Eon: A Language and Runtime System for Perpetual Systems

Thomas Mayer

Seminar:
Ad-Hoc Networks

Final Presentation: 05. 08.08

Overview

- Introduction
  - Design concepts
- The Language
  - Basics
  - Syntax
  - Runtime / Compiler
- Evaluation
  - Deployment
  - Usability / Performance study

Overview

- Introduction
  - Design concepts
- The Language
  - Basics
  - Syntax
  - Runtime / Compiler
- Evaluation
  - Deployment
  - Usability / Performance study

Motivation

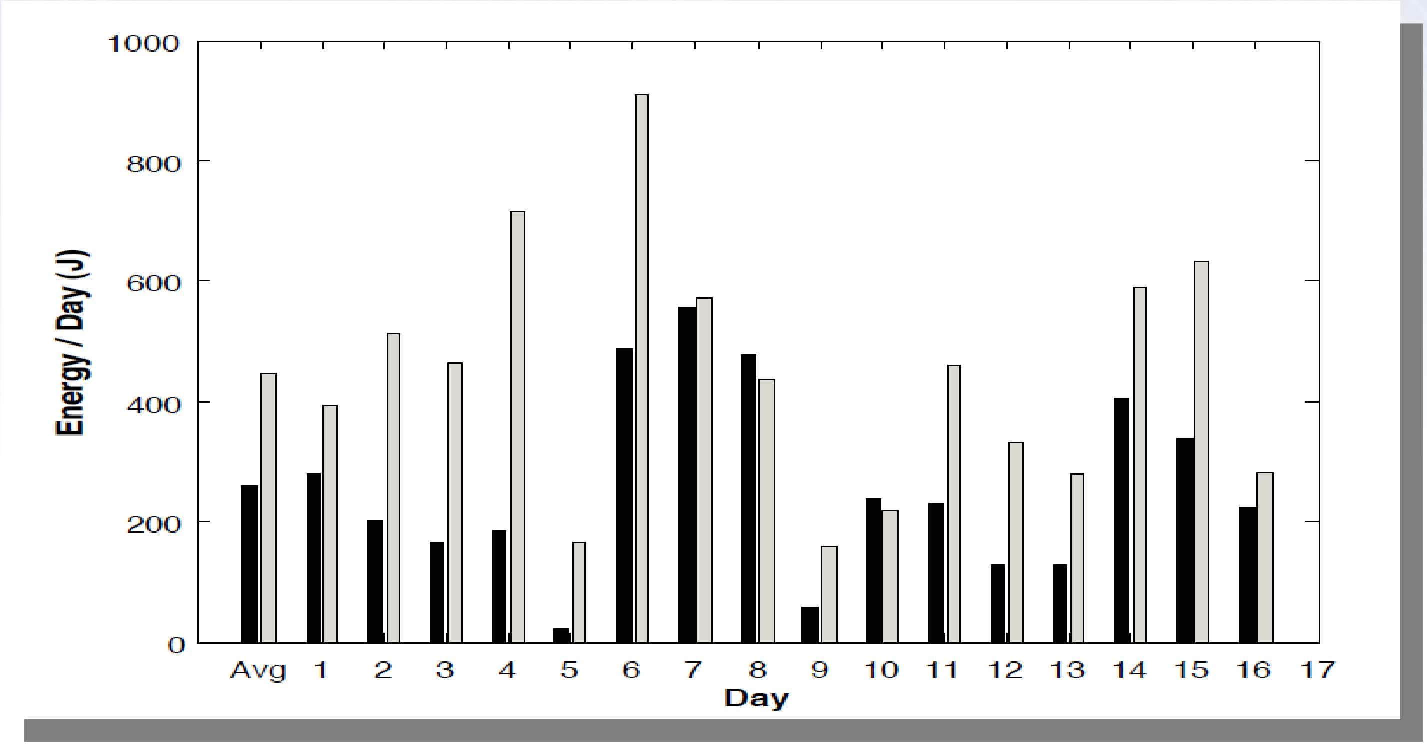
- Sensors everywhere
- Sensors with long lifetime needed
  - For long term experiments
  - Less maintainance
  - ...
- Solution:
  - Larger battery ??
  - Energy efficient programming for perpetual systems

Motivation (2)

- Perpetual system
  - Harvests energy from environment
  - Tries to survive without deadtimes
- Adaptive system
  - Dynamic energy availability
  - Varying energy costs
  - Heterogeneous hardware platforms

Motivation (3)

- Basically 2 problems for the system
  - Predicting weather (runtime)
  - Reacting appropriately (program)



Design concepts

- Energy aware programming language
  - Dynamic reactions
- Abstract energy states
  - Abstraction from hardware
- Meta language
  - Reuse of existing code
- Controll language
- Ease of use more important than complexity

Overview

- Introduction
  - Design concepts
- The Language
  - Basics
  - Syntax
  - Runtime / Compiler
- Evaluation
  - Deployment
  - Usability / Performance study

Basic idea

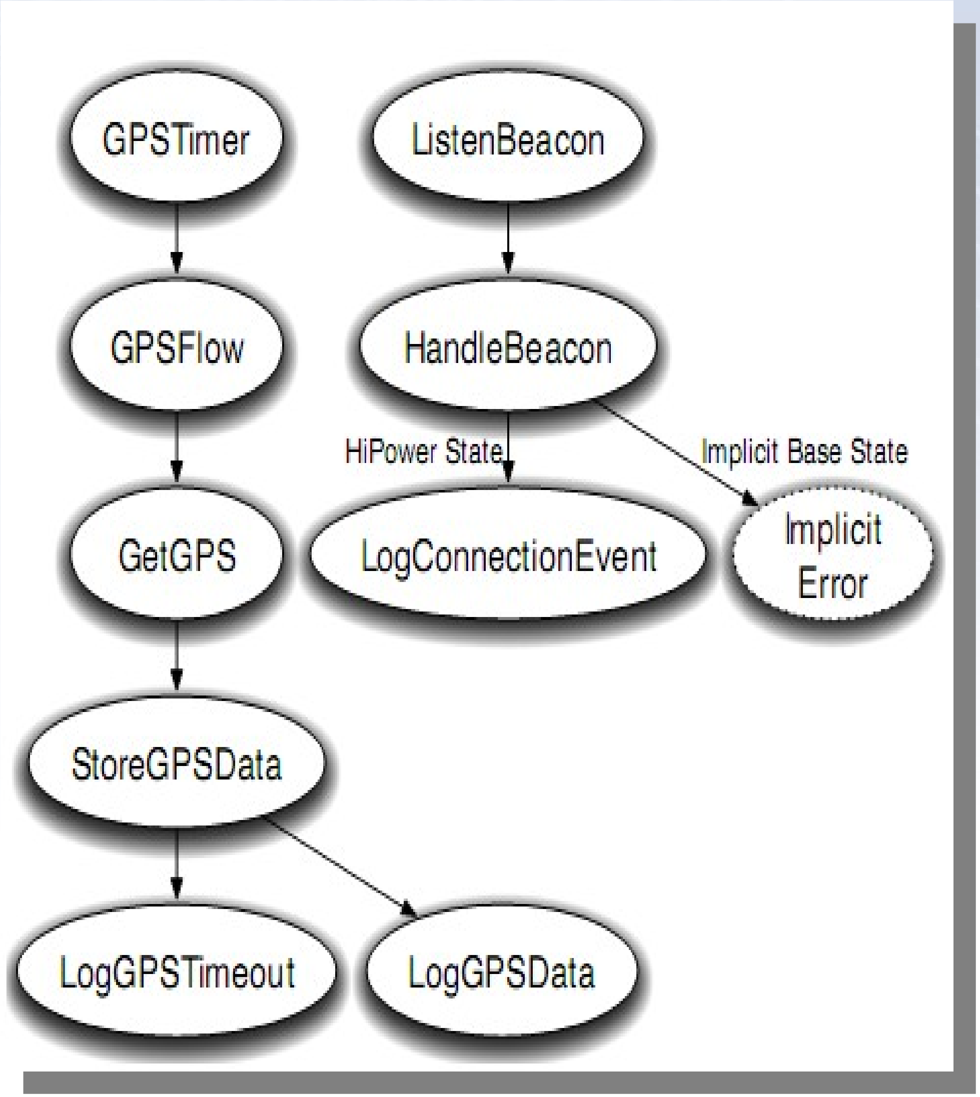
- Control language
- Flows
  - Sequence of actions
  - In response to external events (timers)
  - Belong to certain energy state
  - Have a defined input and output
- Separate logic and energy adaption

