Storage-centric Wireless Sensor Networks for Smart Buildings

Baobing Wang

Department of Electrical and Computer Engineering
Institute for Systems Research
University of Maryland, College Park
briankw@umd.edu

IPSN 2013, Philadelphia, PA

April 7, 2013

Joint work with Prof. John S. Baras and Dr. Shah-An Yang
Motivation

- Buildings consume over 40% of total energy in the EU and US.
  - Main: HVAC (heating, ventilation, and air conditioning), lighting
  - Electric plug-loads: nearly 30% in commercial buildings
- Wireless Sensor Networks are critical for energy-efficient buildings.
  - Collect real-time data for smart HVAC, and lighting
  - Collect historical data for energy usage pattern analysis
- Difficult to design efficient and reliable WSNs for Smart Buildings.
  - Collaboration across multiple engineering domains
  - Complex cyber-physical interactions
  - Component reusability
  - Massive data collection and processing

How can we make the job easier for systems engineers?
Motivation

- Buildings consume over 40% of total energy in the EU and US.
  - Main: HVAC (heating, ventilation, and air conditioning), lighting
  - Electric plug-loads: nearly 30% in commercial buildings
- Wireless Sensor Networks are critical for energy-efficient buildings.
  - Collect real-time data for smart HVAC, and lighting
  - Collect historical data for energy usage pattern analysis
- Difficult to design efficient and reliable WSNs for Smart Buildings.
  - Collaboration across multiple engineering domains
  - Complex cyber-physical interactions
  - Component reusability
  - Massive data collection and processing

How can we make the job easier for systems engineers?
Motivation

- Buildings consume over 40% of total energy in the EU and US.
  - Main: HVAC (heating, ventilation, and air conditioning), lighting
  - Electric plug-loads: nearly 30% in commercial buildings
- Wireless Sensor Networks are critical for energy-efficient buildings.
  - Collect real-time data for smart HVAC, and lighting
  - Collect historical data for energy usage pattern analysis
- Difficult to design efficient and reliable WSNs for Smart Buildings.
  - Collaboration across multiple engineering domains
  - Complex cyber-physical interactions
  - Component reusability
  - Massive data collection and processing

How can we make the job easier for systems engineers?
Motivation

- Buildings consume over 40% of total energy in the EU and US.
  - Main: HVAC (heating, ventilation, and air conditioning), lighting
  - Electric plug-loads: nearly 30% in commercial buildings
- Wireless Sensor Networks are critical for energy-efficient buildings.
  - Collect real-time data for smart HVAC, and lighting
  - Collect historical data for energy usage pattern analysis
- Difficult to design efficient and reliable WSNs for Smart Buildings.
  - Collaboration across multiple engineering domains
  - Complex cyber-physical interactions
  - Component reusability
  - Massive data collection and processing

How can we make the job easier for systems engineers?
Q1: How to develop an integrated framework for the design and evaluation across multiple engineering domains?
  - WSNDesign: Model-based Systems Design Framework
    - Model libraries and integration
    - Theoretical performance estimation
    - Automatic code generation and integrated simulation
    - Reduce the complexity of system analysis

Q2: How to store and retrieve large amount of sensor readings efficiently?
  - Flash-based Data Storage and Retrieval
    - Node-level energy-efficient data storage system
    - Distributed database system supporting $\epsilon$-approximate querying
Q1: How to develop an integrated framework for the design and evaluation across multiple engineering domains?

- **WSNDesign**: Model-based Systems Design Framework
  - Model libraries and integration
  - Theoretical performance estimation
  - Automatic code generation and integrated simulation
  - Reduce the complexity of system analysis

Q2: How to store and retrieve large amount of sensor readings efficiently?

- **Flash-based Data Storage and Retrieval**
  - Node-level energy-efficient data storage system
  - Distributed database system supporting $\epsilon$-approximate querying
Q1: How to develop an integrated framework for the design and evaluation across multiple engineering domains?

- **WSNDesign:** Model-based Systems Design Framework
  - Model libraries and integration
  - Theoretical performance estimation
  - Automatic code generation and integrated simulation
  - Reduce the complexity of system analysis

Q2: How to store and retrieve large amount of sensor readings efficiently?

