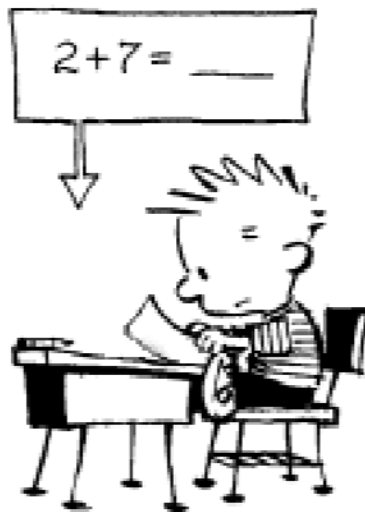


SwissQM: A Virtual Machine for Sensor Networks

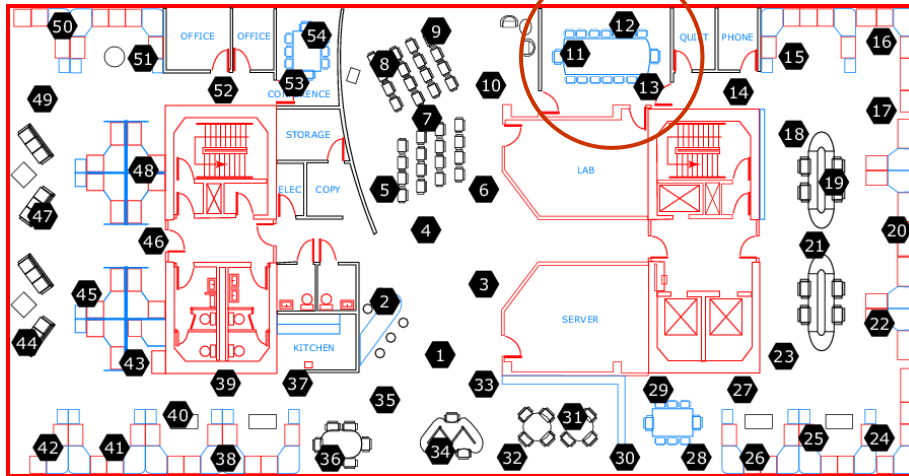
Gustavo Alonso (alonso@inf.ethz.ch)
work with René Müller and Donald Kossmann
Dept. of Computer Science, ETH Zürich, Switzerland



The truth about wireless sensor networks

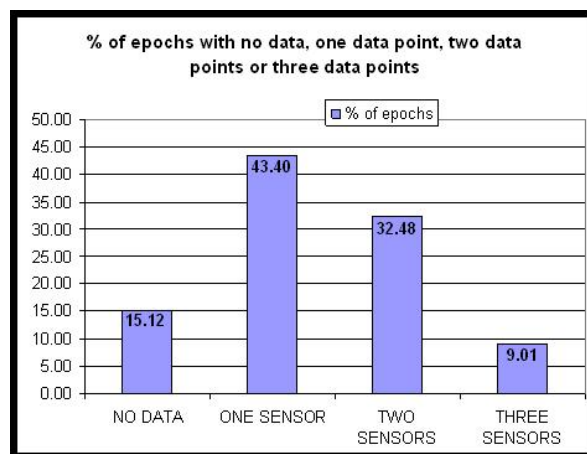


example 1: Intel Lab experiment

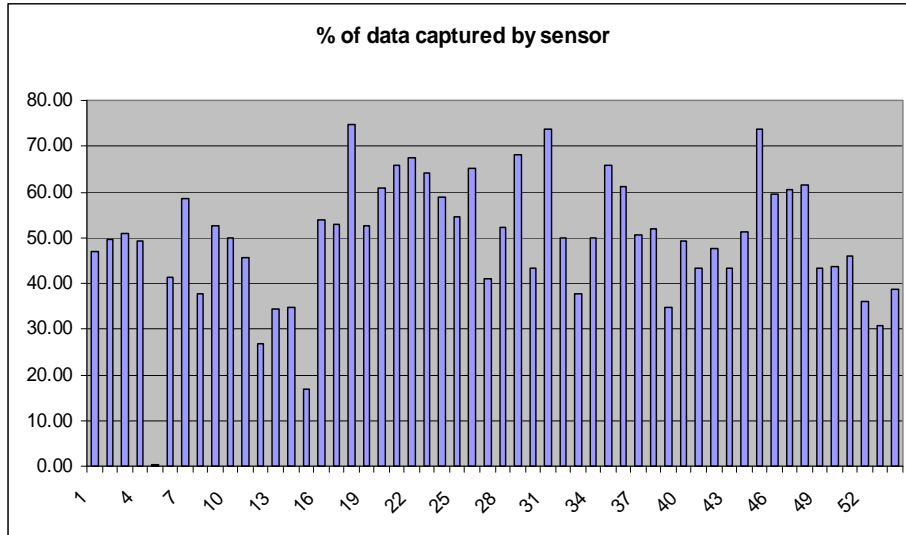


missing data

- sensors (TinyDB) report every 31 s (epoch = 31 s)



lots of missing data



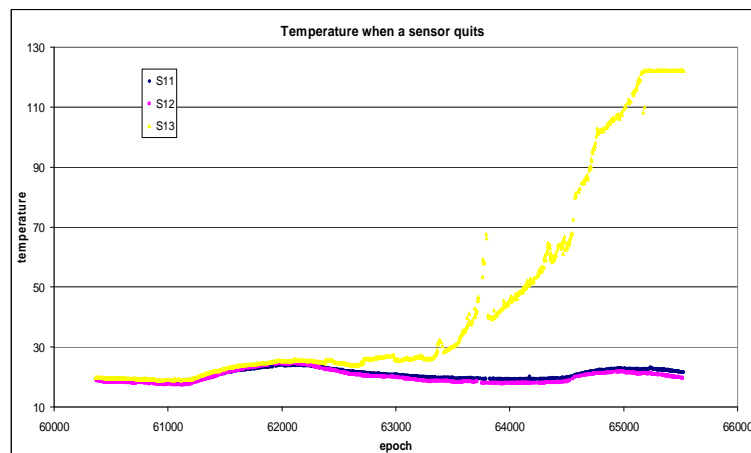
2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

