μSETL: A Set Based Programming Abstraction for Wireless Sensor Networks

Author: Mohammad Sajjad Hossain, A.B.M. Alim Al Islam, Milind Kulkarni, Vijay Raghunathan

By:Yi Shao

Background

- Abstraction: programming specification to tell you what to do but not how to do.
- Microprogramming: node level programming, low level of abstraction complex, difficult to express network level function
- Macroprogramming: network-centric view, specifying node group, complexity hidden, diffictult to express node level interaction
- Set-based programming: compactness,clarity and readability, loss of efficiency, lower speed

Content Overview

- µSETL Expressing example
- Language specification
  - Data types
  - Feature constructor
- Implementation
  - System Architecture
  - Levels of Components
    - µSETL compiler
    - Run-Time Environment
- Evaluation
- Conclusion

µSETL Example

Idiom	*µ*SETL expression
Nodes with even IDs	`{N(i) | i % 2 == 0}`
Neighbors of the current node	`{i | distance(i) == 1}`
Temperature data that are greater than *x*	`{N(i).temp | i IN all, N(i).temp > x}`

- Collection of event handlers
- High level expression
- View of individual node
- Applicable to common behavior

Data Types

- Node
  - Attributes : location, identifier, temperature, light
  - Example: N(i).light
- Set
  - Example:

novolatile {2, 3, 4}	(Non-volatile) set of integers
{N(2), N(5), N(7)}	Set of nodes
{i | distance(i, x) == 1}	One hop neighborhood of *x*
{N(i).light | iINy}	Light data of the nodes in set *y*

- Other types: integer, float, char and string

Constructs

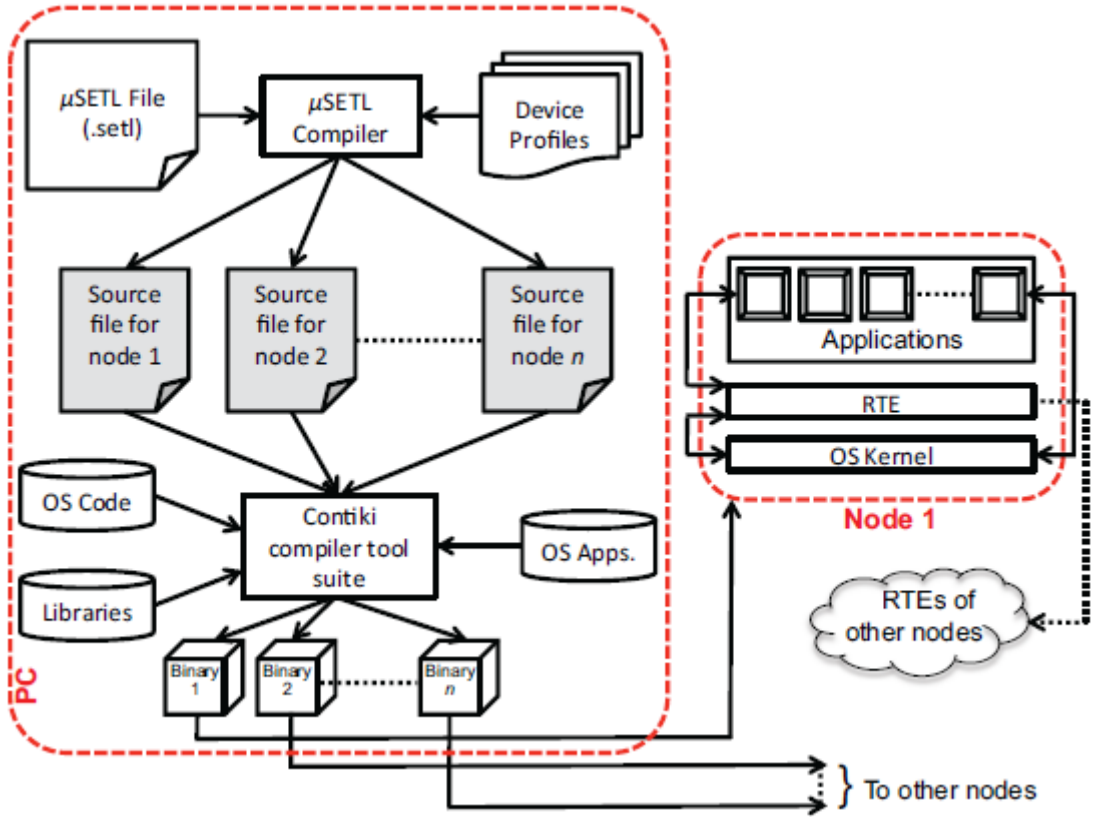
- Periodic Block: event handler with time triggered

