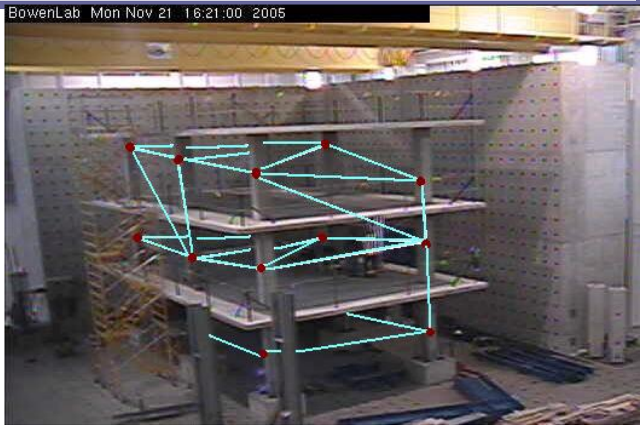
Macroprogramming Sensor Networks for DDDAS Applications

Asad Awan

Department of Computer Science

PURDUE UNIVERSITY

Wireless Sensor Networks



- Integrating computing with the "physical world"
  Sense → Process data → Consume
  – Dynamic data-driven system
- Large-scale self-organized network of tiny low-cost nodes with sensors
  – Resource constrained nodes:
    - CPU: 7 MHz
    - Memory: 4KB data, 128KB program
    - Bandwidth: 32 kbps
    - Power: 2 AA batteries

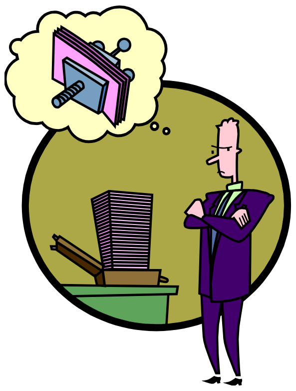
- Challenge: programming the "network" to efficiently collect and process data

WSN: DDDAS Challenges

- Low level details
  - Resource constraints
  - Conserving battery life for long term unattended operation
  - Developing distributed algorithms for self-organization
    - Communication and data routing between nodes
    - Maintain scalability as the number of nodes in the network grow
    - Resilience to dynamic changes (e.g., failures)
- Data processing challenges
  - Spatial and temporal correlation of data from several independent sources
  - Processing of disparate measurement information to estimate/analyze the “actual” physical phenomenon
- Providing a simple & high level interface for end-users to program data processing algorithms and global system behavior without the need to understand low-level issues

Macroprogramming WSNS

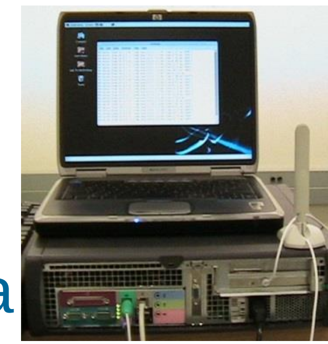
- The traditional approach to DS programming involves writing “network-enabled” programs for each node
  - The program specifies interactions between modules rather than the expected system behavior
  - This paradigm raises several issues:
    - Program development is difficult due to the complexity of *indirectly encoding the system behavior* and *catering to low-level details*
    - Program debugging is difficult due to hidden side effects and the complexity of interactions
    - Lack of a formal distributed behavior specification precludes verification of compliance to “expected” behavioral properties
- Macroprogramming entails programming the system wide behavior of the WSN
  - Hides low system-level details, e.g., hardware interactions, network messaging protocols etc.



Reprogramming?

- Over-the-air reprogramming is a highly desirable feature for WSN systems
  - Deployment costs are high and nodes are often inaccessible or remotely located
- Reasons to reprogram
  - Iterative development cycles
    - Change the fidelity or type of measurements
    - Update data processing features
  - Removal of bugs
- Challenges: (1) Preserving system behavioral properties, (2) Allowing code reuse and versioning, (3) Minimizing update costs

Heterogeneous Sensor Networks



- Resource constraints of nodes necessitates use of heterogeneous devices in the network
  - High data rate sensors, e.g.,  disp. sensor
  - CPU/memory intensive processing, e.g., FFT
  - Bandwidth bottlenecks and radio range
  - Persistent storage
- Heterogeneity can be supported by deploying a hierarchical network
- The macroprogramming architecture should uniformly encompass heterogeneous devices
  - Supporting platform agnostic application development is trivial
- Challenge: Designing an architectural model that scales performance as resources increase

