



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Kellojen synkronointi itsestabiloituvasti

Hajautetut algoritmit -seminaari

Mikko Pervilä

16.11.2007

Matemaattis-luonnontieteellinen tiedekunta



Esitelmän rakenne

- **Mahdottomuus**, eli katsaus ongelmakenttään
 - Atomikellot
- **Ratkaisuja**, eli mitä saadaan aikaiseksi
 - Olemassa olevia algoritmeja, niillä saatavia tuloksia
- Kellopulssiin perustuva **synkroninen algoritmi**
 - Yksityiskohtia, simulointia
- **Semisynkroninen algoritmi**
 - Karkeammin esitelty, johdattelua toimintaan



Mitä oletetaan jo tunnetuksi?

- Itsestabiloituvuus käsitteenä
 - Viat eivät saa muuttaa algoritmin toimintaa
- Bysanttilaisten kenraalien ongelma
 - n = prosessien (solmujen) lukumäärä
 - f = vikaantuneiden lukumäärä
 - Jos $f > n/3$, ratkeamaton yleisessä muodossa
 - Poikkeustapaukset sivuutetaan



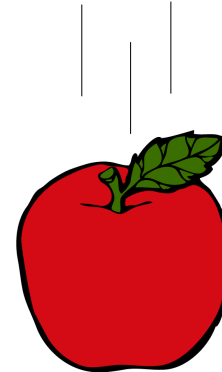
Pääasiallinen lähde

- Dolev, S., Welch, J.L., Self-stabilizing clock synchronization in the presence of Byzantine faults. *Journal of the ACM*, 51,5(syyskuu 2004), sivut 780-799.
- Muut lähteet kirjoitelmassa
 - Linkki seminaarin kotisivuille: [pervila.pdf](#)



Mahdottomuus: kevyttä fysiikkaa

- Unohdetaan suhteellisuusteoria
- Ajatellaan teoreettista *ideaalikelloa*
 - Näyttää aina ajan tarkasti
- Todelliset kellot toteutetaan mekaanisesti tai elektronisesti
 - Tavasta riippumatta aiheutuu virheitä
 - Kelloon vaikuttavat useat tekijät: painovoima, lämpötila, säteily, sähkövirta

