Elosztott rendszerek: Alapelvek és paradigmák
Distributed Systems: Principles and Paradigms

Maarten van Steen[1] Kitlei Róbert [2]

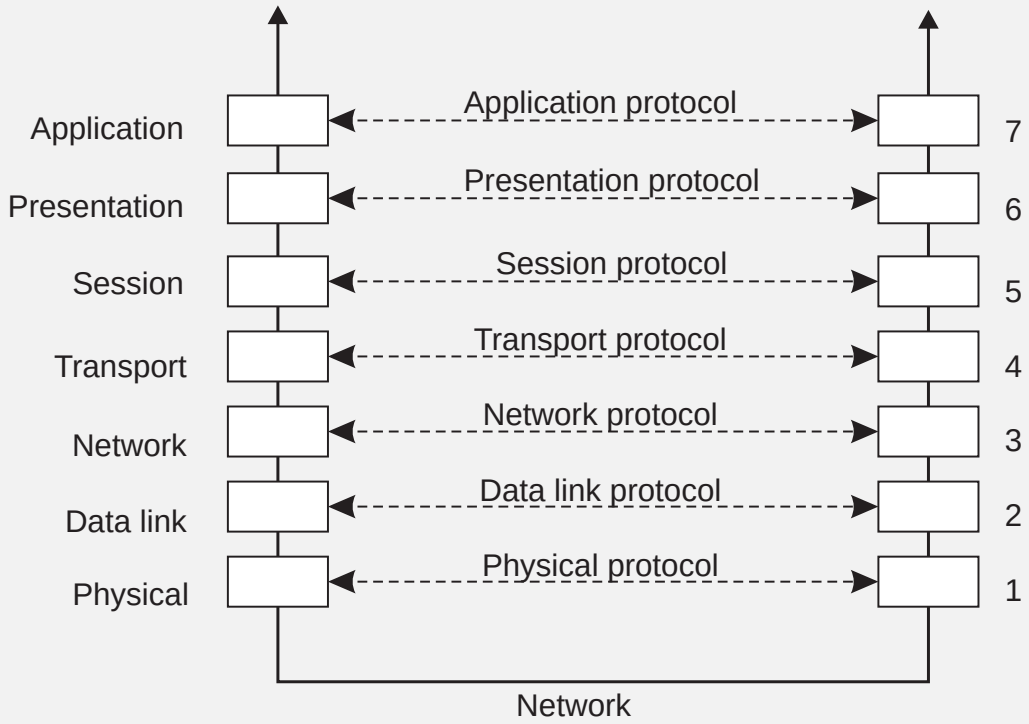
[1]VU Amsterdam, Dept. Computer Science
[2]ELTE Informatikai Kar

4. rész: Kommunikáció

2015. május 24.

Tartalomjegyzék

Az ISO/OSI hálózatkezelési modell



Hátrányok

- Csak az üzenetküldésre koncentrál
- Az (5) és (6) rétegek legtöbbször nem jelennek meg ilyen tisztán
- Az elérési átlátszóság nem teljesül ebben a modellben

Az alsó rétegek

A rétegek feladatai

- Fizikai réteg: a bitek átvitelének fizikai részleteit írja le
- Adatkapcsolati réteg: az üzeneteket keretekre tagolja, célja a hibajavítás és a hálózat terhelésének korlátozása
- Hálózati réteg: a hálózat távoli gépei között közvetít csomagokat útválasztás (routing) segítségével

Szállítási réteg

Absztrakciós alap

A legtöbb elosztott rendszer a szállítási réteg szolgáltatásaira épít.

A legfőbb protokollok

- TCP: kapcsolatalapú, megbízható, sorrendhelyes átvitel
- UDP: nem (teljesen) megbízható, általában kis üzenetek (datagram) átvitele

Csoportcímzés

IP-alapú többcímű üzenetküldés (multicasting) sokszor elérhető, de legfeljebb a lokális hálózaton belül használatos.

Köztesréteg

Szolgáltatásai

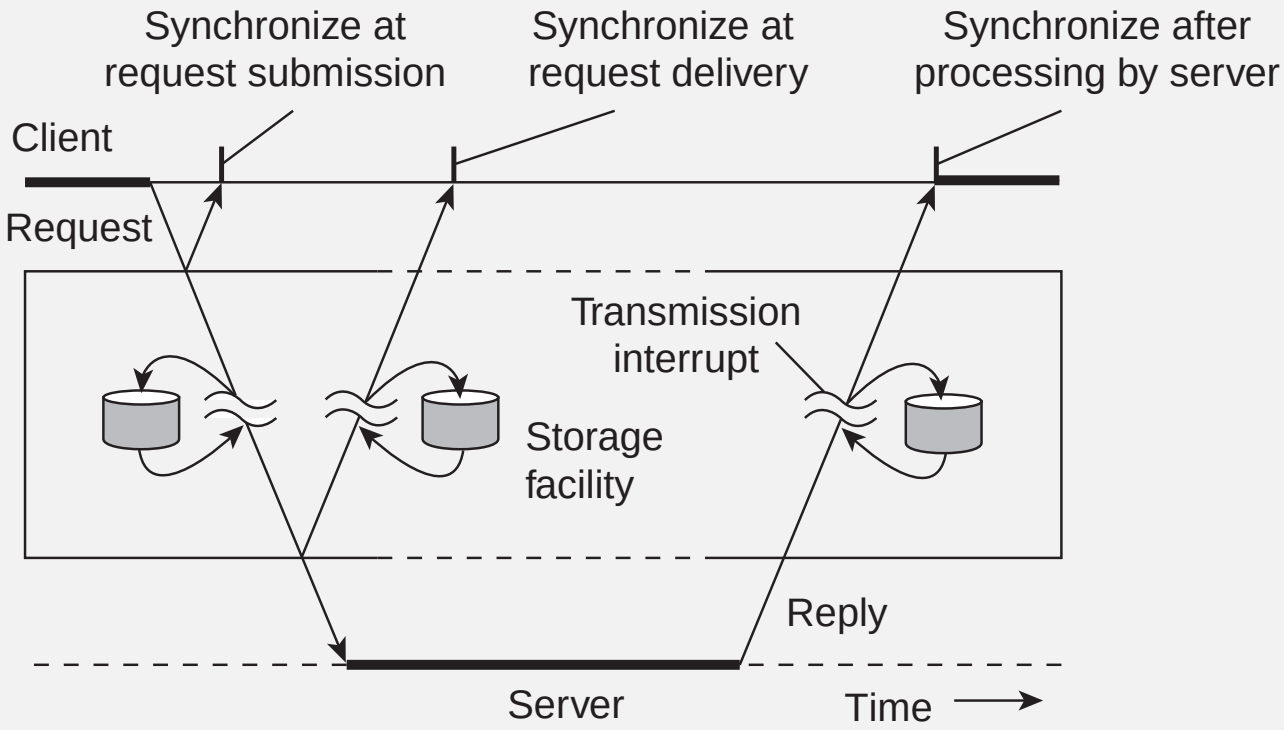
A köztesrétegbe (middleware) olyan szolgáltatásokat és protokollokat szokás sorolni, amelyek sokfajta alkalmazáshoz lehetnek hasznosak.

- Sokfajta kommunikációs protokoll
- Sorosítás ((de)serialization, (un)marshalling), adatok reprezentációjának átalakítása (elküldésre vagy elmentésre)
- Elnevezési protokollok az erőforrások megosztásának megkönnyítésére
- Biztonsági protokollok a kommunikáció biztonságossá tételére
- Skálázási mechanizmusok adatok replikációjára és gyorsítótárazására

Alkalmazási réteg

Az alkalmazások készítőinek csak az alkalmazás-specifikus protokollokat kell önmaguknak implementálniuk.

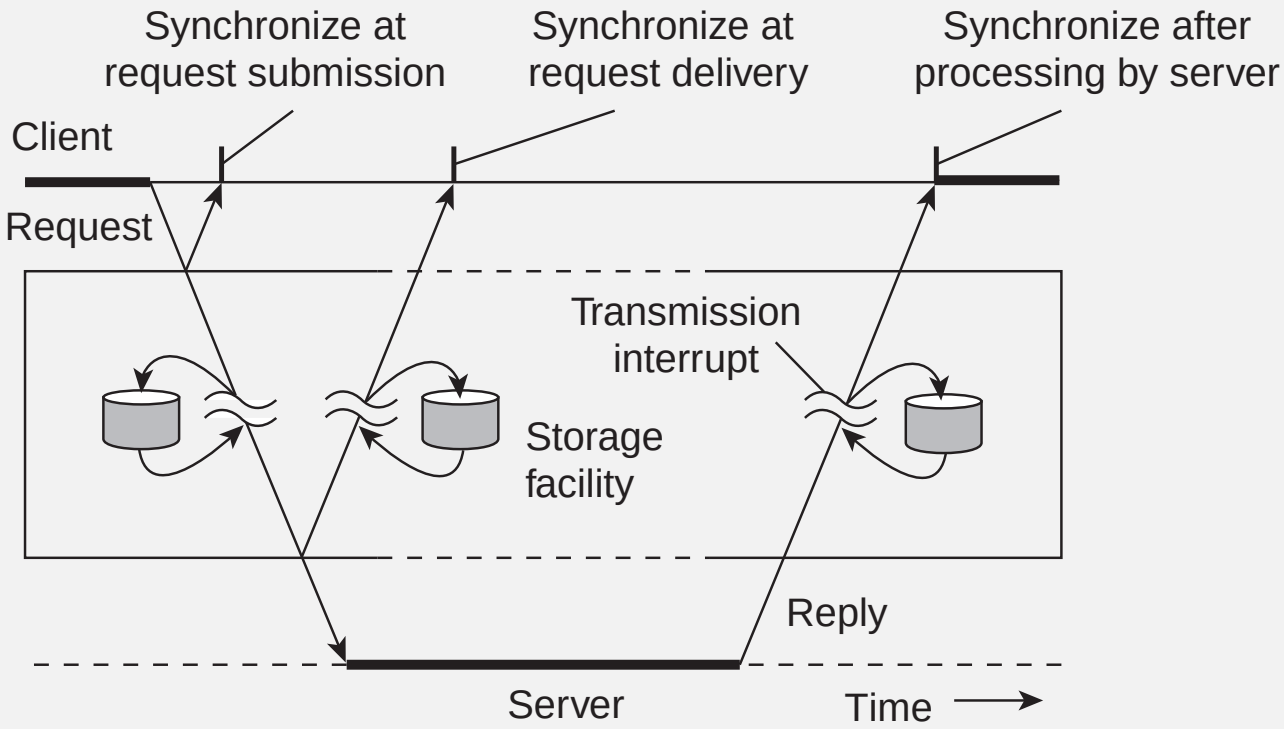
A kommunikáció fajtái



A kommunikáció lehet...