Example

- Turtle tracking application
  - Track GPS movement of endangered species

Graph representation

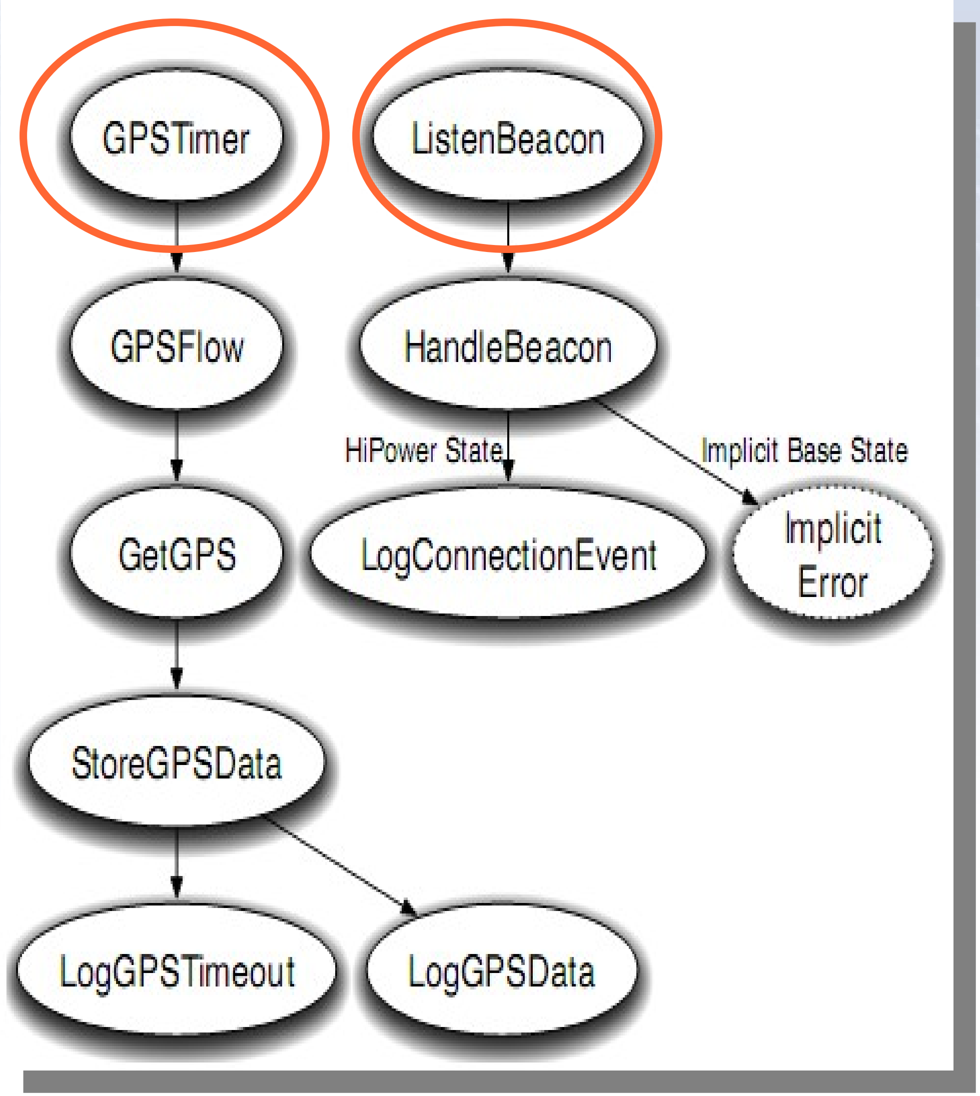
- Abstraction as dataflow graph
  - Source Nodes
  - Concrete Nodes
  - Abstract Nodes
  - Conditional Flows



Graph representation – source node

- Abstraction as dataflow graph
- Source Nodes
  - Feed data into other nodes

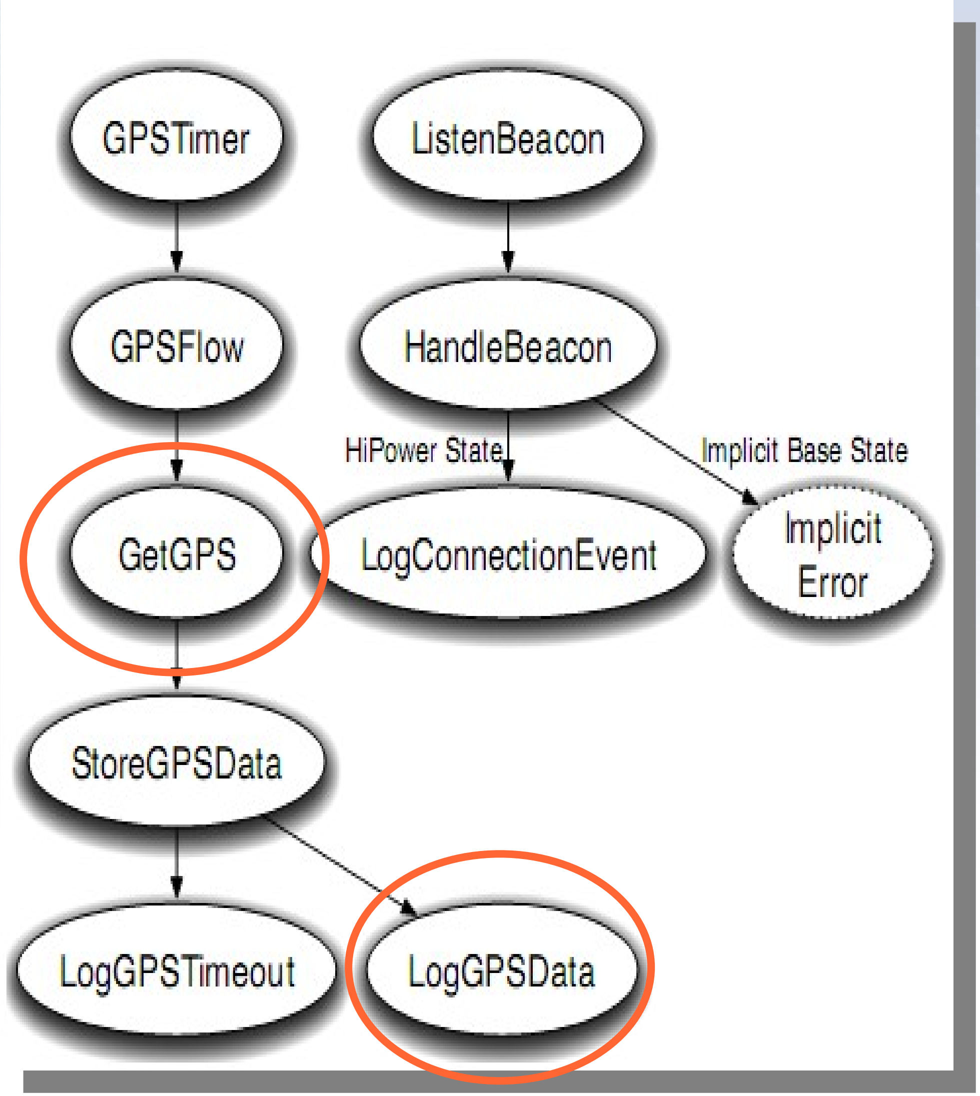
```
// Source Node Declaration
// SYNTAX: NODENAME () => (OUTPUTS);
ListenBeacon() => (msg_t msg);
GPSTimer() => ();
```