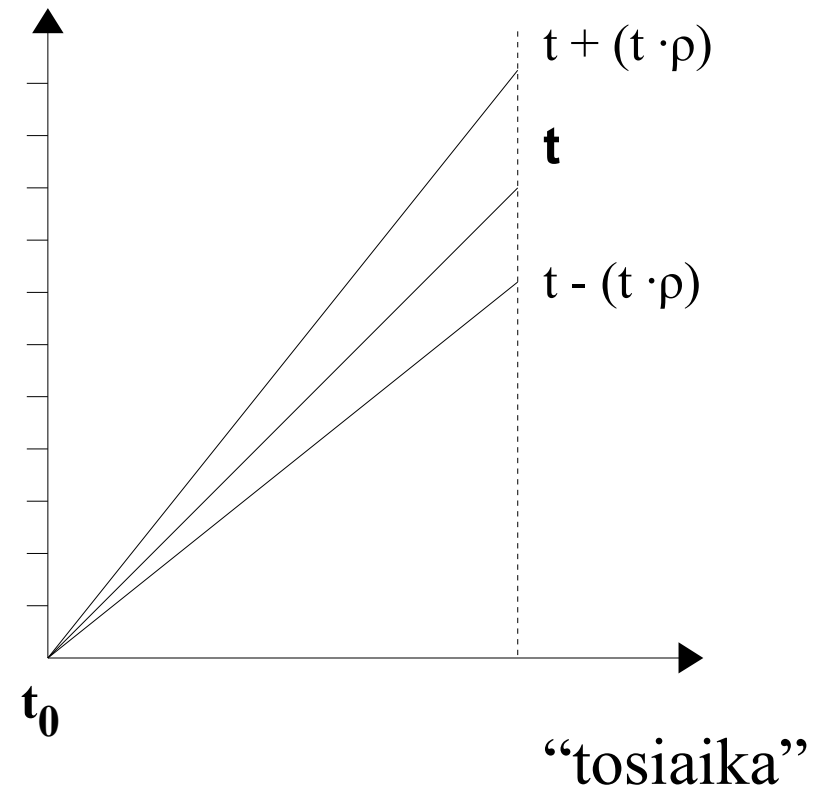




Mahdottomuus: ajelehtiminen

- Kellonaika esitetään rajoitetulla tarkkuudella
- Erotusta ideaalikelloon kutsutaan *ajelehtimiseksi*
- Kello joko *edistää* tai *jätättää*
- Virhe toistuu joka askeleella
- Virhettä merkitään ρ (rho)

kellonaika





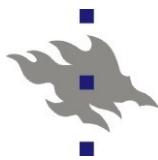
Mahdottomuus: ideaalikello

- Ideaalikello voisi mitata vaikkapa SI-sekunteja
- Miten ideaalikello toteutetaan?
- *The second is the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium 133 atom.* (Lähde: http://www.bipm.org/en/si/base_units/)

