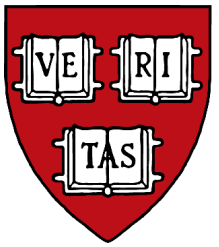
Market-Based Programming Paradigms for Sensor Networks

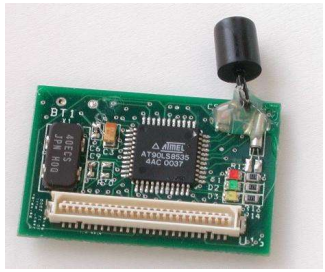
Geoff Mainland, David C. Parkes, and Matt Welsh
Division of Engineering and Applied Sciences
Harvard University

{mainland, parkes, mdw}@eecs.harvard.edu



Introduction: Sensor Networks

WeC (1999)



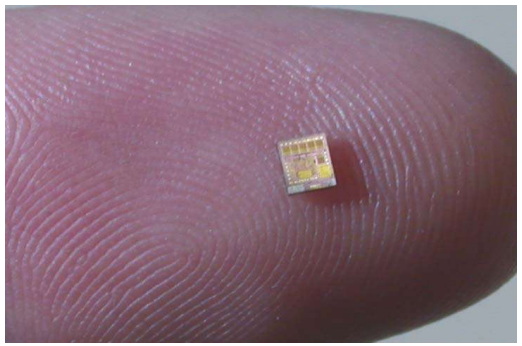
Rene (2000)

Dot (2001)

Integration of sensing, communication, and computation

- Low-power, wireless “motes” with CPU, sensors, low-bitrate radio

MICA (2002)



Speck (2003)

Telos (2004)

The problem...

Programming sensor networks is **hard!!**

Programmer has to deal with many low-level details:

- Scheduling sampling and communication on each sensor node
- Ensuring reliable operation despite message loss and node failure
- Managing limited energy and radio bandwidth

How do we do this today?

- Low-level programming models (e.g., TinyOS)
- Higher-level tools (e.g., TinyDB)