- időleges (transient) vagy megtartó (persistent)
- aszinkron vagy szinkron

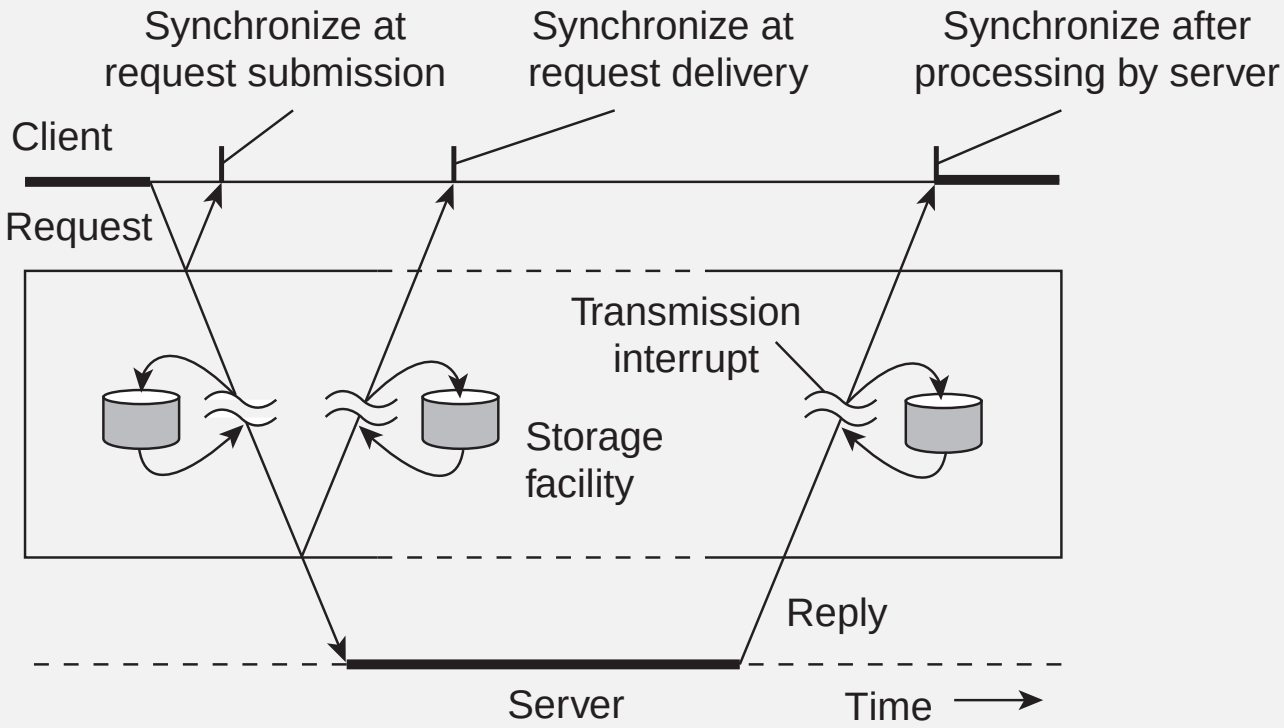
A kommunikáció fajtái



Időleges vs megtartó

- Megtartó kommunikáció: A kommunikációs rendszer hajlandó huzamosan tárolni az üzenetet.
- Időleges kommunikáció: A kommunikációs rendszer elveti az üzenetet, ha az nem kézbesíthető.

A kommunikáció fajtái



A szinkronizáció lehetséges helyei

- Az üzenet elindításakor
- Az üzenet beérkezésekor
- A kérés feldolgozása után

Kliens–szerver modell

Általános jellemzők

A kliens–szerver modell jellemzően időleges, szinkron kommunikációt használ.

- A kliensnek és a szervernek egyidőben kell aktívnak lennie.
- A kliens blokkolódik, amíg a válasz meg nem érkezik.
- A szerver csak a kliensek fogadásával foglalkozik, és a kérések kiszolgálásával.

A szinkron kommunikáció hátrányai

- A kliens nem tud tovább dolgozni, amíg a válasz meg nem érkezik
- A hibákat rögtön kezelni kell, különben feltartjuk a klienst
- Bizonyos feladatokhoz (pl. levelezés) nem jól illeszkedik

Üzenetküldés

Üzenetorientált köztesréteg (message-oriented middleware, MOM)

Megtartó, aszinkron kommunikációs architektúra.

- Segítségével a folyamatok üzeneteket küldhetnek egymásnak
- A küldő félnek nem kell válaszra várakoznia, foglalkozhat mással
- A köztesréteg gyakran hibatűrést biztosít

RPC: alapok

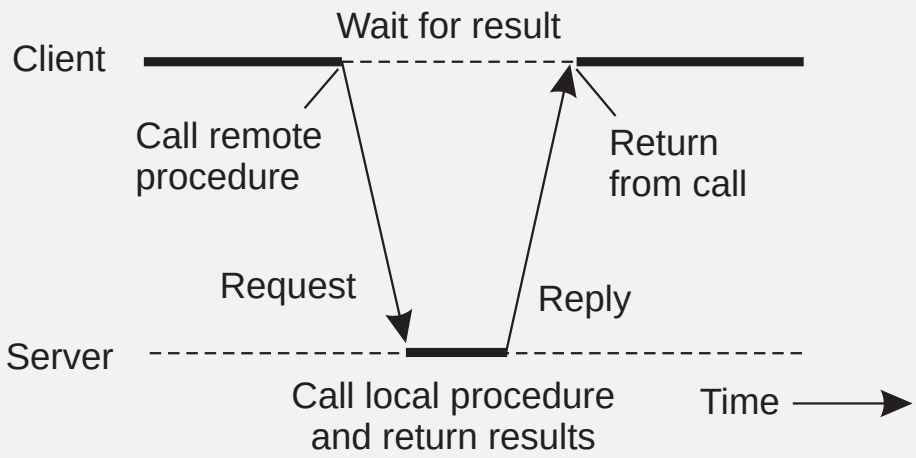
Az RPC alapötlete

- Az alprogramok használata természetes a fejlesztés során
- Az alprogramok a jó esetben egymástól függetlenül működnek („fekete doboz”),
- ... így akár egy távoli gépen is végrehajthatóak

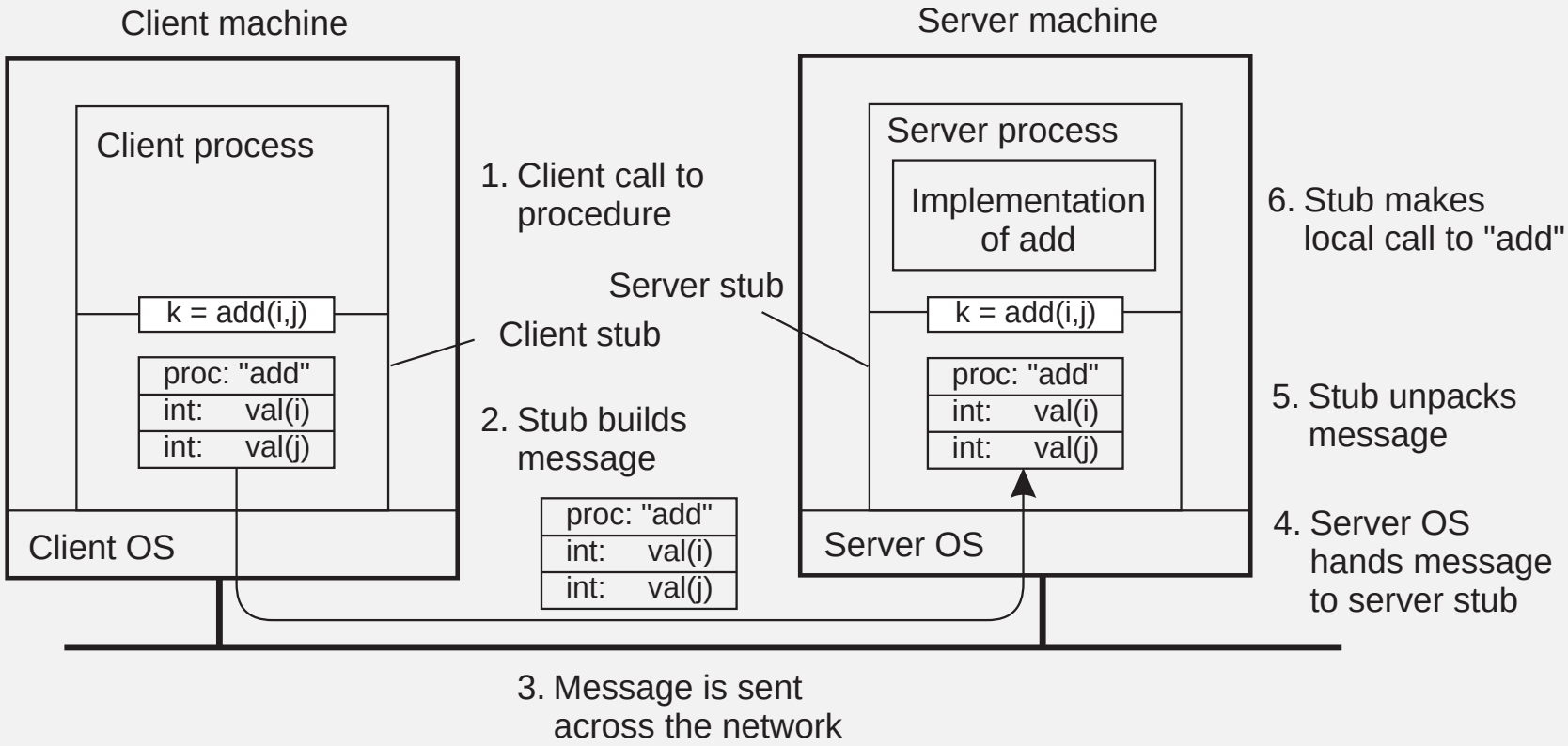
Távoli eljáráshívás (remote procedure call, RPC)