  Example: period 5000 do
  ……..(body)
  end

- Monitor Block: event handler with state triggered

  Example: monitor neighborhood, xyz do
  ……(body)
  end

System Architecture



An overview of the µSETL architecture.

µSETL compiler

-----from abstraction to C code

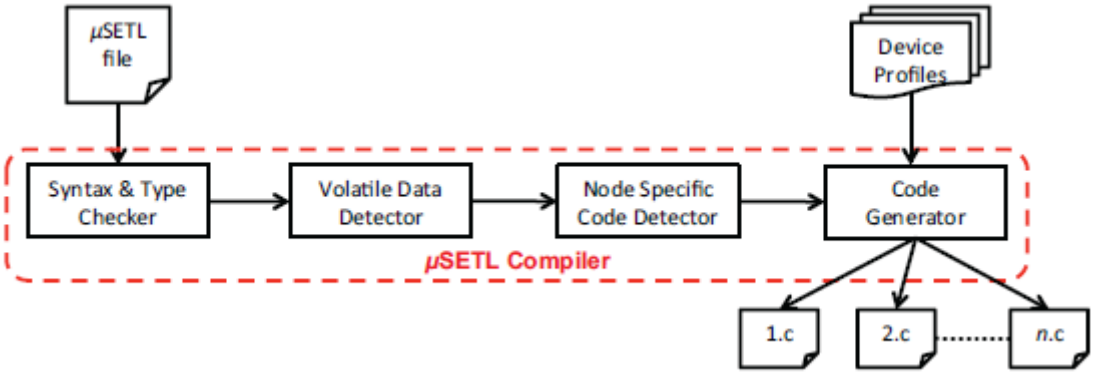
Contiki

-----from C code to binary

RTE

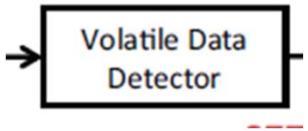
-----communication, data collection to application

Components



- Emits node-specific source code
- Volatile data detection
- Combine with device information

Volatile Data Detector



To detect volatile variables

- Variable is volatile if
  - Members in set or other variable refers to attributes changing In time(distance of node; sensor reading)
  - Variable has dependency on other volatile variables

Types of volatile data

- Distance information
- Resource data
- Data received using receive()

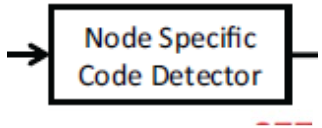
```
neighbors := {i | distance(i) == 1}
```

```
temp := {N(i).temp | i IN all}
```

```
received := {i:i8 | receive(base_station, i)}
```

µSETL Compile:

Node Specific Code Detector



To detect code for target node

- Node specific by:
  - Name target node for a code
  - By conditional statements
  - Quasi-constants

```
@base_station@ #
    ......
#
```

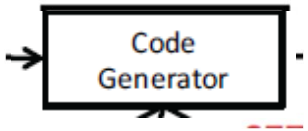
if-then-else..

if N(node_id).has(CAMERA)

- Improve run-time efficiency
- Allow code reuse
- Decrease binary size
- Save consumption in nodes(cpu, memory etc.)
- Save power

µSETL Compile:

