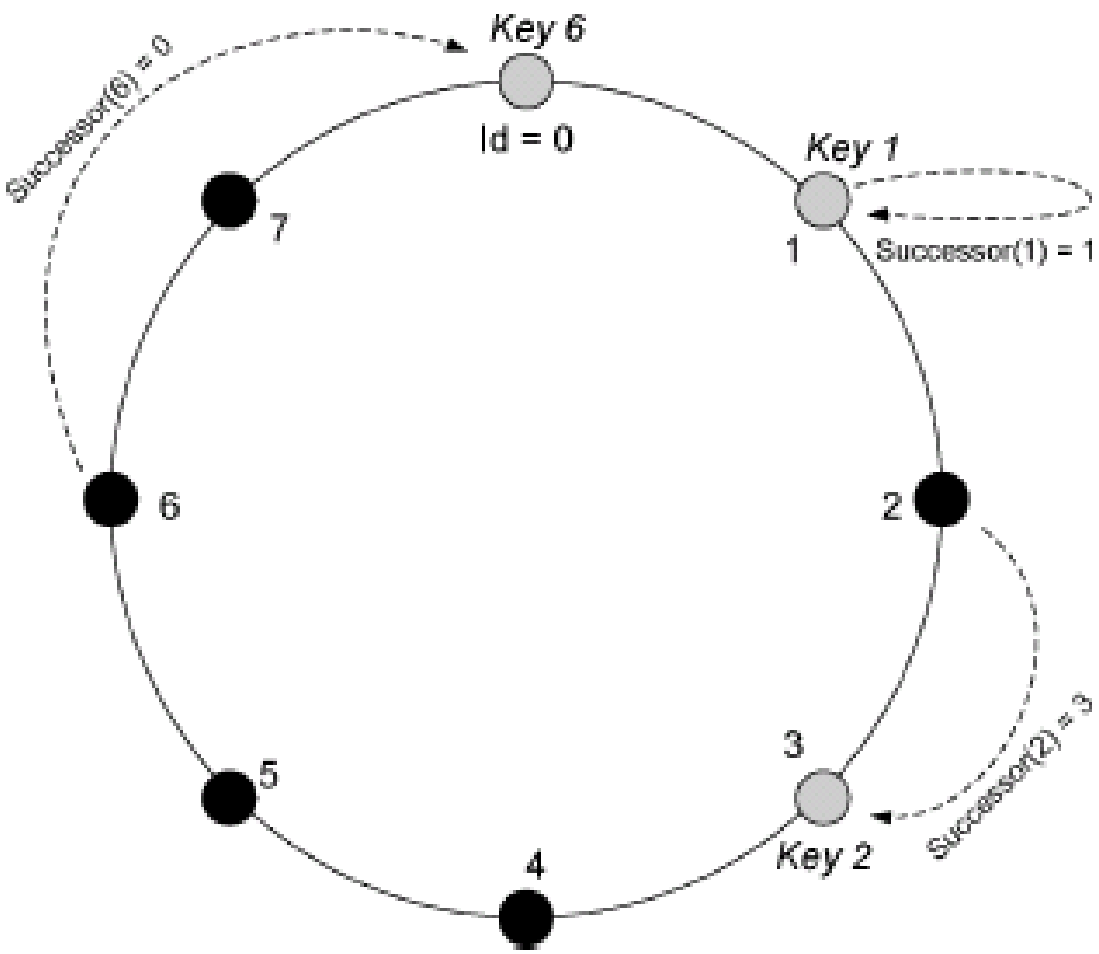
- *mecanisme de cautare inteligenta*
  - fiecare peer paseaza interogarile unui subset de vecini, selectati pe baza performantei in interogarile precedente

Arhitecturi structurate - Chord

Chord isi bazeaza localizarea fisierelor pe o tabela de rutare distribuita mai multor noduri (DHT – distributed hash table)

- asociaza fiecarui nod un identificator de m biti obtinut prin aplicarea unei functii hash adresei IP a nodului respectiv
- asociaza fiecarui fisier un identificator de m biti obtinut prin aplicarea functiei hash pe cheia fisierului
  - cei doi identificatori mai sunt numiti nod si cheie
- **m** poate fi 128 sau 160 (dupa metoda de hash folosita)
  - spatiul identificatorilor este foarte mare
  - doar o parte a lui este folosita pentru identificatorii nodurilor
- informatia despre un fisier cu cheia k este pastrata de un nod care are cel mai mic identificator id>=k;
  - el se numeste succesorul lui k, succ(k)

Un spatiu (inel) de identificatori cu m = 3



Exemplu

Inelul are trei noduri
0, 1, si 3

Nodurile pastreaza
info despre trei chei
1, 2 si 6

- Cu o functie *hash* *consistenta* cheile sunt distribuite uniform nodurilor
- **Principala problema**: data fiind o cheie **k** sa se afle cat mai rapid **succ(k)**

Gasirea succ(k) - solutia naiva

- nodurile sunt organizate in inel si fiecare nod p pastreaza informatii despre
  - nodul din inel care **succede** p notat succede(p)
  - nodul care **precede** p notat precede(p)
- cand p este solicitat sa rezolve cheia k la adresa succ(k)
  - p returneaza adresa proprie cand precede(p) < k <= p
  - p trimite cererea lui precede(p) cand k <= precede(p)
  - p trimite cererea lui succede(p) cand k > p
  - in ultimele doua cazuri operatiile sunt repetate de nodul care primeste cererea
- Complexitate de ordinul numarului de noduri

Solutia Chord – continutul finger table

- Fiecare nod **p** pastreaza o tabela de rutare - finger table, FTp cu cel mult **m** intrari (m = numar biti ai cheilor)
- o intrare contine (in principal)
  - un identificator Id si
  - adresa (IP si port) nodului succ (Id)
- intrarile in **tabela nodului p** corespund unor chei mai mari decat p cu valori date de puteri ale lui 2: **1**, **2**, **4** …
  - intrarea **i** corespunde cheii **p** + $\mathbf{2^{i-1}}$ unde **i = 1..m**

    si contine **succ (p +** $\mathbf{2^{i-1}}$**)**
  - prima intrare din tabela lui p contine **succ (p+1)**

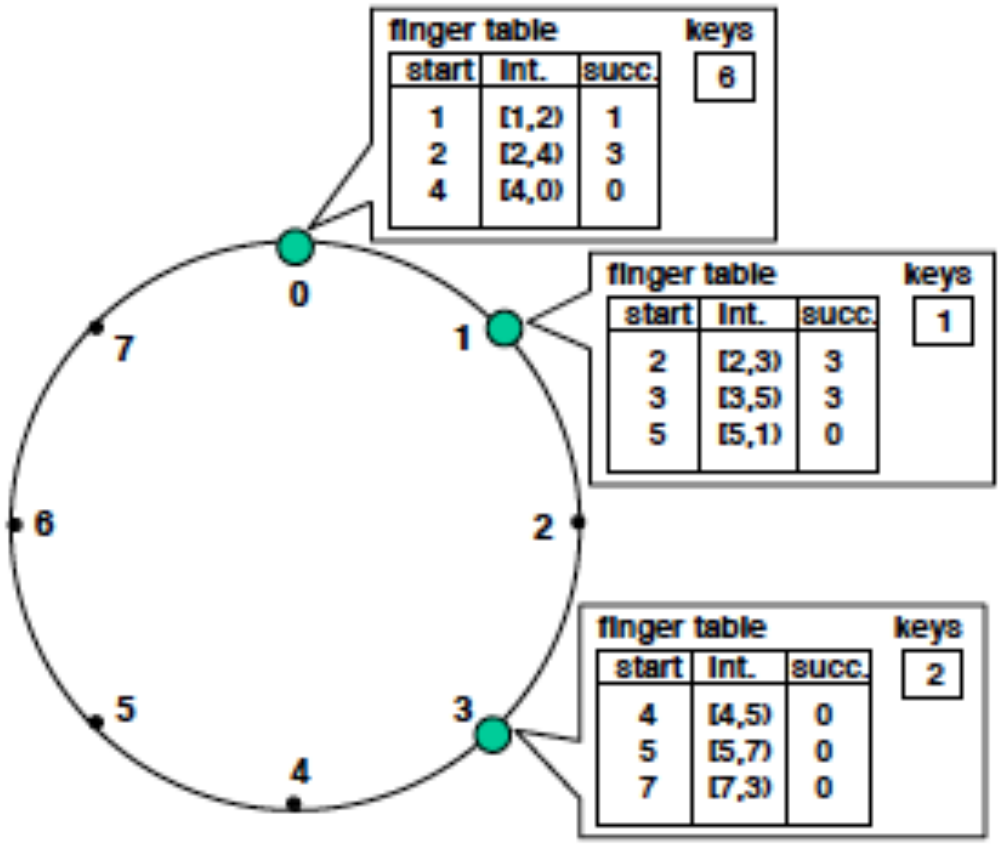
Exemplu: Tabele de dirijare pentru o retea cu m=3,

nodurile existente au identificatorii 0, 1 si 3

si cheile pastrate de aceste noduri sunt 6, 1 si 2

Tabela din nodul 1 puncteaza catre nodurile succesoare ale identificatorilor $1+2^0$=**2**, $1+2^1$=**3** si $1+2^2$=**5**