A távoli gépen futtatandó eljárás[a] eléréséhez hálózati kommunikációra van szükség, ezt eljáráshívási mechanizmus fedi el.

[a] tekintsük az alprogram szinonímájának



RPC: a hívás lépései



1. A kliensfolyamat lokálisan meghívja a klienscsonkot.
2. Az becsomagolja az eljárás azonosítóját és paramétereit, meghívja az OS-t.
3. Az átküldi az üzenetet a távoli OS-nek.
4. Az átadja az üzenetet a szervercsonknak.
5. Az kicsomagolja a paramétereket, átadja a szervernek.
6. A szerver lokálisan meghívja az eljárást, megkapja a visszatérési értéket.
7. Ennek visszaküldése a klienshez hasonlóan zajlik, fordított irányban.

RPC: paraméterátadás

A paraméterek sorosítása

A második lépésben a klienscsonk elkészíti az üzenetet, ami az egyszerű bemásolásnál összetettebb lehet.

- A kliens- és a szervergépen eltérhet az adatábrázolás (eltérő bájtsorrend)
- A sorosítás során bájtsorozat készül az értékből
- Rögzíteni kell a paraméterek kódolását:
  - A primitív típusok reprezentációját (egész, tört, karakteres)
  - Az összetett típusok reprezentációját (tömbök, egyéb adatszerkezetek)
- A két csonknak fordítania kell a közös formátumról a gépeik formátumára

RPC: paraméterátadás

RPC paraméterátadás szemantikája

- Érték–eredmény szerinti paraméterátadási szemantika: pl. figyelembe kell venni, hogy ha (a kliensoldalon ugyanoda mutató) hivatkozásokat adunk át, azokról ez a hívott eljárásban nem látszik.
- Minden feldolgozandó adat paraméterként kerül az eljáráshoz; nincsen globális hivatkozás.

Átlátszóság

Nem érthető el teljes mértékű elérési átlátszóság.

Távoli hivatkozás

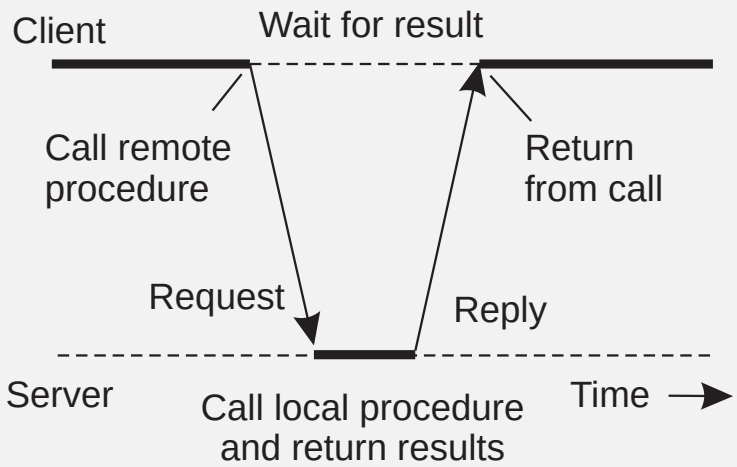
Távoli hivatkozás bevezetésével növelhető az elérési átlátszóságot:

- A távoli adat egységesen érhető el
- A távoli hivatkozásokat át lehet paraméterként adni

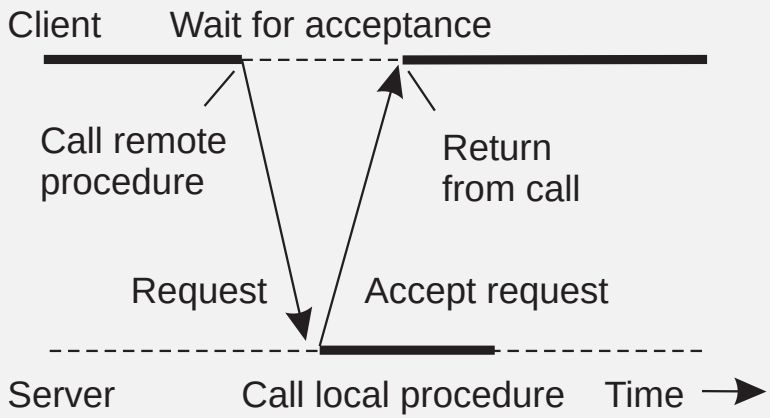
Aszinkron RPC

Az RPC „javítása”

A szerver nyugtázza az üzenet megérkezését. Választ nem vár.



(a)

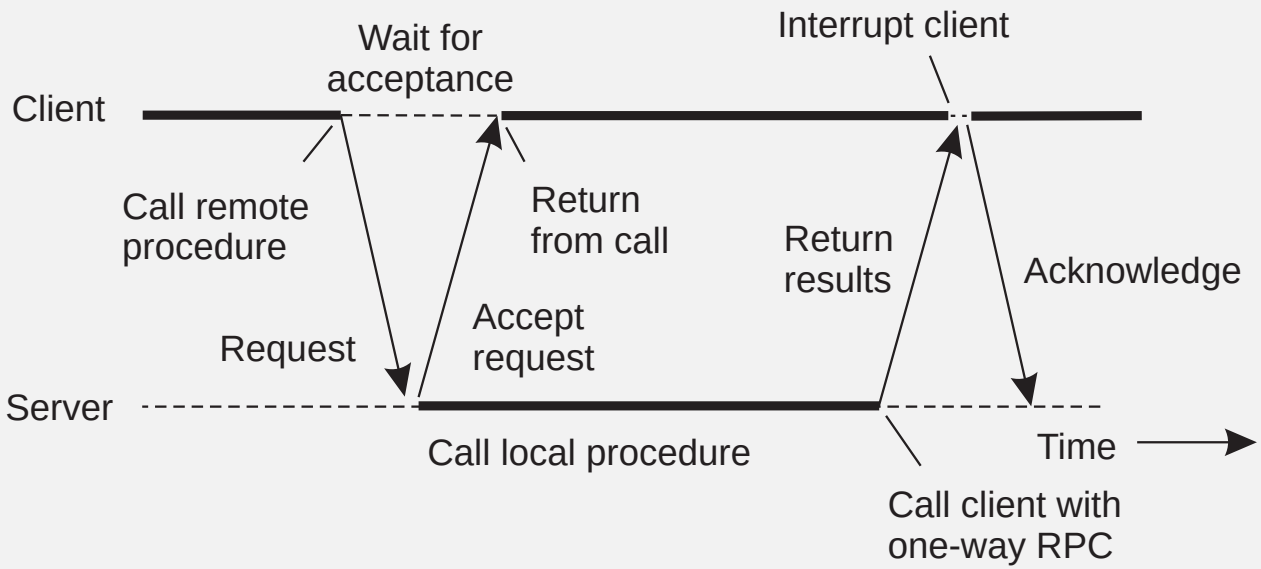


(b)