Graph rep. – concrete node

- Abstraction as dataflow graph
- Concrete Node
  - Node that links to C/nesC code

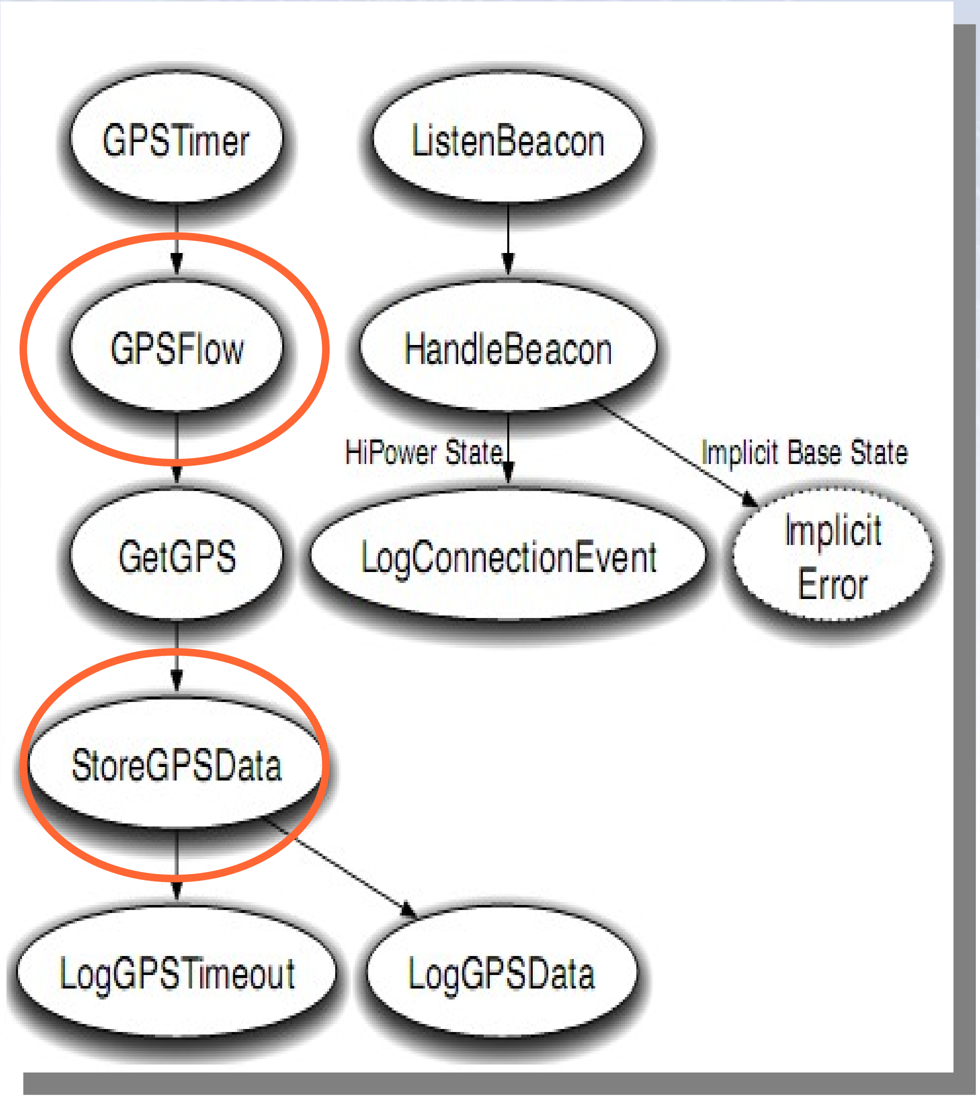
```
// Concrete Node Declaration
// SYNTAX: NODEAME (INPUTS) => (OUTPUTS);
GetGPS() =>
   (GpsData_t data, bool valid);
LogGPSData(GpsData_t data bool valid)
        => ();
```



Graph rep. – abstract node

- Abstraction as dataflow graph
- Abstract Node
  - Dataflow through concrete / abstract nodes

```
// Abstract Nodes and Predicate Flows
// SYNTAX: ABSTRACT[[type,..][state]] =
// CONCRETE->...CONCRETE;
GPSFlow = GetGPS -> StoreGPSData;
StoreGPSData:[*,gotfix][*] = LogGPSData;
StoreGPSData:[*,*][*] = LogGPSTimeout;
```