- succesorul cheii 2 este nodul 3 (primul cu cheia >= 2)
- succesorul cheii 3 este nodul 3
- succesorul lui 5 este nodul 0



Gasirea succ (k) in Chord

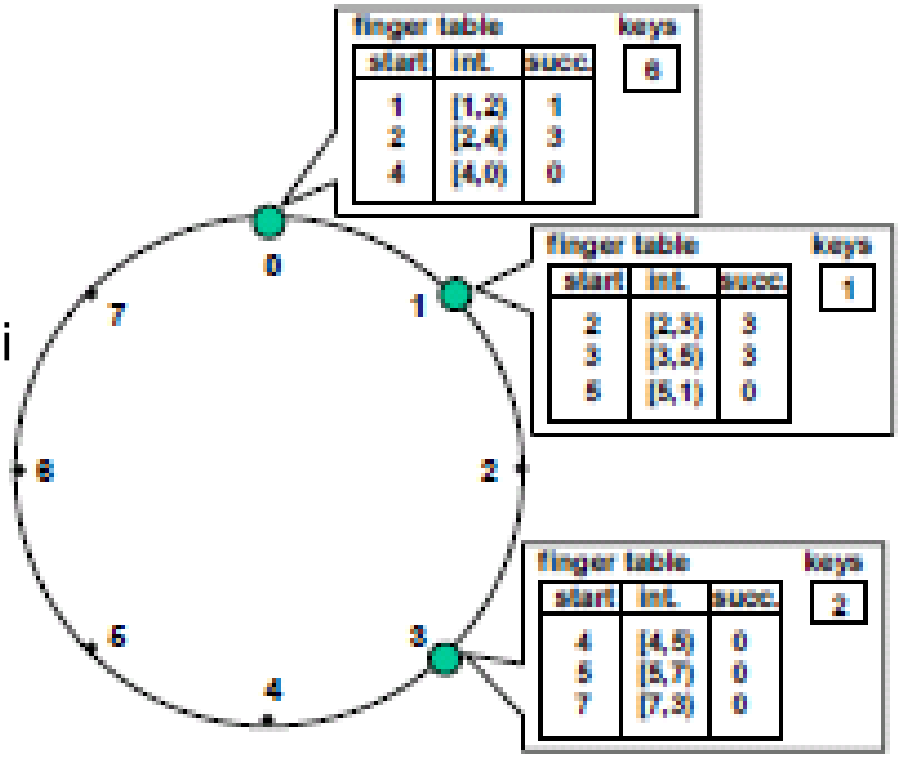
Cererea de rezolvare a cheii **k** poate fi adresata oricarui nod **p**

p trebuie sa gaseasca nodul **q** care este succesorul lui **k** prin cautare in inel in sensul acelor de ceasornic

1. **Cand** k = p → succ (k) este chiar nodul p
2. **Cand** p < k <= succ(p+1) → succ (k) este nodul care succede imediat p (este in prima pozitie a tabelei Finger a nodului p)

Exemple

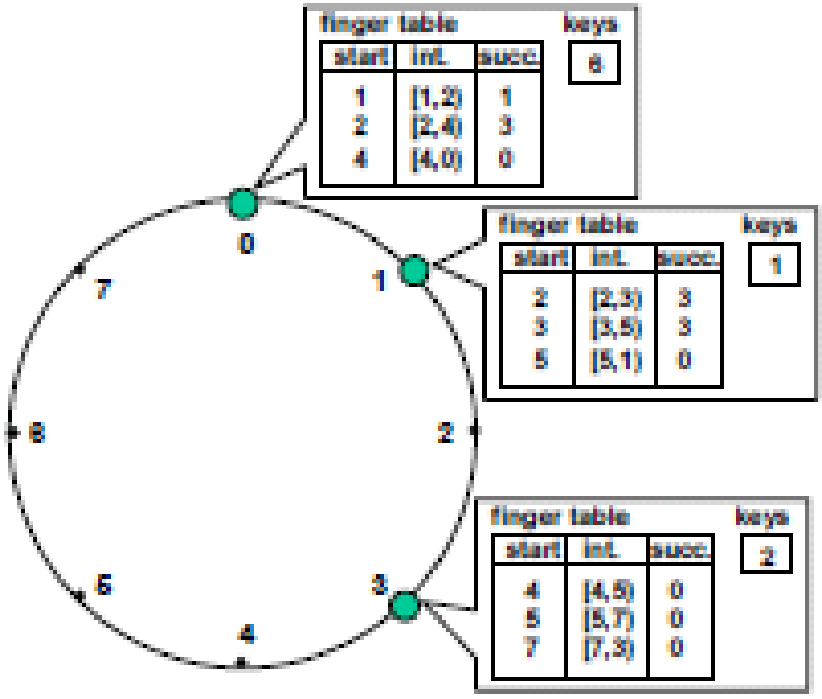
1. cererea cu cheia k=1 este adresata nodului p=1; deoarece k=p rezulta succ(k)=1
2. cererea cu cheia k=2 este adresata nodului p=1; in tabela finger a nodului 1, exista o intrare pentru 2, din care rezulta ca succ(k) = 3;



3. p nu cunoaste succ(k) – in tabela sa de rutare nu exista o intrare pentru k
- p cauta in tabela o intrare pentru un nod j al carui ID **precede** imediat k si cere de la j ID-ul nodului succ(k)
- procesul se repeta daca **j** nu are o intrare pentru k

Ex. cererea cu cheia k=1 este adresata nodului p=3

- deoarece 1 nu este in tabela sa, p cere nodului 0 (care este in tabela si precede 1) sa gaseasa succ(k)
- 1 este in prima pozitie a tabelei nodului 0 si are succ(1) = 1
- nodul 0 trimite nodului 3 raspusul succ(k) = 1



Gasirea succ (k) - exemplu

In fig. o intrare in tabela finger contine: <nr. ordine intrare, nodul succesor corespunzator>