Késleltetett szinkronizált RPC

Késleltetett szinkronizált RPC

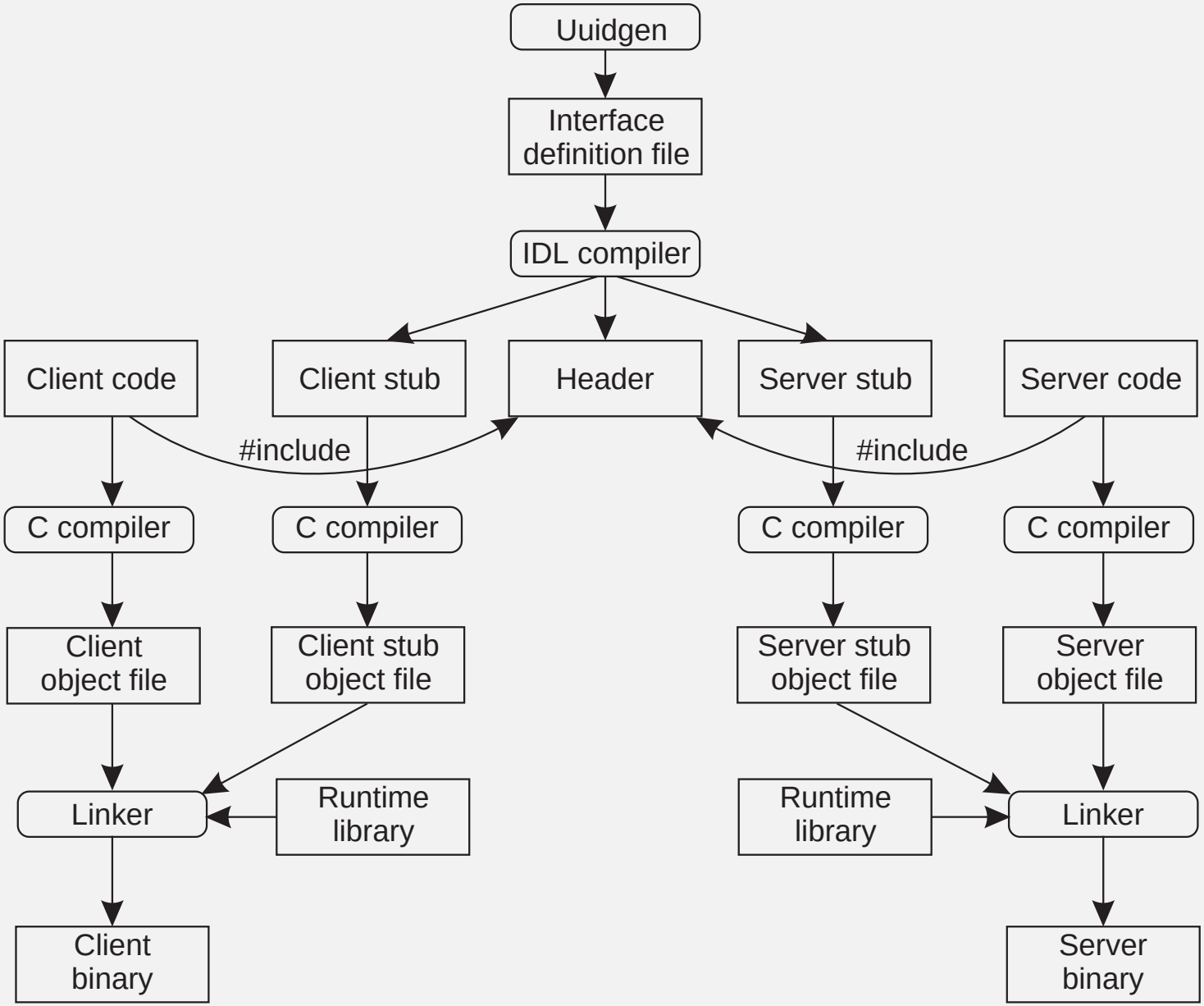
Ez két aszinkron RPC, egymással összehangolva.



További lehetőség

A kliens elküldheti a kérését, majd időnként lekérdezheti a szervertől, kész-e már a válasz.

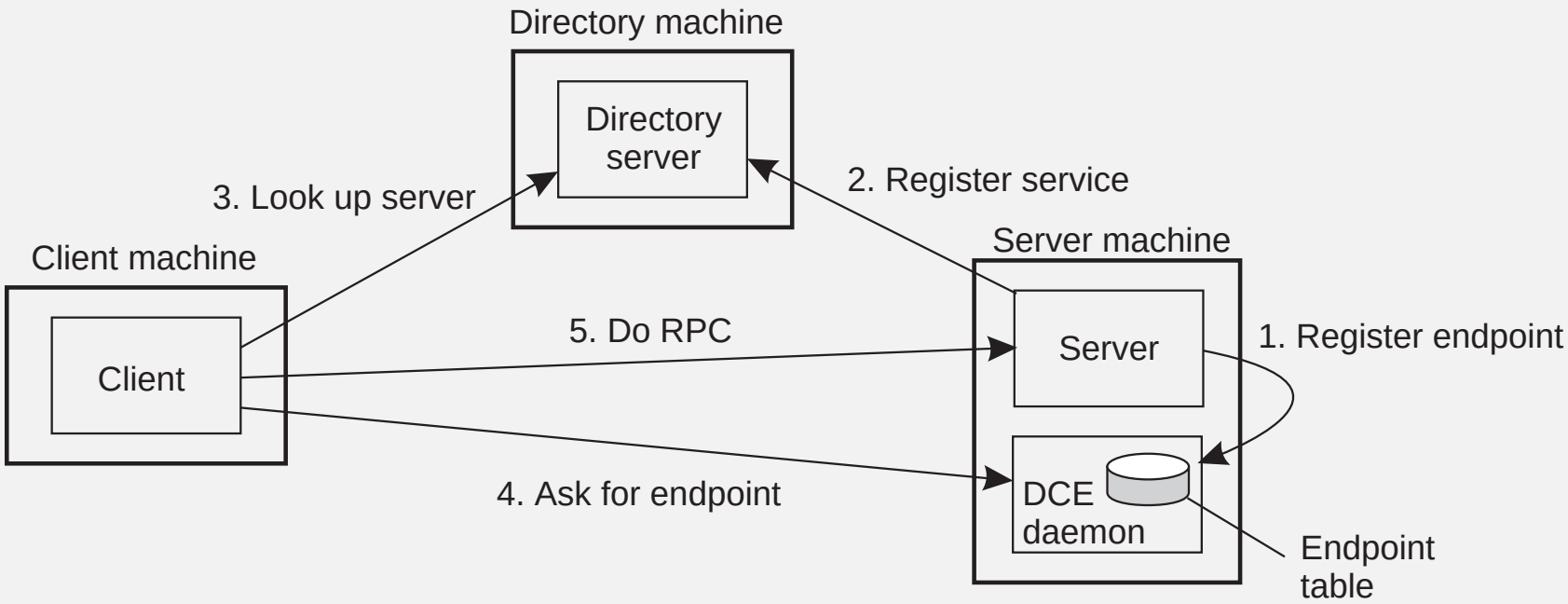
RPC: a használt fájlok



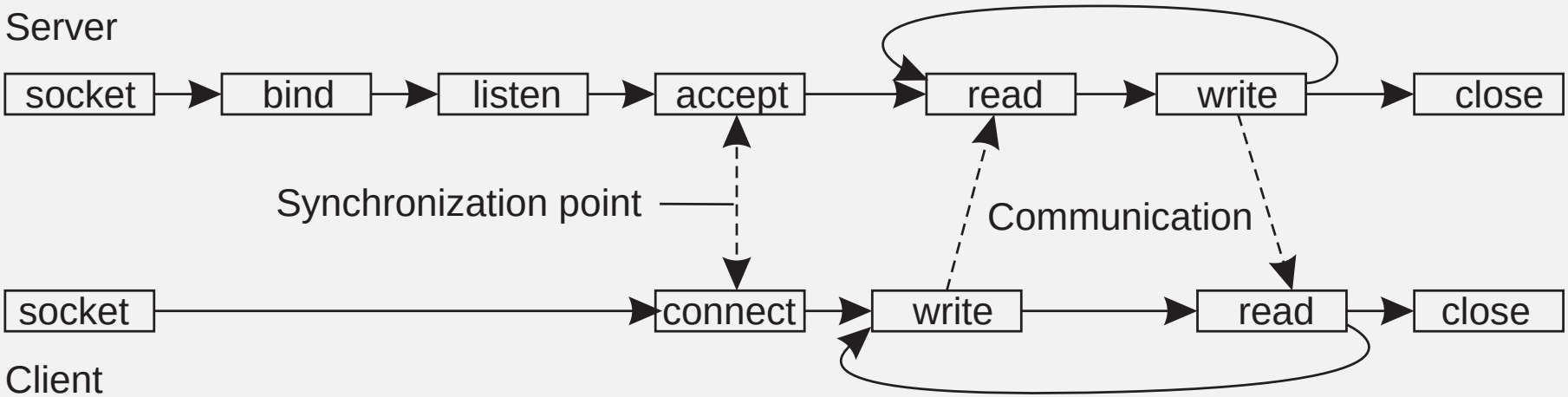
RPC: a kliens csatlakozása a szerverhez

A kliens

1. A szolgáltatások katalógusba jegyzik be (globálisan és lokálisan is), melyik gépen érhetőek el. (1-2)
2. A kliens kikeresi a szolgáltatást a katalógusból. (3)
3. A kliens végpontot igényel a démontól a kommunikációhoz. (4)



Időleges kommunikáció: socket



Socket: példa Python nyelven

Szerver

```
import socket
HOST = ''
PORT = SERVERPORT
srvsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srvsock.bind((HOST, PORT))
srvsock.listen(N)                    # legfeljebb N kliens várakozhat
clsock, addr = srvsock.accept() # lokális végpont + távoli végpont címe
while True: # potenciálisan végtelen ciklus
  data = clsock.recv(1024)
  if not data: break
  clsock.send(data)
clsock.close()
```

Kliens

```
import socket
HOST = 'distsys.cs.vu.nl'
PORT = SERVERPORT
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send('Hello, world')
data = s.recv(1024)
s.close()
```

Üzenetorientált köztesréteg

Működési elv

A köztesréteg várakozási sorokat (queue) tart fenn a rendszer gépein.
A kliensek az alábbi műveleteket használhatják a várakozási sorokra.

