Programmēšanas abstrakcijas

Ģirts Strazdiņš
Kurss “Bezvadu sensoru tīkli” [B]
Datorikas fakultāte
Latvijas Universitāte
23.nov.2011.

Kas ir galvenais sensoru tīklu uzdevums?

Kas ir galvenais sensoru tīklu uzdevums?

Savākt informāciju!

Informācijas savākšana ietver

- Jēlo datu interpretāciju
- Informācijas apstrāde, agregācija
- Rezultātu nogādāšana līdz apstrādes vietai

Kāpēc nesūtīt visu uz bāzes staciju?

Kāpēc nesūtīt visu uz bāzes staciju?

- Komunikācija tērē enerģiju
- Radio kapacitāte ierobežota

Galvenās pieejas

- Vaicājumu (query) bāzētas pieejas
- Aģentu bāzētas pieejas
- Makro programmēšana

Makro programmēšana

- Ļauj programmēt visu tīklu kopumā
- Kompilators globālo programmu nokompilē individuāliem mezgliem

Makro programmēšanas priekšrocības un trūkumi

+ Ļauj domāt par visu tīklu kopumā
- Šāda paradigmas maiņa nav vienkārša
- Sarežģīta kompilatoru būve

Makro programmēšanas piemērs: Pleaides

- Specifiska valoda, kas kompilējas uz n TinyOS programmām

```
1: #include "pleiades.h"
2: boolean nodelocal isfree=TRUE;
3: nodeset nodelocal neighbors;
4: node nodelocal neighborIter;

5: void reserve(pos dst) {
6:  boolean reserved=FALSE;
7:  node nodeIter,reservedNode=NULL;
8:  node n=closest_node(dst);
9:  nodeset loose nToExamine=add_node(n, empty_nodeset());
10: nodeset loose nExamined=empty_nodeset();


11: if(isfree@n) {
12:  reserved=TRUE; reservedNode=n;
13:  isfree@n=FALSE;
14:  return;
15: }

16: while(!reserved && !empty(nToExamine)){
17:  cfor(nodeIter=get_first(nToExamine);nodeIter!=NULL;
         nodeIter = get_next(nToExamine)){
18:   neighbors@nodeIter=get_neighbors(nodeIter);
19:   for(neighborIter@nodeIter=get_first(neighbors@nodeIter);
         neighborIter@nodeIter!=NULL;
         neighborIter@nodeIter=get_next(neighbors@nodeIter)){
20:    if(!member(neighborIter@nodeIter,nExamined))
21:     add_node(neighborIter@nodeIter,nToExamine);
22:   }
23:   if(isfree@nodeIter){
24:    if(!reserved){
25:     reserved=TRUE; reservedNode=nodeIter;
26:     isfree@nodeIter=FALSE;
27:     break;
28:    }
29:   }
30:   remove_node(nodeIter,nToExamine);
31:   add_node(nodeIter,nExamined);
32:  }
33: }
34:}
```

Ko dara šī programma?

```
1: #include "pleiades.h"
2: boolean nodelocal isfree=TRUE;
3: nodeset nodelocal neighbors;
4: node nodelocal neighborIter;

5: void reserve(pos dst) {
6:  boolean reserved=FALSE;
7:  node nodeIter,reservedNode=NULL;
8:  node n=closest_node(dst);
9:  nodeset loose nToExamine=add_node(n, empty_nodeset());
10: nodeset loose nExamined=empty_nodeset();


11: if(isfree@n) {
12:  reserved=TRUE; reservedNode=n;
13:  isfree@n=FALSE;
14:  return;
15: }

16: while(!reserved && !empty(nToExamine)){
17:  cfor(nodeIter=get_first(nToExamine);nodeIter!=NULL;
         nodeIter = get_next(nToExamine)){
18:   neighbors@nodeIter=get_neighbors(nodeIter);
19:   for(neighborIter@nodeIter=get_first(neighbors@nodeIter);
         neighborIter@nodeIter!=NULL;
         neighborIter@nodeIter=get_next(neighbors@nodeIter)){
20:    if(!member(neighborIter@nodeIter,nExamined))
21:     add_node(neighborIter@nodeIter,nToExamine);
22:   }
23:   if(isfree@nodeIter){
24:    if(!reserved){
25:     reserved=TRUE; reservedNode=nodeIter;
26:     isfree@nodeIter=FALSE;
27:     break;
28:    }
29:   }
30:   remove_node(nodeIter,nToExamine);
31:   add_node(nodeIter,nExamined);
32:  }
33: }
34:}
```

Ko dara šī programma?