Ex. Rezolvarea cheii k=**26** pornind din nodul **1**

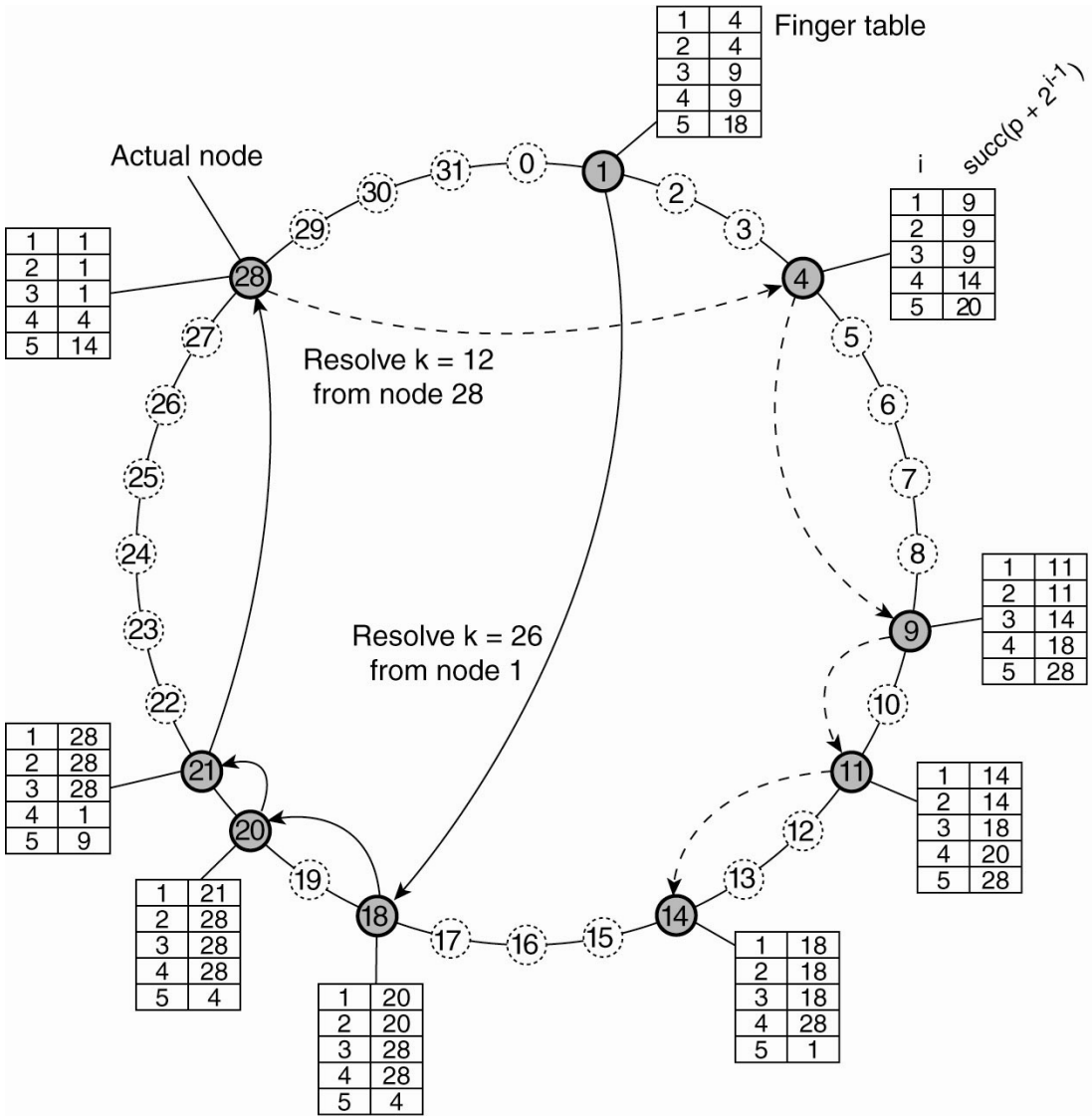
cel mai apropiat predecesor al lui 26 in tabela FT_1 a nodului 1 este 18

in FT_{18} este 20

in FT_{20} este 21

Succesorul imediat al nodului 21 este 28 care este si cel mai mic nod >= 26

nodul 21 trimite lui 1 rezultatul succ(26) = 28



Gasirea succ (k) – exemplu 2

Ex. se cauta cheia 12 pornind din nodul 28, trecand prin (linie punctata) nodurile

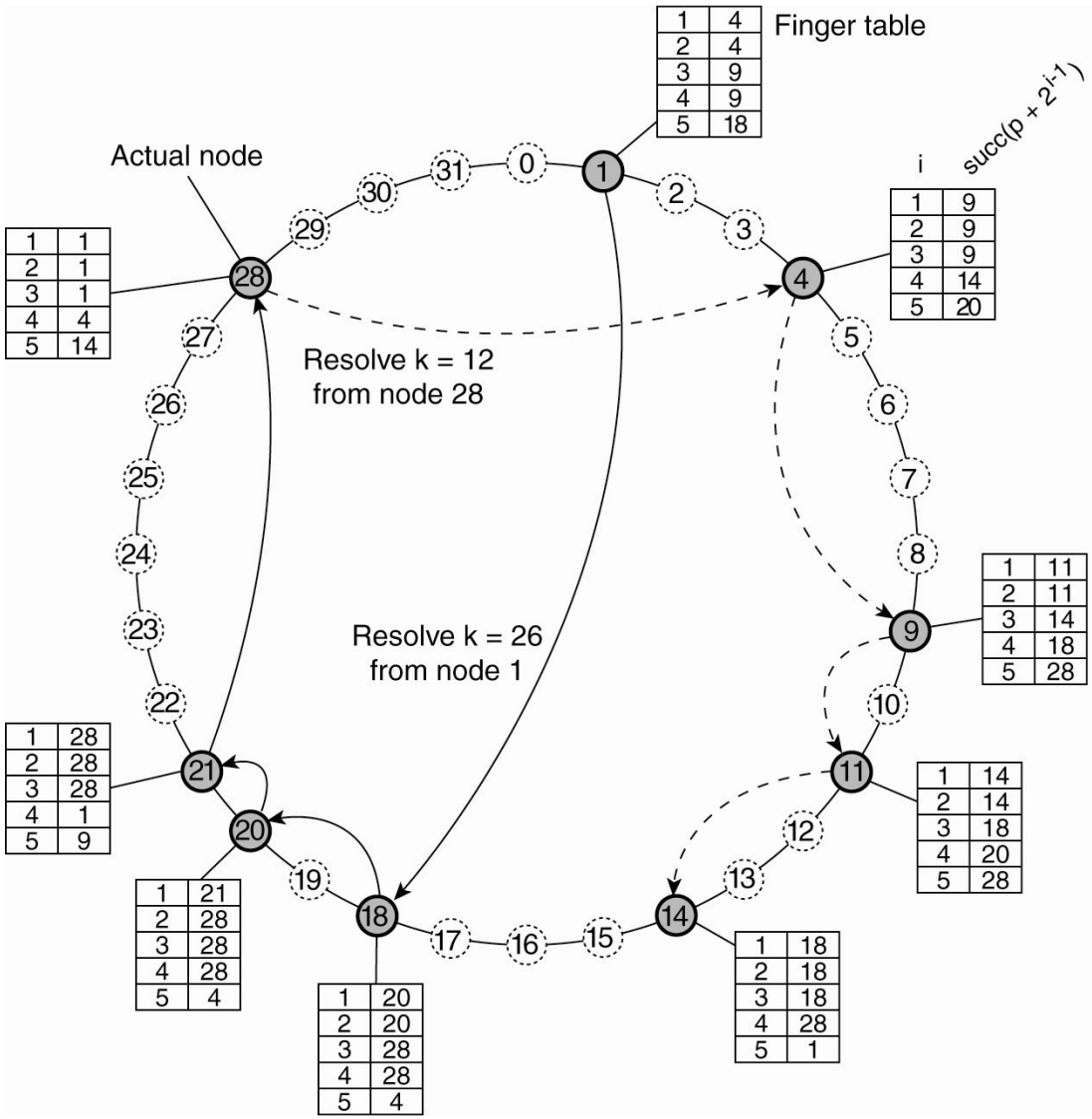
4, 9, si 11; deoarece succesorul imediat al lui 11 este nodul 14 nodul 11 trimite lui 28 raspunsul 14 = succ(12)

Complexitate

Distanta intre nodul care trateaza cererea si predecesorul cautat q se injumatateste in fiecare pas si

initial, distanta este cel mult 2^m
→in **m** pasi distanta va deveni 1

→ cautarea cere O(log N) pasi cu N noduri in retea



Intrarea unui nod in retea

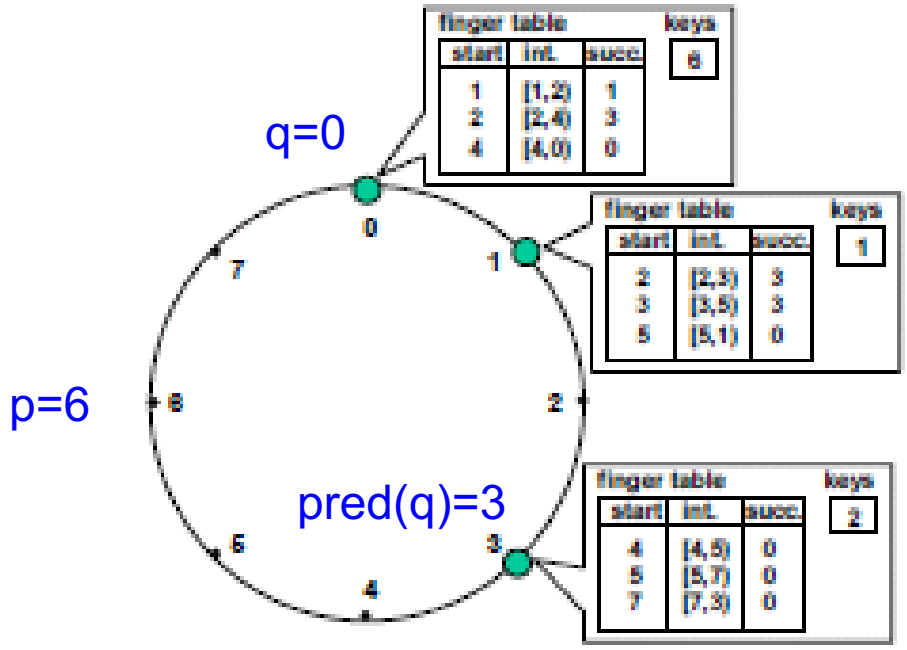
Nodul **p** intra in retea – trebuie plasat in intervalul dintre doua noduri (unul cu ID mai mic, altul cu ID mai mare) intre care nu se mai afla alte noduri