Objective

To develop a second generation operating system suite that facilitates rapid macroprogramming of efficient self-organized distributed data-driven applications for WSN

Outline

- Challenges
- Related work
- Our approach
- Current status
- Future directions

Related Work

- TinyOS
  - Low footprint: applications and OS are tightly coupled
  - Costly reprogramming: update complete node image
  - Aimed at resource constrained nodes
- SOS
  - Interacting modules compose an application
  - OS and modules are loosely coupled
  - Modules can be individually updated: low cost
  - Lack of sufficient safety properties
  - Aimed at resource constrained nodes
- Maté – application specific virtual machine
  - Event driven bytecode modules run over an interpreter
  - Domain specific interpreter
  - Very low cost updates of modules
  - Major revision require costly interpreter updates
  - Ease to program using simple scripting language
  - Implemented for constrained nodes
- Impala
  - Rich routing protocols
  - Rich software adaptation subsystem
  - Aimed at resource rich nodes

Related Work

- TinyDB
  - An application on top of TinyOS
  - Specification of data processing behavior using SQL queries
  - Limitations in behavioral specifications (due to implementation)
  - Difficult to add new features or functionality
  - High footprint
- High level macroprogramming languages
  - Functional and intermediate programming languages
  - Programming interface is restrictive and system mechanisms can not be tuned
  - No mature implementations exist
  - No performance evaluation is available

Outline

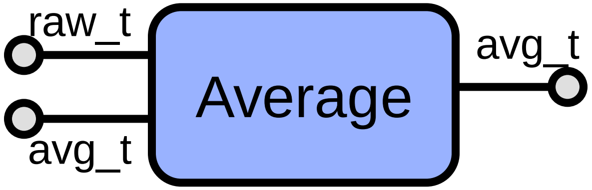
- Challenges
- Related work
- Our approach
- Current status
- Future directions

Application Model

- Macroprogramming (application) centric OS design: top down approach
- Application model:
  - Application is composed of data processing components called processing elements (PE)
  - Application is a specification of data-driven macro system behavior:
    - An annotated connection graph of PEs
    - Capability based naming of devices in the heterogeneous network
    - PE deployment map: assignment of tasks to named devices (sets) in the heterogeneous net.