Figure 1. A street-parking application in Pleiades.

Aģentu bāzētas programmas

- Kas ir aģents?

Aģentu bāzētas programmas

- Aģents ir programma, kas ceļo starp mezgliem
- Nevis dati nāk pie programmas, bet programma iet pie datiem
- Aģenti autonomi pārvietojas
- Vienlaikus tīklā var darboties vairāki aģenti
- Kaut kas ļoti eksotisks

Aģentu priekšrocības

- Ļoti plašas iespējas
- Mazs datu pārraides apjoms (?)
- Spēcīga decentralizācija, apstrāde seko datiem
- Spēja dinamiski reaģēt uz izmaiņām tīklā

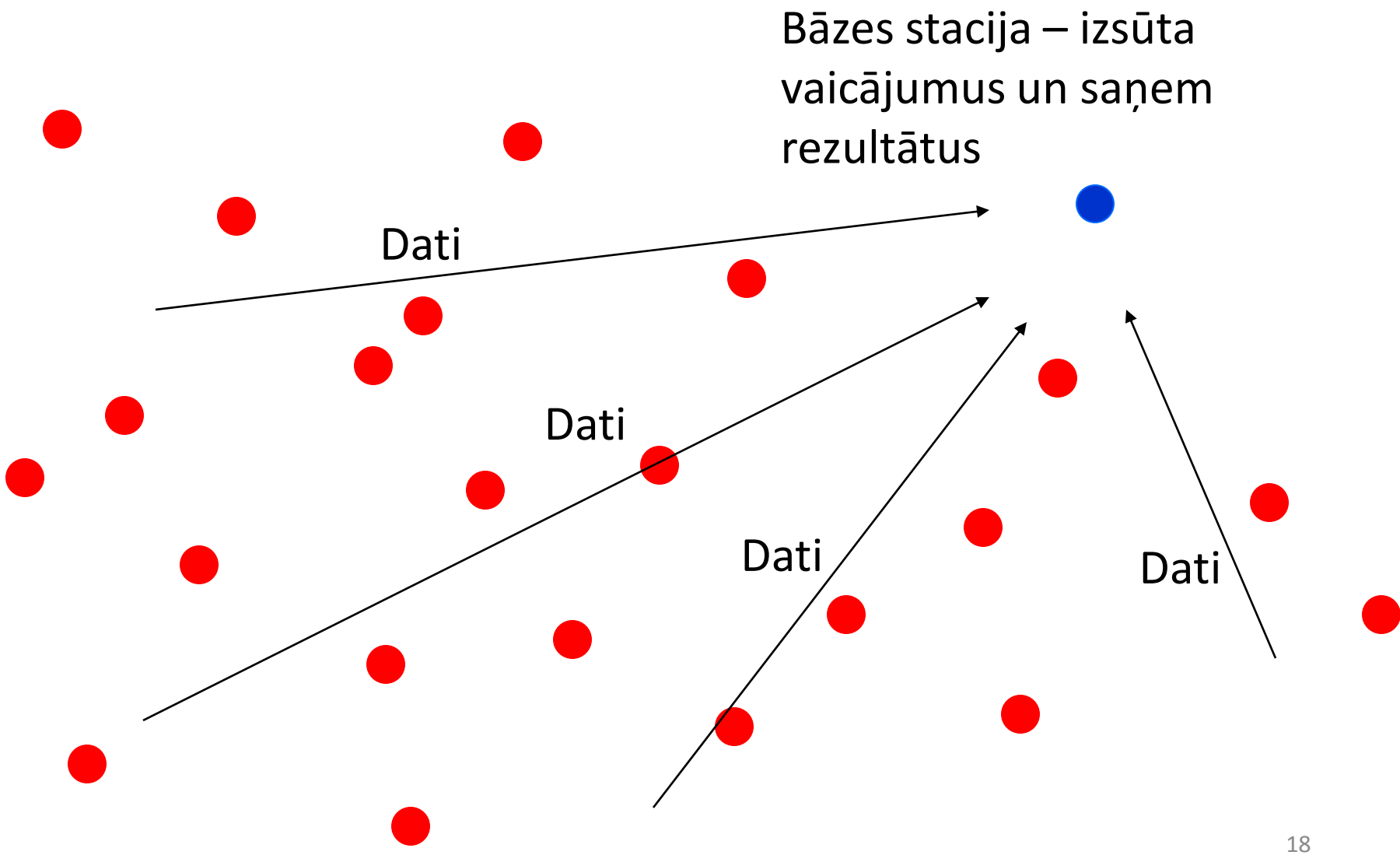
Aģentu trūkumi

- Sarežģīti modelēt tīkla komponentes un to sadarbību
- Grūti paredzēt izmaiņas, uz kurām jāreaģē
- Sarežģīta testēšana un atkļūdošana

Vaicājumu bāzētas pieejas

- Tīklā iesūta vaicājumu, atpakaļ nāk atbilde
- Parasti centralizēti risinājumi: bāzes stacija vaicā, tīkls atbild
- Vispopulārākā pieeja – vienkārša realizācija

Tipiska vaicājumu pieejas arhitektūra



Pieejas pamatmērķis

- Sensoru tīkls kā dalīta (distributed) datu bāze
- Problēmas:
  - Ierobežota atmiņa
  - Nepieciešams kopīgs *duty cycle*
  - Mezgli parasti nav uzticami
  - Dati var būt trokšņaini
  - Parasti datiem svarīgs arī laiks un vieta

Vaicājumu pieejas priekšrocības

- Vienkārša realizācija
- Abstrahēšanās no tīkla topoloģijas

Vaicājumu pieejas trūkumi

- Pārsvarā triviāla datu apstrāde
- Vaicājums parasti vai nu konkrētam mezglam, vai visam tīklam