Mai precis, **p** trebuie plasat inainte de **q = succ (p)** si dupa **pred(q)**

Ex. nodul **6** este plasat intre nodurile **3** si **0**

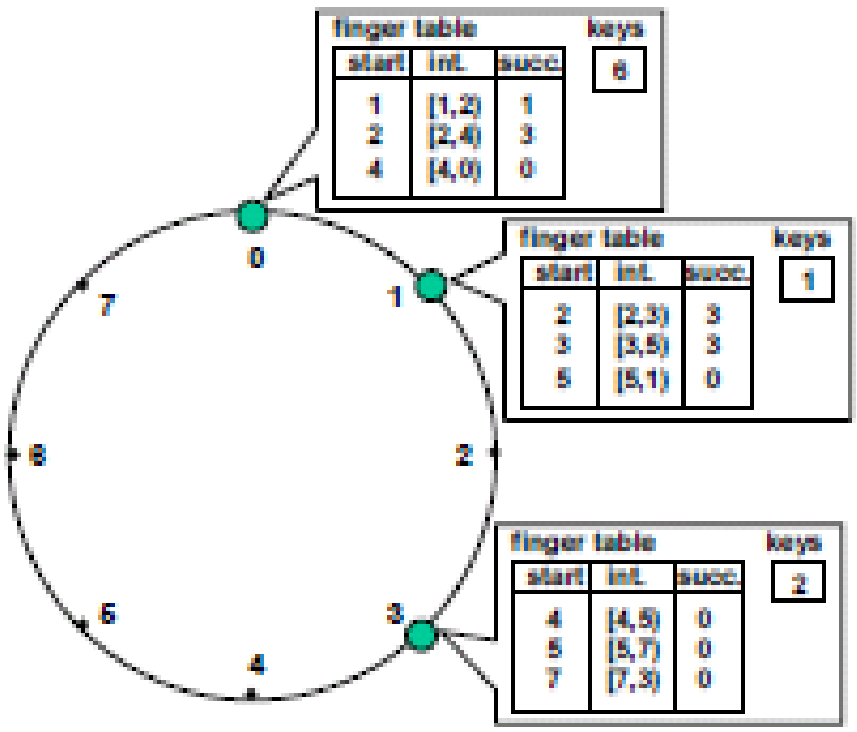
Nodul **p** contacteaza un nod existent oarecare, care

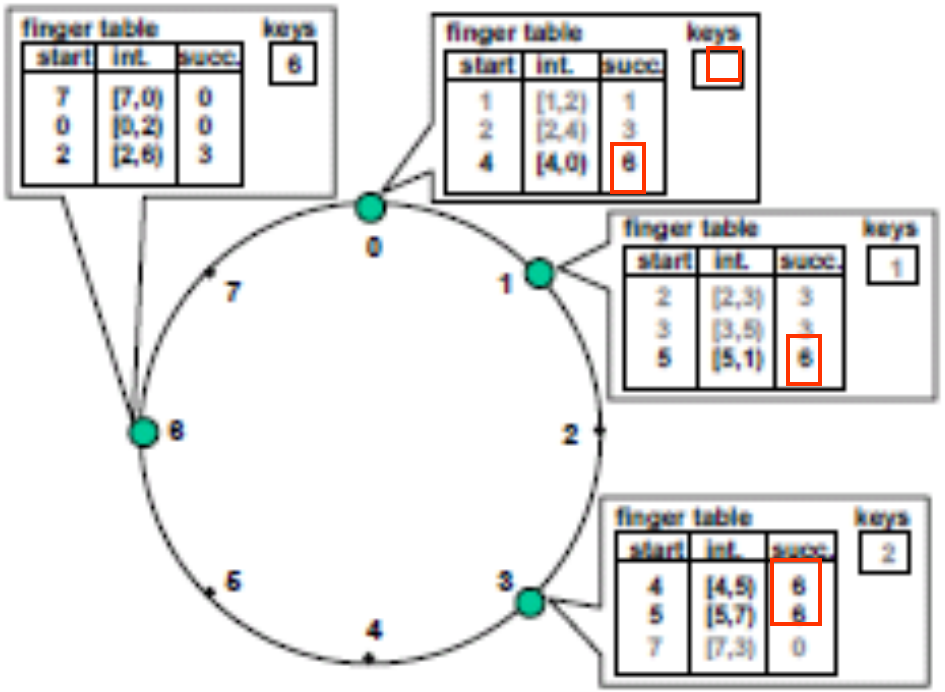
- trimite cerere de cautare pentru **q = succ(p)**
- **p** este inclus in inel intre **pred(q)** si **q**
- se modifica:
  predecesorii lui **q** si **p**,
  cheile pastrate de **q** si **p**
  tabelele Finger ale altor noduri



Intrarea unui nod in retea (2)

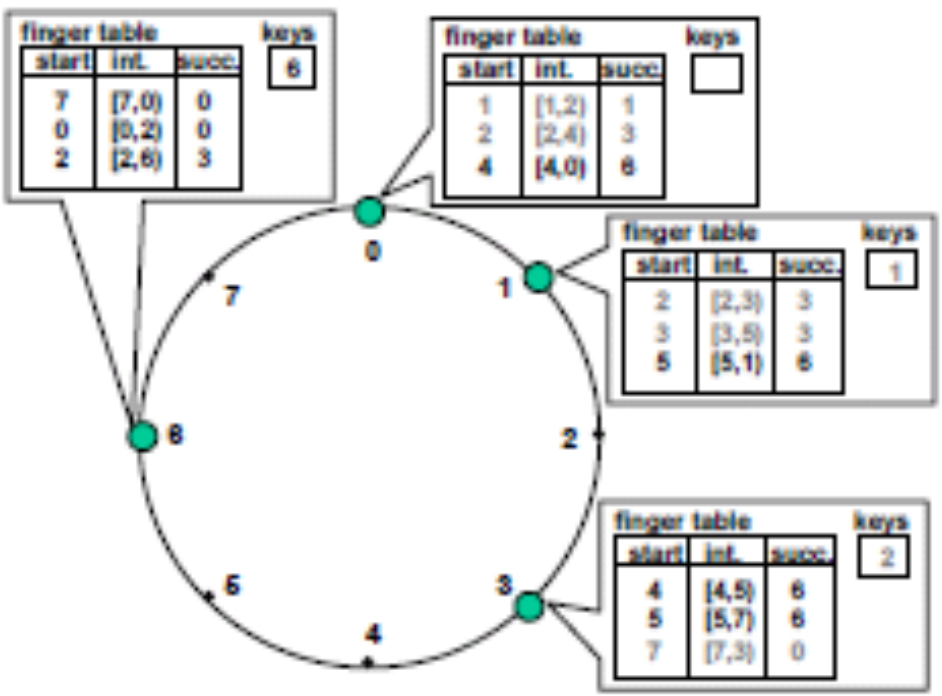
Ex. predecesorul lui 0 devine 6; predecesorul lui 6 devine 3
succesorul lui 3 devine 6; succesorul lui 6 devine 0
cheia 6 trece de la nodul 0 la 6
Se actualizeaza tabelele Finger ale tuturor nodurilor