- **Flash-based Data Storage and Retrieval**
  - Node-level energy-efficient data storage system
  - Distributed database system supporting $\epsilon$-approximate querying
Contributions

Q1: How to develop an integrated framework for the design and evaluation across multiple engineering domains?

- **WSNDesign: Model-based Systems Design Framework**
  - Model libraries and integration
  - Theoretical performance estimation
  - Automatic code generation and integrated simulation
  - Reduce the complexity of system analysis

Q2: How to store and retrieve large amount of sensor readings efficiently?

- **Flash-based Data Storage and Retrieval**
  - Node-level energy-efficient data storage system
  - Distributed database system supporting $\epsilon$-approximate querying
WSNDesign Overview: Model Libraries

Applications
(System Services)
(Network Models)
(Physical Models)
(Cyber System Models)

Wireless Sensor Networks
(System Models)

Computation/Algorithms, Data Presentation, Communication Protocols

Applications (Requirements)

System Services (Information-oriented)

Service Models (Distributed Data Store and Retrieval)

Network Models (Communication and Management)

Physical Systems (Functions and Resource)

Physical Models (Functions and Performance)

Environment & BECS

Cyber System Models

Mapping

Figure: Hierarchy of System Models
WSNDesign Overview: Design Flow

Figure: Integrated Design Environment
Develop model libraries using SysML, Modelica and Simulink

Model Wireless Sensor Networks
- Physical platforms: CPU, sensor, RF transceiver, and battery
- MAC layer protocols: Low Power Listener, CSMA/CA Channel Access, CSMA/CA Sender, MAC Controller, Slot Manager, Queue Manager, TDMA Sender, Receiver ...
- Wireless channels: radio propagation models, channel fading models, and bit error rates

Model Cyber Systems
- Phenomenon: interface between the event-triggered domain and continuous-time domain
- Environment: propagation of phenomenon signals
- Control logic

Case study: building thermal control system
Develop model libraries using SysML, Modelica and Simulink

Model Wireless Sensor Networks

- Physical platforms: CPU, sensor, RF transceiver, and battery
- MAC layer protocols: Low Power Listener, CSMA/CA Channel Access, CSMA/CA Sender, MAC Controller, Slot Manager, Queue Manager, TDMA Sender, Receiver ...
- Wireless channels: radio propagation models, channel fading models, and bit error rates

Model Cyber Systems

- Phenomenon: interface between the event-triggered domain and continuous-time domain
- Environment: propagation of phenomenon signals
- Control logic

Case study: building thermal control system
Develop model libraries using SysML, Modelica and Simulink

- Model Wireless Sensor Networks
  - Physical platforms: CPU, sensor, RF transceiver, and battery
  - MAC layer protocols: Low Power Listener, CSMA/CA Channel Access, CSMA/CA Sender, MAC Controller, Slot Manager, Queue Manager, TDMA Sender, Receiver ...
  - Wireless channels: radio propagation models, channel fading models, and bit error rates

- Model Cyber Systems
  - Phenomenon: interface between the event-triggered domain and continuous-time domain
  - Environment: propagation of phenomenon signals
  - Control logic

- Case study: building thermal control system
Develop model libraries using SysML, Modelica and Simulink

Model Wireless Sensor Networks
- Physical platforms: CPU, sensor, RF transceiver, and battery
- MAC layer protocols: Low Power Listener, CSMA/CA Channel Access, CSMA/CA Sender, MAC Controller, Slot Manager, Queue Manager, TDMA Sender, Receiver ...
- Wireless channels: radio propagation models, channel fading models, and bit error rates

Model Cyber Systems
- Phenomenon: interface between the event-triggered domain and continuous-time domain
- Environment: propagation of phenomenon signals
- Control logic

Case study: building thermal control system
Complexity of system analysis

- Exponential exploration space: $D^N$
- Overall system = a series of local analysis + composition rules
- Reduced complexity: $\sum_{i=1}^{k} D^{n_i}$

Drawbacks of existing work

- Ad-hoc partitioning
- Rely upon general rules of thumb
- Rely upon the expertise of systems engineers