Both approaches require *manual* resource management

- This is limiting and often error-prone

Example application: Tracking







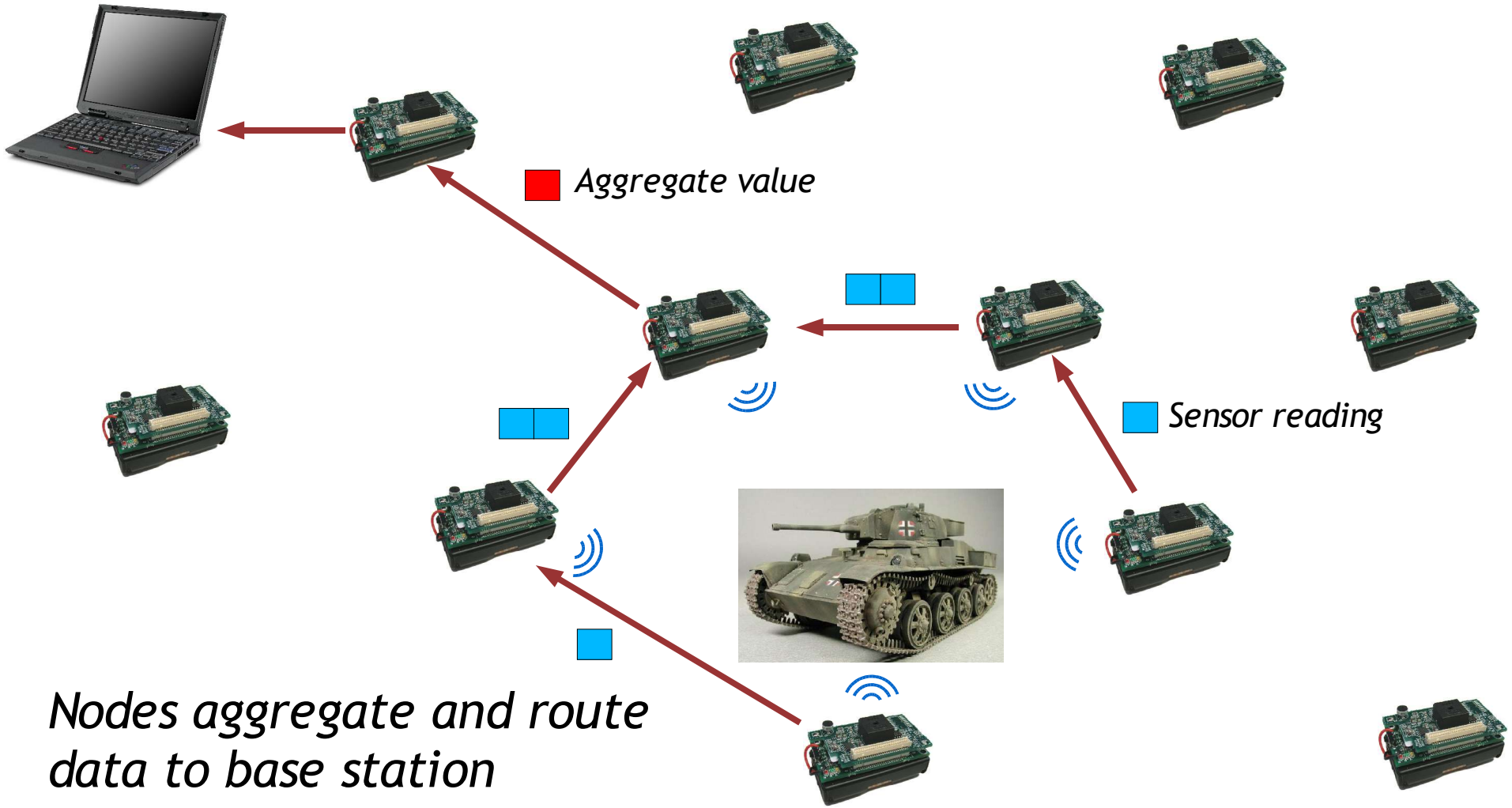






Nearby nodes detect target

Example application: Tracking



How is this done today?

Programmers define a *static schedule* for each node

- e.g., "Every *N* seconds, sample the magnetometer, then transmit data if above threshold *T*"

Might use more sophisticated data aggregation techniques

- Nodes combine results into a single aggregate value to save bandwidth
- e.g., Compute the centroid of sensor readings within some area

Assumes communication is coordinated across nodes

- e.g., That parent in routing tree is listening for incoming messages

The actions performed by each node have a deep impact on:

- Accuracy and latency of data returned to the base station
- Radio contention
- Battery lifetime

Problem Definition

Static scheduling of sensor nodes is clearly suboptimal

Does not take into account spatial or topological differences

- Nodes closer to target might need to sample more frequently
- Nodes closer to the root of a spanning tree use radio more than leaves

Ideally, nodes should *self-schedule* based on their local state

- Node should decide locally which operations to perform and how often
- Driven by interaction with environment

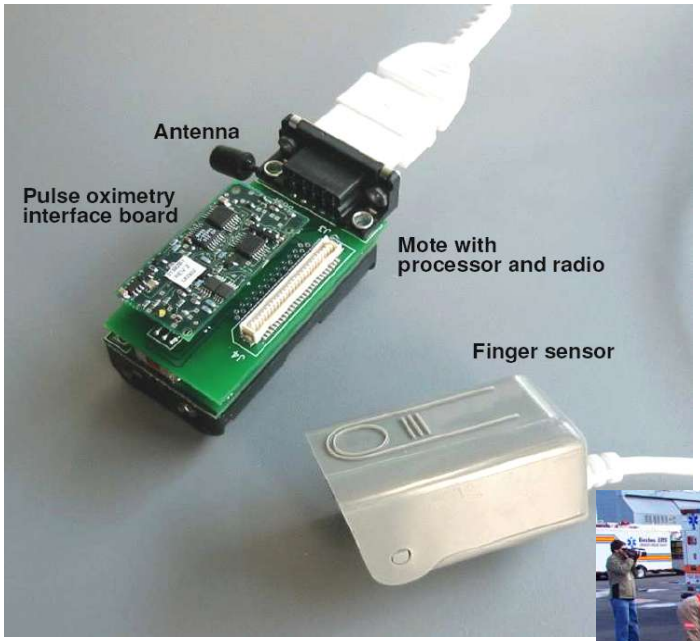
Node operation should *adapt* to changing conditions

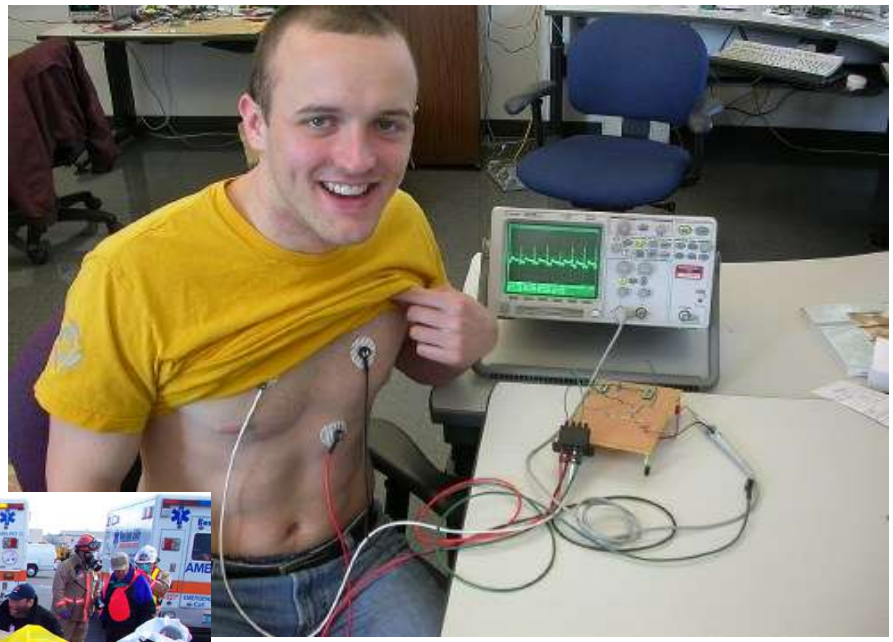
- e.g., If interesting event happens nearby, node might ramp up sampling rate