5

fail dirty

- what you read might not be real



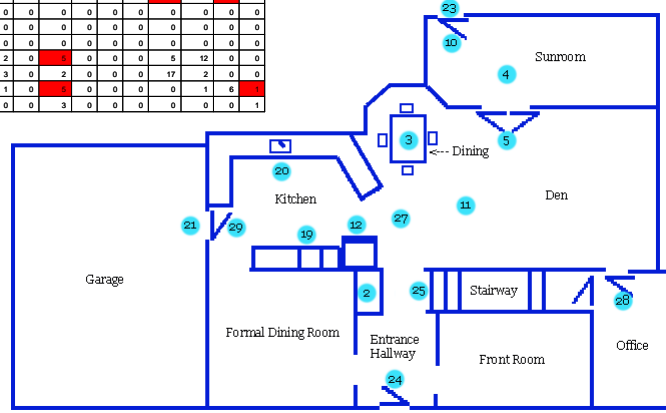
2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

6

example 2: GeorgiaTech home

	0	3	4	5	10	11	12	19	20	21	23	24	25	27	28	29
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	5	1	0	0	0	0	0	0	10	0	0
4	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1
5	0	1	1	3	0	6	0	0	0	0	0	0	0	2	4	0
10	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0
11	0	6	0	8	0	6	5	1	1	0	0	0	1	3	9	0
12	0	4	0	0	0	1	2	2	9	0	0	0	0	0	3	1
19	0	0	0	0	0	1	2	5	0	0	0	0	0	0	0	1
20	0	0	0	1	0	1	7	8	19	0	0	0	0	3	3	2
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	2	0	1	0	1	2	0	19	0	0	0	0	5	12	0
27	0	3	0	0	0	9	3	0	2	0	0	0	0	17	2	0
28	0	0	0	1	0	10	1	0	19	0	0	0	0	0	1	6
29	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1

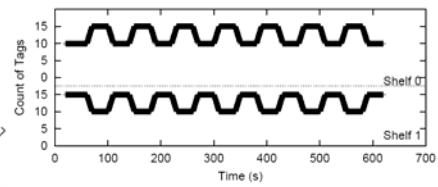
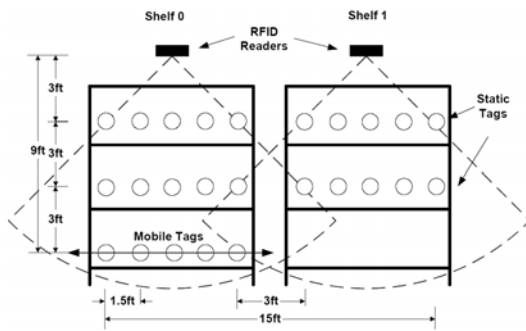


2.2.2007

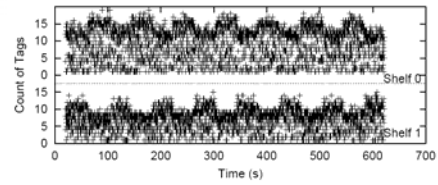
Gustavo Alonso (alonso@inf.ethz.ch)

7

example 3: RFID labels



(a) Reality



(b) Application query results using raw RFID data

2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

8

Except for wired sensor networks connected to the power grid, any type of wireless sensor network will be suffering from these effects:

power restrictions
bandwidth restrictions
limited reliability
system dynamics
impedance mismatch
scalability issues



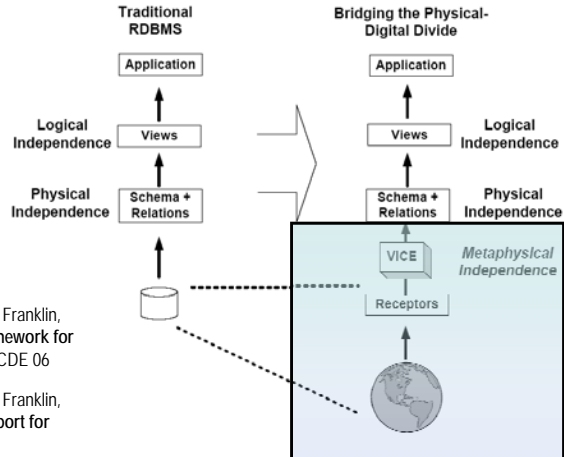
Murphy loves potatoes
K. Langedoen et al.