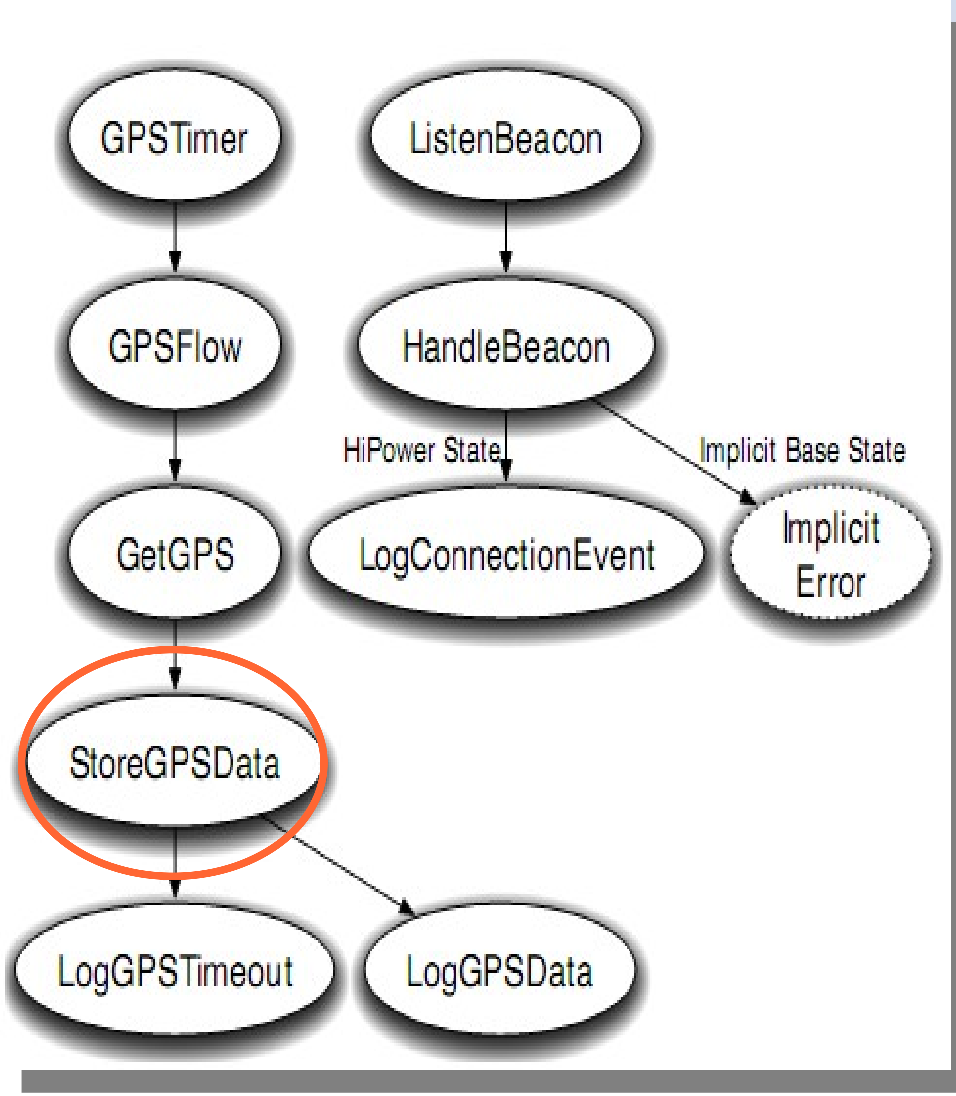


Graph rep. – conditional flow

- Abstraction as dataflow graph
- Conditional Flow
  - Use of predicate types

```
// Abstract Nodes and Predicate Flows
// SYNTAX: ABSTRACT[[type,..][state]] =
// CONCRETE->...CONCRETE;
GPSFlow = GetGPS -> StoreGPSData;
StoreGPSData:[*,gotfix][*] = LogGPSData;
StoreGPSData:[*,*][*] = LogGPSTimeout;
```

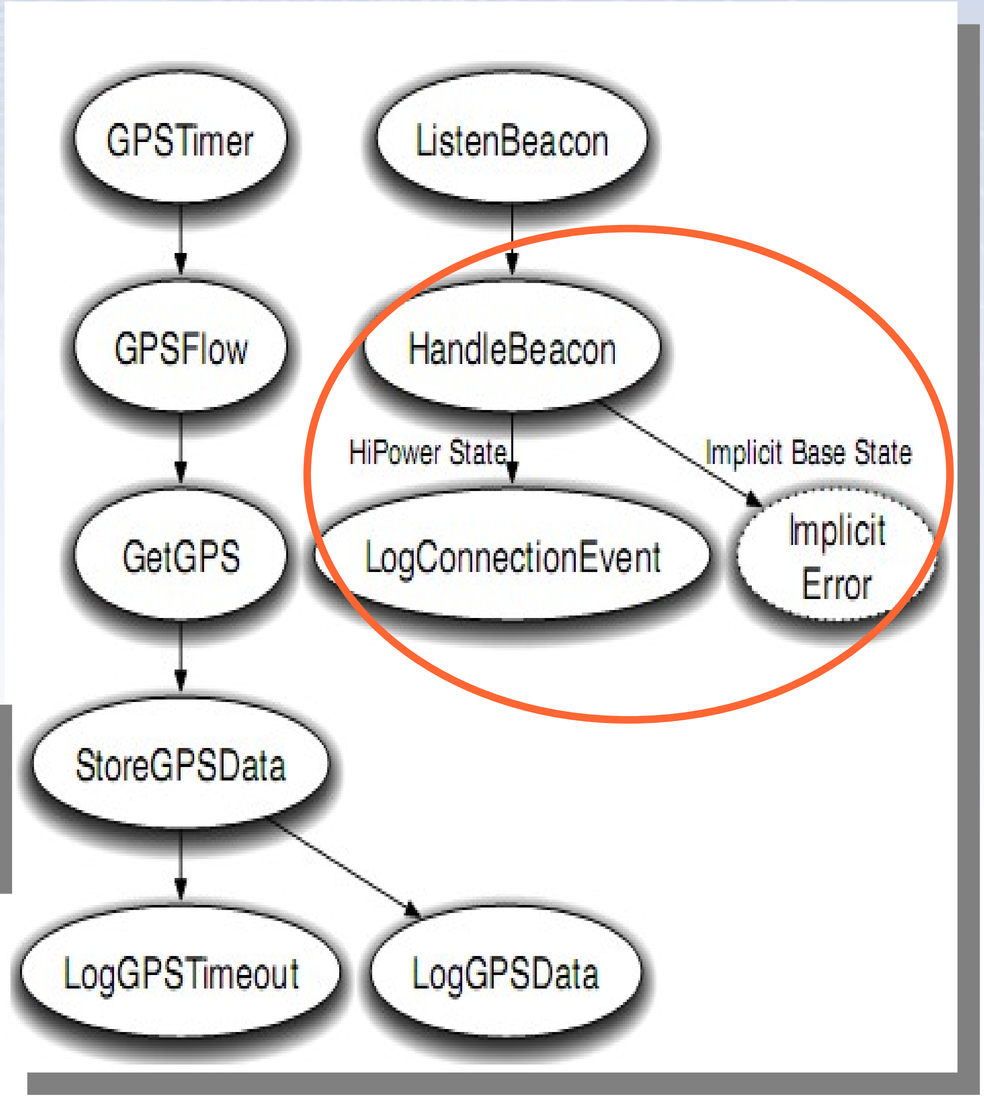
```
// Predicate Types
// SYNTAX: typedef PRED_TYPE PRED_TEST
typedef gotfix TestGotFix;
```



Graph rep. – conditional flow (2)

- Abstraction as dataflow graph
- Conditional Flow
  - Can be used in conjunction with energy levels

```
// Abstract Node using Energy Predicates
HandleBeacon:[*,*][HiPower]
        = LogConnectionEvent;
```