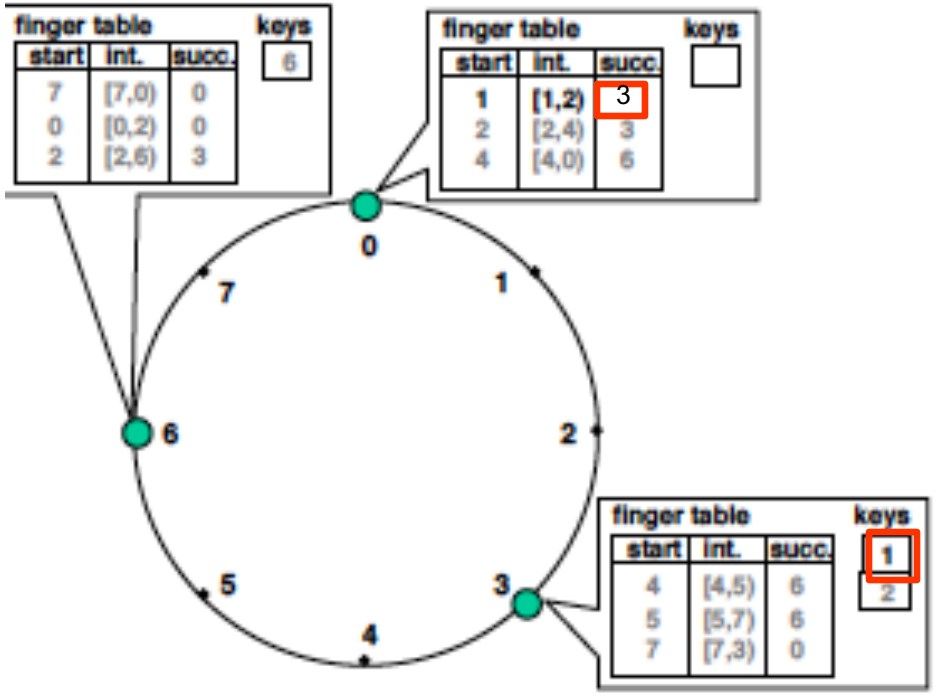




Iesirea unui nod din retea

- Cheile pastrate de nodul iesit sunt asignate succesorului
- Ex. nodul 1 iese
  - cheia **1** pastrata de nodul **1** trece la nodul **3**
  - se modifica tabela Finger a nodului **0**





Defectarea nodurilor si replicarea datelor

Chiar daca nodul **n** cade, nodurile care au intrare pentru **n** in tabela finger trebuie sa poata gasi **succesorul** lui n

In plus, cererile in derulare nu trebuie oprite

Solutionare defectari la cautare unui succesor – doua mecanisme

- fiecare nod pastreaza o lista cu cei mai apropiati **r succesori** din inel
  - cand un nod observa caderea succesorului, el ia urmatorul nod din lista
- mecanism de time-out daca nodul invocat nu raspunde
  - la time-out, nodul alege un alt finger din tabela dirijând cererea pe alta cale

Replicare – datele asociate cu o cheie sunt pastrate in **k** noduri ce succed cheia

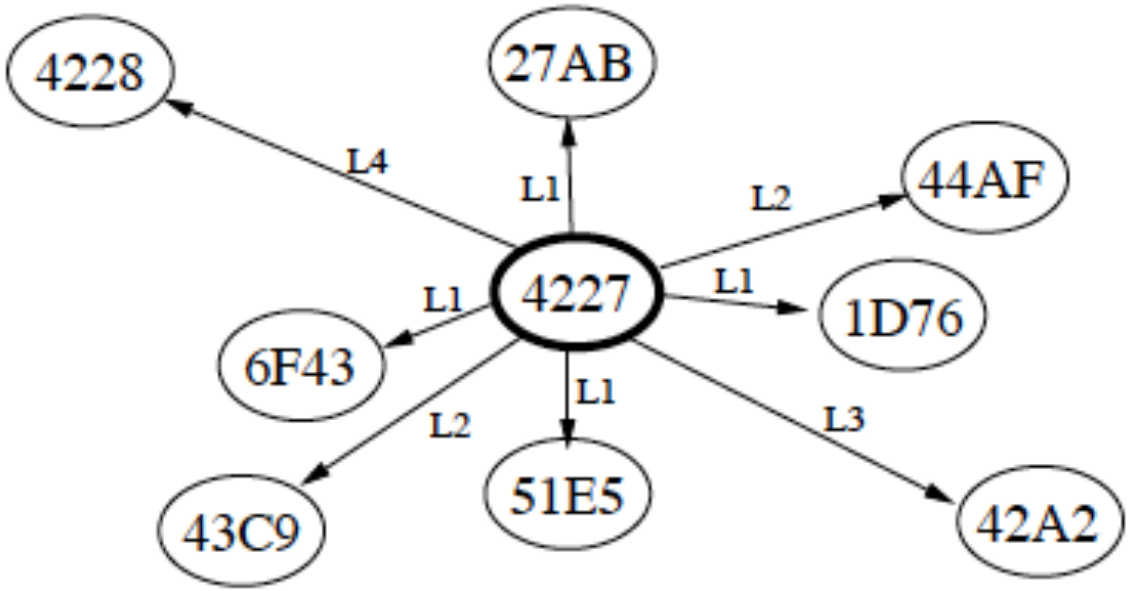
  - cand unul din cele k noduri iese din inel, se creeaza o noua replica pentru a mentine numarul k de replici

Arhitecturi Structurate - Tapestry

- Bazat pe o structura de date distribuita (*Plaxton mesh*) optimizata pentru localizarea obiectelor dupa nume
- Nodurile si obiectele din sistem au identificatori din acelasi spatiu de nume
  - un id este un sir de cifre intr-o baza specificata (de ex. hexazecimala)
  - id-urile sunt distribuiti uniform in spatiul de nume
  - node-ID identifica unic nodurile Tapestry
  - GUID (Globally Unique IDentifier) identifica obiectele (specifice aplicatiei)
- Fiecare identificator de obiect **G** este mapat pe un nod unic $\mathbf{G_R}$, denumit radacina identificatorului - identifier's *root*
- Daca exista un nod N cu N_{ID} = G, acesta este considerat radacina lui G
  - altfel, radacina are un node-ID "apropiat" de G
- Fiecare nod contine **pointeri** la nodurile vecine (neighbor links) si **pointeri** (object pointers) la noduri care memoreaza obiectele corespunzatoare unor GUIDs

Legaturile cu nodurile vecine

- Fiecare nod pastreaza **legaturi** la vecini cu care partajeaza prefix-uri
- Figura arata legaturile nodului cu node-ID = 4227
- Vecinii sunt grupati pe **nivele**, dupa lungimea prefixului partajat
  - ex. nivelul 2, **L2** include vecini cu care nodul partajeaza un **prefix de lungime 1** adica 44AF si 43C9
  - cel mai apropiat nod este numit primar, ceilalti sunt secundari



Calea unui mesaj

Mesajele catre radacina lui G sunt trimise vecinilor ale caror **node-ID**uri sunt progresiv mai apropiate de G (au in comun prefixe mai lungi)

Un exemplu: calea de la nodul cu *ID* = 5230, la nodul *ID* = 42AD

Cifre rezolvate de la stanga la dreapta: 4*xxx* → 42*xx* → 42A*x* → 42AD

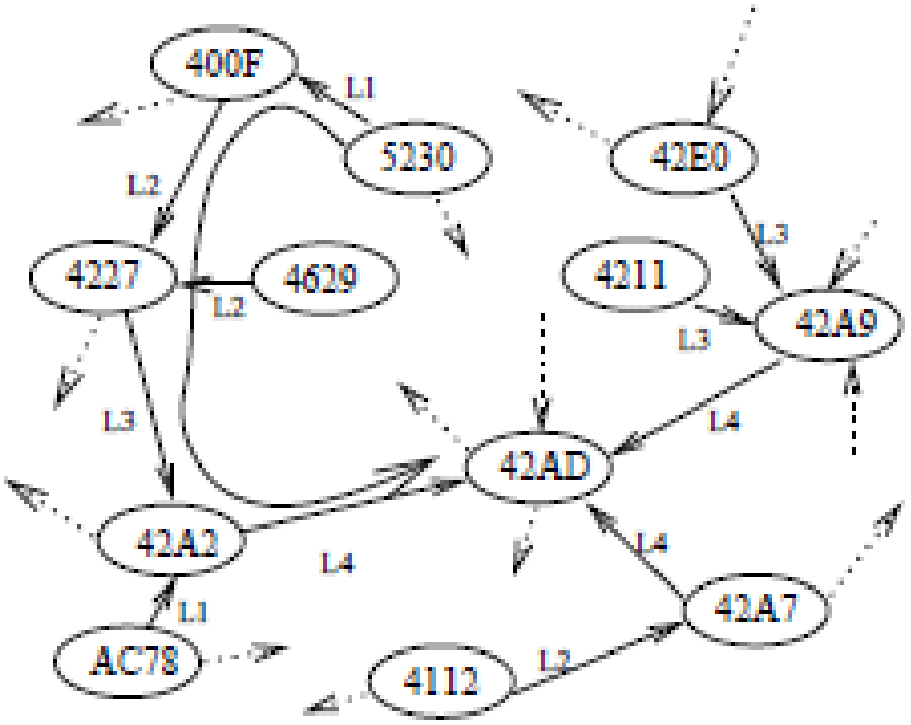
– nodul 5230 cauta vecinul care partajeaza prefixul 4xxx cu destinatia, inspectand vecinii din grupul de nivel 1