Research on sensor networks today:



virtual device (VICE)

- A programmable system for coping with the nasty side of WSN

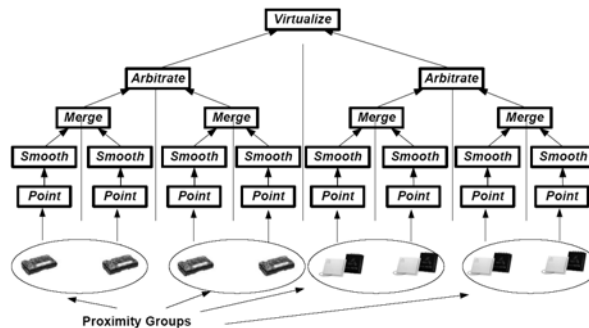


Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, Jennifer Widom: A Pipelined Framework for Online Cleaning of Sensor Data Streams. ICDE 06

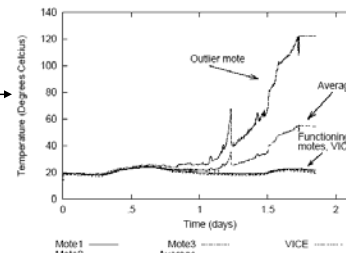
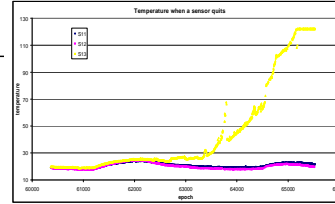
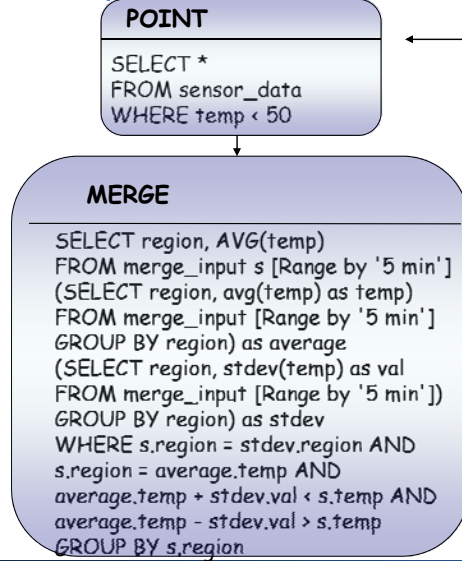
Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, Jennifer Widom: Declarative Support for Sensor Data Cleaning, Pervasive 2006

structure of the VICE

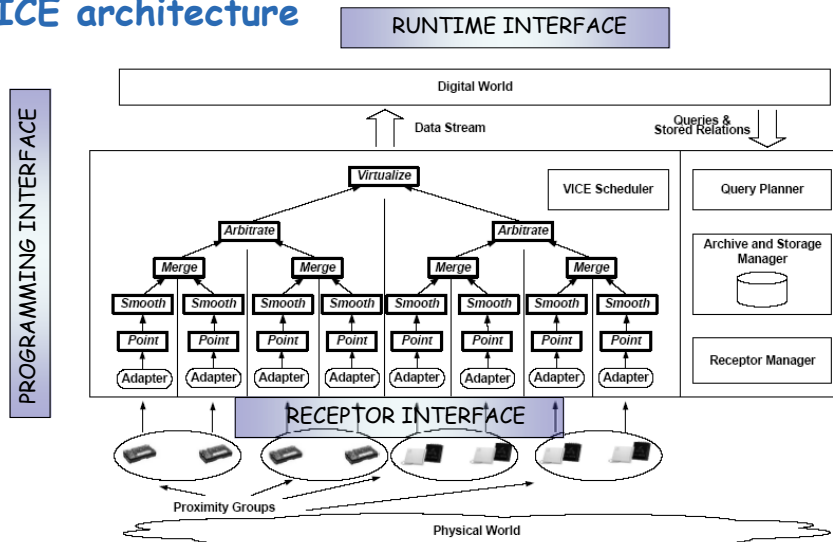
- point: operators on a single value of a stream
- smooth: window operators over a set of values of a stream
- merge: aggregation of data from different streams
- arbitrate: conflict resolution across receptors of the same type
- virtualize: user defined functions operating on arbitrary combinations of streams



example of a VICE



VICE architecture



Many similar examples of efforts to build architectures and systems that can be used to implement and deploy better data acquisition systems based on sensors

But today there is a fundamental limitation:

Few of the tools and infrastructures available for developing real, complex distributed systems are available for sensor networks. Too much complexity is exposed to the developer and user of the data.



Step 1: SwissQM



René Müller, Gustavo Alonso, Donald Kossmann: A Virtual Machine For Sensor Networks. EuroSys 2007

Programming Sensor Networks

- In practice, very low-level
- Operating Systems
 - Contiki, NutOS, TinyOS
- Languages
 - C, nesC, Assembly
- Platform/target specific
- Programming almost directly on hardware
- Cumbersome
- "We work on things you can throw against the wall". [Lars Bak, JavaOne '02]
- Requires very skilled programmer
- Limits functionality because of the cost of development

Right Level of Abstraction?