- (Sarežģīti apstrādāt telpas un laika atribūtus)

Vaicājumu pieejas piemērs: TinyDB

- SQL-tipa dalītā datu bāze sensoru tīkliem
- Darbojas kā slānis virs TinyOS
- Visi sensoru mezgli kā daļa no tabulas
- Katrs sensoru lasījums kā viens ieraksts
- Vaicājumus izsūta un atbildes savāc bāzes stacija

- Sīkāk rakstā: S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “TinyDB: an acquisitional query processing system for sensor networks,” ACM Transactions on Database Systems (TODS), vol. 30, no. 1, pp. 122–173, 2005.

TinyDB vaicājuma piemērs

```
SELECT nodeid, light, temp
  FROM sensors
  SAMPLE PERIOD 1s FOR 10s
```

Lasa gaismu un temperatūru ik sekundi, 10 sekunžu garumā

TinyDB materializācijas punkti

- Lai veidotu apakšvaicājumus, kārtošanu, *join* operācijas un buferēšanu, izmanto datu uzkrāšanu atsevišķos mezglos, sauktos par materializācijas punktiem

CREATE
  STORAGE POINT recentlight SIZE 8
  AS (SELECT nodeid, light FROM sensors
  SAMPLE PERIOD 10s)

TinyDB JOIN piemērs

```
SELECT COUNT(*)
  FROM sensors AS s, recentLight AS rl
  WHERE rl.nodeid = s.nodeid
  AND s.light < rl.light
  SAMPLE PERIOD 10s
```

TinyDB Agregācija

- Datu agregāciju veic pēc iespējas tuvu datu avotam, notiek automātiski

SELECT AVG(volume), room FROM sensors
WHERE floor = 6
GROUP BY room
HAVING AVG(volume) > *threshold*
SAMPLE PERIOD 30s

