Active Sensor Networks

Written by Phillip Levis
David Gay
David Culler
Presented by Stephen Nann

Introduction

- Active Sensor Networking
  - In-network dynamic computations.
  - Scripting allows for data processing near source.
  - Reduces network traffic and saves energy.
  - Uncontrolled environments result in bad data.
    - Requires reprogramming of motes already in deployment.
  - Programs need support for a large application domain.
    - Requires a new architecture for in-network programming models.

- Uncontrolled environments result in bad data.
  - Requires reprogramming of motes already in deployment.
  - Programs need support for a large application domain.
    - Requires a new architecture for in-network programming models.

Maté VM Limitations

- Concise programs for only one application domain.
- Explicit synchronization operations have to be implemented by programmer.
- Propagation done with explicit code forwarding and a simple local broadcast.
- Complex and limited instruction set.
- Only 3 types of event handlers.

Base Requirements

- Flexibility
  - Support wide range of applications.
- Concurrency
  - Automatic implementation of safe and effective parallelism.
- Propagation
  - Forward code reliably and efficiently with support for larger programs.

Application Specific Virtual Machines (ASVM)

- Hardware and Software independence.
- Programming with high level languages.
- Reprogrammable with small bytecoded capsules.
- Protects motes from malicious programs.
- customizable instruction set and event triggering.
- Uses Trickle algorithm for propagation.
- Supports large application domain.

Template Components

- Concurrency Manager
  - Handles the execution of code within threads.
  - Ensures race-free and deadlock-free execution.
- Scheduler
  - Executes threads in FIFO Round Robin fashion with fine granularity.
- Capsule Manager
  - Handles the storage and loading of capsules.
  - Propagates capsules.
ASVM Design

- Extension Components
  - Handlers – code routines that run in response to system events.
  - Operations - functional code units.
  - Capsules – store the code sections for propagation.

ASVM Concurrency Manager

- Operations note access to a shared resource.
- New capsules are analyzed for shared resources.
- Handlers are only allowed to run if it can access all the required shared resources.
- Two Phase Locking
  - Start – hold all resources, can release with execution.
  - Finish – releases all of the held resources.
- Reboots when new code arrives to reset all variables.
- Option for explicit synchronization.

ASVM Scheduler

- Threads are executed by fetching the next bytecode from the capsule store.
- byteLength allows the Scheduler to keep track of the threads PC.
- Allows multiple threads to run concurrently.

ASVM Extensions

- Handlers
  - Defined by user to react to certain system events.
  - Connected to operations.
  - Trigger a thread to execute code in response to a specified event.
  - Examples: Timers, Route forwarding request, ASVM boot.
**ASVM Extensions**

- **Operations**
  - Defined by user to perform application specific computations.
  - Primitives – language specific
    - Ex. jump
    - ASVM supports a particular language by including the primitives it uses to compile.
  - Functions – language independent
    - Ex. Send()
    - User tailors to a specific application by implementing appropriate functions and handlers.

- **Primitives – language specific**
  - Example: jump

- **Functions – language independent**
  - Example: Send()

**Building ASVM**

- **Description File with scripts defined.**
- **Specify 3 things:**
  - Language
  - Functions
  - Handlers
- **The toolchain generates TOS source code and maps opcodes.**

**Evaluation: Flexibility**

- **Languages**
  - TinyScript
    - BASIC-like dynamic typing.
    - No dynamic allocation.
  - Motlle
    - C-like syntax.
    - Supports vectors, lists, strings, and first-class functions.
    - All handlers transmitted in one capsule. No incremental changes to running programs.

**Evaluation: Applications**

- **RegionsVM**
  - Vehicle tracking written in TinyScript.
- **QueryVM**
  - Periodic data collection with in-network aggregation.
  - Written in TinySQL.
  - Aggregation libraries compiled to Motlle.
  - TinySQL supports new aggregation queries written in Motlle.

**Evaluation: Concurrency**

- 24 shared resources and a 128 byte handler.
- Locking and Unlocking
  - Few microseconds.
- Looking for a shared resource
  - < 1 ms
- Manager can provide shared data safety with little overhead cost.

**Evaluation: Propagation**

- **Trickle algorithm.**
- 100 byte handler injected.
- Average reprogram ≈ 40 s.
- Worst case ≈ 85 s.
- ASVM only had to send 71 bytes handler rather than 19k image.
Critique

- Difficult to predict what data will look like from each mote.
- Programming in high level language increases execution time due to interpretation.
- Initial code image is much larger.
- Increased overhead due to race and deadlock checking.
- Propagation technique is slow and requires 3 transmissions.

Maté Comparisons

- ASVM has 6% more overhead.
- ASVM has 20% better energy efficiency with propagation.
- ASVM provides customizable instructions and event handling.
- ASVM provides automatic race and deadlock checking.
- ASVM provides a more energy efficient propagation algorithm with support for larger programs.
- ASVM allows for greater language and application flexibility.

Questions