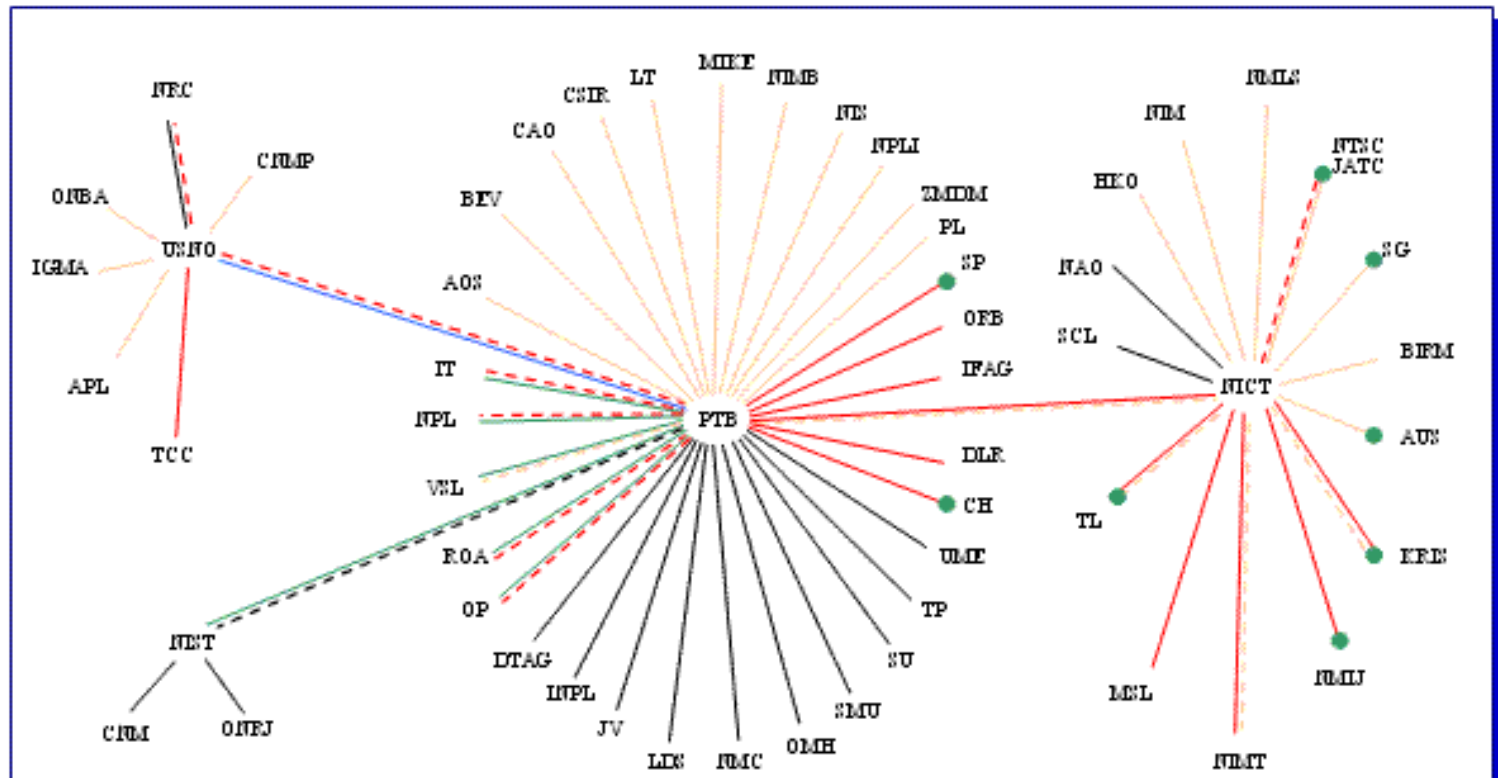


Mahdottomuus: atomikellot

- Universal Coordinated Time (UTC)
 - Tahdistetaan aurinkoon lisäämällä karkaussekunteja
 - Lähteenä International Atomic Time
- International Atomic Time (TAI)
 - Ei välitä taivaankappaleiden tai maan liikkeistä
 - 250 laboratoriota, 50 atomikelloa



Mahdottomuus: atomikellot



ORGANIZATION OF THE INTERNATIONAL TIME LINKS

April 2006

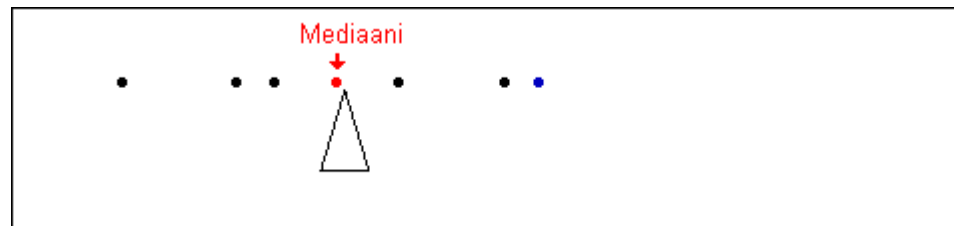
- Laboratory equipped with TWSTFT (not yet used)
 - TWSTFT by Ku band with X band back-up
 - TWSTFT link
 - GPS CV single-channel link
 - - - GPS CV single-channel back-up link
- GPS CV multi-channel link
 - - - GPS CV multi-channel back-up link
 - GPS CV dual frequency link
 - - - GPS CV dual frequency back-up link





Mahdottomuus: atomikellot

- Menemättä liikaa yksityiskohtiin, TAI on painotettu keskiarvo.
 - Siihenkin tehdään pieniä korjauksia: $-6 \cdot 10^{-15}$ viime vuonna
 - Tarkkuus noin $(1,2) \times 10^{-15}$
- Keskiarvolla on joitakin ikäviä ominaisuuksia:
 - Juha Puranen / Tilastotieteen laitos:
<http://noppa5.pc.helsinki.fi/uudet/da1htm/sanasto.html>



- Uuden atomikellon rakennus?



Ensimmäinen neljännes ohitse

1/4



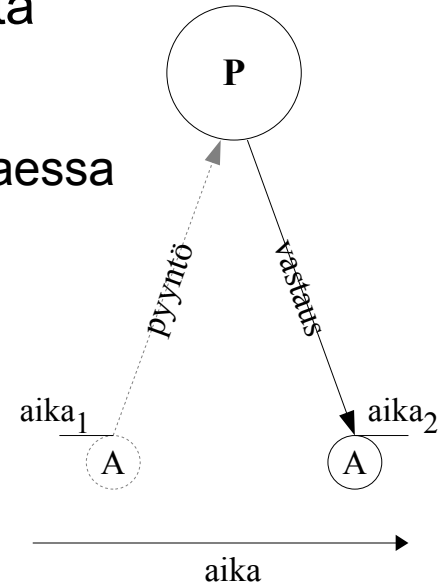
Ratkaisuja: mitä on tehty aiemmin?