Agregācijas funkcijas

- Ir dažas gatavas: AVG, MIN, MAX
- Agregātfunkcijas izmanto datu stāvokļus s_i
- Var definēt savas. Jānodrošina 3 funkcijas:
  - Inicializācija $s_0 = i(x)$
  - Apvienošana $m(s_i, s_j)$
  - Evaluators $e(s_i)$

Agregātfunkcijas piemērs: AVG

- $s_i = \langle sum_i, count_i \rangle$
- $s_0 = i(x) = \langle x, 1 \rangle$
- $m(\langle sum_i, count_i \rangle, \langle sum_j, count_j \rangle)$
  $= \langle sum_i + sum_j, count_i + count_j \rangle$
- $e(\langle sum_i, count_i \rangle) = sum_i / count_i$

Agregātfunkcijas ar laika logu

```
SELECT WINAVG(volume, 30s, 5s)
  FROM sensors
  SAMPLE PERIOD 1s
```

- Tas ir *sintakses cukurs*, to pašu var panākt ar

```
CREATE
  STORAGE POINT vol SIZE 30
  AS (SELECT volume FROM sensors
  SAMPLE PERIOD 1s)

SELECT AVG(volume) FROM vol
  SAMPLE PERIOD 5s
```

TinyDB notikumu apstrāde

- Datu savākšanu var iniciēt ar lokālu, iepriekš definētu, notikumu

ON EVENT bird-detect(loc):