Our objective

- Visualize and quantitate the complexity of system analysis
- Help system engineers understand the impact of their decisions
- Give hints to system engineers to improve their designs
Complexity of system analysis

- Exponential exploration space: $D^N$
- Overall system = a series of local analysis + composition rules
- Reduced complexity: $\sum_{i=1}^{k} D^{n_i}$

Drawbacks of existing work

- Ad-hoc partitioning
- Rely upon general rules of thumb
- Rely upon the expertise of systems engineers

Our objective

- Visualize and quantitate the complexity of system analysis
- Help system engineers understand the impact of their decisions
- Give hints to system engineers to improve their designs
 Complexity of system analysis

- Exponential exploration space: $D^N$
- Overall system = a series of local analysis + composition rules
- Reduced complexity: $\sum_{i=1}^{k} D^{n_i}$

 Drawbacks of existing work

- Ad-hoc partitioning
- Rely upon general rules of thumb
- Rely upon the expertise of systems engineers

 Our objective

- Visualize and quantitate the complexity of system analysis
- Help system engineers understand the impact of their decisions
- Give hints to system engineers to improve their designs
Figure: Interactive Tool (GUI)
WSNDesign: Future Work

- **Model Library Development**
  - Transform and import TinyOS libraries: SysML-TinyOS profile
  - Integrate SysML-Modelica profile with IBM Rational Rhapsody
  - Enrich the Physical Model Library

- **Code Generation and Simulation Integration**
  - Generate TinyOS configuration scripts
  - Generate Modelica wrapper components
  - Synchronize the IBM Rhapsody simulator (SysML) with TOSSIM (TinyOS) and OpenModelica simulator (Modelica)

- **Complexity Reduction**
  - Parse hierarchical SysML Parametric Diagrams
  - Generate improved block diagrams from the analytical result
WSNDesign: Future Work

- **Model Library Development**
  - Transform and import TinyOS libraries: SysML-TinyOS profile
  - Integrate SysML-Modelica profile with IBM Rational Rhapsody
  - Enrich the Physical Model Library

- **Code Generation and Simulation Integration**
  - Generate TinyOS configuration scripts
  - Generate Modelica wrapper components
  - Synchronize the IBM Rhapsody simulator (SysML) with TOSSIM (TinyOS) and OpenModelica simulator (Modelica)

- **Complexity Reduction**
  - Parse hierarchical SysML Parametric Diagrams
  - Generate improved block diagrams from the analytical result
WSNDesign: Future Work

- **Model Library Development**
  - Transform and import TinyOS libraries: SysML-TinyOS profile
  - Integrate SysML-Modelica profile with IBM Rational Rhapsody
  - Enrich the Physical Model Library

- **Code Generation and Simulation Integration**
  - Generate TinyOS configuration scripts
  - Generate Modelica wrapper components
  - Synchronize the IBM Rhapsody simulator (SysML) with TOSSIM (TinyOS) and OpenModelica simulator (Modelica)

- **Complexity Reduction**
  - Parse hierarchical SysML Parametric Diagrams
  - Generate improved block diagrams from the analytical result
Flash-based Data Storage and Retrieval

System Models
- Application Models (Functionality and Performance Reqs.)
  - Tracking
  - Detection
  - Monitoring
  ... Application-specific

Service Models (Distributed Data Store and Retrieval)
- Query
- Naming
- Location
- Syn
  ... Mapping

Network Models (Communication and Management)
- MAC
- Routing
- Mobility
- Data
- Topology Control
- Power Control
  ... Mapping

Physical Models (Functions and Performance)
- Sensor
- Actuator
- Router
- Base Station
- Wireless Channel
  ... Mapping

Cyber System Models
- Phenomena
- HVAC
  ... Mapping

Modelica Building Library (Lawrence Berkeley Lab)

- Aggregated: average, peak for each hour
- Approximate: power usage pattern
- Not every reading is needed!
- Centralized data collection
- In-situ data storage ✓
- Flash memory: high capacity, energy efficient
Flash-based Data Storage and Retrieval

- **Aggregated**: average, peak for each hour
- **Approximate**: power usage pattern

Not every reading is needed!