PUT	Üzenetet tesz egy várakozási sor végére
GET	Blokkol, amíg a sor üres, majd leveszi az első üzenetet
POLL	Nem-blokkoló módon lekérdezi, van-e üzenet, ha igen, leveszi az elsőt
NOTIFY	Kezelőrutint telepít a várakozási sorhoz, amely minden beérkező üzenetre meghívódik

Üzenetközvetítő

Üzenetsorkezelő rendszer homogenitása

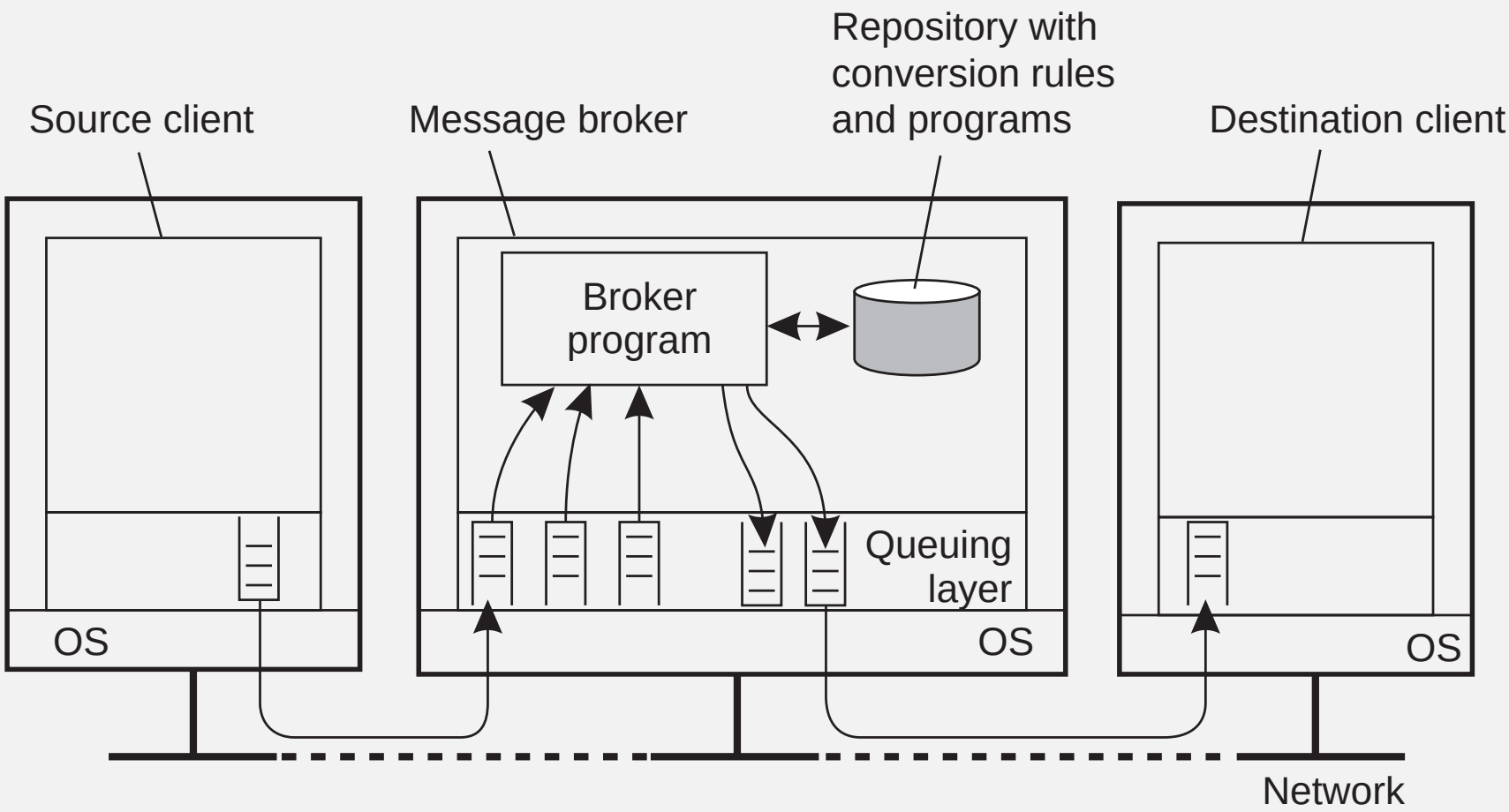
Az üzenetsorkezelő rendszerek feltételezik, hogy a rendszer minden eleme közös protokollt használ, azaz az üzenetek szerkezete és adatábrázolása megegyező.

Üzenetközvetítő

üzenetközvetítő (message broker): Olyan központi komponens, amely heterogén rendszerben gondoskodik a megfelelő konverziókról.

- Átalakítja az üzeneteket a fogadó formátumára.
- Szerepe szerint igen gyakran átjáró (application-level gateway, proxy) is, azaz a közvetítés mellet további (pl. biztonsági) funkciókat is nyújt
- Az üzenetek tartalmát is megvizsgálhatják az útválasztáshoz (subject based vagy object based routing) $\Rightarrow$ Enterprise Application Integration

Üzenetközvetítő

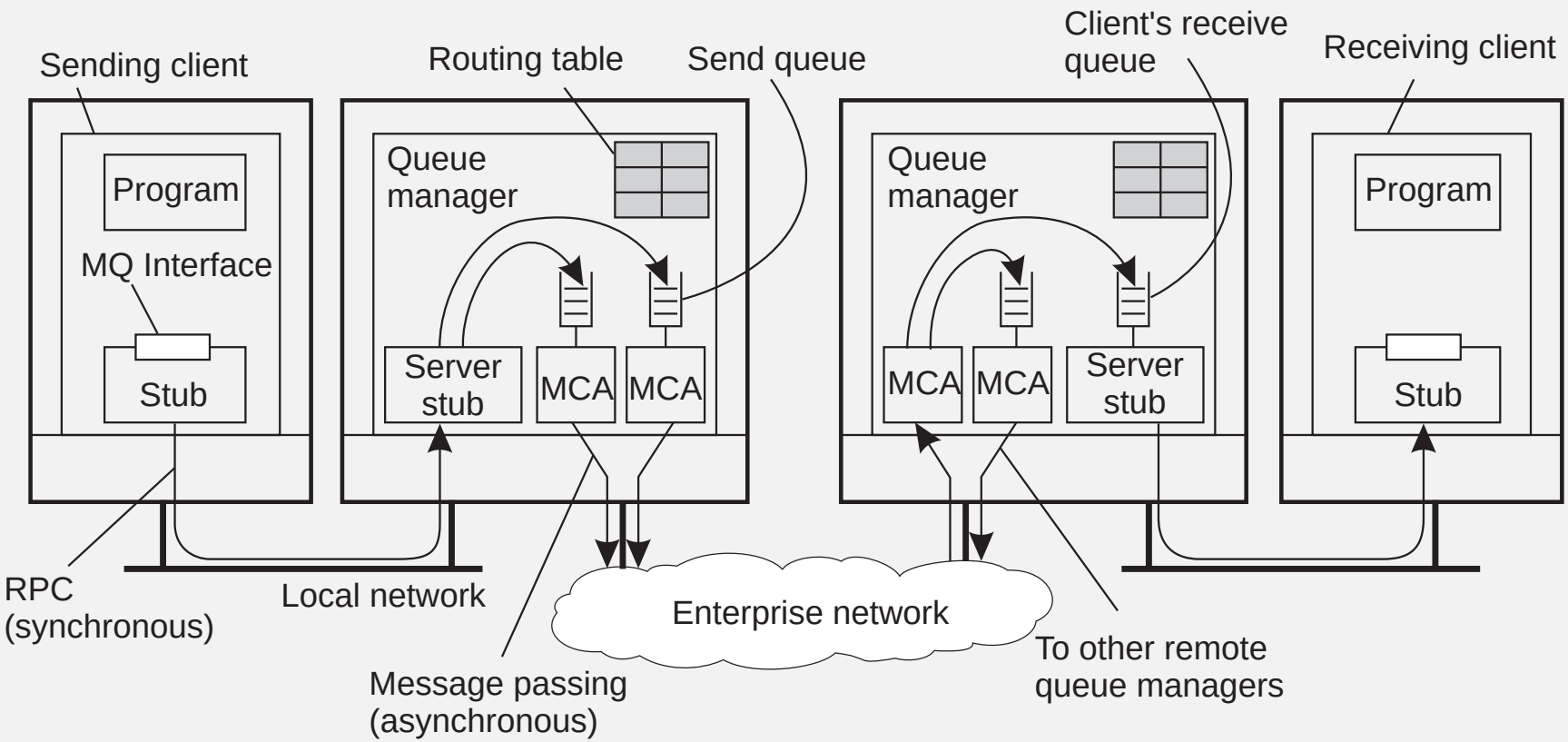


Példa: WebSphere MQ (IBM)

Működési elv

- Az üzenetkezelők neve itt sorkezelő (queue manager); adott alkalmazásoknak címzett üzeneteket fogadnak
  - Az üzenetkezelőt össze lehet szerkeszteni a kliensprogrammal
  - Az üzenetkezelő RPC-n keresztül is elérhető
- Az útválasztótáblák (routing table) megadják, melyik kimenő csatornán kell továbbítani az üzenetet
- A csatornákat üzenetcsatorna-ügynökök (message channel agent, MCA) kezelik
  - Kiépítik a hálózati kapcsolatokat (pl. TCP/IP)
  - Ki- és becsomagolják az üzeneteket, és fogadják/küldik a csomagokat a hálózatról

Példa: WebSphere MQ (IBM)