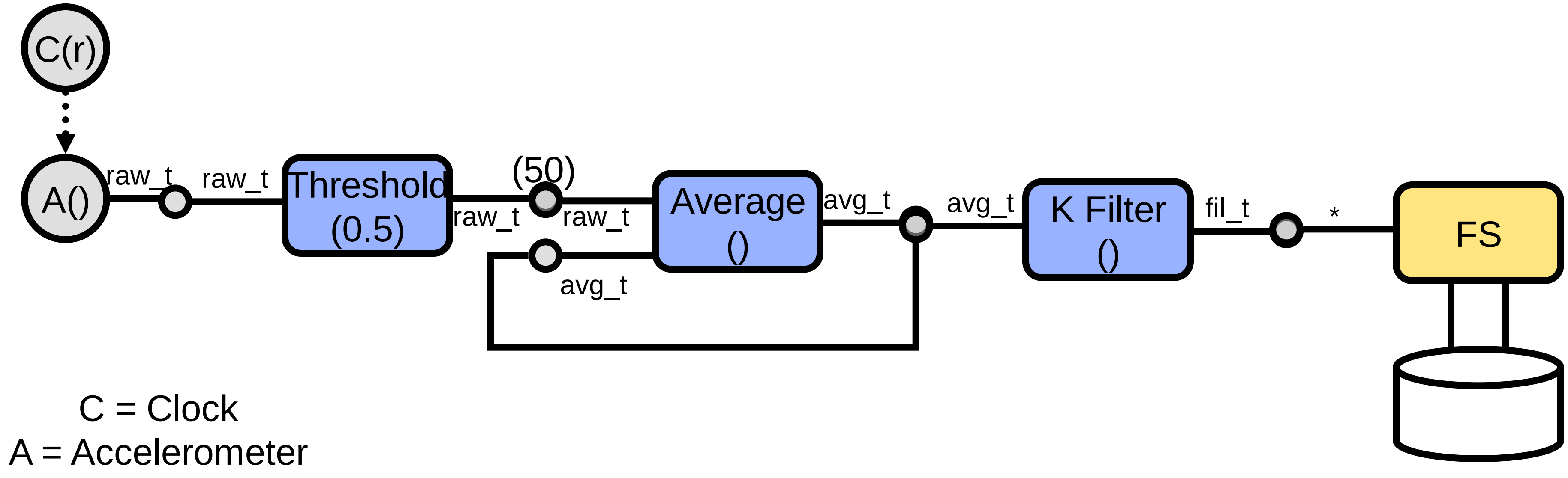
Processing Elements

- Defines "typed" input/output interfaces
  - Implemented as data queues
- Performs a data processing operation on input data
  - Programmed in C
  - Transactional behavior
    - Reads input → processes data → writes output → commits output enqueue & input dequeue
    - Concurrency safety: independent of underlying system's concurrency model
- Conceptually a single unit of execution
  - Isolation properties
    - Enables independent arch, scaling
  - Asynchronous execution
  - Code reusability



Connection Graph

- A data-driven macro specification of system behavior
- Connection of instances of data sources (ports), PEs and services using an annotated graph
- Typed safety: connection interfaces are statically type checked
- Deterministic system behavior
- A simple example:



The Application

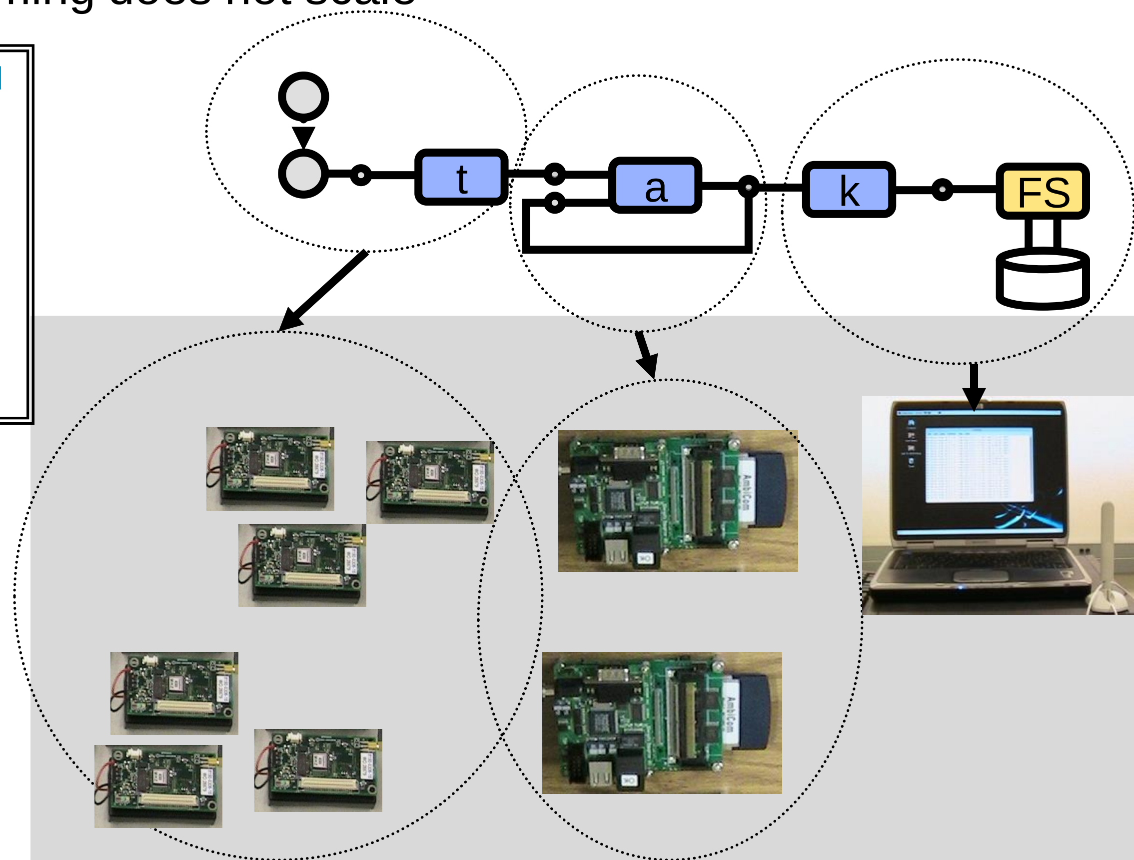
- Device naming (addressing) the last piece in the puzzle:
  - Devices are identified based on their capability sets
    - For example, devices with photo sensors, devices with fast CPU
    - Implemented as masks
    - Individual node naming does not scale

```
@ ACCELEROMETER_SENSOR_NODES: threshold
@ FAST_CPU_NODES: average
@ SERVER_NODE: k_filter, FS

TRIGER(CLOCK(1,rate)[0]) → ACC_SENSOR(2,)[0]
ACC_SENSOR(2,)[0] → threshold(3,0.5)
threshold(3,0.5)[0] –(50)→ average(4,)[0]
average(4,)[0] → k_filter(5,) | –(5)→ average(4,)[1]
k_filter(5,) → FS(1,)
```