– nodul 400F – cauta vecinul cu prefix 42xx in grupul de nivel 2

– nodul 4227 - vecinul cu prefix 42Ax

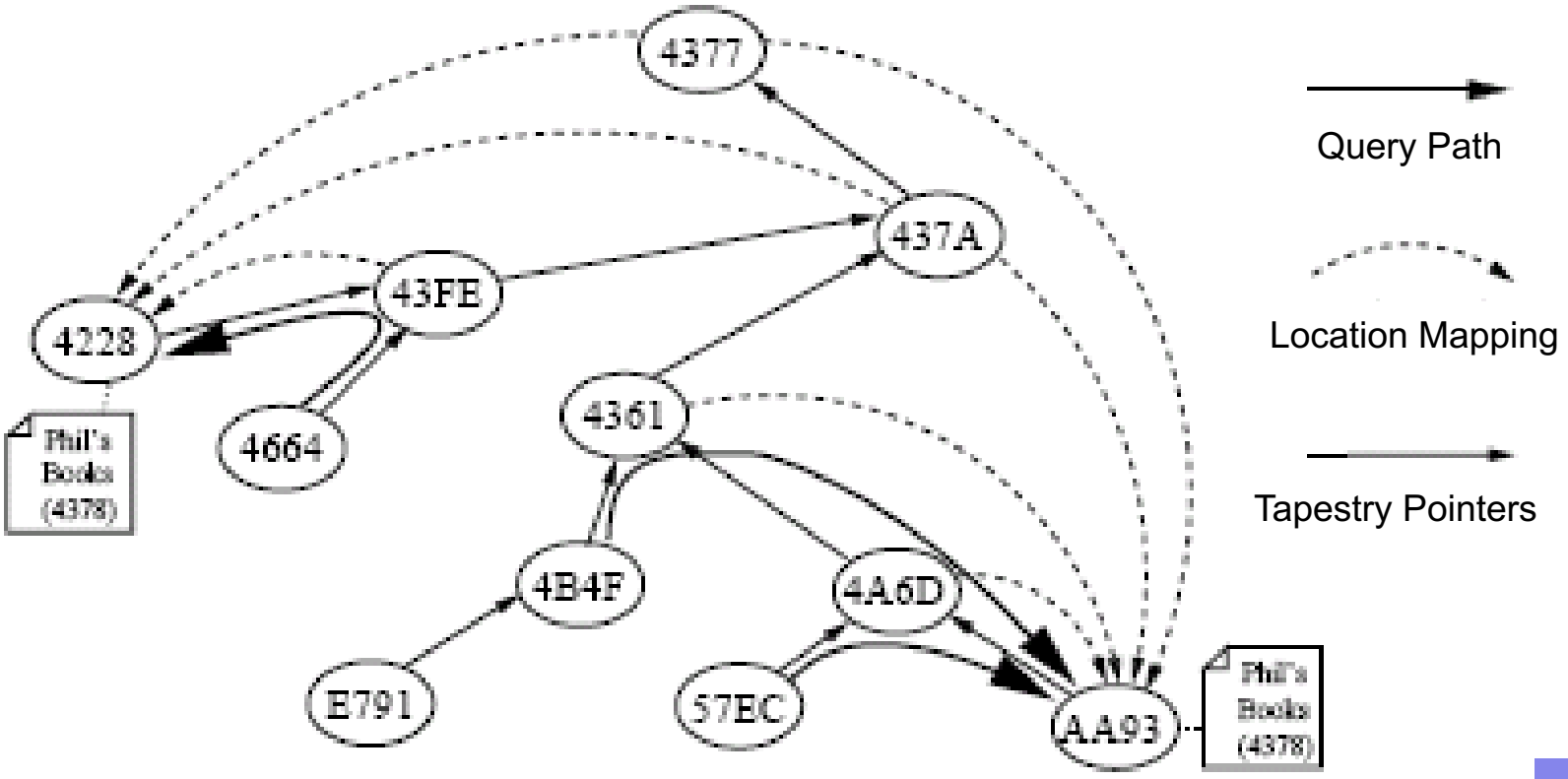
– nodul 42A2 – vecinul 42AD

Cand o cifra a numelui nu poate fi potrivita cu un vecin, Tapestry cauta o cifra apropiata in tabelul de rutare (*surrogate routing*)



Publicarea obiectelor Tapestry

Serverul S care a memorat un obiect (ex. 4378) publica periodic acest obiect prin trimiterea unui mesaj catre nodul radacina (4377)
Fiecare nod pe calea de publicare memoreaza pointeri (Location Mappings) la serverele care memoreaza obiectele, de ex. <4378,S>
Orice nod Tapestry pastreaza mapari de locatii ale cópiilor obiectelor in ordinea distantei de la nod la replici

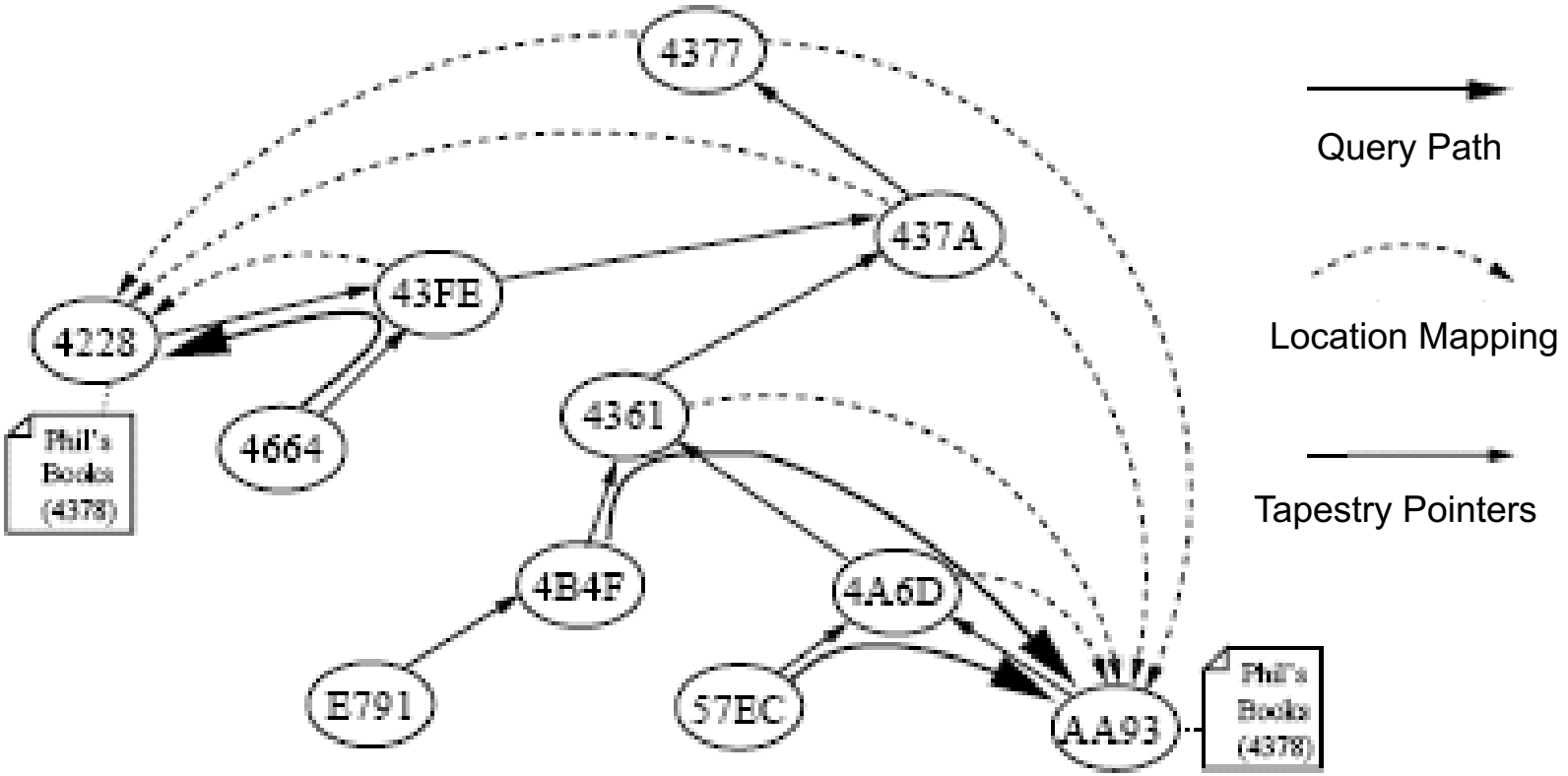


Ruta Tapestry catre obiect

Mai multe noduri trimit mesaje la obiectul 4378 din diferite puncte ale retelei

Fiecare mesaj este rutat catre nodul radacina 4377

Cand mesajul intersecteaza calea de publicare, el urmeaza pointerul catre copia obiectului (care este cea mai apropiata de nodul de origine)



Arhitecturi Structurate - Freenet

- Sistem pur descentralizat, slab structurat
  - foloseste similaritatile intre identificatorii fisierelor si nodurilor pentru a estima locatia fisierului
    - Ocoleste inundarea sau alegerea aleatoare a vecinilor pentru transmiterea mesajelor de interogare
    - Foloseste un mod de propagare similar cu backtracking-ul pentru a cauta un fisier.