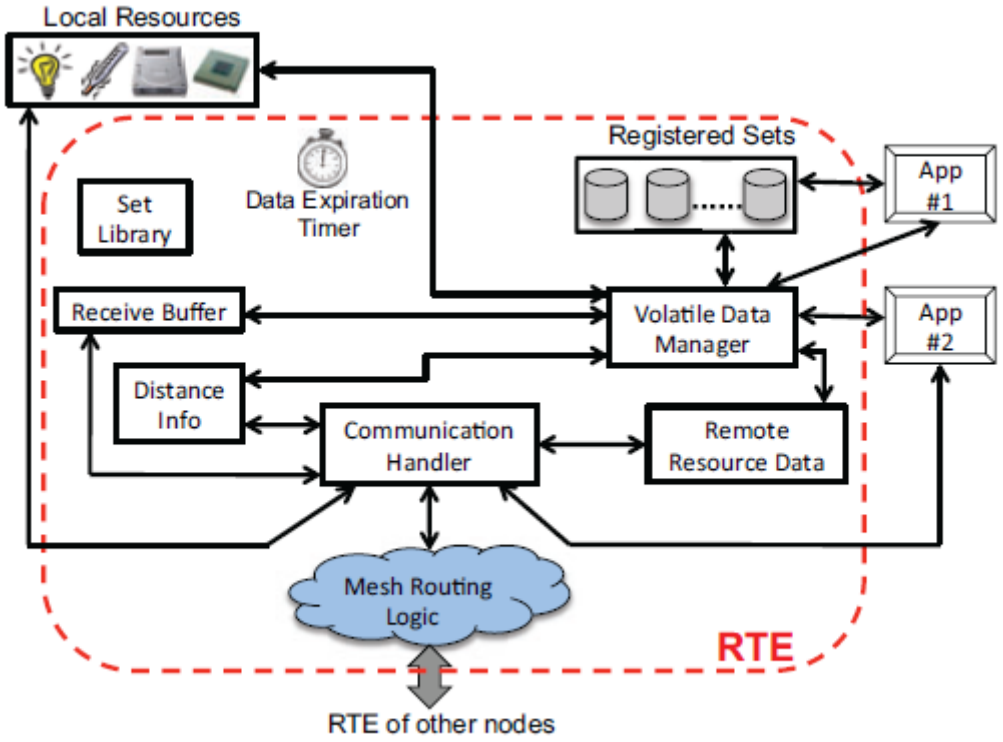
Code Generator



To generate C code for Contiki operating system

- Task performed
  - Code optimization: delete empty loop
    example: period 4000 do…..
  - Calculating a variable : complicated detection overheads
    example: A-→B-→C-→D-→E-→….

Run Time Environment(RTE)



Support for μSETL program

- ----communication among nodes
- ----sensing remote data
- ----set operation
- ----avoid packet collision
- ----update volatile variable

The run-time environment (RTE) in μSETL.

Volatile Data Manager

VDM tasks

- Set registration: during application initialization
- Data update procedure
  - Duration time for volatile data
  - Distance information: ping request <---->ping reply
  - Resource data: resource request <---->resource reply
  - Receive(): waiting for send()
- ➢ Avoid delay and synchronization
- Data consistency
  - No shared variables
  - Wrong loop conditioned by volatile variables

Case1

Modified Surge

Surge: a data collection where base station periodically gather data

```
1  period 4000 do
2    send(base_station, N(node_id).temp);
3  end
4
5
6
```

(a) *Distributed Surge*

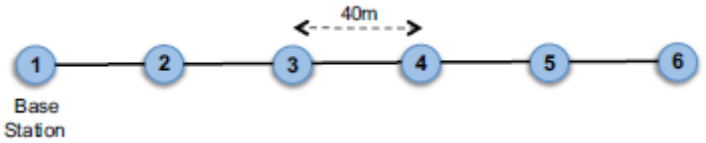
```
set_param(TS_PERIOD, 4000);
temp := {N(i).temp | i IN all};
monitor temp do
  print(temp);
end
```

(b) *Centralized Surge*

```
neighbors := {i | distance(i) == 1};
temp := {N(i).temp | i IN neighbors};
period 4000 do
  avg:i8 := average(temp);
  send(base_station, avg);
end
```

(c) *Modified Surge*

μSETL code for different versions of Surge.