Example Application: Surge in TinyOS

Nodes send their ID+ temp sensor value every x seconds

```

#include Surge;
#include SurgeCmd;

module SurgeM {
  provides {
    interface StdControl;
  }
  uses {
    interface ADC;
    interface Timer;
    interface Leds;
    interface StdControl as Sounder;
    interface Send;
    interface Receive as Bcast;
    interface RouteControl;
  }
}

implementation {
  enum {
    TIMER_GETADC_COUNT = 1,
    TIMER_CHIRP_COUNT = 10,
  };
  bool sleeping;
  bool focused;
  bool rebroadcast_adc_packet;
  TOS_Msg pMsgBuffer;
  norace uint16_t gSensorData;
  bool gSendBusy;
  int timer_rate;
  int timer_ticks;

  static void initialize() {
    timer_rate = INITIAL_TIMER_RATE;
    atomic gSendBusy = FALSE;
    sleeping = FALSE;
    rebroadcast_adc_packet = FALSE;
    focused = FALSE;
  }

  task void SendData() {
    SurgeMsg *pReading;
    uint16_t len;
    if (pReading = (SurgeMsg *)call
        Send.getBuffer(&gMsgBuffer, &len)) {
      pReading->type = SURGE_TYPE_SENSORREADING;
      pReading->parentaddr = call
        RouteControl.getParent();
      pReading->reading = gSensorData;
      if ((call Send.send(
          &gMsgBuffer, sizeof(SurgeMsg)))
          != SUCCESS)
        atomic gSendBusy = FALSE;
    }

    command result_t StdControl.init() {
      call Leds.init();
      initialize();
      return SUCCESS;
    }

    command result_t StdControl.start() {
      return call Timer.start(TIMER_REPEAT,
          timer_rate);
      return SUCCESS;
    }

    command result_t StdControl.stop() {
      return call Timer.stop();
    }

    event result_t Timer.fired() {
      timer_ticks++;
      if (timer_ticks % TIMER_GETADC_COUNT
          == 0)
        call ADC.getData();
    }

    if (focused && timer_ticks % TIMER_CHIRP_COUNT == 0) {
      call Sounder.start();
      if (focused && timer_ticks % TIMER_CHIRP_COUNT == 1) {
        call Sounder.stop();
        return SUCCESS;
      }
    }

    async event result_t ADC.dataReady(uint16_t data) {
      atomic {
        if (!gSendBusy) {
          SendBusy = TRUE;
          gSensorData = data;
          post SendData();
        }
      }
      return SUCCESS;
    }

    event result_t Send.sendDone(TOS_MsgPtr pMsg,
        result_t success) {
      atomic gSendBusy = FALSE;
      return SUCCESS;
    }

    event TOS_MsgPtr Bcast.receive(
        payload, uint16_t
        SurgeCmdMsg *pCmdMsg = (Sur
        if (pCmdMsg->type == SURGE
            timer_rate = pCmdMsg->rate;
            call Timer.stop();
            call Timer.start(TIMER_RE
            } else if (pCmdMsg->type ==
                sleeping = TRUE;
                call Timer.stop();
                call Leds.greenOff();
                call Leds.yellowOff();
            } else if (pCmdMsg->type == S
                TYPE_MAKEUP) {
  }
}

```

Last 26 lines +
code for Routing
omitted

The declarative approach

- Use a declarative system
 - Queries instead of Programs
 - Query Processing Engine at the sensor nodes
 - Systems
 - TinyDB [Madden]
 - Cougar [Gehrke]
- Surge application as SQL-like query (TinyDB)

```
SELECT nodeid,temp  
SAMPLE PERIOD x s.
```

Appropriate Abstraction Levels?

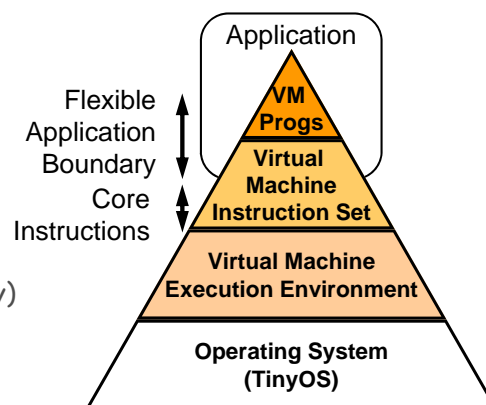
- Low-level Programming
 - Advantages:**
 - Full flexibility
 - Full control of the node
 - Disadvantages:**
 - Cumbersome to program
 - Very platform dependent
 - Dynamic code-updates are expensive (update = binary image)
- Declarative (Queries) High-level Programming
 - Advantages:**
 - Platform independent
 - Declarative (what not how)
 - Queries relatively small → cheap updates
 - Disadvantages:**
 - Limited expressiveness (based on query language after all)

Requirements

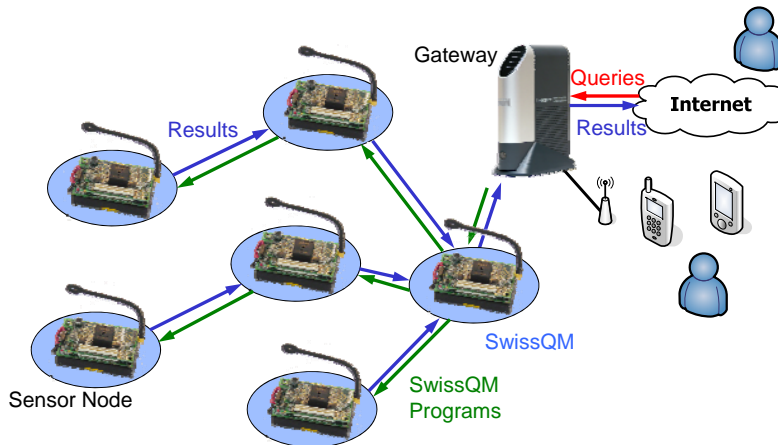
- We want a system that is
 - Language independent (SQL, XQuery, Java, new languages, Webservices ...)
 - Turing complete
 - User-defined functions
 - Capable of pushing down complex processing functions all the way to the sensors → reduction of messages sent in the network
- Solution: Virtual Machine tailored to data acquisition in sensor networks.