- Centralized data collection
- In-situ data storage ✓
- Flash memory: high capacity, energy efficient
Flash-based Data Storage and Retrieval

- Aggregated: average, peak for each hour
- Approximate: power usage pattern

Not every reading is needed!

- Centralized data collection
- In-situ data storage
- Flash memory: high capacity, energy efficient
Flash-based Data Storage and Retrieval

- **Aggregated**: average, peak for each hour
- **Approximate**: power usage pattern

Not every reading is needed!

- **Centralized** data collection
- **In-situ** data storage ✓
- **Flash memory**: high capacity, energy efficient
Problem Formulation

- $N$ motes, sampling once every $\tau$ seconds
  - $r_{id,t} = \langle id, t, key, v_2, \ldots, v_d \rangle$

- $\epsilon$-Approximate Querying
  - Given a dataset $\Upsilon$, retrieve $O \subseteq \Upsilon$ s.t. the approximate version $\tilde{\Upsilon}$ computed from $O$ satisfies:

$$L_\infty(O, \Upsilon) \triangleq L_\infty(\tilde{\Upsilon}, \Upsilon) = \max_{id=1}^N \max_{t=t_1}^{t_2} \| (\tilde{r}_{id,t} - r_{id,t}) \times w \|_\infty \leq \epsilon$$

- Problem of $\epsilon$-Approximate Querying
  - Error bound $\epsilon$ must be specified by users
  - What $\epsilon$ can lead to satisfactory results? $\leftrightarrow$ Difficult to decide
  - $\epsilon$ is too tight: over-qualified result, energy waste
  - $\epsilon$ is too loose: re-issue the query, duplicated data retrieval
Problem Formulation

- \( N \) motes, sampling once every \( \tau \) seconds
  - \( r_{id,t} = \langle id, t, key, v_2, \ldots, v_d \rangle \)

- \( \epsilon \)-Approximate Querying
  - Given a dataset \( \Upsilon \), retrieve \( O \subseteq \Upsilon \) s.t. the approximate version \( \tilde{\Upsilon} \) computed from \( O \) satisfies:
    \[
    L_\infty(O, \Upsilon) \triangleq L_\infty(\tilde{\Upsilon}, \Upsilon) = \max_{id=1}^{N} \max_{t=t_1}^{t_2} \| (\tilde{r}_{id,t} - r_{id,t}) \times w \|_\infty \leq \epsilon
    \]

- Problem of \( \epsilon \)-Approximate Querying
  - Error bound \( \epsilon \) must be specified by users
  - What \( \epsilon \) can lead to satisfactory results? \( \leftarrow \) Difficult to decide
  - \( \epsilon \) is too tight: over-qualified result, energy waste
  - \( \epsilon \) is too loose: re-issue the query, duplicated data retrieval
Problem Formulation

- $N$ motes, sampling once every $\tau$ seconds
  - $r_{id,t} = \langle id, t, key, v_2, \ldots, v_d \rangle$

- $\epsilon$-Approximate Querying
  - Given a dataset $\Upsilon$, retrieve $O \subseteq \Upsilon$ s.t. the approximate version $\tilde{\Upsilon}$ computed from $O$ satisfies:

  $$L_\infty(O, \Upsilon) \triangleq L_\infty(\tilde{\Upsilon}, \Upsilon) = \max_{id=1}^N \max_{t=t_1}^{t_2} \| (\tilde{r}_{id,t} - r_{id,t}) \times w \|_\infty \leq \epsilon$$

- Problem of $\epsilon$-Approximate Querying
  - Error bound $\epsilon$ must be specified by users
  - What $\epsilon$ can lead to satisfactory results? ← Difficult to decide
    - $\epsilon$ is too tight: over-qualified result, energy waste
    - $\epsilon$ is too loose: re-issue the query, duplicated data retrieval
**Problem Formulation**

**Incremental $\epsilon$-Approximate Querying**

- $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
- $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
- Each $\rho_i$ retrieves $O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq \text{key} \leq k_2)\} \subseteq \Upsilon$
- Incremental: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
- Correct: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- Q2.1: How to get $\Psi_i$ efficiently? $\Rightarrow$ HybridStore
- Q2.2: How to compute $\Delta_i$ efficiently? $\Rightarrow$ HybridDB
Problem Formulation