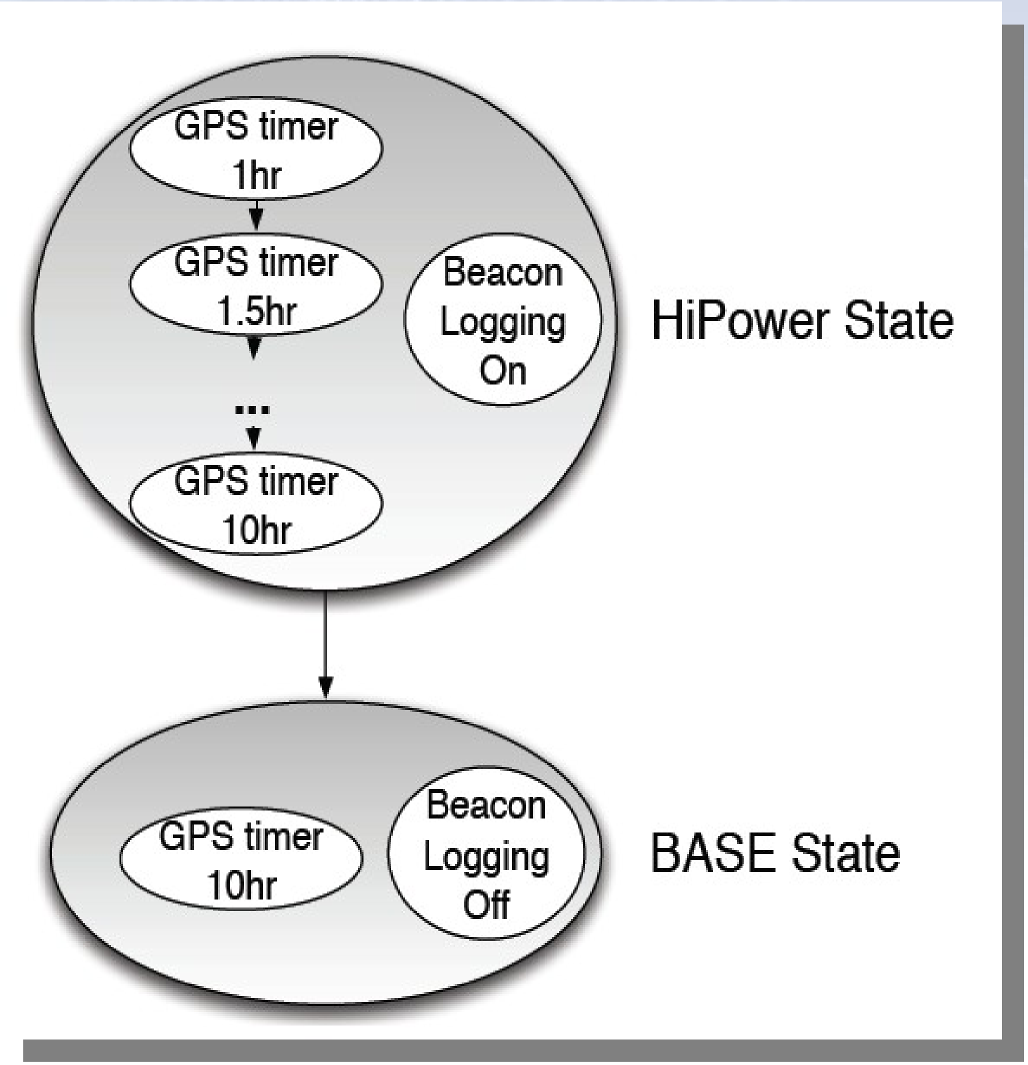


Power levels

- Discreet levels representing battery state
  - No utilities
- Implicit BASE state
- Higher ordered states are more desirable
  - And more energy intensive
- Used in
  - Conditional flows
  - State based parameters

Power levels (2)

- Timer intervals can be adjusted
  - Handled by the Runtime
- Features can be disabled
  - Hardware
  - Software
  - Data quality
    - Energy state based paths



Criticism ?

- No fine grain adjustments
  - Only timer frequency
  - State ordering instead of utilities
  - Discreet states
    - Adjustment of data quality can only be done in discreet steps
- Code has to be wrapped or structured differently

Runtime

- Goals
  - Broad array of low-power hardware
  - Online measurements
  - No training
  - Low overhead
- Ensure that the right paths are chosen
  - Predict state
  - Own consumption
  - Weather forecast

Energy adaption algorithm

- Highest fidelity while avoiding two states
  - Full battery
    - Higher level of fidelity could be provided
    - Energy is wasted
  - Empty battery
    - Sudden deadtimes could occur
    - Execution of high priority flows is prevented
- Anything in between is equivalent good

Energy adaption algorithm (2)

- Performs a search on possible states
  - Initial: Highest state with lowest timer-freq.
  - Lower state until stable (on short interval T_i)
  - Future prediction: $2^n \cdot T_i \quad n=\{1 \ldots N\}$
  - Binary search on timers

Energy attribution

- Measure consumption for the path
  - Hardware support
- Downside, Eon requires
  - Fuel gauge
  - Fine-grained current measurement
- Energy consumption can be allocated
  - Energy production/loss

Energy source model

- Energy production in following days == energy production in recent days

$$E(t+1) = \alpha E(t) + (1-\alpha) E(t-1)$$

Compiler

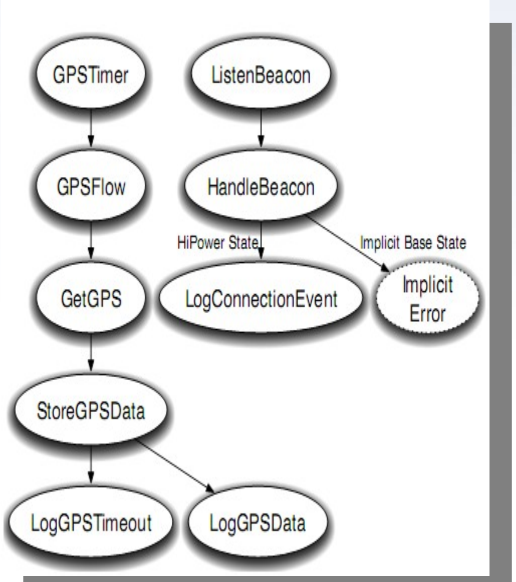
- Compilation in 3 steps
  - Nodes are built
  - Edges are attributed
  - User supplied code is linked

```
// Predicate Types
// SYNTAX: typedef PRED_TYPE PRED_TEST
typedef gotfix TestGotFix;

// Source Node Declaration
// SYNTAX: NODENAME () => (OUTPUTS);
ListenBeacon() => (msg_t msg);
GPSTimer() => ();

// Concrete Node Declaration
// SYNTAX: NODEAME (INPUTS) => (OUTPUTS);
GetGPS() =>
   (GpsData_t data, bool valid);
LogGPSData(GpsData_t data bool valid)
        => ();
LogGPSTimeout(GpsData_t data bool valid)
        => ();
LogConnectionEvent(msg_t msg) => ();

// Regular Sources
// SYNTAX: source NODENAME => NODENAME;
source ListenBeacon => HandleBeacon;
```



Compiler - Simulation

- Trace-based
  - Feed with weather data
- Test different adaption policies without deployment
- Profile energy behavior
  - Locate bottlenecks