Application-specific Virtual Machines

- Virtual Machine on sensor node
- Instruction Set = Elementary Instruction + Application-specific extension
- Functionality can be implemented
 - In bytecode (composability)
 - As additional instruction



SwissQM: Scalable WIreleS Sensor Query Machine

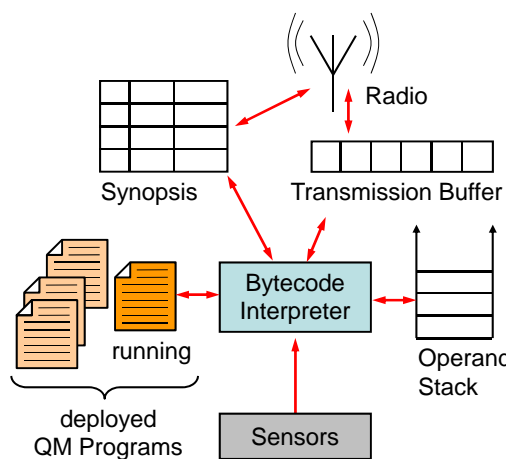


2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

23

SwissQM on the Sensor Node



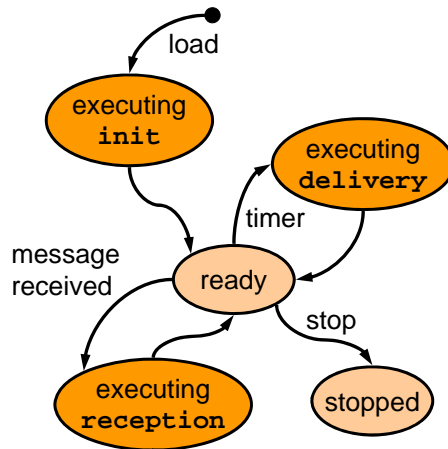
- Stack-based VM
- Integer Arithmetic only
- 59 bytecode instructions
 - 40 = JVM
 - 19 = sensor network specific
- Transmission Buffer
- Synopsis
- Multi-tasking
- Two platforms

2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

24

Code Execution in SwissQM



- Streaming data
 - Some code has to be executed periodically
- QM Program has 3 Code Sections, which are executed when
 - when the program is loaded (**init** section)
 - when a tuple is due (**delivery** section)
 - when a message is received (**reception** section)

Bytecode Instructions

JVM Instructions:

- 16 Stack Instructions
- 11 Arithmetic and Logical Instructions (iadd, isub, iand, ior ...)
- 13 Control Instructions (if_icmple, iflt, ...)

SwissQM "core" Instructions:

- 9 Buffer Instructions (iload, istore, iload_sy, ...)
- 7 Sensor Instruction (get_temp, get_light, ...)
- 2 Transmission Instructions (send, send_sy)

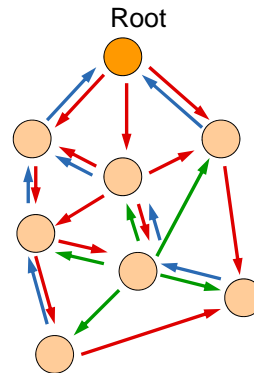
Application-specific:

- 1 In-network Aggregation (merge)

SwissQM - Messaging in the Network

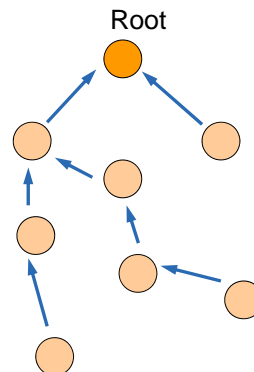
- Message Types:
 - Program Messages
 - Result Messages
 - Command Messages (Node Reset, Program Stop,...)
- Traffic Pattern
 - "Root to all": Program and Command Messages
 - "All to Root": Result Messages
 - "Local Broadcasts": Routing updates and time synchronisation

multihop



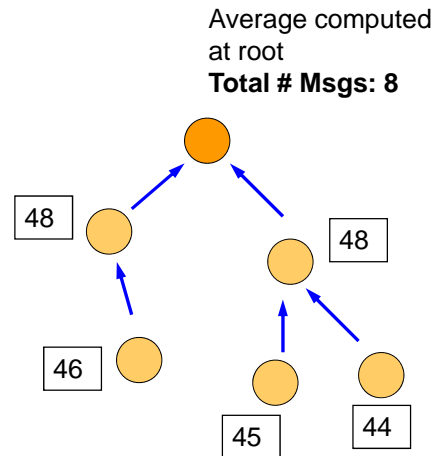
Routing of Result Messages

- **Spanning Tree** for routing of Result Messages
- Each node knows has a parent node
- Based on TinyOS' **MintRoute** (to be changed)
- Time synchronisation piggy-backed on routing messages
- Tree is also used for in-network aggregation



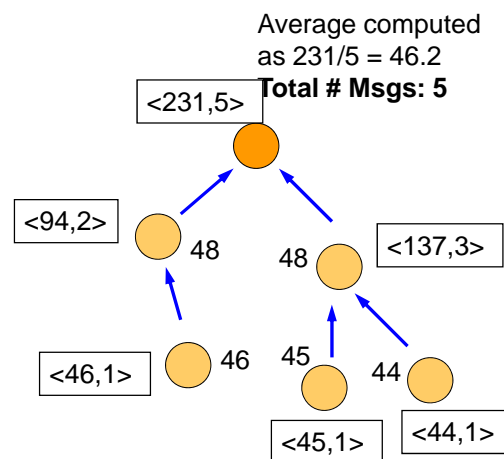
In-network Aggregation

- In-network aggregation can *reduce* the number of *messages* (save energy)
- TAG Approach: Tiny AGgregation [Madden OSDI'02]
- Example: *What is the average temperature in the network?*