**Incremental $\epsilon$-Approximate Querying**

- $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
- $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
- Each $\rho_i$ retrieves
  
  $O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq \text{key} \leq k_2)\} \subseteq \Upsilon$

- Incremental: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$

- Correct: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

**Q2.1:** How to get $\Psi_i$ efficiently? $\implies$ HybridStore

**Q2.2:** How to compute $\Delta_i$ efficiently? $\implies$ HybridDB
Problem Formulation

- **Incremental $\epsilon$-Approximate Querying**
  - $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
  - $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
  - Each $\rho_i$ retrieves
    $O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq key \leq k_2)\} \subseteq \Upsilon$
  - Incremental: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
  - Correct: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- Q2.1: How to get $\Psi_i$ efficiently? $\Rightarrow$ HybridStore
- Q2.2: How to compute $\Delta_i$ efficiently? $\Rightarrow$ HybridDB
Problem Formulation

**Incremental $\epsilon$-Approximate Querying**

- $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
- $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
- Each $\rho_i$ retrieves $O_i \subseteq \Psi_i = \{r_{id, t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq key \leq k_2)\} \subseteq \mathcal{Y}$
- **Incremental**: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
- **Correct**: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- **Q2.1**: How to get $\Psi_i$ efficiently? $\Rightarrow$ HybridStore
- **Q2.2**: How to compute $\Delta_i$ efficiently? $\Rightarrow$ HybridDB
Problem Formulation

- **Incremental ε-Approximate Querying**
  - $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
  - $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
  - Each $\rho_i$ retrieves
    - $O_i \subseteq \Psi_i = \{r_{id,t} | (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq \text{key} \leq k_2)\} \subseteq \Upsilon$
  - **Incremental**: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
  - **Correct**: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- Q2.1: How to get $\Psi_i$ efficiently? $\Rightarrow$ HybridStore
- Q2.2: How to compute $\Delta_i$ efficiently? $\Rightarrow$ HybridDB
Problem Formulation

- **Incremental $\epsilon$-Approximate Querying**
  - $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
  - $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
  - Each $\rho_i$ retrieves
    - $O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq \text{key} \leq k_2)\} \subseteq \Upsilon$
  - Incremental: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
  - Correct: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- **Q2.1:** How to get $\Psi_i$ efficiently? $\Rightarrow$ HybridStore
- **Q2.2:** How to compute $\Delta_i$ efficiently? $\Rightarrow$ HybridDB
Problem Formulation

- **Incremental \( \epsilon \)-Approximate Querying**
  - \( Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\} \), where \( \rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\} \)
  - \( 0 \leq \epsilon_{i+1} < \epsilon_i \) and \([t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]\)
  - Each \( \rho_i \) retrieves
    \[
    O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq key \leq k_2)\} \subseteq \Upsilon
    \]
  - Incremental: \( \Delta_i = O_i \setminus O_{i-1} \) and \( \Delta_i \cap \Delta_j = \emptyset \) for all \( i \neq j \)
  - Correct: \( O_i \) can be constructed from \( \Delta_1, \ldots, \Delta_i \), and \( L_\infty(O_i, \Psi_i) \leq \epsilon_i \)

- **Q2.1:** How to get \( \Psi_i \) efficiently? \( \Rightarrow \) HybridStore
- **Q2.2:** How to compute \( \Delta_i \) efficiently? \( \Rightarrow \) HybridDB
Problem Formulation

**Incremental ε-Approximate Querying**

- \( Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\} \), where \( \rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\} \)
- \( 0 \leq \epsilon_{i+1} < \epsilon_i \) and \( [t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}] \)
- Each \( \rho_i \) retrieves
  \[ O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq key \leq k_2)\} \subseteq \Upsilon \]
- **Incremental:** \( \Delta_i = O_i \setminus O_{i-1} \) and \( \Delta_i \cap \Delta_j = \emptyset \) for all \( i \neq j \)
- **Correct:** \( O_i \) can be constructed from \( \Delta_1, \ldots, \Delta_i \), and \( L_\infty(O_i, \Psi_i) \leq \epsilon_i \)