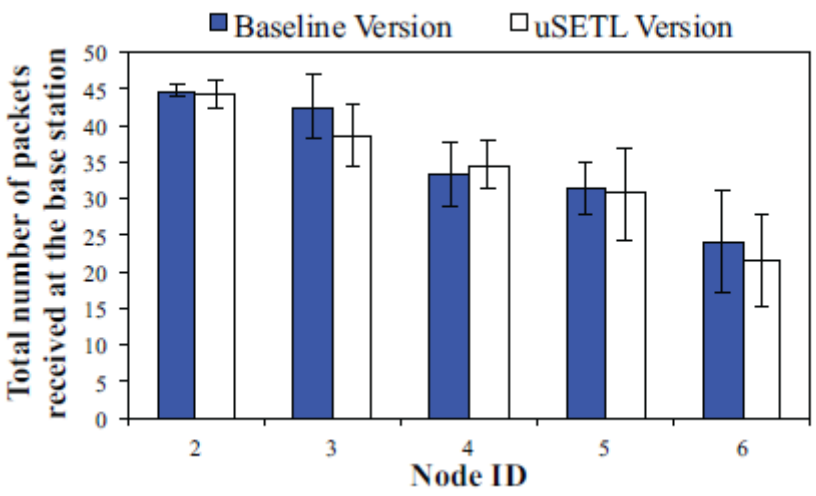


Topology used for *Modified Surge*. Nodes had a transmission range of 50m and an interference range of 100m.

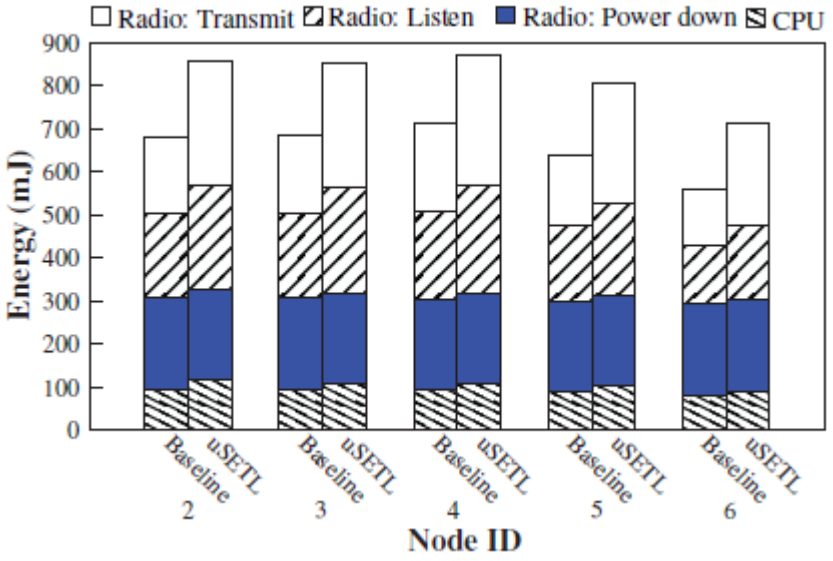
Program	Code size (Approximate # of lines)	
	Baseline Version	μSETL Program
Modified Surge	195	6

Result: concise code
less RAM needed

Packet amount & Energy Cost



Total number of data packets received at the base station for *Modified Surge*.