In-network Aggregation

- State of AVERAGE aggregation
 - *Sum*
 - *Count*
- In-network aggregation
 - Tree topology
 - Only one message per link
 - Children send data before parent (Scheduling/Time sync)



Merge Instruction

- Merge aggregation data from the transfer buffer with local synopsis
- General Layout
 - n Grouping Expressions
 - m Aggregate Expressions
- Example


```
SELECT parent,
MAX(light), MIN(light),
AVG(temp)
GROUP BY parent
```
- Aggregation State Record


```
<parent, max_light,
min_light, <sum_temp
count_temp> >
```

n = 1, m = 3
- merge Function


```
merge(1, 3, AVG, MIN, MAX)
```

 - Arguments Pushed on stack
 - Merge function is invoked as merge instruction.

Example: TinyDB Query (No Aggregation)

- ```
SELECT nodeid, light+temp
FROM sensors
SAMPLE PERIOD 30s
```
  - Code of compiled query is executed when
    - **Timer fired** → Tuple is due
    - **Message is received** → message (in transmission buffer) is forwarded
    - No reception section → implicit forwarding
    - Program size: 10 bytes
- ```
.section delivery, "@30s"
get_nodeid
istore 0
get_light
get_temp
iadd
istore 1
send_tb

.section reception
send_tb # forward
```
-

Example: TinyDB Aggregation Query

- `SELECT MAX(temp)`
`FROM sensors`
`WHERE nodeid>15`
`SAMPLE PERIOD 4s`
- Code of compiled query is executed when
 - **Timer fired** → Tuple is due
 - **Message is received** → Aggregation data needs to be merged
 - Program size: 17 bytes

```
.section delivery, "@4s"
  get_nodeid
  ipushb 15
  if_icmple skip
  get_temp # local data
  istore 0
  iconst_2 # MAX agg.
  iconst_1 # num aggs
  iconst_0 # num groups
  merge
  skip: send_sy

.section reception
  iconst_2 # MAX agg.
  iconst_1 # num aggs
  iconst_0 # num groups
  merge
```

Example: Program Execution

Query:
`SELECT MAX(light) FROM sensors`
`SAMPLE PERIOD 10s`

SwissQM Program:
`.section init`
→ `iconst_0`
`istore_sy 0`

`.section reception`
→ `iload_sy 0`
→ `iload 0`
→ `dup2`
→ `if_icmple 12`
`swap`
`12:istore_sy 0`
→ `pop`



Event:

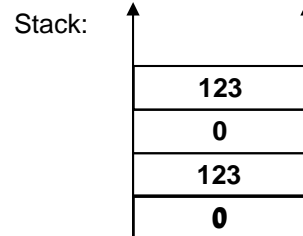
program execution done

Transmission Buffer:

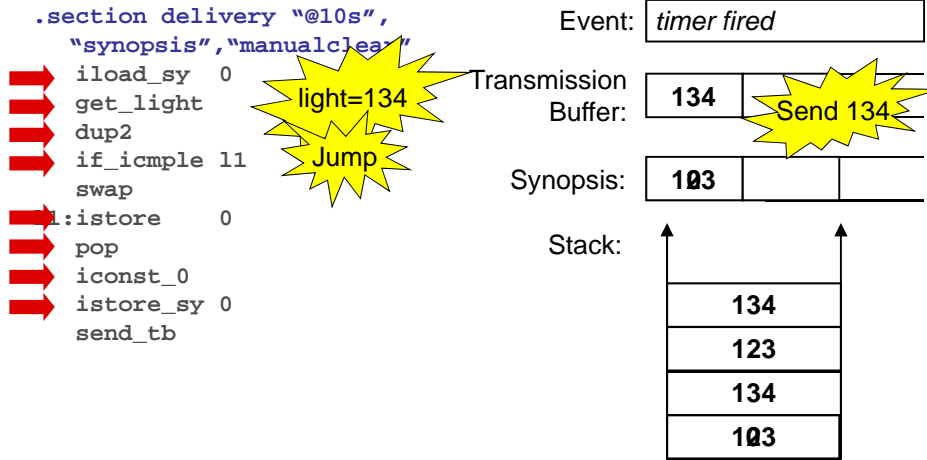
123		
-----	--	--

Synopsis:

103		
-----	--	--



Example: Program Execution (2)



Step 2: Gateway Infrastructure



René Müller, Gustavo Alonso: **Efficient Sharing of Sensor Networks**. IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS) 2006.