- Cristianin algoritmi
 - Jonka avulla saadaan arvioitua tiedonsiirtoviive
- Berkeleyyn algoritmi
 - Keskiarvojen avulla muodostettu konsensus
- Lamportin aikaleimat
 - Muunnetaan ongelmaa helpommaksi



Ratkaisuja: Cristianin algoritmi

- Haetaan kellonaika luotetulta palvelimelta
 - Arvioidaan kauanko siirtoon meni
 - Huomioidaan siirtoaika kellonaikaa asettaessa
- Lievä oletus: tieto luotettavaa
 - Palvelin ja tiedonsiirto luotettavia
 - Autentikointi- ja salausmenetelmät
- Vahva oletus: palvelimella oikea kellonaika
 - Atomikello?





Ratkaisuja: Cristianin algoritmi

- Kun lähetetään palvelimelle pyyntö,
 - kirjoitetaan pyyntöön aikaleima ”vanhalla” kellolla
- Palvelin kopioi aikaleiman mukaan vastaukseen
- Kun vastaanotetaan palvelimelta vastaus
 - kirjoitetaan vastaukseen aikaleima ”vanhalla” kellolla
- Lopputuloksena *arvio* siirtoviiveestä d

$$d = \frac{(aika_1 - aika_2)}{2}$$



Ratkaisuja: Berkeleyyn algoritmi

- Tutkii, mitä kellonaikoja löytyy ja laskee näistä yhteisen arvon kaikille
 - Välittää tiedon *muutoksena* vanhaan kellonaikaan
- Käyttää keskiarvoa, kykenee valitsemaan johtajan
 - *Myös* mikäli johtaja katoaa tai monistuu verkon osittumisen seurauksena
- Jättää huomioimatta ”selvästi vialliset” kellonajat
 - Määrittys esimerkiksi parametrilla **b**
 - Oikean arvon valinta vaatii ennakointikykyä



Ratkaisuja: Berkeleyn algoritmi

- Keskiarvon käyttö ei takaa, että kello olisi oikeassa ajassa algoritmin suorituksen jälkeen
 - Mutta suunta lienee oikea
 - Palaamme keskiarvoihin vielä hieman myöhemmin



- Mikä arvo on parametrille **b** kaikkein sopivin?
 - Liian pieni: unohdetaan suurin osa osallistujista
 - Liian suuri: poikkeavien arvojen merkitys kasvaa



Ratkaisuja: Lamportin aikaleimat

- Lamportin aikaleimat muuntavat ongelmaa
 - Käyttävät hyväkseen osittaista järjestysrelaatiota →
 - → voidaan mieltää ”syy-seuraus-suhteena”
 - Järjestämätöntä osaa kutsutaan rinnakkaisiksi
 - Lisäksi siirrytään *loogiseen* kelloon
- Monotonisesti kasvavan laskurin arvoilla merkitään ne tapahtumat, jotka on tarpeen pystyä järjestämään
 - Esimerkiksi lähetettävät viestit ja kuittaukset
- Tarkempi läpikäynti sivuutetaan
 - Timo Alangon luentokalvot ovat eräs hyvä lähde (simulointia)
<http://www.cs.helsinki.fi/u/alanko/hj/K06/kalvot/ch4.ppt>



Puoliaika!

1/2



Synkroninen algoritmi

- Valitettavan tylsä, käytiin läpi jo edellisellä viikolla!
 - Esitelmää yritetty painottaa uusiksi
- Tutkitaan oletuksia
- Simuloidaan
- Mietitään ongelmatapausta