- Alte caracteristici importante
  - Focus pe securitate – integritatea fisierelor
  - Anonimitatea sursei care publica continut
    - Nu se poate identifica sursa care publica documente fara consimtamantul autorilor (copyright)
  - Replicarea datelor pentru a creste disponibilitatea si performanta

- Fisierele sunt identificate prin chei binare unice
  - cheia = hash pe un scurt text descriptiv
  - ex. text/□philosophy□/sun-□tzu/□art-□of-□war
  - Obs. autorii sugereaza modul de alegere a textului descriptiv (de ex. fisiere directoare)
- Fiecare nod pastreaza in memoria locala
  - fisiere pe care le poate trimite altui nod, la cerere
  - o tabela de dirijare dinamica
    - cu perechi (cheie, urmatorul nod)
    - dirijarea nu cere potrivirea exacta a cheii de cautare cu o cheie identica din tabela, fiind suficienta gasirea unei chei apropiate de cheia de cautare

Cautarea unui fisier

- Utilizatorul trimite un mesaj de cerere propriului nod, cu o cheie de fisier si o valoare *hops-to-live*
- Cand un nod primeste o cerere
  - verifica memoria proprie si intoarce ca raspuns fisierul, daca este gasit, precum si o nota ca el a fost sursa datelor
  - daca fisierul nu este gasit in nodul curent, nodul gaseste cheia cea mai apropiata in tabela sa de rutare si transmite cererea nodului corespunzator
- Cand un nod primeste un raspuns
  - **memoreaza** fisierul in cache (memoria proprie)
  - **creaza o noua intrare** in tabela de dirijare, asociind sursa datelor cu cheia de cautare
  - **paseaza datele** nodului aflat pe calea inversa interogarii,
  - (pentru securitate) poate inlocui in **mesajul de raspuns** sursa datelor cu propria sa identitate

Alte actiuni de cautare

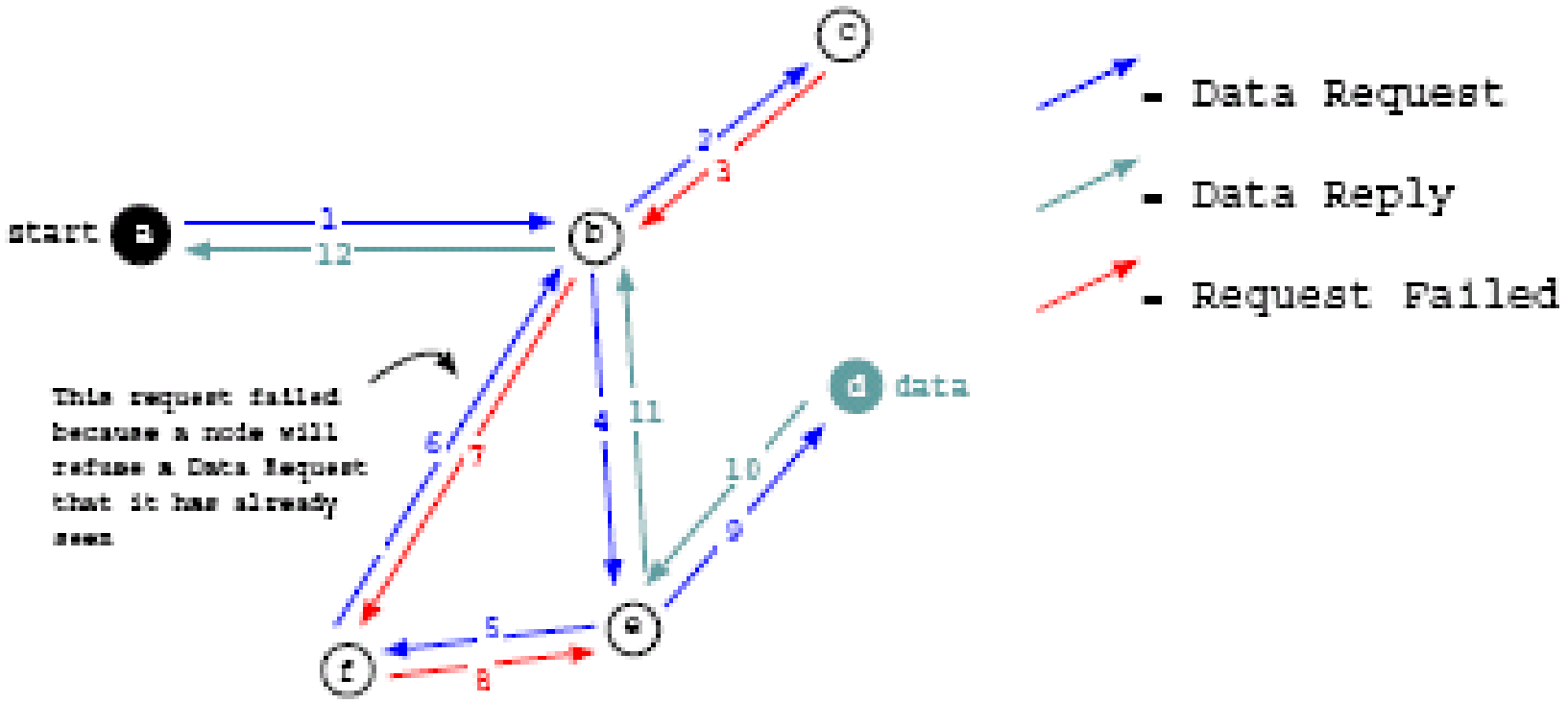
- Daca un nod nu poate pasa cererea catre nodul cu cheia cea mai apropiata (acesta este temporar indisponibil) el alege nodul avand urmatoarea cheie apropiata care poate prelua cererea
- daca nu exista un alt candidat pentru pasarea cererii, nodul trimite pe calea de retur un mesaj de eroare; vecinul care primeste mesajul incearca un alt nod
  - Procesul de cautare este similar cu procedura backtracking
- daca hops-to-live ajunge la zero se trimite un mesaj de eroare pe calea de retur

Solutii pentru evitare “gatuirilor”

- nodurile pot micsora valoarea hops-to-live pentru a reduce incarcarea retelei
- nodurile pot “uita” de cererile in asteptare pentru a elibera memoria in care acestea sunt pastrate.

Exemplu

- Nodul b trimite eroare la primirea mesajului 6, care este o copie a mesajului 1 primit anterior
- eroarea se propaga la nodul e (mesaje 7, 8) care incearca o cale alternativa, spre d (mesaj 9)
- d raspunde cu datele cerute



Efecte

- calitatea rutarii se imbunatateste in timp deoarece
  - nodurile se specializeaza in localizarea unor seturi de chei similare
  - nodurile se specializeaza in stocarea unor seturi de fisiere cu chei similare
- mecanismul de cerere face ca fisierele populare sa fie replicate (transparent)
- pe masura procesarii cererilor, creste conectivitatea retelei
  - nodurile creaza intrari noi in tabelele de dirijare, pentru noduri necunoscute anterior

Memorarea datelor

- Pentru a insera date, un utilizator **alege un text** descriptiv si **creaza cheia**
  - trimite un mesaj de **inserare** propriului nod
    - mesajul contine **cheia** si valoarea **hops-to-live** (care va determina numarul de noduri care vor memora datele)
- Mesajul de inserare este tratat ca o crere de cautare a unui fisier
- Raspunsul primit de nod este
  - fisierul cu cheia specificata, daca el exista deja – se propune o alta cheie
  - un rezultat all clear (=succes) daca se atinge limita hops-to-live fara a detecta coliziuni de cheie
    - se trimite fisierul de inserat care va fi propagat de-a lungul caii stabilite de cererea initiala si va fi memorat in fiecare nod de-a lungul caii