René Müller, Gustavo Alonso, Donald Kossman: **SwissQM: Next Generation Data Processing in Sensor Networks**. CIDR 2007

Example: User-defined Function

- Exponential Weighted Moving Average Filter (EWMA)
- $y_k = \alpha y_{k-1} + (1 - \alpha) u_k$
- Implementation of UDF in C-like language

```
int ewma(int u,int alpha) {
    static int y1 = 0;
    int y;
    y = (alpha*y1+
        (10-alpha)*u)/10;
    y1 = y;
    return y;
}
```

State allocated on Synopsis

- E.g., used in SQL query

```
SELECT nodeid, ewma(light, 8)
FROM sensors SAMPLE PERIOD 4s
```

- UDFs are weaved into bytecode program (inlined)
- UDF can keep state
 - Static variables in UDF
 - State is stored in Synopsis buffer

Example: User-defined Function (Code)

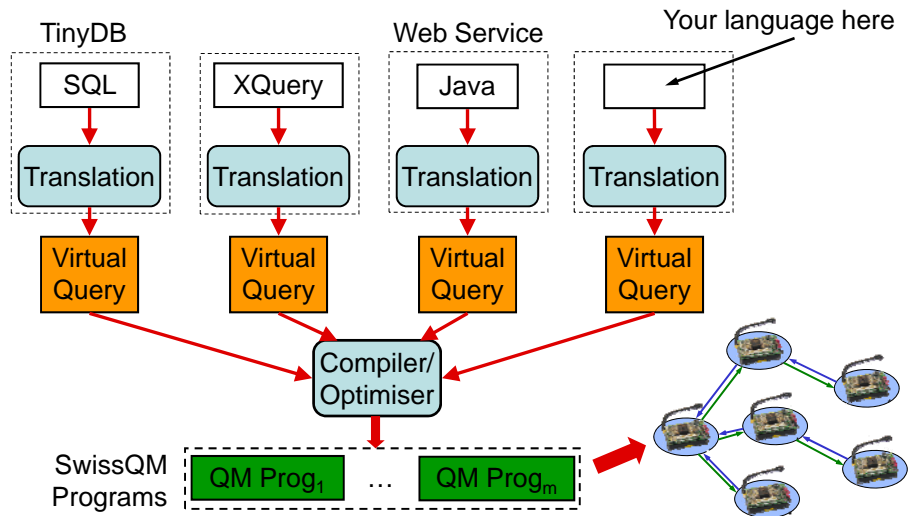
```
.section init
    iconst_0
    istore_sy 0 # y_0 = 0

.section delivery "@4s",
    "synopsis","manualclear"
    get_light # sample u_k
    iload_sy 0 # load y_{k-1}
    isub # u_k - y_{k-1}
    ipushb 8
    idiv # (u_k - y_{k-1}) / 8
    iload_sy 0 # load y_{k-1}
    iadd # y_{k-1} + (u_k - y_{k-1}) / 8
    dup # duplicate y_k
    istore_sy 0 # y_{k-1} ← y_k
    istore 1 # store y_k

    get_nodeid
    istore 0 # store id
    sendtb
```

- Program size: 22 bytes
- Dissemination: 2 Messages

SwissQM + Gateway System



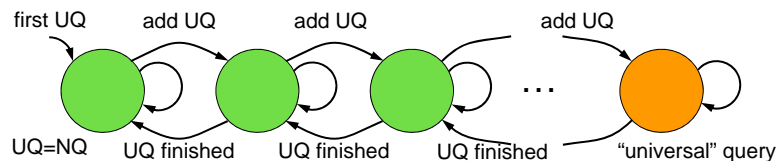
2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

39

Query Merging - Universal Query [MASS2006]

- Universal Network Query (NQ) is able to answer **all** user queries (UQ)
- `SELECT *`
`FROM sensors`
`SAMPLE PERIOD <as short as possible>`
- But too expensive (energy consumption, messages)
- **Idea:** Merge all UQs into a single NQ, but keep it *as specific as possible*.

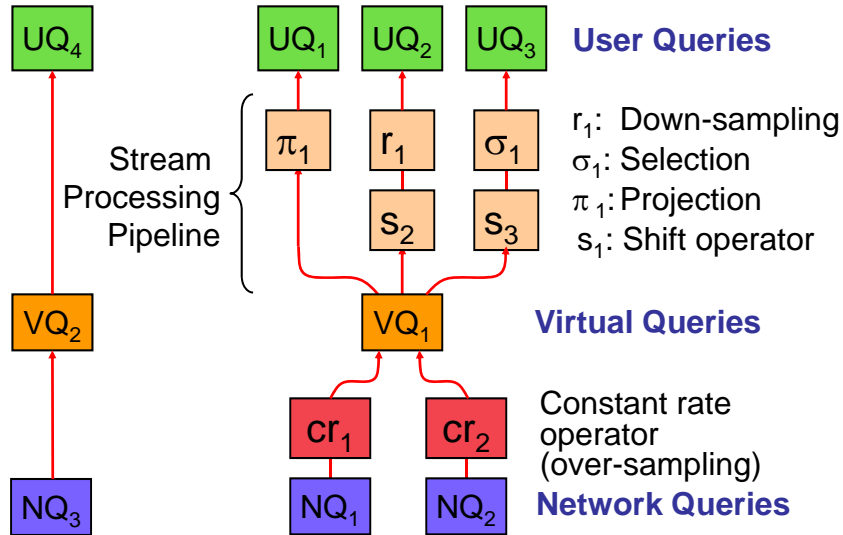


2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

40

Query Merging and Result Processing



2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

41

Example

Set of Queries (arriving in order):