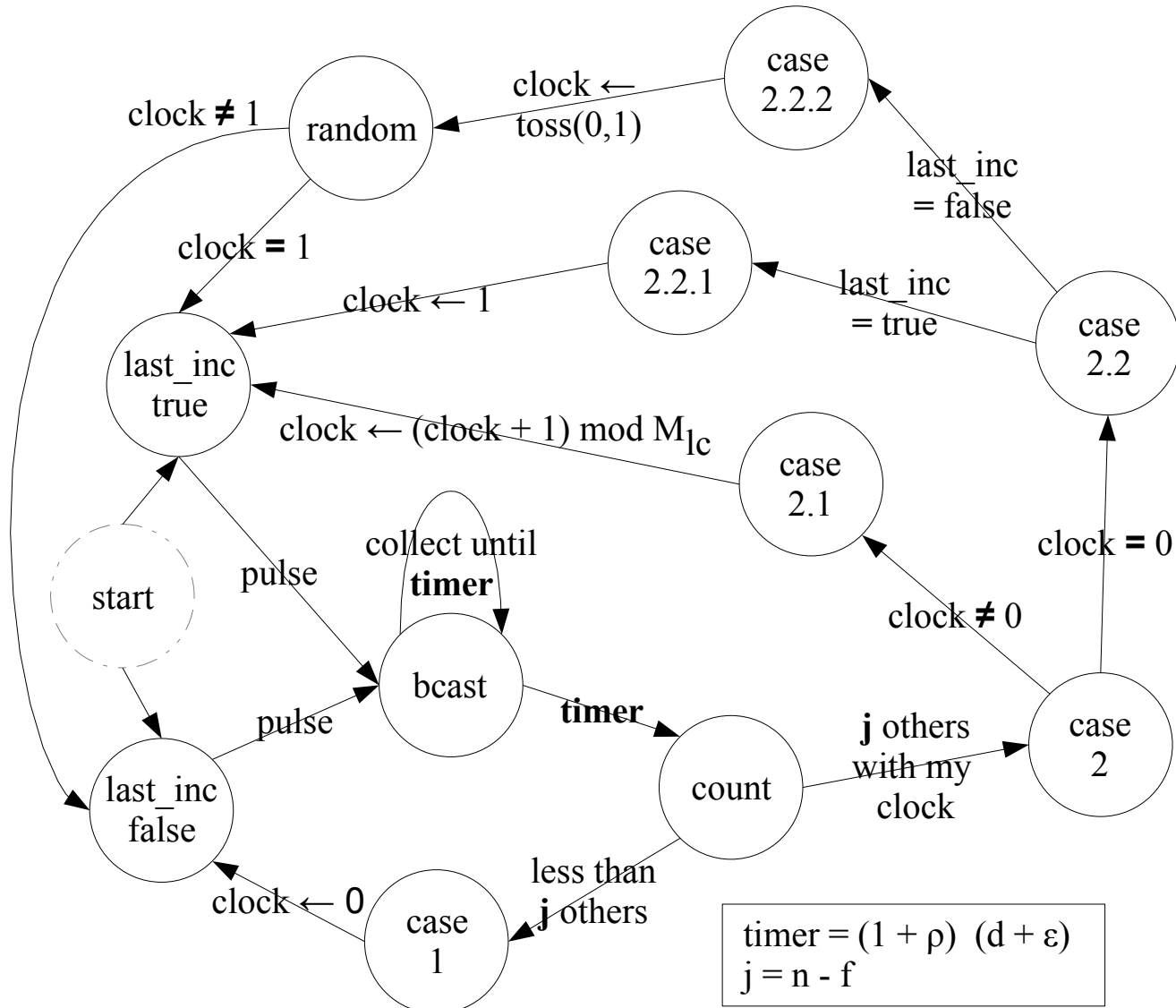
Synkroninen algoritmi: ohjelmakoodina

```
01 when pulse occurs:
02   broadcast clocki
03   collect clock values until  $(1 + \rho)(d + \varepsilon)$  time has elapsed on the physical clock
04   if  $|\{j | \text{clock}_i = \text{clock}_j\}| < n - f$  then (case 1)
05     clocki  $\leftarrow 0$ 
06     last_incrementi  $\leftarrow \text{false}$ 
07   else (case 2)
08     if clocki  $\neq 0$  then (case 2.1)
09       { clocki  $\leftarrow (\text{clock}_i + 1) \bmod M_{1c}$ 
10         last_incrementi  $\leftarrow \text{true}$  }
11     else (case 2.2)
12       if last_incrementi = true then (case 2.2.1) clocki  $\leftarrow 1$ 
13       else (case 2.2.2) clocki  $\leftarrow \text{toss}(0, 1)$ 
14       if clocki = 1 then last_incrementi  $\leftarrow \text{true}$ 
15       else last_incrementi  $\leftarrow \text{false}$ 
```