Overview

- Introduction
  - Design concepts
- The Language
  - Basics
  - Syntax
  - Runtime / Compiler
- Evaluation
  - Deployment
  - Usability / Performance study

Deployment

- Deployment driven approach on developing Eon
- Different environments/applications
  - Turtle tracking
    - Car tracking
  - Remote camera application

Turtle tracking

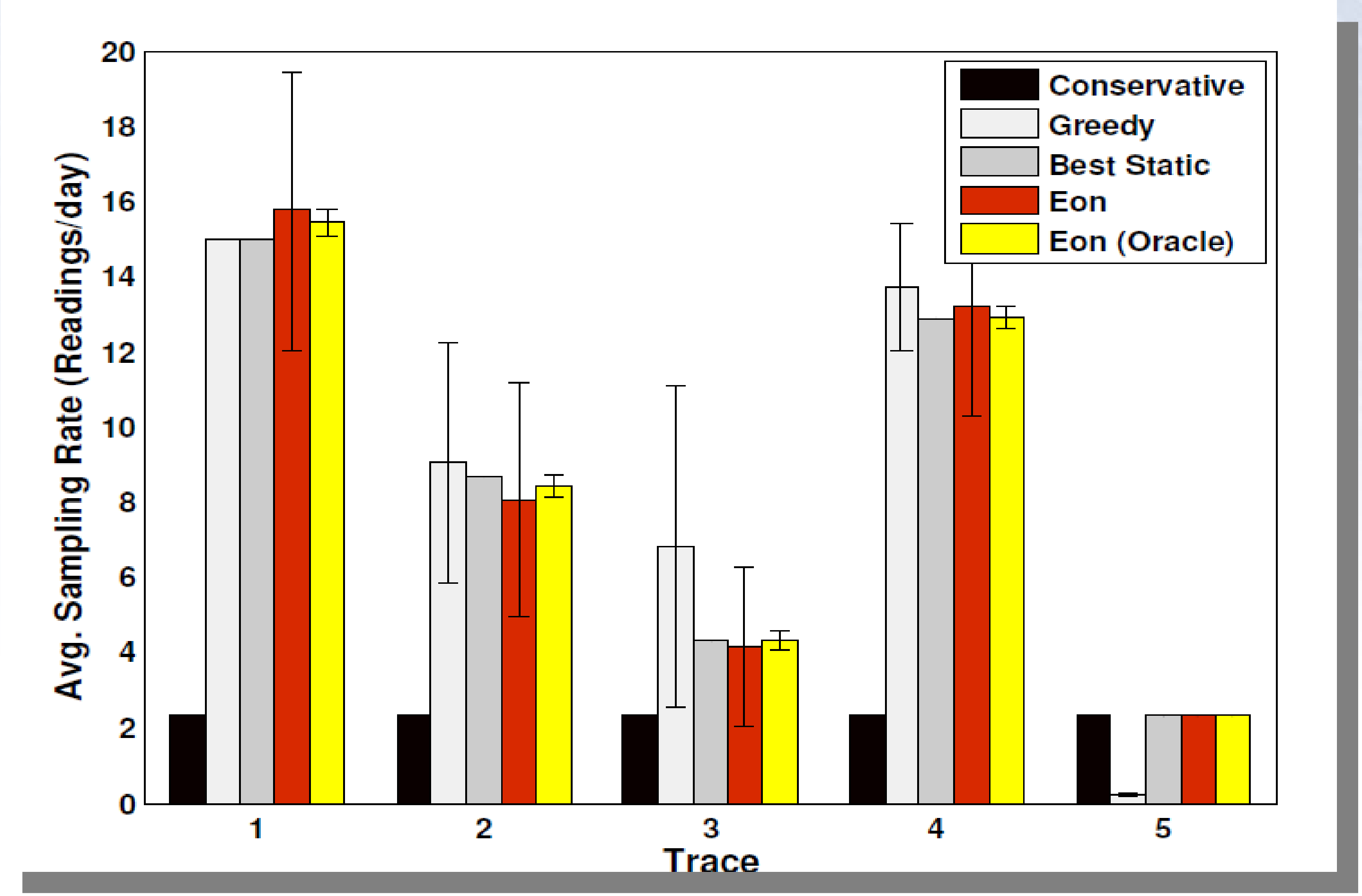
- Solar powered node fixed on turtle shell
- Report GPS data to scientists
  - Until now: had to be done manually
- Result: Failed
  - Turtles spent 98% of time underwater
  - Early hibernation
- Car tracking had to be done to get evaluation data

Adaption study

- Loop acquired data in simulator
  - 3 months of data from 2 weeks
- GPS sampling rate can be changed
  - Conservative, static
  - Greedy, static
  - Best sustainable
  - Eon (Predictor)
  - Eon (Oracle)
- 5 different devices