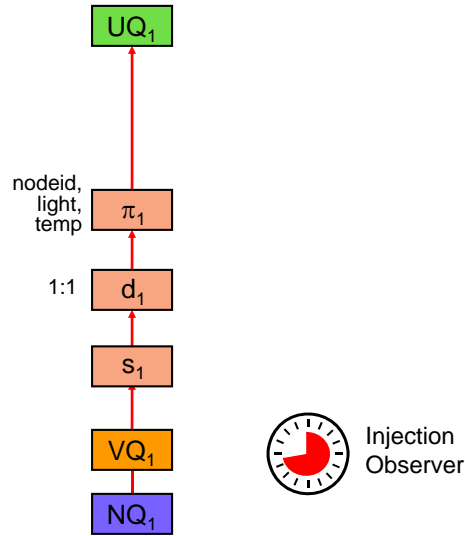
1. `SELECT nodeid, light, temp FROM sensors WHERE nodeid=3 SAMPLE PERIOD 15000`
2. `SELECT nodeid, light, temp FROM sensors WHERE nodeid=3 SAMPLE PERIOD 60000`
3. `SELECT light FROM sensors WHERE nodeid=3 SAMPLE PERIOD 5000`
4. `SELECT nodeid, light FROM sensors SAMPLE PERIOD 5000`

2.2.2007

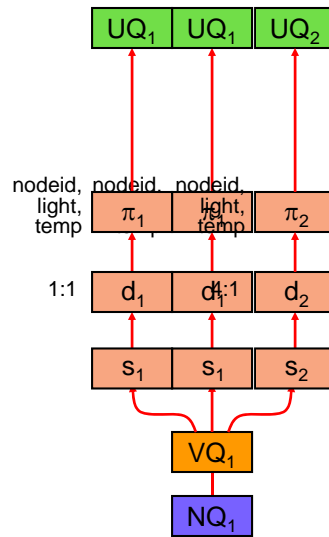
Gustavo Alonso (alonso@inf.ethz.ch)

42

UQ₁: SELECT nodeid, light, temp FROM sensors WHERE nodeid=3 SAMPLE PERIOD 15000

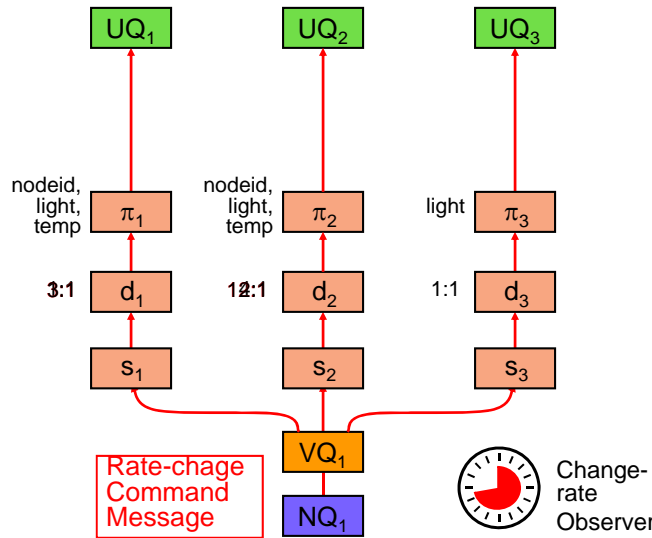


UQ₂: SELECT nodeid, light, temp FROM sensors WHERE nodeid=3 SAMPLE PERIOD 60000 ... 15000



... 15000

UQ₃: SELECT light FROM sensors WHERE nodeid=3 SAMPLE PERIOD 5000



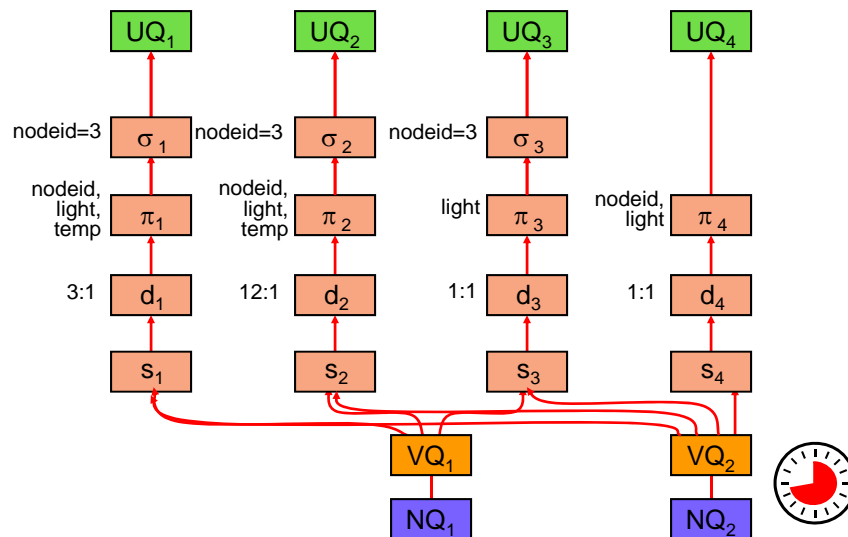
2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

45

... WHERE nodeid=3 ...

UQ₄: SELECT nodeid, light FROM sensors SAMPLE PERIOD 5000



2.2.2007

Gustavo Alonso (alonso@inf.ethz.ch)

46

Conclusions

- Sensor networks need better programming platforms
- SwissQM is the means to an end (automatic adaptation, optimisation, complex algorithms,...)
- Increases abstraction level at the network interface
- Powerful instruction set → short programs → eases dissemination
- Future Sensors?
 - May have more memory and CPU power
 - But radio bandwidth and reliability still an issue
 - Cost-efficiency
 - The basic problems will not change

Try it yourself

Download SwissQM at
<http://swissqm.inf.ethz.ch>

SwissQM