Pieniä eroja kirjoitelmaan



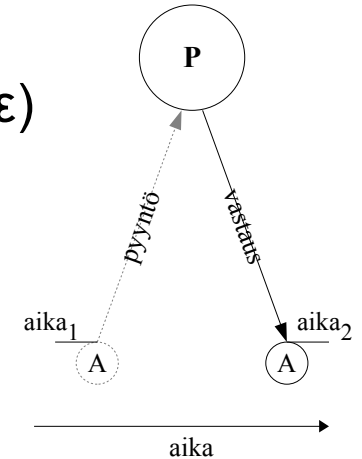
Synkroninen algoritmi: tilakoneena





Synkroninen algoritmi: oletuksia ja muuttujia

- Yläraja viestien odottelemiselle: $(1 + \rho)(d + \varepsilon)$
 - ρ = fyysisen kellon ajalehtimisen yläraja
 - d = arvio tiedonsiirtoviiveelle
 - ε = arviointivirhe, satunnainen elementti
 - Jos kestää yli tämän, käsitellään puuttuvana



- Toimivat ja vikaantuneet prosessit
 - Järjestelmään liittyminen ja siitä poistuminen julkisia
 - Jokainen prosessi pystyy laskemaan osallistujien määrän n
 - Vikaantuneiden lukumäärä f arvioidaan ylärajaksi $n/3$
- Mitä jos f kasvaa yli $n/3$?
 - Simuloidaan seuraavaksi hieman



Synkroninen algoritmi: simulointia

lf = (last_increment_i ← false)

	toss									
P1	155	123	198	198	15		21	lf 0	1	2
P2	155	156	157	158	lf 0		lf 0	lf 0	lf 0	lf 0
P3	155	156	157	158	lf 0	...	lf 0	lf 0	1	2
P4	155	156	157	6	7		225	255	1	1

$M_{lc} = 8, 2^8 = 256, [0-255]$

	toss									
P1	lf 0	1	2	3	4		255	0	1	
P2	lf 0	1	2	3	4		255	0	1	
P3	lf 0	1	2	3	4	...	255	0	1	...
P4	5	127	127	lf 0	lf 0		lf 0	lf 0	1	



Synkroninen algoritmi: toiminnan nopeuttaminen

- Palautuneet prosessit pitäisi saada nopeammin mukaan kuvioihin!
 - Kiinalainen jakojäännös
 - Nähty viime viikolla, esimerkki Brunbergin kalvoissa
- Jaetaan looginen kello (a, b, \dots, n) pienempään laskuriin ja koodataan kellonarvot vastaamaan monikoita
 - Ajetaan kellopulssiin perustuvaa algoritmia jokaiselle laskurille
 - Jokainen laskuri a, b, \dots, n pyörähtää ympäri nopeammin
 - Muistuttaa hieman hedelmäpeliä
- Näin toimimalla 64 bitin looginen kello saadaan stabiloitumaan $381 \cdot 2^{2(n-f)}$ pulssin jälkeen
 - n ei siis voi olla kovin suuri



Enää vartti jäljellä!

3/4



Semisynkroninen algoritmi

- Ei käytä keskitettyä kellopulssia
- Nimi **semisynkroninen** lienee peräisin siitä, että algoritmi käyttää kahta fyysiseen kelloon perustuvaa laskuria
- Tutkitaan oletuksia
- Kevyt johdanto algoritmin toimintaan
- Satunnaisuus tekee simuloinnista hankalaa
 - Ehkä vähän epäuskottavaakin