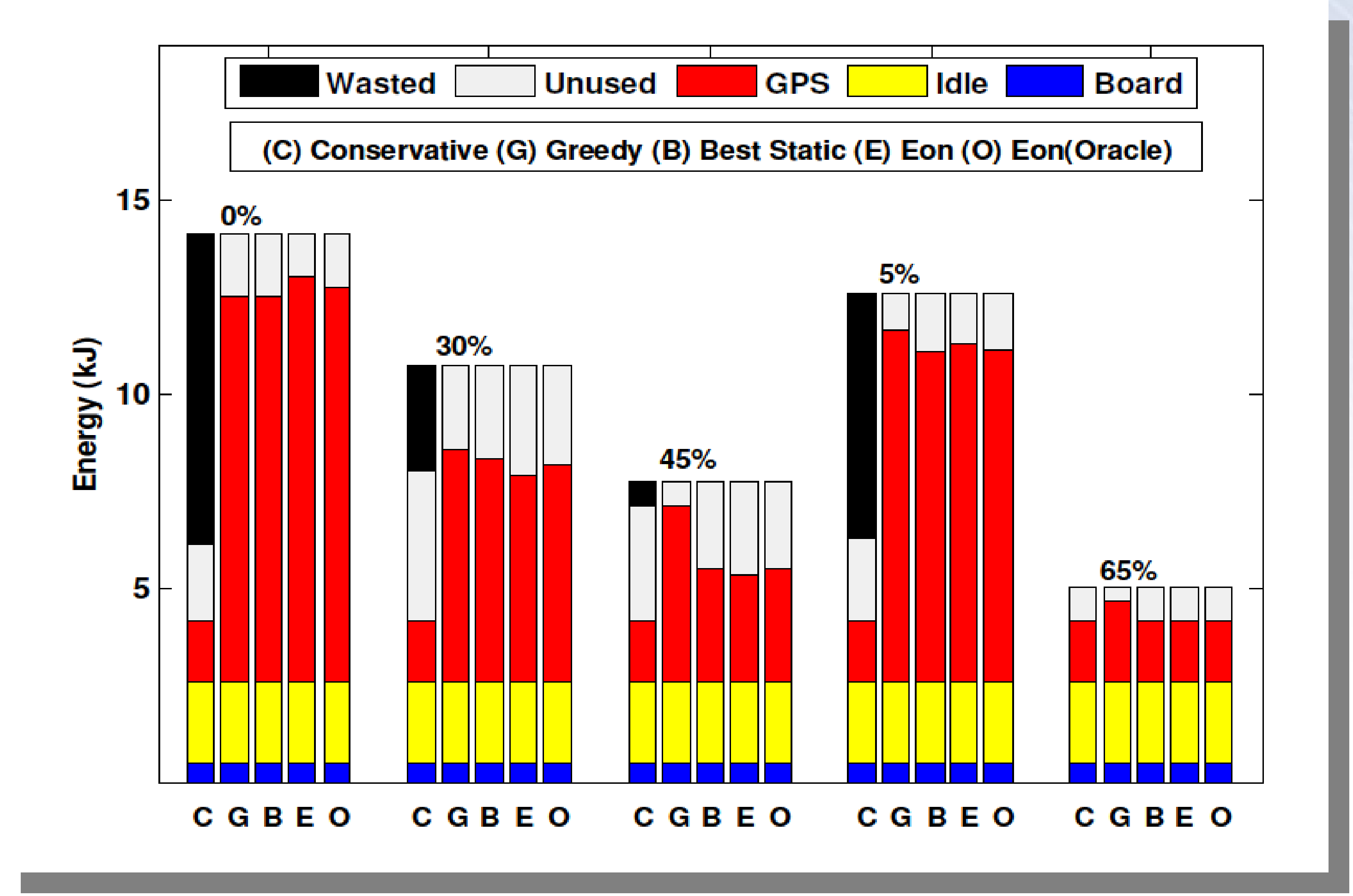
Adaption study - results

- Sampling rates of the devices



Adaption study – results (2)

- Energy consumption by board parts / strategy



Remote camera

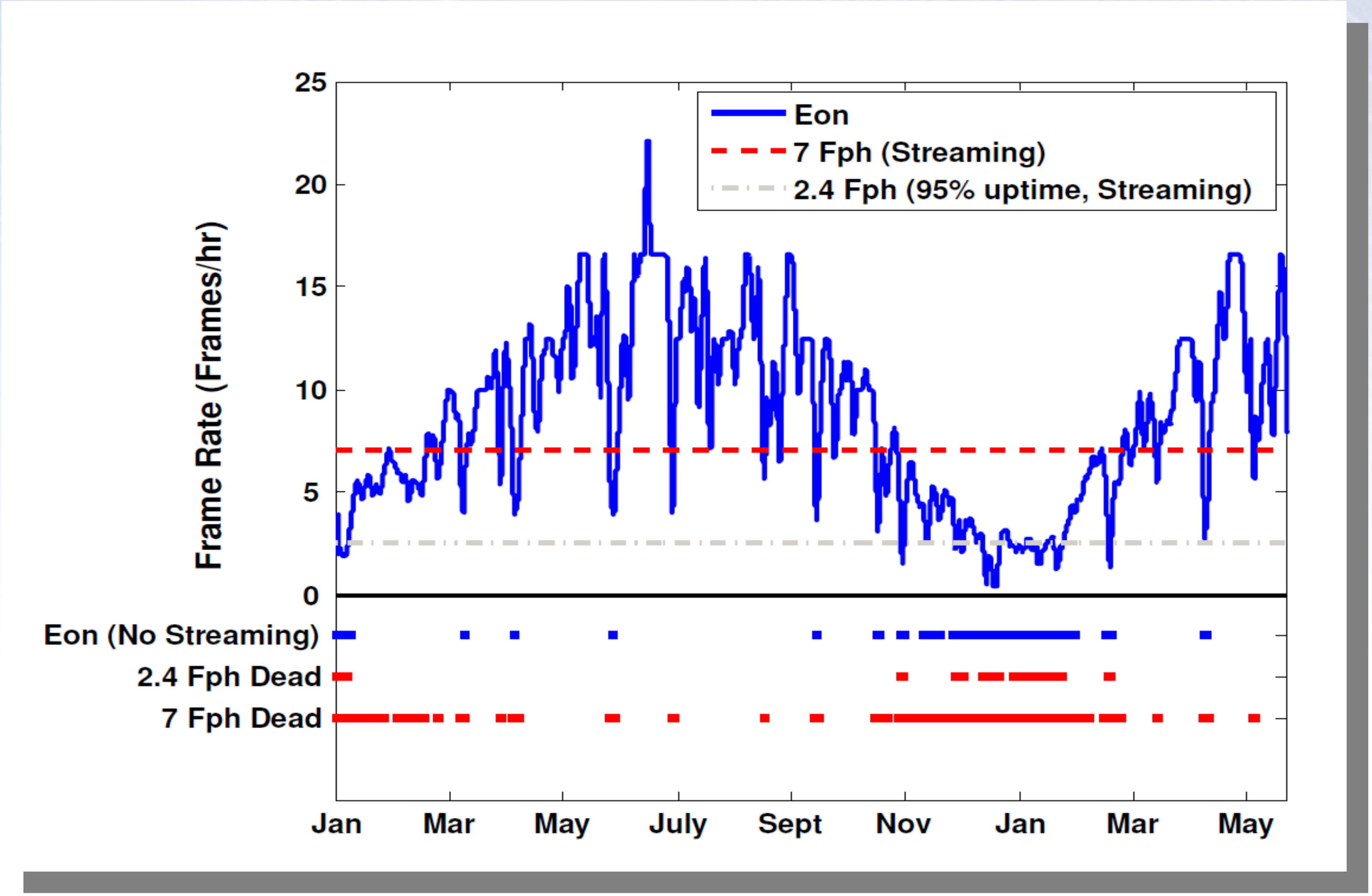
- Image streaming
  - High power state
- Image storage
  - Low power
- Building whole application in < 3 hours

Energy consumption

- Collect solar traces
  - Map solar intensity to power output of cells
  - Use climate tables to produce long term data
- Policies
  - 2.4 Fph, static
  - 7 Fph, static
  - Eon (stream / query)