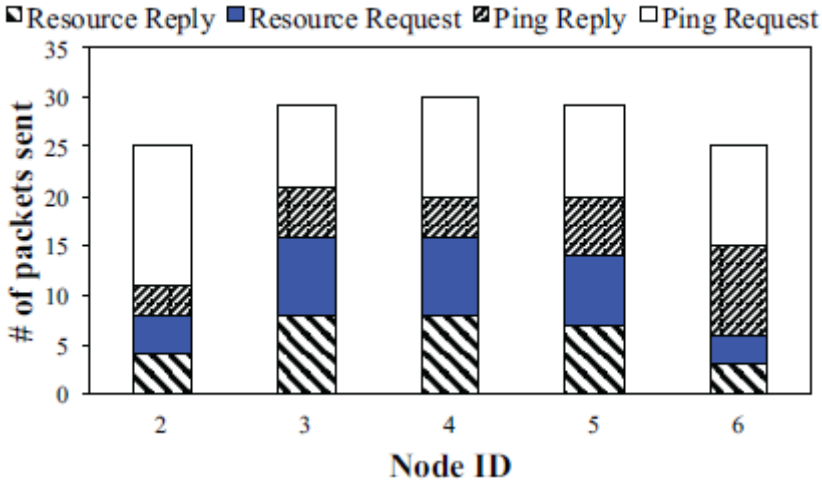
Energy consumed by the nodes for *Modified Surge*.

Result: node further from base station have more packet loss

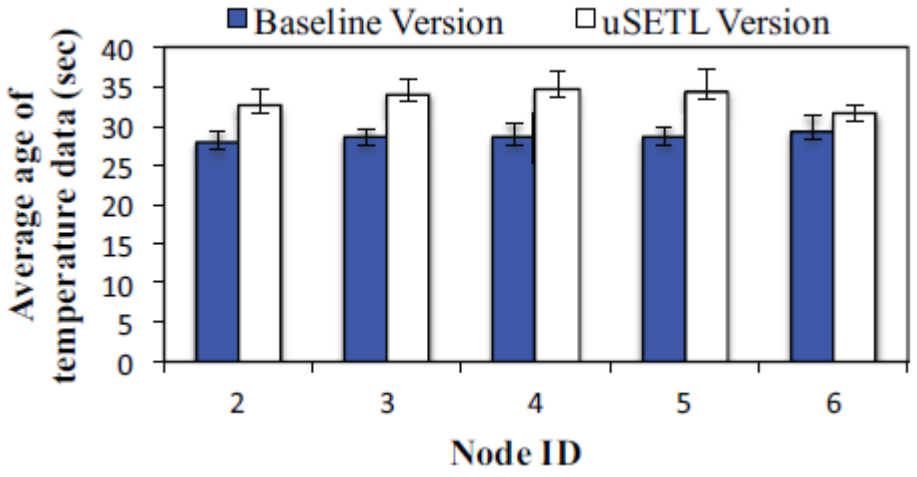
Result: µSETL consumes slightly more energy due to overhead in RTE

Modified Surge

Update Packets & Time efficiency



Number of non-data packets sent by the RTE on each node for *Modified Surge*.



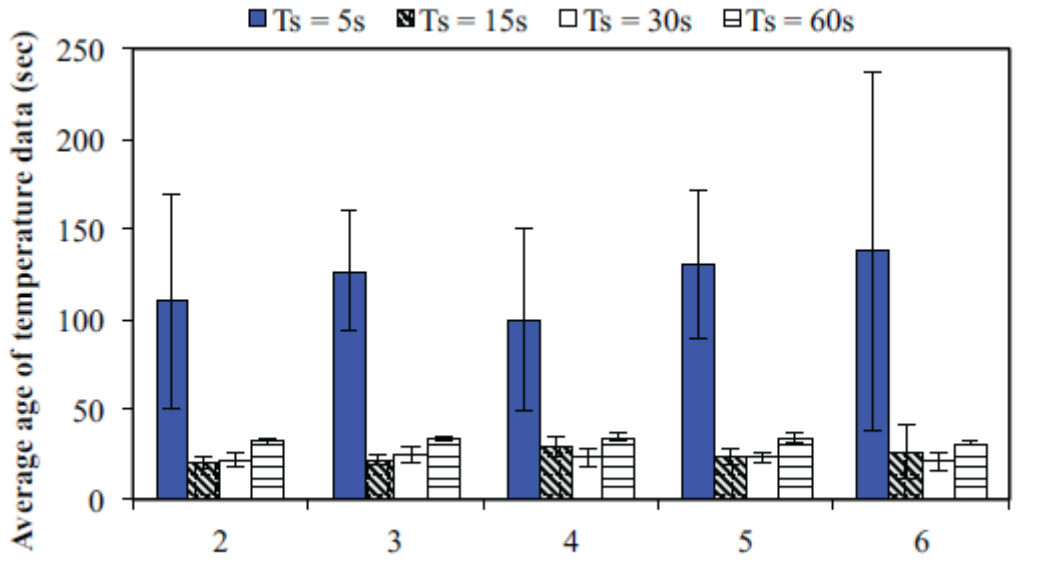
Comparison of the age of temperature data between the *μ*SETL and the baseline versions of *Modified Surge*.