The Application

- Device naming (addressing) the last piece in the puzzle:
  - Devices are identified based on their capability sets
    - For example, devices with photo sensors, devices with fast CPU
    - Implemented as masks
    - Individual node naming does not scale

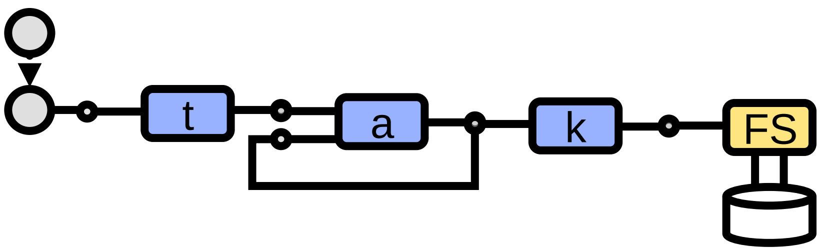
```
@ ACCELEROMETER_SENSOR_NODES: threshold
@ FAST_CPU_NODES: average
@ SERVER_NODE: k_filter, FS

TRIGER(CLOCK(1,rate)[0]) → ACC_SENSOR(2,)[0]
ACC_SENSOR(2,)[0] → threshold(3,0.5)
threshold(3,0.5)[0] –(50)→ average(4,)[0]
average(4,)[0] → k_filter(5,) | –(5)→ average(4,)[1]
k_filter(5,) → FS(1,)
```