Energy consumption - results

- Graph including deadtimes / query mode

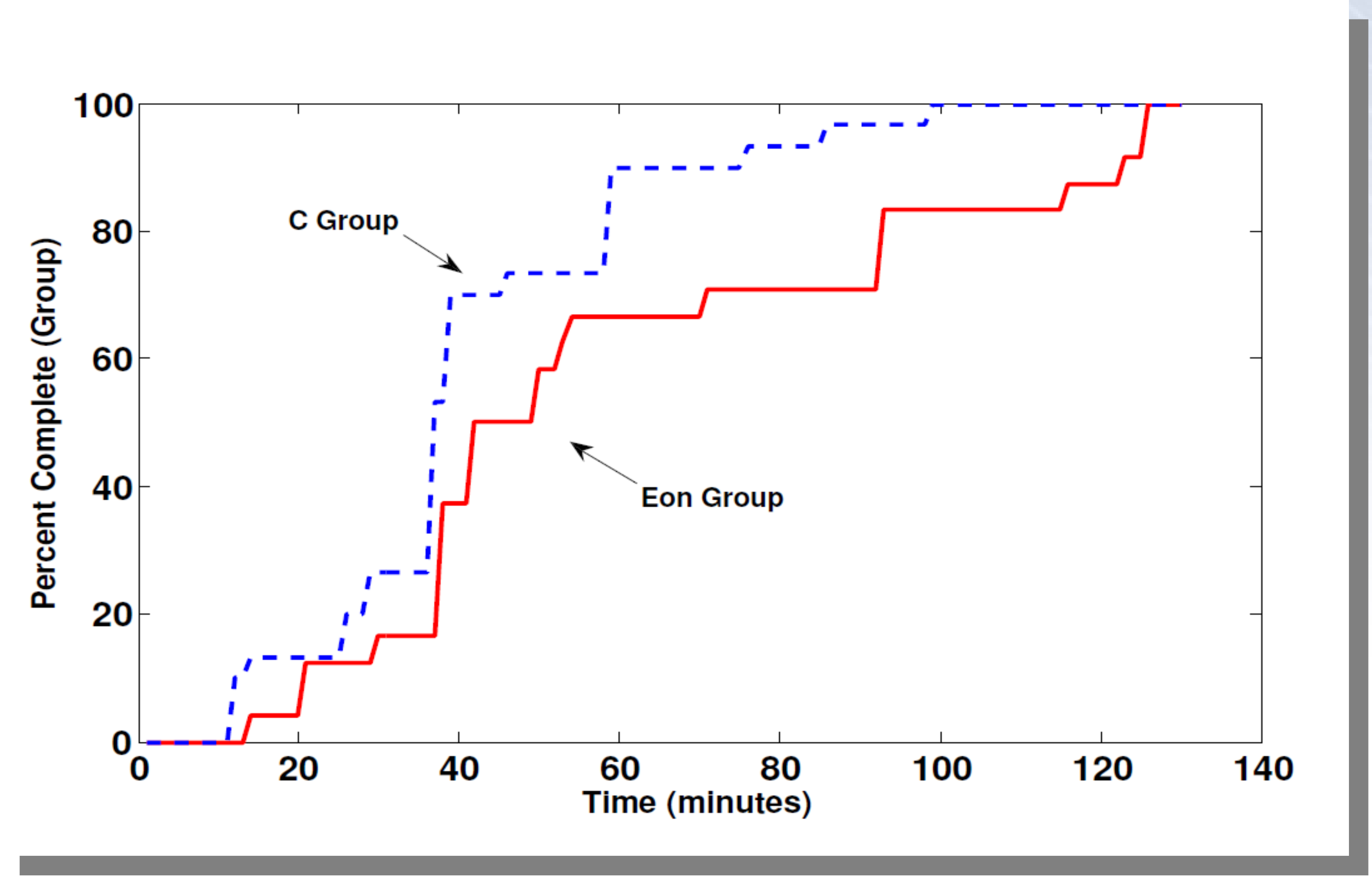


User study

- Programming sensor applications
  - Setup
    - Group of experiences C programmers
      - Provided with the same solar energy predictor Eon uses
    - Group of first time Eon users
- 1st Application samples data and saves it
- 2nd Get the most samples without running out of battery

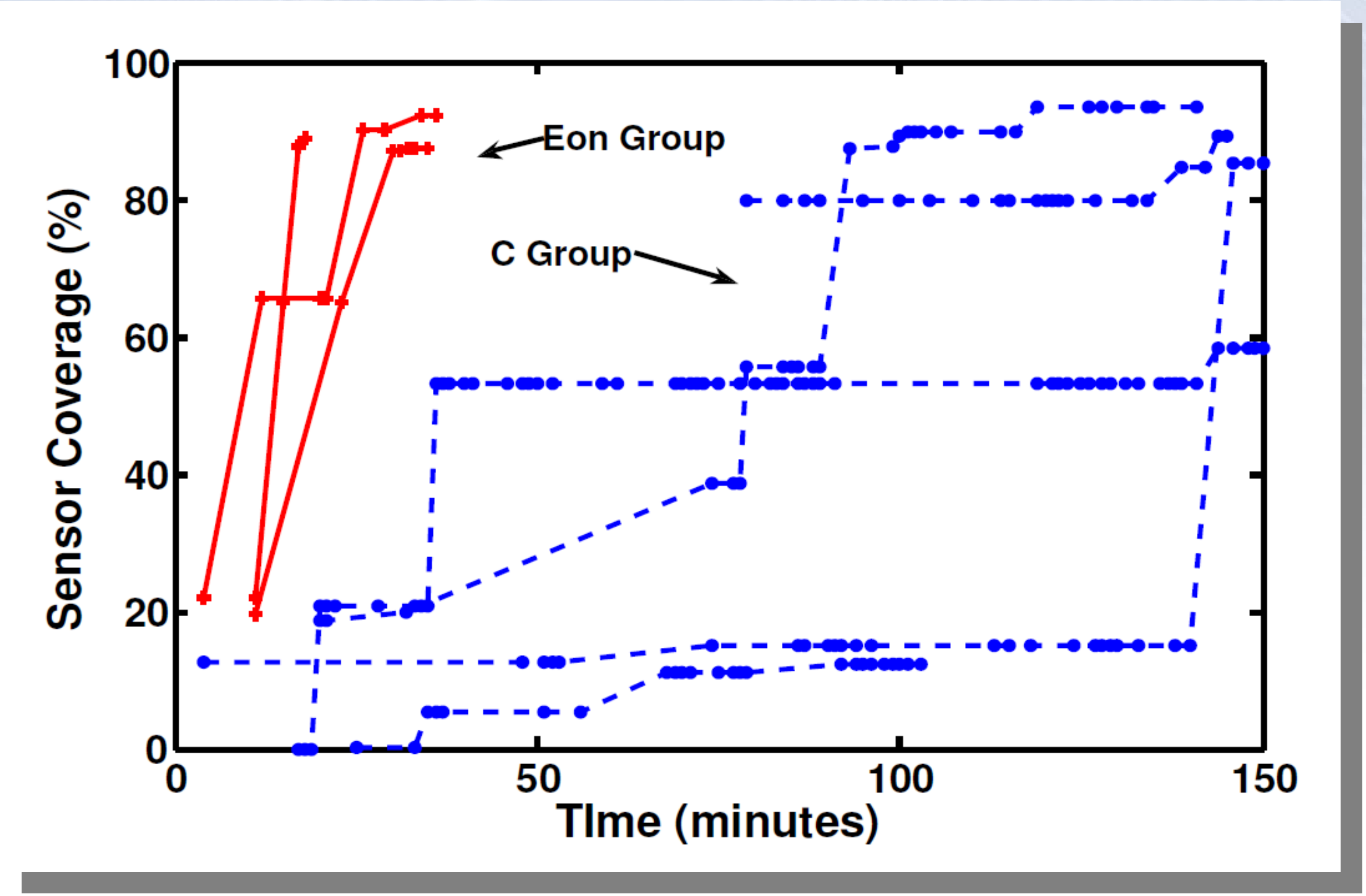
User study - results

- Task 1 has been finished



User study – results (2)

- Task 2 has been finished



Conclusion

- Benefits
  - Ease of use
  - Only approach that targets energy adaption at programming level
  - Proven efficiency
- Downsides
  - Existent code has to be rewritten

Fin

Thank you for your attention !