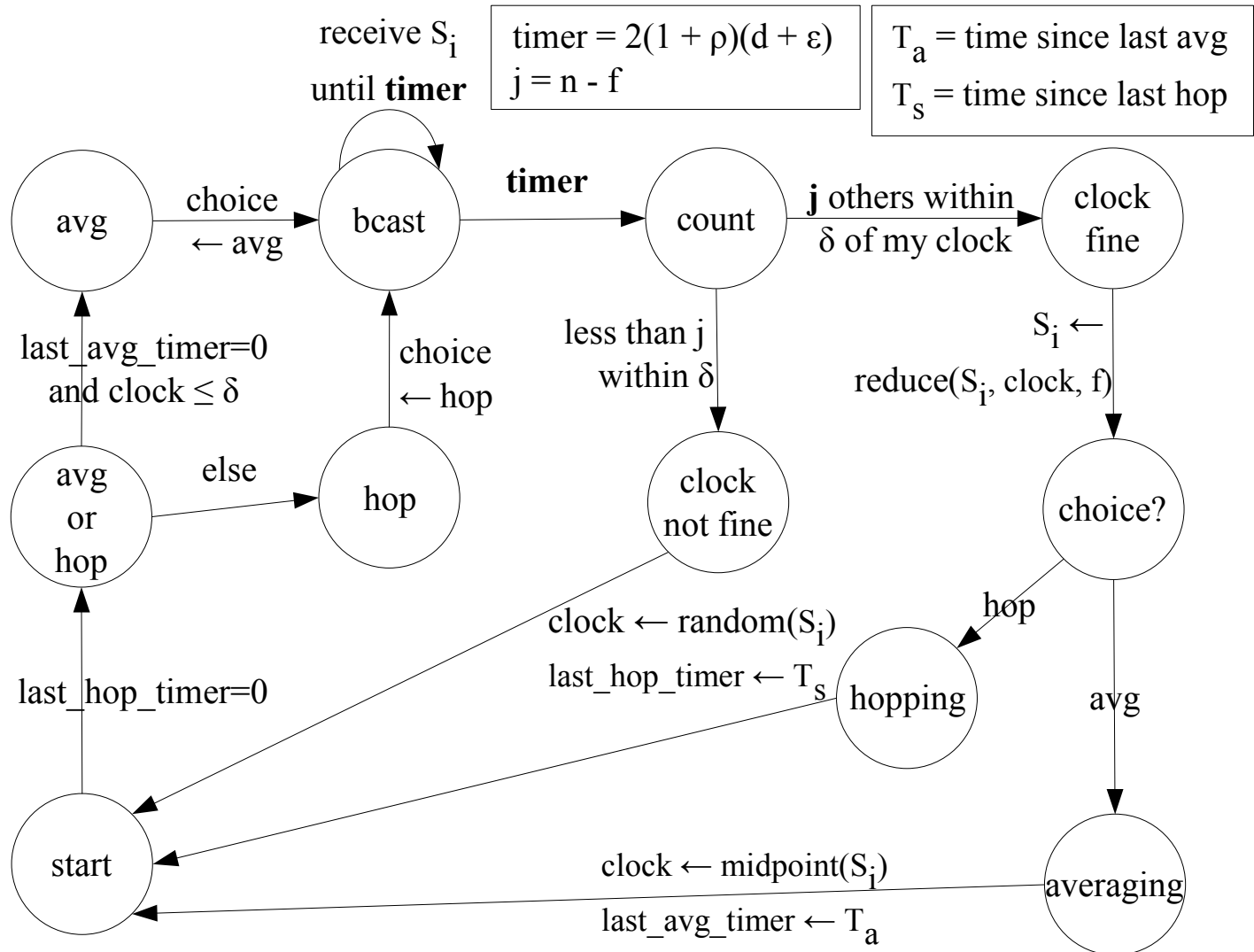


Semisynkroninen algoritmi: ohjelmakoodina

```
01 if last_hop_timeri = 0 then
02   if last_avg_timeri = 0 and clocki ≤ δ then
03     choicei ← “average”
04   else choicei ← “hop”
05   broadcast “clock-request” message
06   let Si be multiset of clock values collected until 2(1 + ρ)(d + ε) time has elapsed
       on the physical clock
07   if n - f elements of Si are within δ of clocki then
08     Si ← reduce( Si , clocki , f )
09     if choicei = “average” then (*do the averaging procedure*)
10       clocki ← midpoint( Si )
11       last_avg_timeri ← Ta (*set physical clock timer - counts down to 0*)
12     else (*do the hopping procedure*)
13       clocki ← random( Si )
14       last_hop_timeri ← Ts (*set physical clock timer - counts down to 0*)
15   else (*less than n - f clock values are within clocki *)
16     clocki ← random(Si )
17     last_hop_timeri ← Ts
```