**Q2.1:** How to get \( \Psi_i \) efficiently? \( \Rightarrow \) HybridStore

**Q2.2:** How to compute \( \Delta_i \) efficiently? \( \Rightarrow \) HybridDB
Problem Formulation

- **Incremental $\epsilon$-Approximate Querying**
  - $Q = \{\rho_1, \rho_2, \ldots, \rho_\lambda\}$, where $\rho_i = \{[t_{i,1}, t_{i,2}], [k_1, k_2], \epsilon_i\}$
  - $0 \leq \epsilon_{i+1} < \epsilon_i$ and $[t_{i+1,1}, t_{i+1,2}] \subseteq [t_{i,1}, t_{i,2}]$
  - Each $\rho_i$ retrieves
    $$O_i \subseteq \Psi_i = \{r_{id,t} \mid (t_{i,1} \leq t \leq t_{i,2}) \land (k_1 \leq \text{key} \leq k_2)\} \subseteq \Upsilon$$
  - **Incremental**: $\Delta_i = O_i \setminus O_{i-1}$ and $\Delta_i \cap \Delta_j = \emptyset$ for all $i \neq j$
  - **Correct**: $O_i$ can be constructed from $\Delta_1, \ldots, \Delta_i$, and $L_\infty(O_i, \Psi_i) \leq \epsilon_i$

- Q2.1: How to get $\Psi_i$ efficiently? $\implies$ HybridStore
- Q2.2: How to compute $\Delta_i$ efficiently? $\implies$ HybridDB
HybridStore Interface [EWSN ’13]

- insert(float key, void* record, uint8_t length)
- select(uint32_t t₁, uint32_t t₂, float k₁, float k₂)

HybridStore Features

- All NAND pages are fully occupied and written purely sequentially
- In-place updates and out-of-place writes are completely avoided
- Process typical joint queries efficiently, even on large-scale datasets
- Data aging without overhead, and simple failure recovery mechanism
HybridDB: Incremental $\epsilon$-Approximate Querying

**HybridDB Interface [TOSN ’13]**

- `approxQuery(uint32_t t_{1,1}, uint32_t t_{1,2}, float k_1, float k_2, float \epsilon_1)`
- `approxUpdate(uint8_t queryID, uint32_t t_{i,1}, uint32_t t_{i,2}, float \epsilon_i)`

**HybridDB Features**

- Support refinement and zoom-in sub-queries
- Retrieve an approximate dataset with arbitrary $L_\infty$ error bound
- Balance trade-offs
  - Energy consumption: sensors $\leftrightarrow$ proxy
  - Response time: current sub-query $\leftrightarrow$ following sub-queries
HybridDB: Incremental $\epsilon$-Approximate Querying

HybridDB Interface [TOSN '13]

- approxQuery(uint32_t $t_{1,1}$, uint32_t $t_{1,2}$, float $k_1$, float $k_2$, float $\epsilon_1$)
- approxUpdate(uint8_t queryID, uint32_t $t_{i,1}$, uint32_t $t_{i,2}$, float $\epsilon_i$)

HybridDB Features

- Support refinement and zoom-in sub-queries
- Retrieve an approximate dataset with arbitrary $L_\infty$ error bound
- Balance trade-offs
  - Energy consumption: sensors $\leftrightarrow$ proxy
  - Response time: current sub-query $\leftrightarrow$ following sub-queries
HybridDB: Testbed Results

- Sensor friendly: 22.5KB ROM, 3.76KB RAM
- Benefits: significant energy savings, much better user experience

![Diagram of sensor placement and latency graph]

Baobing Wang (UMD)  WSNs for Smart Buildings  April 7, 2013
HybridDB: Future Work

Import into WSNDesign using SysML-TinyOS profile
Conclusion

- Proposed a model-based design framework for the design of WSNs in the context of Smart Buildings $\implies$ **System-level Design**
  - Hierarchy of model libraries, model transformation and integration
  - Composition rules, and system performance estimation
  - Code generation, and multi-simulator integration
  - Reduction of system analysis complexity

- Proposed and implemented an abstraction for *in-situ* data storage and retrieval $\implies$ **System Implementation**
  - Efficient light-weight data storage system
  - Distributed incremental $\epsilon$-approximate querying