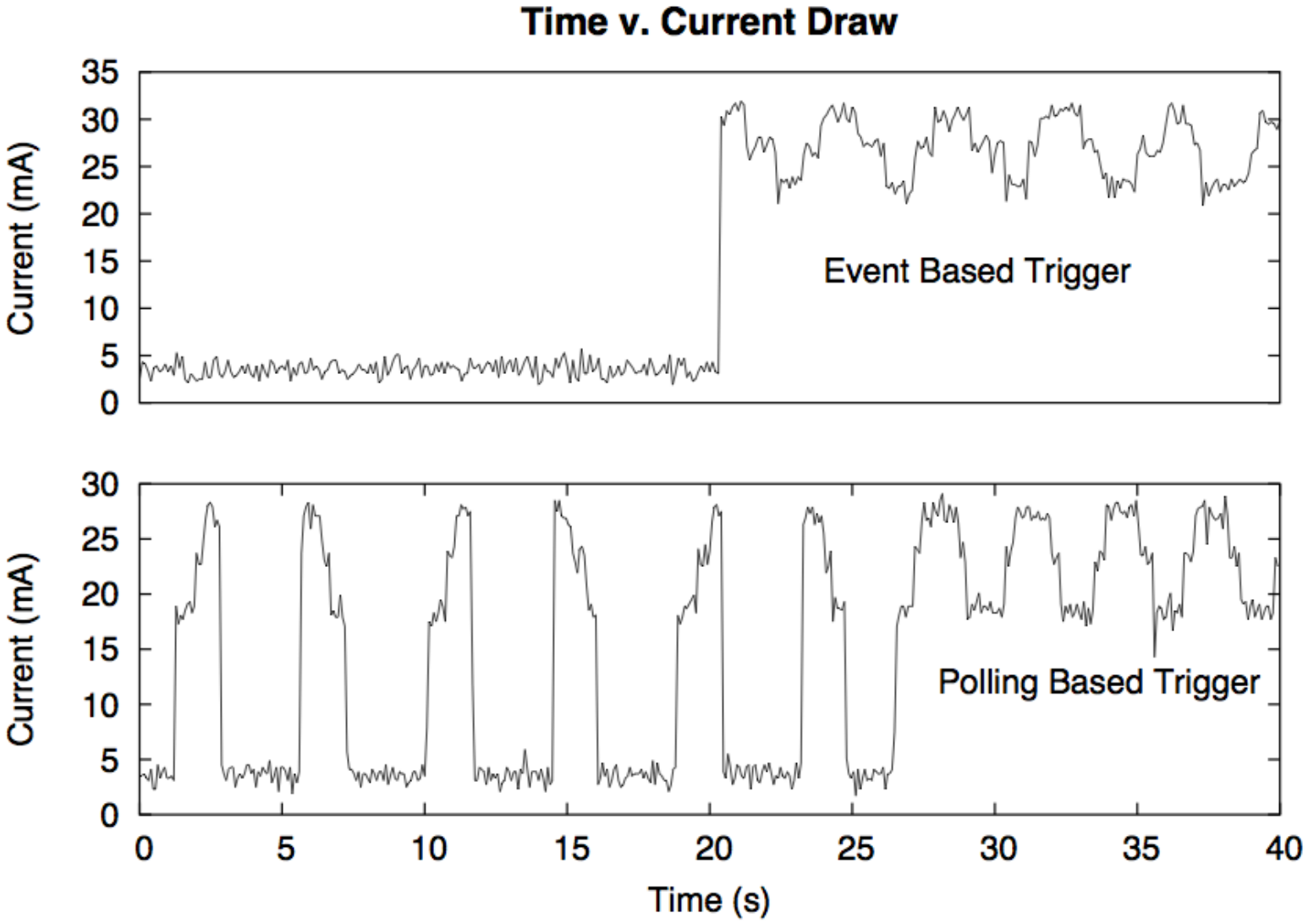
SELECT AVG(light), AVG(temp), event.loc

FROM sensors AS s

WHERE dist(s.loc, event.loc) < 10m

SAMPLE PERIOD 2 s FOR 30 s

Notikumu izmantošanas enerģijas ietaupījums



Notikums var arī apstādināt

```
SELECT AVG(light), AVG(temp),
  FROM sensors AS s
  SAMPLE PERIOD 2 s FOR 30 s
  STOP ON EVENT(bird-away)
```

Var arī no otras puses

SELECT nodeid, temp

WHERE temp > thresh

OUTPUT ACTION SIGNAL hot(nodeid, temp)

SAMPLE PERIOD 10s

Piemēram, hot() notikuma apstrāde nozīmē “ieslēgt ventilāciju”

Sample rate var aizstāt ar lifetime

SELECT nodeid, accel

FROM sensors

LIFETIME 30 days

- Lasīt tik bieži, lai baterijas pietiktu 30 dienām

TinyDB atbalstīto vaicājumu tipi

- Monitorings – mērīt temperatūru kokos ik 5min
- Tīkla veselības uzraudzība – pārbaudīt bateriju līmeni ik 1h
- Statusa pārbaude – iegūt momentuzņēmumu par aktuālo gaismas daudzumu visā telpā
- Aktuatoru vaicājumi – noteiktos apstākļos ieslēgt/izslēgt apkuri, izziņot trauksmi
- Pēcapstrāde – saglabāt lielus datu apjomus lokāli, pārsūtīt vēlāk

TinyDB vaicājuma izplatīšana

- Vaicājumi vairāku tipu:
  - Visiem mezgliem
  - Konkrētam mezglam i
  - Mezgliem reģionā <x1, y1, x2, y2>

- Katram vaicājumam tiek uzbūvēts Semantic Routing Tree (SRT)

SRT būve

- Ideja – vaicājums jānodod visiem mezgliem, kas ar varbūtību p > 0 ievāks vajadzīgos datus

- Katram vaicājumam ir konstants parametrs A, pēc kura atfiltrē datu vācējus. Piemēram, location vai nodeId.