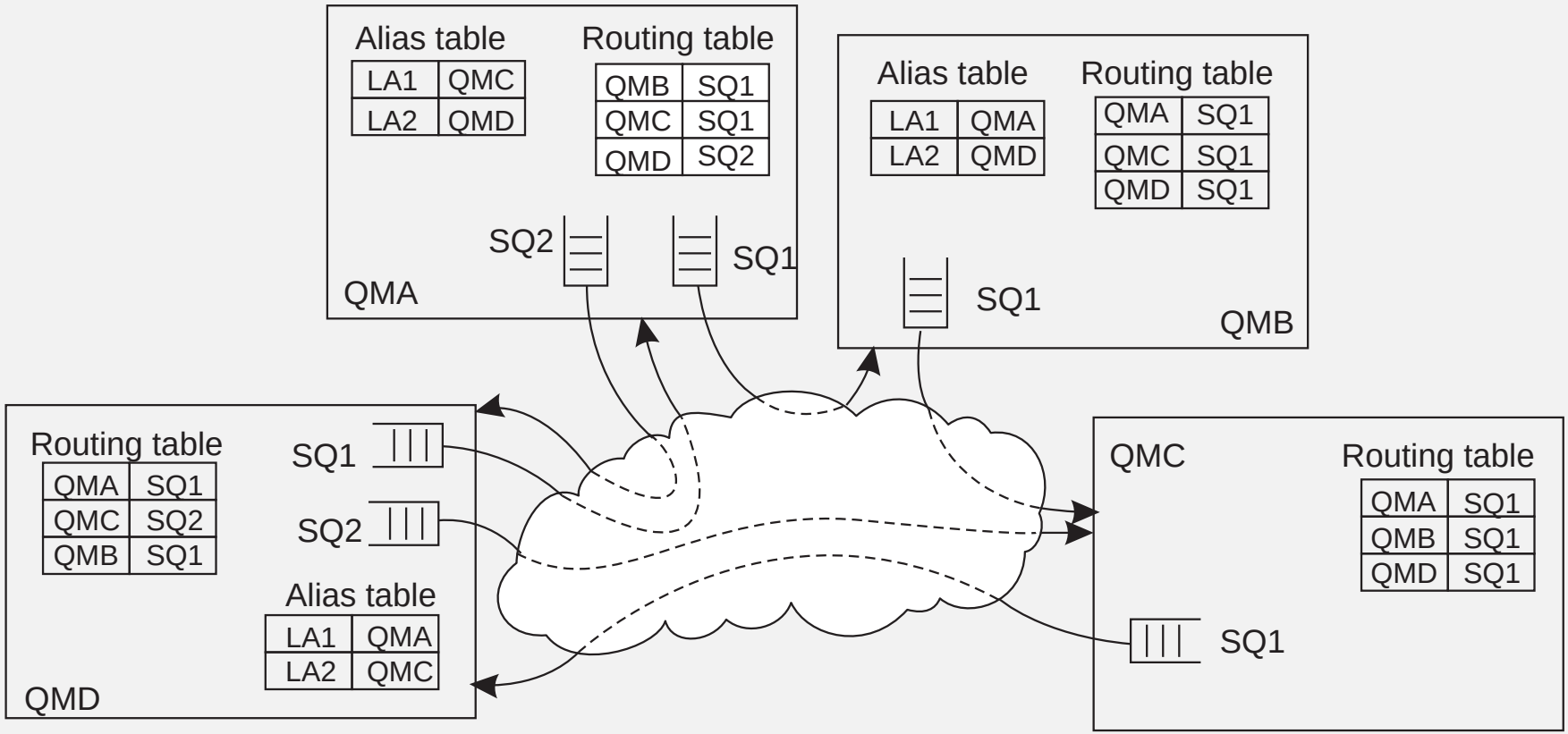


- A csatornák egyirányúak
- A sorkezelőkhöz beérkező üzenetek automatikusan továbbítódnak a megfelelő lokális MCA-hoz
- Az útválasztás paramétereit kézzel adják meg

Példa: WebSphere MQ (IBM)

Álnevek

Távoli üzenetkezelőhöz álnevet (alias) is lehet rendelni, ez csak a lokális üzenetkezelőn belül érvényes.



Folyamatos média

Az idő szerepe

Az eddig tárgyalt kommunikációfajtákban közös, hogy diszkrét ábrázolásúak: az adategységek közötti időbeli kapcsolat nem befolyásolja azok jelentését.

Folyamatos ábrázolású média

A fentiekkel szemben itt a továbbított adatok időfüggőek.
Néhány jellemző példa:

- audio
- videó
- animációk
- szenzorok adatai (hőmérséklet, nyomás stb.)

Folyamatos média

Adatátviteli módok

Többfajta megkötést tehetünk a kommunikáció időbeliségével kapcsolatban.

- aszinkron: nem ad megkötést arra, hogy **mikor** kell átvinni az adatot
- szinkron: az egyes adatcsomagoknak megadott időtartam alatt célba kell érniük
- izokron vagy izoszinkron[a]: felső és alsó korlátot is ad a csomagok átvitelére; a remegés (jitter) így korlátozott mértékű

[a] α-=(fosztóképző), ἴσος=egyenlő, σύν=együtt, χρόνος=idő

Folyam

Adatfolyam

adatfolyam: Izokron adatátvitelt támogató kommunikációs forma.

Fontosabb jellemzők

- Egyirányú
- Legtöbbször egy forrástól (source) folyik egy vagy több nyelő (sink) felé
- A forrás és/vagy a nyelő gyakran közvetlenül kapcsolódik hardverelemekhez (pl. kamera, képernyő)
- egyszerű folyam: egyfajta adatot továbbít, pl. egy audiocsatornát vagy csak videót
- összetett folyam: többfajta adatot továbbít, pl. sztereo audiót vagy hangot+videót

Folyam: QoS

Szolgáltatás minősége

A folyamokkal kapcsolatban sokfajta követelmény írható elő, ezeket összefoglaló néven a szolgáltatás minőségének (Quality of Service, QoS) nevezzük. Ilyen jellemzők a következők:

- A folyam átvitelének „sebessége”: bit rate.
- A folyam megindításának legnagyobb megengedett késleltetése.
- A folyam adategységeinek megadott idő alatt el kell jutniuk a forrástól a nyelőig (end-to-end delay), illetve számíthat az oda-vissza út is (round trip delay).
- Az adategységek beérkezési időközeinek egyenetlensége: remegés (jitter).

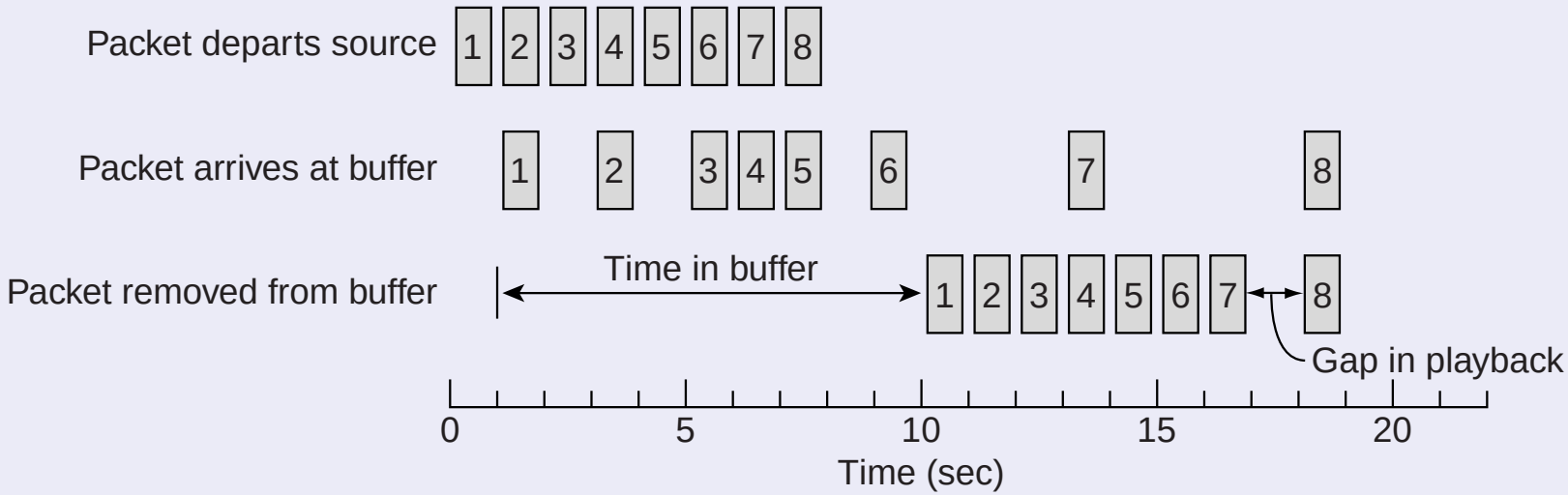
Folyam: QoS biztosítása

Differenciált szolgáltatási architektúra

Több hálózati eszköz érhető el, amelyekkel a QoS biztosítható. Egy lehetőség, ha a hálózat routerei kategorizálják az áthaladó forgalmat a beérkező adatcsomagok tartalma szerint, és egyes csomagfajtákat elsőbbséggel továbbítanak (differentiated services).