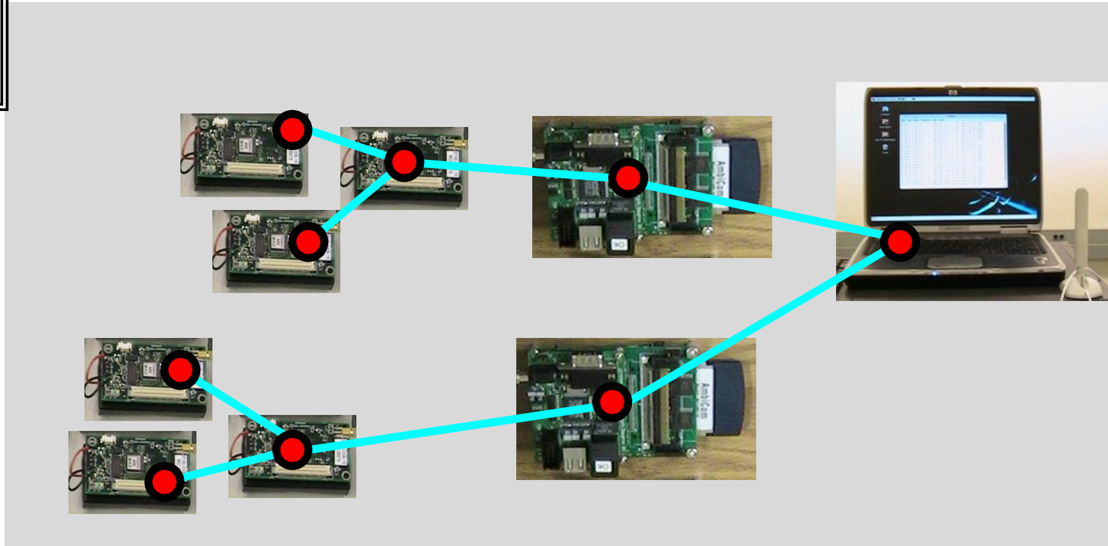


The Application

- Device naming (addressing) the last piece in the puzzle:
  - Devices are identified based on their capability sets
    - For example, devices with photo sensors, devices with fast CPU
    - Implemented as masks
    - Individual node naming does not scale

```
@ ACCELEROMETER_SENSOR_NODES: threshold
@ FAST_CPU_NODES: average
@ SERVER_NODE: k_filter, FS

TRIGER(CLOCK(1,rate)[0]) → ACC_SENSOR(2,)[0]
ACC_SENSOR(2,)[0] → threshold(3,0.5)
threshold(3,0.5)[0] –(50)→ average(4,)[0]
average(4,)[0] → k_filter(5,) | –(5)→ average(4,)[1]
k_filter(5,) → FS(1,)
```

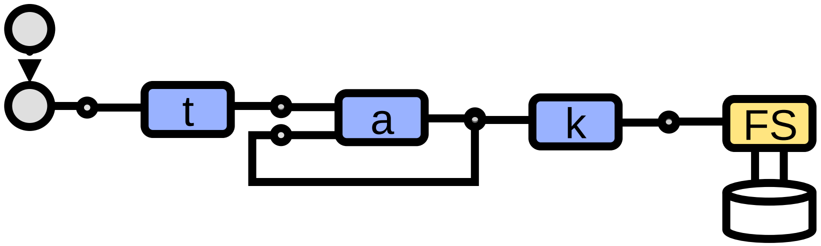




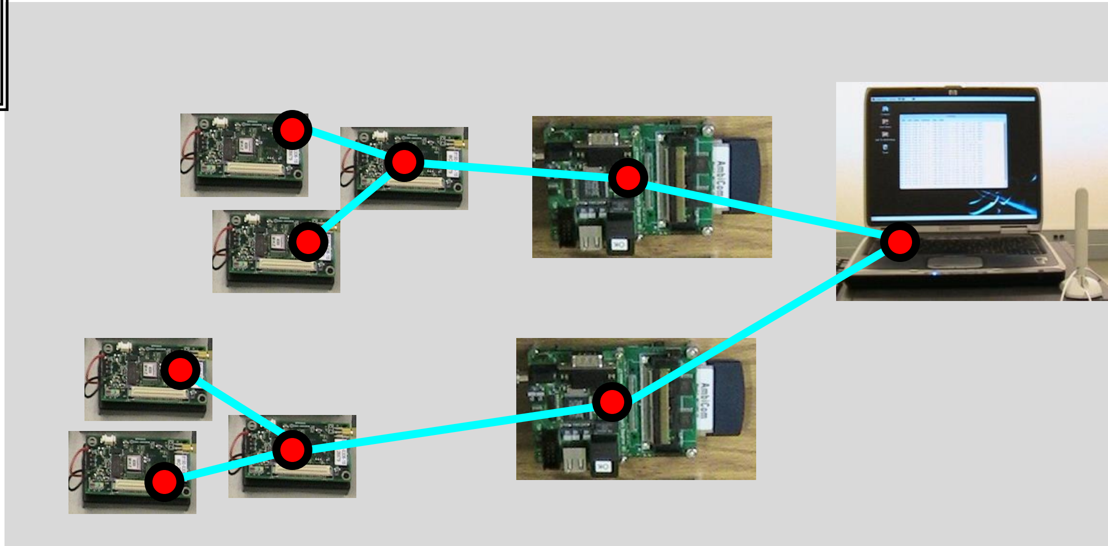
The Application

- Device naming (addressing) the last piece in the puzzle:
  - Devices are identified based on their capability sets
    - For example, devices with photo sensors, devices with fast CPU
    - Implemented as masks
    - Individual node naming does not scale

```
@ ACCELEROMETER_SENSOR_NODES: threshold
@ FAST_CPU_NODES: average
@ SERVER_NODE: k_filter, FS

TRIGER(CLOCK(1,rate)[0]) → ACC_SENSOR(2,)[0]
ACC_SENSOR(2,)[0] → threshold(3,0.5)
threshold(3,0.5)[0] –(50)→ average(4,)[0]
average(4,)[0] → k_filter(5,) | –(5)→ average(4,)[1]
k_filter(5,) → FS(1,)
```