Efecte

- noile fisiere sunt plasate selectiv in nodurile care au deja fisiere cu chei similare
- inserarea de date este folosita de un **nou nod** pentru a informa celelalte noduri de intrartea lui in reteaua P2P
- o **incercare de atacare** a unui fisier existent (cerere de inserare a unui fisier corupt cu aceeasi cheie cu a unui fisier valid existent) va replica fisierul valid pe calea de intoarcere la nodul care a facut cererea
  - nu face decat sa difuzeze si mai mult fisierul existent

Mai multe despre replicare

- ***Replicare Pasiva***
  - Se produce natural cand nodurile cer continut - se copiaza continut de la sursa la destinatie
- ***Replicare bazata pe Cache***
  - se creaza copii cand continutul parcurge noduri intermediare intre sursa si destinatie
    - creste disponibilitatea
    - localizare mai usoara in cererile ulterioare
  - folosita in Freenet

Replici Pro-Active

– schema recomandata - replicarea obiectelor direct proportional cu radacina patrata a ratei de interogare

$$p \propto \sqrt{q_r},$$

- p este numarul de replici ale obiectului
- q_r este rata de interogare a obiectului

– doua strategii

- replicare a datelor de-a lungul caii de la sursa la destinatie
- replicile sunt plasate aleator

Managementul increderii in Sisteme Distribuite

Bazat pe un articol de

Huaizhi Li & Mukesh Singhal

Modele de incredere

- criptografie cu chei publice
  - sistem de certificate X.509
  - PGP
- modelul “resurrecting duckling”
  - relatie master-slave intre entitati:
    - una care trimite o cheie secreta si
    - alta care accepta cheia si ramane fidela masterului pana cand relatia este intrerupta (de master, time-out sau alt eveniment)
- modelul distribuit de incredere

Modelul distribuit de incredere

- Increderea este tranzitiva in anumite conditii
  - Ex. daca A are incredere in B si B are incredere in C, nu rezulta neaparat ca A are incredere in C
- A are incredere in C daca
  - B recomanda explicit lui A increderea sa in C
  - A are incredere in B ca sfatuitor (reccomender)
  - A poate aprecia recomandarea lui B si poate decide cat de mult va avea incredere in C
- Modelul defineste doua tipuri de relatii de incredere:
  - *incredere directa*
  - *incredere recomandata*

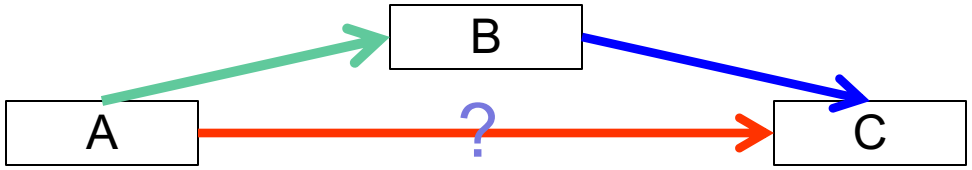


Table 1. Direct trust value.

Value	Meaning	Explanation
–1	Distrust	Completely untrustworthy
0	Ignorance	Can't decide
1	Minimal	Lowest trust
2	Average	Mean trustworthiness
3	Good	Trusted by major population
4	Complete	Fully trustworthy

Valorile folosite in continuare pentru calculul increderii

Table 2. Recommender trust value.

Value	Meaning	Explanation
–1	Distrust	Completely untrustworthy
0	Ignorance	Can't decide
1	Minimal	The entity itself judges the reliability of recommender's recommendation.
2	Average	
3	Good	
4	Complete	

Protocol simplu de recomandare

- A are nevoie de serviciul lui D dar nu stie nimic despre D
- A cere o recomandare de la B
- B paseaza cererea lui C
- C trimite un raspuns cu valoarea increderii lui in D
- Calea AxBxCxD este *calea de recomandare*
- Valoarea de incredere (trust value) a tintei *T*, *tv_T este*

$$tv\_T = [rtv(1)/4]*[rtv(2)/4]*...*[rtv(i)/4]*...*[rtv(n)/4]*tv(T)$$

  - *rtv*(*i*) este valoarea increderii in sfatuitorul *i* pe calea de recomandare (valoarea este in gama de la 1 la 4)
  - *tv*(*T*) este valoarea increderii in tinta *T* returnata de ultimul sfatuitor
- Daca exista mai multe cai de recomandare
  - valoarea finala a increderii in tinta este media valorilor calculate pe diferitele cai

Slabiciunile modelului

- Nu considera recomandarile false
- Presupune ca un sfatuitor cu o valoare ridicata a increderii de recomandare face mereu recomandari sigure, ceeace nu este adevarat intotdeauna
- Nu are mecanisme de monitorizare si re-evaluare a increderii (care este dinamica)

Incredere bazata pe recomandari

Se coteaza o interactiune satisfacatoare cu 1 si o plangere cu 0.

Criteriul de incredere este **T(u,t)** – raportul dintre satisfactia cumulativa ponderata pe care **u** o primeste de la peers si numarul total de interactiuni pe care u le are intr-un sistem P2P

$$T(u,t) = \frac{\sum_{v \in P,\, v \neq u} S(u,v,t) \bullet Cr(v,t)}{\sum_{v \in P,\, v \neq u} I(u,v,t)}$$

T(u,t) - valoarea increderii in *u* evaluata de alti peers, pana la tranzactia *t*

***P** este* un set de peers in sistemul P2P

u si *v* sunt peers in sistem, *u, v Є P*;

S(u,v,t) - gradul de satisfactie pe care *v* il are fata de *u* pana la tranzactia *t*

Cr(v,t) - factorul de ponderare a satisfactiei lui *v* = *increderea in v ca sfatuitor*

I(u,v,t) - numar de interactiuni pe care *u* le are cu *v* pana la tranzactia *t*

Reformulare T(u,t)

$S(u,v,t)$ ● $Cr(v,t)$ este aproximat prin $I(u,v,t)$ – $C(u,v,t)$ ● $T(v,t)$.

$C(u,v,t)$ denota plangerile lui *v* impotriva lui *u*.

$C(u,v,t)$ ● $T(v,t)$ indica plangerile lui *v* contra lui *u* filtrate cu valoarea increderii in v evaluata de alti peers, pana la tranzactia t

Definitia lui $T(u,t)$ devine

$$T(u,t) = 1 - \frac{\sum_{v \in P,\, v \neq u} C(u,v,t) \bullet T(v,t)}{\sum_{v \in P,\, v \neq u} I(u,v,t)}$$

$T(u,t)$ este in domeniul (0, 1).

Valori mari $T(u,t)$ denota ca *u* este mai de incredere

Criteriul de decizie al increderii introduce doua praguri C1 si C2:

- *C*1 defineste numarul minim de interactiuni cerute
- *C2* specifica valoarea minima a increderii

Daca $I(u,t) > C1$ si $T(u,t) > C2$, atunci *u* este de incredere.

Un anumit numar minim de interactiuni sunt necesare pentru a imbunatati acuratetea calculului

Formula a doua considera evaluari pozitive si negative

→ produce rezultate mai precise

Neajunsuri

- Criteriul de decizie din a doua formula cere un numar minim de interactiuni
  - dezavantajeaza peers nou sositi
- Factorul de ponderare Cr(v,t) din prima ecuatie este valoarea de incredere intr-un peer ca sfatuitor
  - sistemul presupune ca un peer cu o valoare de incredere mai mare da un feedback mai sigur, ceeace nu este intotdeauna adevarat
- Modelul nu face diferenta intre feedback-uri mai vechi si cele noi
  - Un feedback mai recent este mai apropiat de comportamentul curent al unui peer

Referinte

A Survey of Peer-to-Peer Content Distribution Technologies
Stephanos Androutsellis-Theotokis & Diomidis Spinellis

Chord: A Scalable Peertopeer Lookup Service for Internet Applications
Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan.

Tapestry: A Resilient Global-scale Overlay for Service Deployment
Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, Member, IEEE, and John D. Kubiatowicz, Member, IEEE

Freenet: A Distributed Anonymous Information Storage and Retrieval System
Vijay Gopalakrishnan, Bujor Silaghi, Bobby Bhattacharjee, and Pete Keleher
Department of Computer Science, University of Maryland, College Park

Trust Management in Distributed Systems
Huaizhi Li & Mukesh Singhal