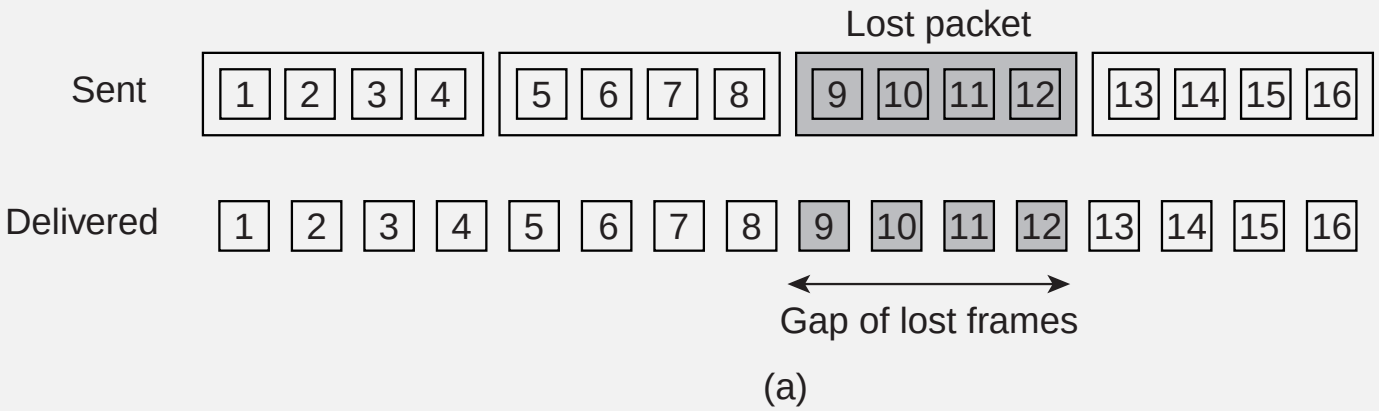
A remegés csökkentése

A routerek pufferelhetik az adatokat a remegés csökkentésére.

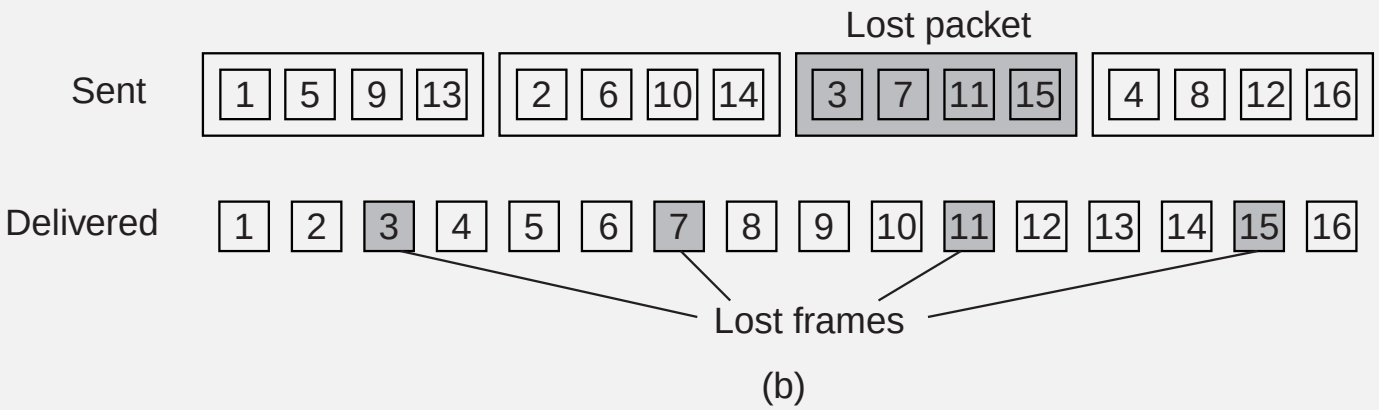


Folyam: QoS biztosítása

A csomagok elveszhetnek útközben. Ennek hatását mérsékelheti, ha a csomagon belül az adatelemek sorrendjét kissé módosítjuk; ennek ára, hogy a lejátszás lassabban indul meg.



(a)

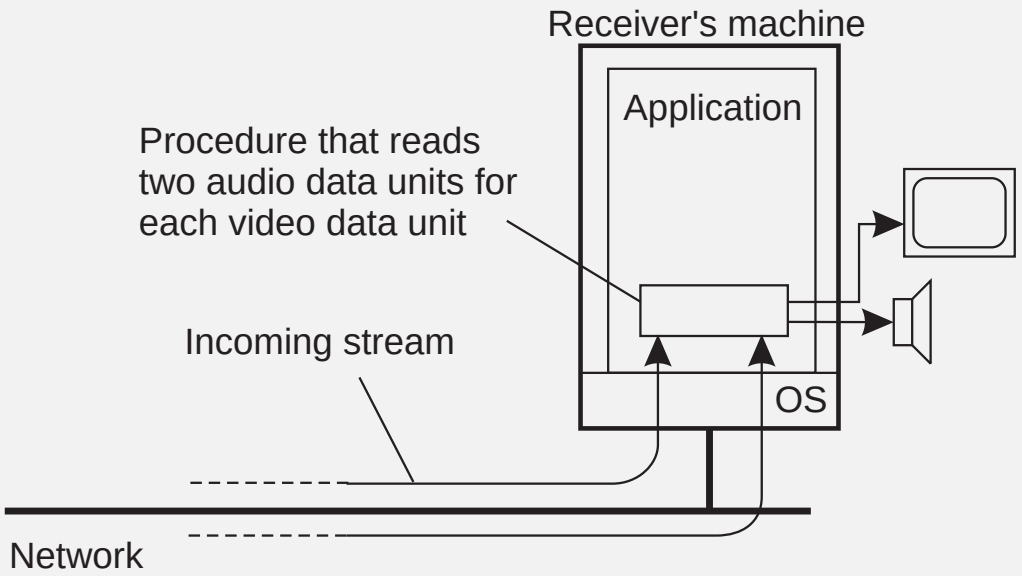


(b)

Összetett folyam szinkronizációja

Szinkronizáció a nyelőnél

Az összetett folyam alfolyamait szinkronizálni kell a nyelőnél, különben időben elcsúszhatnának egymáshoz képest.



Multiplexálás

Másik lehetőség: a forrás már eleve egyetlen folyamot készít (multiplexálás). Ezek garantáltan szinkronban vannak egymással, a nyelőnél csak szét kell őket bontani (demultiplexálás).

Alkalmazásszintű multicasting

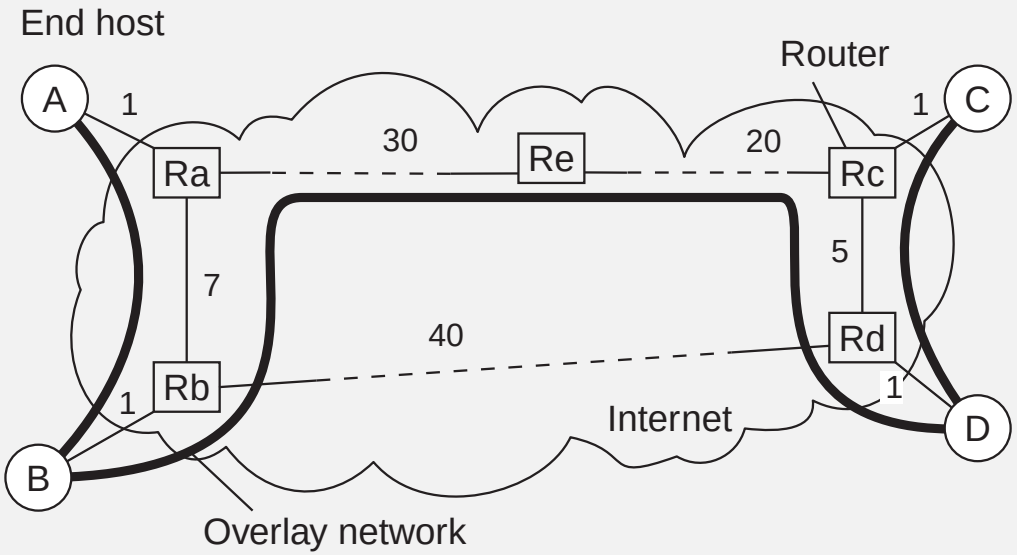
A hálózat minden csúcsának szeretnénk üzenetet tudjunk küldeni (multicast). Ehhez hierarchikus overlay hálózatba szervezzük őket.

Chord struktúrában tárolt fa készítése

- A multicast hálózatunkhoz generálunk egy azonosítót, így egyszerre több multicast hálózatunk is lehet egy rendszerben.
- Tegyük fel, hogy az azonosító egyértelműen kijelöl egy csúcsot a rendszerünkben[a]. Ez a csúcs lesz a fa gyökere.
- Terv: a küldendő üzeneteket mindenki elküldi a gyökérhez, majd onnan a fán lefele terjednek.
- Ha a P csúcs csatlakozni szeretne a multicast hálózathoz, csatlakozási kérést küld a gyökér felé. A P csúcstól a gyökérig egyértelmű az útvonal[b]; ennek minden csúcsát a fa részévé teszünk (ha még nem volt az). Így P elérhetővé válik a gyökértől.

[a] Ez az azonosító ún. rákövetkezője; a technikai részletek később jönnek.
[b] Részletek szintén később.

Alkalmazásszintű multicasting: költségek



- Kapcsolatok terhelése: Mivel overlay hálózatot alkalmazunk, előfordulhat, hogy egy üzenetküldés többször is igénybe veszi ugyanazt a fizikai kapcsolatot.
  Példa: az $A \rightarrow D$ üzenetküldés kétszer halad át az $Ra \rightarrow Rb$ élen.
- Stretch: Az overlayt követő és az alacsonyszintű üzenetküldés költségének hányadosa.
  Példa: $B \rightarrow C$ overlay költsége 71, hálózati 47 $\Rightarrow$ $stretch = 71/47$.