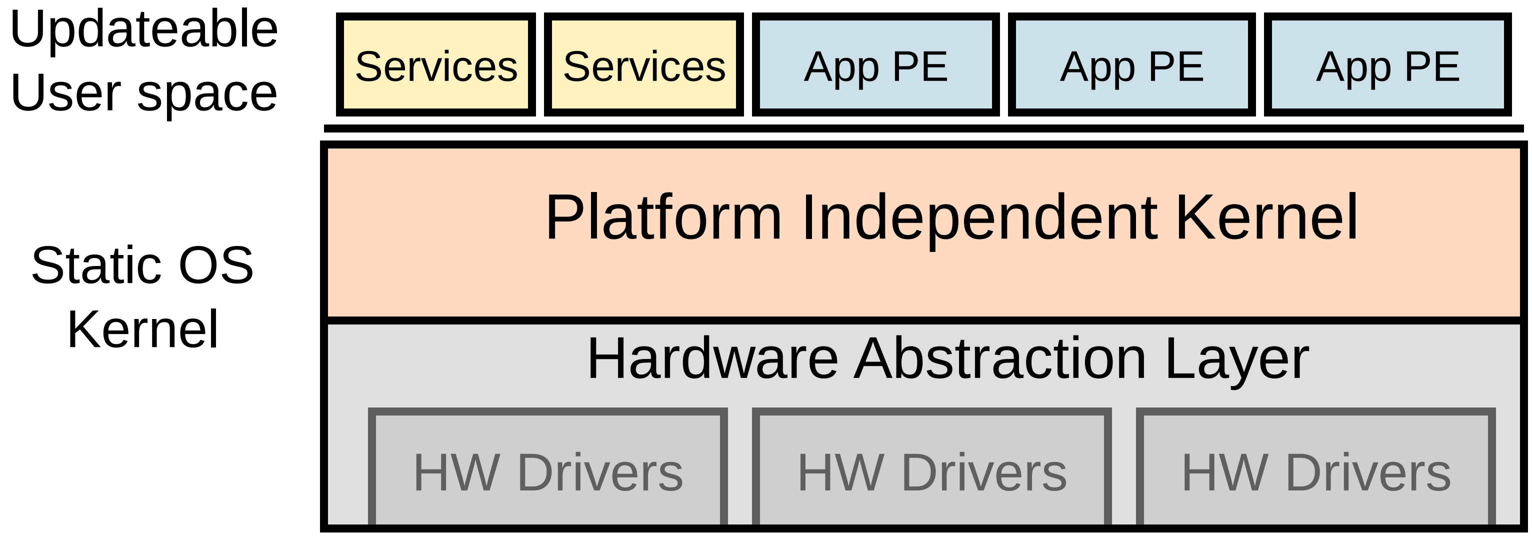


Application updation?



OS Design

- Each node has a static OS kernel
  - Consists of platform depend and platform independent layers
- Each node runs service modules
- Each node runs a subset of the components that compose a macro-application