Result: control packets to communicate among nodes updating data in 200 seconds

Result: μSETL is older version data less efficiency for set operation

Modified Surge

Data Consistency

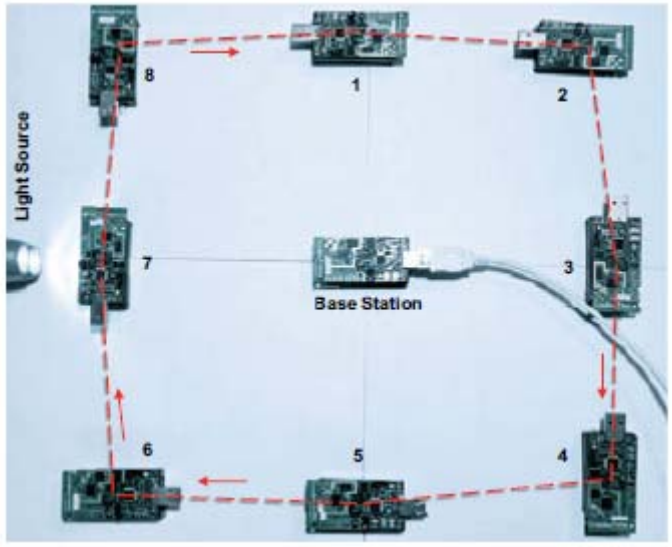


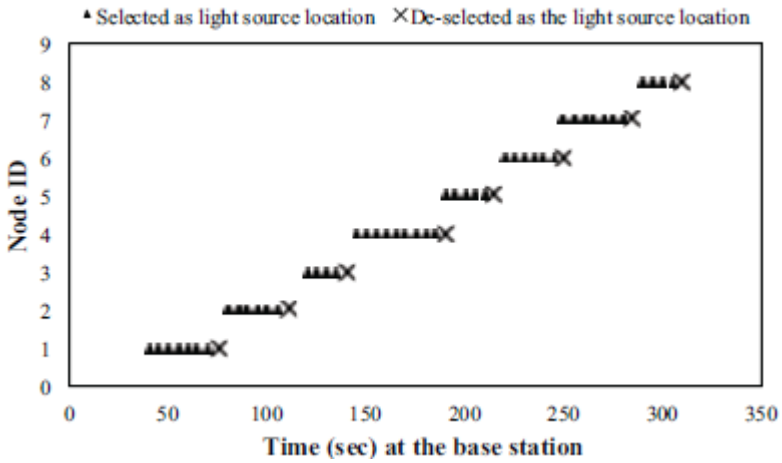
Effect of T_s on the age of temperature data for *Modified Surge*

Result: the shorter data life time, the fresher data, more update control packet, more collision.

Case2

Object Tracking





- Eliminating extra code, save resource by node specific
- Avoid control overhead by monitor block

Conclusion

- µSETL is a set based abstraction for WSN microprogramming
- Set theory provides macro level simplicity, easy for programmer
- It decrease code size efficiently more than 90%