Járványalapú algoritmusok

Alapötlet

- Valamelyik szerveren frissítési műveletet (update) hajtottak végre, azt szeretnénk, hogy ez elterjedjen a rendszerben minden szerverhez.
- Minden szerver elküldi a változást néhány szomszédjának (messze nem az összes csúcsnak) lusta módon (nem azonnal)
- Tegyük fel, hogy nincs olvasás-írás konfliktus a rendszerben.

Két alkategória

- Anti-entrópia: Minden szerver rendszeresen kiválaszt egy másikat, és kicserélik egymás között a változásokat.
- Pletykálás (gossiping): Az újonnan frissült (megfertőzött) szerver elküldi a frissítést néhány szomszédjának (megfertőzi őket).

Járvány: anti-entrópia

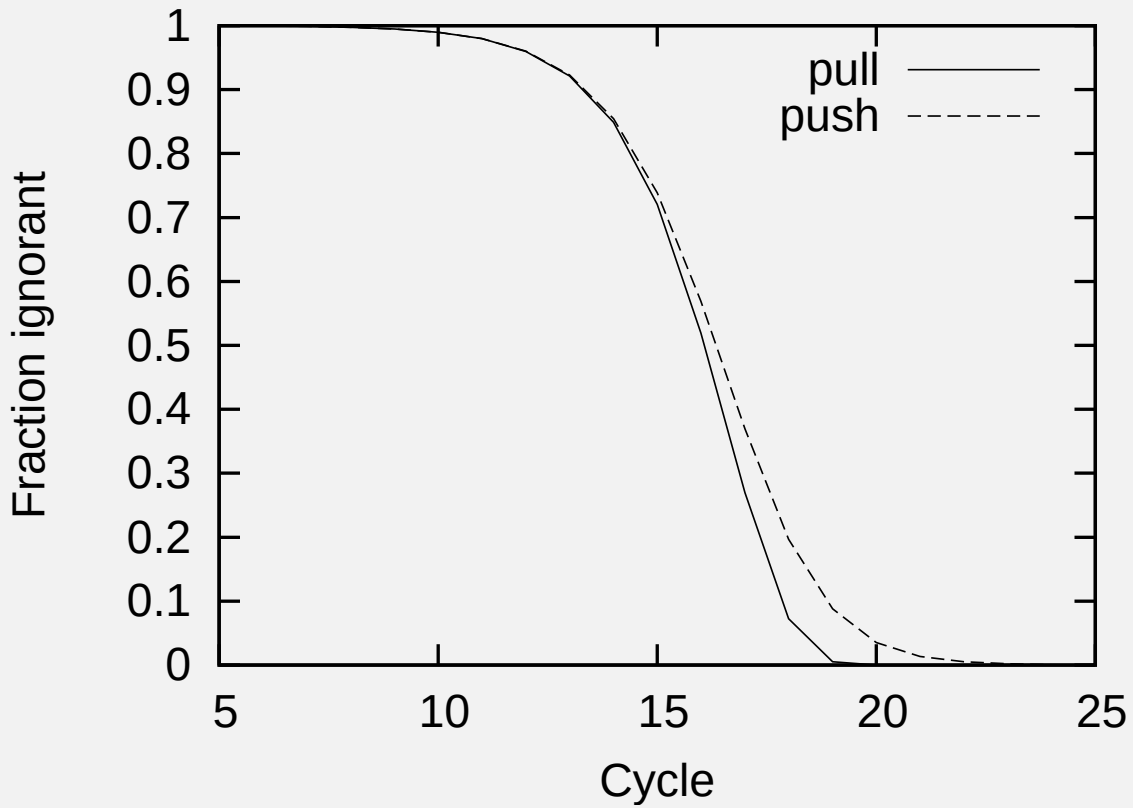
A frissítések cseréje

- P csúcs Q csúcsot választotta ki.
- Küldés (push): P elküldi a nála levő frissítéseket Q-nak
- Rendelés (pull): P bekéri a Q-nál levő frissítéseket
- Küldés–rendelés (push–pull): P és Q kicserélik az adataikat, így ugyanaz lesz mindkettő tartalma.

Hatékonyság

A küldő–rendelő megközelítés esetében $\mathcal{O}(log(N))$ nagyságrendű forduló megtétele után az összes csúcshoz eljut a frissítés.
Egy fordulónak az számít, ha mindegyik csúcs megtett egy lépést.

Járvány: anti-entrópia: hatékonyság



Járvány: pletykálás

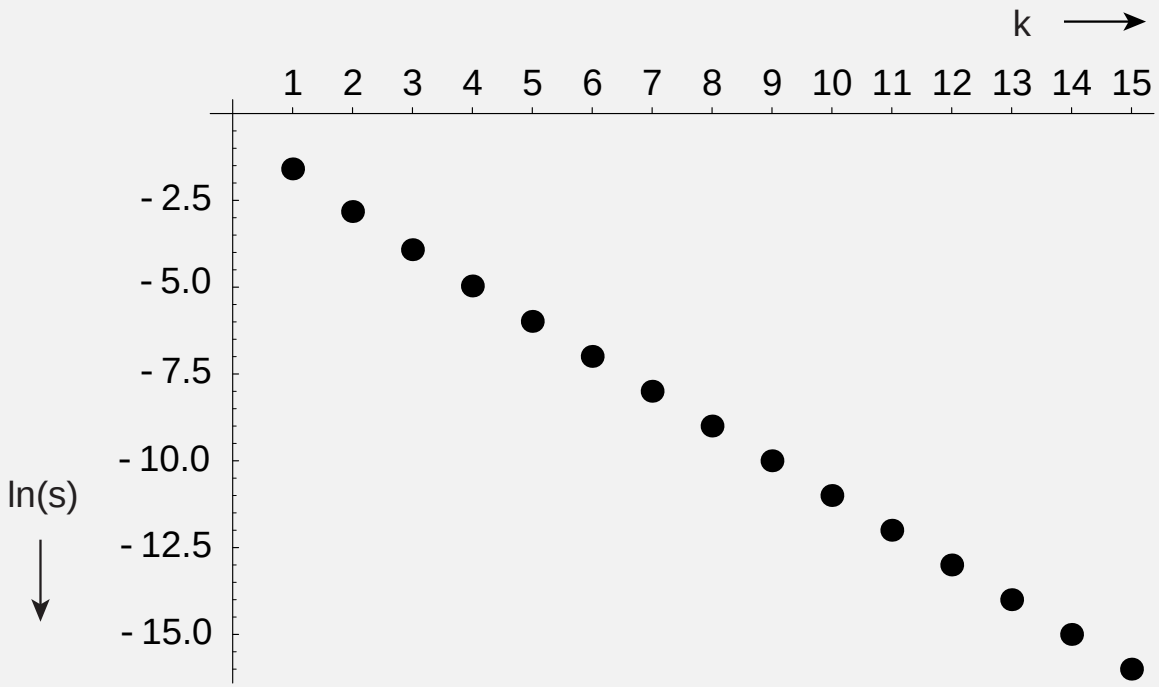
Működési elv

Ha az S szerver új frissítést észlelt, akkor felveszi a kapcsolatot más szerverekkel, és elküldi számukra a frissítést.
Ha olyan szerverhez kapcsolódik, ahol már jelen van a frissítés, akkor $\frac{1}{k}$ valószínűséggel abbahagyja a frissítés terjesztését.

Járvány: pletykálás: hatékonyság

Hatékonyság

Kellően sok szerver esetén a tudatlanságban maradó szerverek (akikhez nem jut el a frissítés) száma exponenciálisan csökken a k valószínűség növekedésével, de ezzel az algoritmussal nem garantálható, hogy minden szerverhez eljut a frissítés.



Consider 10,000 nodes		
k	s	N_s
1	0.203188	2032
2	0.059520	595
3	0.019827	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.000336	3

Járvány: értékek törlése

A törlési művelet nem terjeszthető

Ha egy adat törlésének műveletét is az előzőekhez hasonlóan terjesztenénk a szerverek között, akkor a még terjedő frissítési műveletek újra létrehoznák az adatot ott, ahová a törlés eljutott.

Megoldás

A törlést speciális frissítésként: halotti bizonyítvány (death certificate) küldésével terjesztjük.

Járvány: értékek törlése

Halotti bizonyítvány törlése

A halotti bizonyítványt nem akarjuk örökké tárolni. Mikor törölhetőek?

- Szemétgyűjtés-jellegű megközelítés: Egy rendszerszintű algoritmussal felismerjük, hogy mindenhová eljutott a bizonyítvány, és ekkor mindenhonnan eltávolítjuk. Ez a megoldás nem jól skálázódik.
- elavuló bizonyítvány: Kibocsátás után adott idővel a bizonyítvány elavul, és ekkor törölhető; így viszont nem garantálható, hogy mindenhová elér.

Járvány: példák

Példa: adatok elterjesztése

Az egyik legfontosabb és legjellemzőbb alkalmazása a járványalapú algoritmusoknak.

Példa: adatok aggregálása

Most a cél a csúcsokban tárolt adatokból új adatok kiszámítása. Kezdetben mindegyik csúcs egy értéket tárol: x_i. Amikor két csúcs pletykál, mindkettő a tárolt értékét a korábbi értékek átlagára állítja:

$$x_i, x_j \leftarrow \frac{x_i + x_j}{2}$$

Mivel az értékek minden lépésben közelednek egymáshoz, de az összegük megmarad, mindegyik érték a teljes átlaghoz konvergál.

$$\bar{x} = \frac{\sum_i x_i}{N}$$