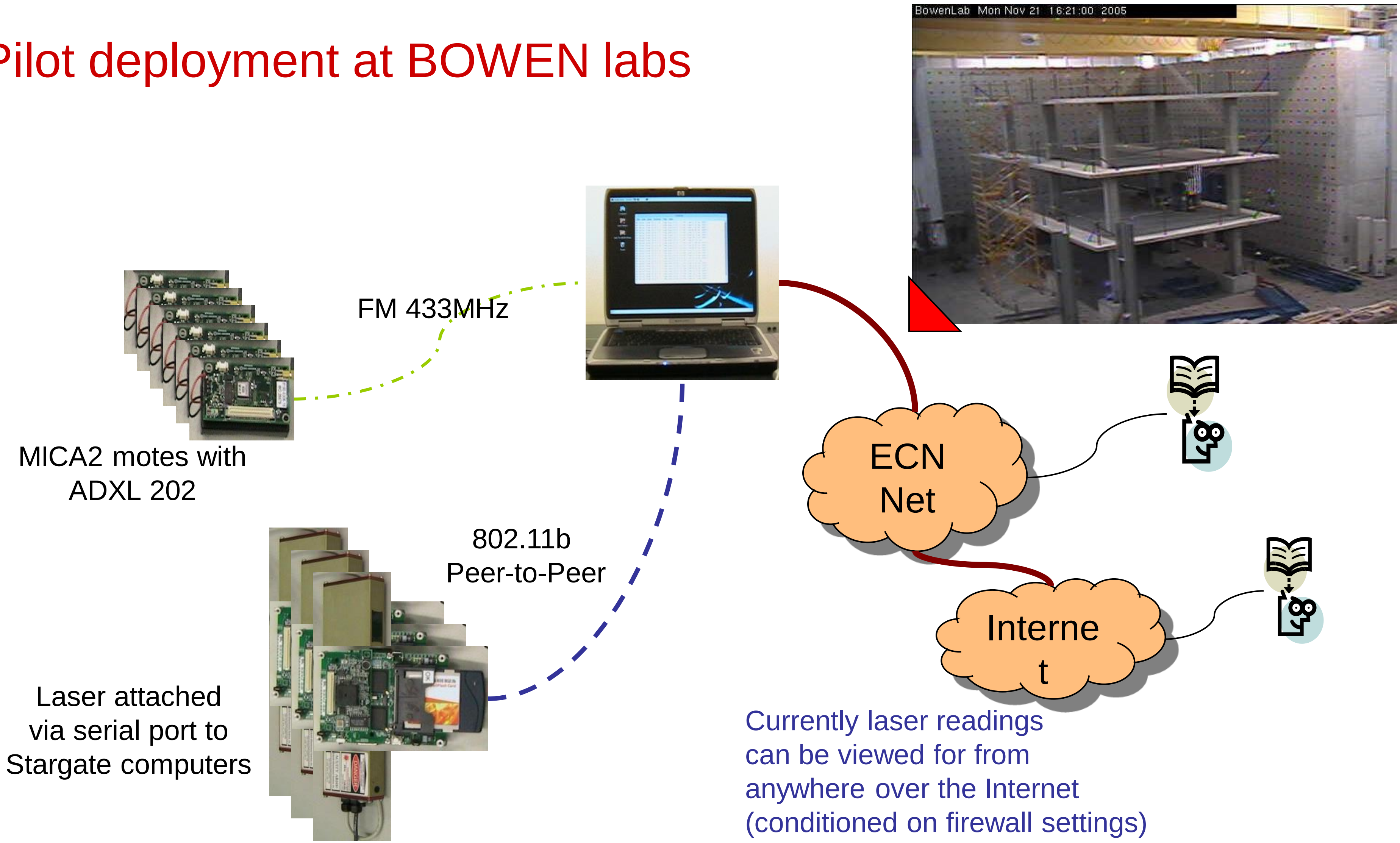


Outline

- Challenges
- Related work
- Our approach
- Current status
- Future directions

WSN @ BOWEN

Pilot deployment at BOWEN labs



Currently laser readings
can be viewed for from
anywhere over the Internet
(conditioned on firewall settings)

Current Status: OS

- We have completed an initial prototype of our operating system for AVR µc (Mica2)
- Introductory paper in ICCS 2006
- Current activities
  - Exhaustive testing and debugging
  - Performance evaluation
  - Enhancing generic routing modules
  - Enhancing application loading service
  - Porting to different platforms (POSIX)

Outline

- Challenges
- Related work
- Our approach
- Current status
- Future directions

Future Directions

- Implement common data processing modules that can be reused
  - E.g., aggregation, filtering, FFT
- Release the OS code
- Complete deployment on a real-world large-scale heterogeneous test bed: BOWEN labs
  - Iteratively develop a DDDAS system for structural health monitoring
- WYSIWYG application design utility, high level functional programming abstractions
- Exploring other application domains
- Exploring distributed algorithms:
  - E.g. PE allocation, routing, aggregation, etc.

Questions?

Thank you!