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

2 + 7 = ___
example 1: Intel Lab experiment

Example of an Intel Lab experiment map. The map shows the layout of the laboratory with various sensors labeled.

missing data

- sensors (TinyDB) report every 31 s (epoch = 31 s)
lots of missing data

% of data captured by sensor

fail dirty

- what you read might not be real
example 2: GeorgiaTech home

example 3: RFID labels

(a) Reality

(b) Application query results using raw RFID data
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

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

POINT

```
SELECT * 
FROM sensor_data 
WHERE temp < 50 
```

MERGE

```
SELECT region, AVG(temp) 
FROM merge_input [Range by '5 min'] 
(SELECT region, avg(temp) as temp) 
GROUP BY region) as average 
(SELECT region, stdev(temp) as val 
FROM merge_input [Range by '5 min']) 
GROUP BY region) as stdev 
WHERE s.region = stdev.region AND 
s.region = average.temp AND 
average.temp + stdev.val < s.temp AND 
average.temp - stdev.val > s.temp 
GROUP BY s.region 
```

VICE architecture

**RUNTIME INTERFACE**

**PROGRAMMING INTERFACE**

**RECEPTOR INTERFACE**

- Digital World
- Data Stream
- Query Planner
- Archive and Storage Manager
- Receptor Manager
- Proximity Groups
- Physical World
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

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

```c
#include Surge; includes Surge; module SurgeM { provides { interface StdControl; } uses { interface ADC; interface Timer; interface Serial as Receive; interface Socket as ROUTEMENT; } implementation { enum { TIMER_GETADC_COUNT = 1, TIMER_CHIRP_COUNT = 10, }; bool sleeping; bool focused; bool rebroadcast_adc_packet; TOS_Msg gMsgBuffer; norace uint16_t gSensorData; bool gfSendBusy; int timer_rate; int timer_ticks; static void initialize() { timer_rate = INITIAL_TIMER_RATE; atomic gfSendBusy = FALSE; sleeping = FALSE; rebroadcast_adc_packet = FALSE; focused = FALSE; } } } 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 (!gfSendBusy) { SendBusy = TRUE; gSensorData = data; post SendData(); } } return SUCCESS; } event result_t Send.sendDone(TOS_MsgPtr pMsg, result_t success) { atomic gfSendBusy = FALSE; } event TOS_MsgPtr Bcast.receive(TOS_MsgPtr pMsg, void* payload, uint16_t payloadLen) { SurgeCmdMsg *pCmdMsg = (SurgeCmdMsg *)payload; if (pCmdMsg->type == SURGE_TYPE_SETRATE) { timer_rate = pCmdMsg->args.newrate; call Timer.stop(); call Timer.start(TIMER_REPEAT, timer_rate); } else if (pCmdMsg->type == SURGE_TYPE_SLEEP) { sleeping = TRUE; call Timer.stop(); call Leds.greenOff(); call Leds.yellowOff(); } else if (pCmdMsg->type == SURGE_TYPE_WAKEUP) { } }
```

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

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
**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

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?

Average computed at root
Total # Msgs: 8

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)

Average computed as $231/5 = 46.2$
Total # Msgs: 5
Merge Instruction

- Merge aggregation data from the transfer buffer with local synopsis
- General Layout
  - n Grouping Expressions
  - m Aggregate Expressions
- Example
  
```sql
SELECT parent,
MAX(light), MIN(light),
AVG(temp)
GROUP BY parent
```

- Aggregation State Record
  
```sql
<parent, max_light,
min_light, <sum_temp
count_temp> >
n = 1, m = 3
```

- merge Function
  - 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
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

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
```

Event: **tuple reception done**

Transmission Buffer: 123

Synopsis: 103

Stack:
```
123
0
123
0
```
Example: Program Execution (2)

```
.section delivery "@10s",
        "synopsis","manualclear"
.iload_sy 0
.get_light
.dup2
.if_icmple l1
.swap
:l1:istore    0
.pop
.iconst_0
.istore_sy 0
.send_tb
```

Event: timer fired
Transmission Buffer: 134
Synopsis: 103
Stack:

Step 2: Gateway Infrastructure


René Müller, Gustavo Alonso, Donald Kossmann: 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

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

- E.g., used in SQL query
  ```sql
  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)

```assembly
.section init
iconst_0
istore_sy 0  # y0 = 0

.section delivery "@4s", "synopsis", "manualclear"
get_light  # sample uk
iload_sy  0  # load yk-1
isub     # u_k-yk-1
ipushb 8  # (u_k-y_k-1)/8
idiv
iload_sy  0  # load yk-1
iadd     # y_k-1+(u_k-y_k-1)/8
dup
istore_sy 0  # y_k-1 \leftarrow y_k
istore 1  # store yk
```

- Program size: 22 bytes
- Dissemination: 2 Messages
SwissQM + Gateway System

TinyDB
- SQL Translation
- Virtual Query

Web Service
- XQuery Translation
- Virtual Query

Compiler/ Optimiser

Your language here

Virtual Query

TinyDB

Web Service

Virtual Query

Virtual Query

Virtual Query

Compiler/ Optimiser

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.

first UQ

add UQ

add UQ

...
Query Merging and Result Processing

User Queries
- \( r_1 \): Down-sampling
- \( \sigma_1 \): Selection
- \( \pi_1 \): Projection
- \( s_1 \): Shift operator

Virtual Queries
- Constant rate operator (over-sampling)

Network Queries

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
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
UQ₁: SELECT light FROM sensors WHERE nodeid=3 SAMPLE PERIOD 5000

UQ₂: SELECT nodeid, light FROM sensors WHERE nodeid=3 SAMPLE PERIOD 5000
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