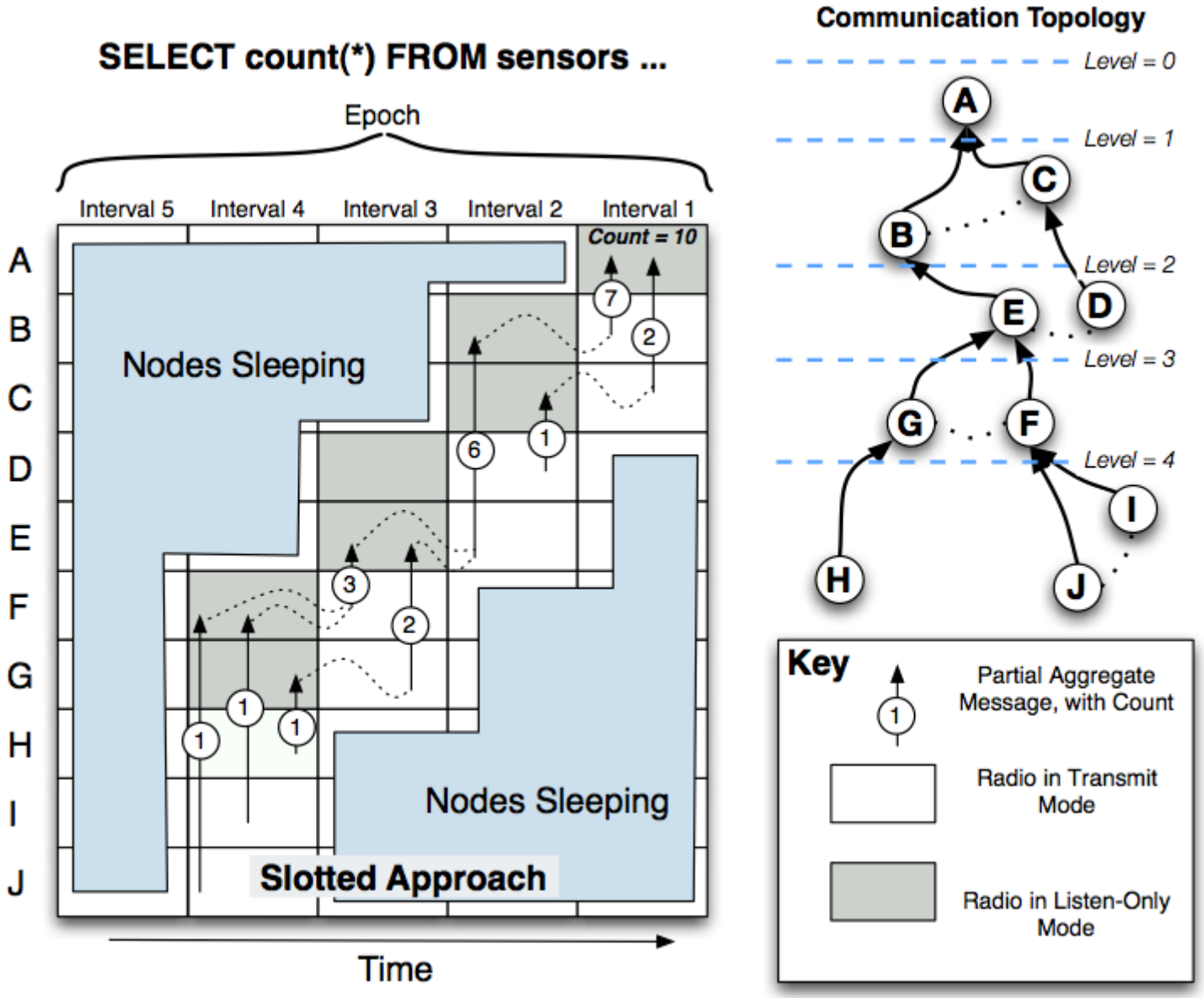
SRT būves soļi

- Bāzes stacija izsūta SRT pieprasījumu ar atribūtu A
- Mezgli, kas dzird:
  - Ja ir iespēja, ka būs iesaistītie bērnu mezgli, pārsūta SRT
  - Visi bērni izrēķina savu A vērtību
  - Kad visi bērni atbildējuši, mezgls pieglabā savu bērnu min un max A vērtības, izvēlas savu vecāku, nosūta tam

SRT būves piemērs

- Pazīmēt uz tāfeles!

TinyDB duty cycle piemērs



TinyDB kopsavilkums

- Dalītā datu bāze sensoru tīkliem
- Slānis virs TinyOS
- Piedāvā vienkāršu saskarni
- Realizē efektīvu agregāciju
- Der vairumam sensoru tīklu, bet ne visiem

8. Eseja

Kādu sensoru tīklu TinyDB nespētu realizēt?

Termiņš: 30.11.2011. 05:00