Semisynkroninen algoritmi: tilakoneena





Semisynkroninen algoritmi: oletuksia ja muuttujia

- Samat oletukset tiedonsiirtoviiveestä ja vikaantuneiden prosessien määrästä
- Kaksi toimintatapaa ja laskuria, T_s ja T_a
 - S = hopping, valitaan satunnaisesti uusi kellonarvo vastaanotetuista
 - A = averaging, pyritään synkronoimaan kellot keskiarvofunktion avulla
- Käyttää keskiarvofunktiota *reduce*, jonka tehtävänä on muodostaa keskiarvo hajautetusti
 - Joka askeleella leikataan suurimmat ja pienimmät arvot pois
 - Suunnilleen puolittaa kellonarvojen joukon



Semisynkroninen algoritmi: toiminta

- T_s ja T_a ovat nollaan väheneviä laskureita, joiden tarkoitus on kiinnittää algoritmin eri toimintatiloihin käyttämä aika
 - Muistuttavat siis hieman munakelloja
 - T_a on moninkertaisesti suurempi
 - Lisäksi keskinäinen suhde on rajoitettu
- Kellonarvot kerätään verkosta **joka** suorituskerralla
- Reduce suoritetaan **myös** ennen satunnaista valintaa
 - Paitsi jos oma kellonaika on liian harhautunut



Semisynkroninen algoritmi: kuristustekniikka

- Suomennetaan reduce vaikkapa arvojen kuristamiseksi
- $n = 10, f = 3, \text{clock}_i = 155$

$$S_i = \{154, 156, 155, 130, 160, 156, 154, 156, 155, 178\}$$

$$S_i \leftarrow \text{reduce}(S_i, 155, 3)$$

$$S_i = \{154, 156, 155, 130, 160, 156, 154, 156, 155, 178\}$$

$$S_i = \{154, 156, 155, 156, 154, 156, 155\}$$



Semisynkroninen algoritmi: kuristustekniikka

- Mikä on δ ?
 - Kynnysarvo oman kellon oikeellisuudelle
 - Rajoittaa keskiarvon valitsemisen laskemisen tiheyttä
 - (Saattaa olla muitakin)
- Semisynkroninen algoritmi toipuu ajassa $T_a n^{6(n-f)}$
 - Jälleen, prosessien lukumäärä n rajoittaa käytettävyyttä



Lyhyt yhteenveto: oikeellisuustodistukset?

- Toimintaa mallinnetaan vuorotellen pelattavana pelinä
- Osapuolina oikeaan tulokseen pyrkivät *onni* ja huonoimpaan mahdolliseen tulokseen pyrkivä *vuorottelija*
 - Synkronisen algoritmin tapauksessa onni pääsee jossakin vaiheessa valitsemaan $(0,1)$ -arvot sopivasti oikein toimiville solmuille
 - Semisynkronoidun algoritmin tapauksessa onni pääsee jossakin vaiheessa tilanteeseen, jossa solmut suorittavat kaikki keskiarvofunktiota
- Yksityiskohdat: omalla vastuulla, omalla ajalla
 - ”□”



Lyhyt yhteenveto: mitä tästä opittiin

- Kellonaikojen synkronoiminen on hankalaa, etenkin hajautetusti
- Voidaan muuntaa perusongelmaa ja ratkaista helpompia osaongelmia
- Algoritmeja yhdistelemällä päästään hieman parempiin tuloksiin
- Kysymyksiä, kommentteja?
 - (älkää kysykö kauhean vaikeita...)