Applications: Emergency Medical Care

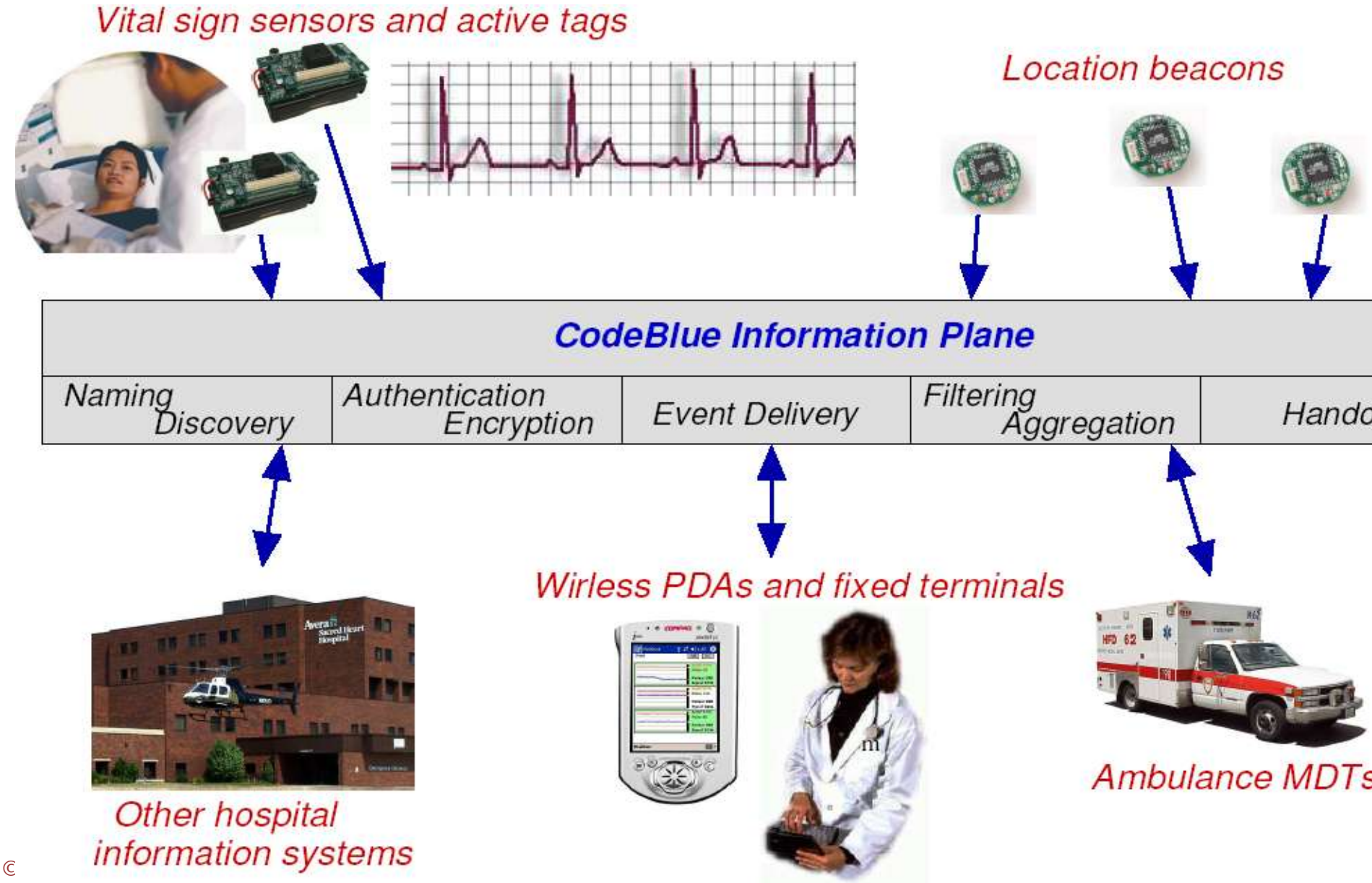
Medics outfit patients with wireless medical sensors

- Deliver real-time data on vital signs, location, etc. to multiple receivers
- Requires better tools for managing energy use and radio congestion





CodeBlue: Protocols and Services for Wireless Medical Devices



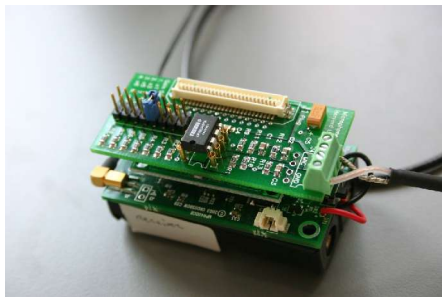
Monitoring Volcanic Eruptions

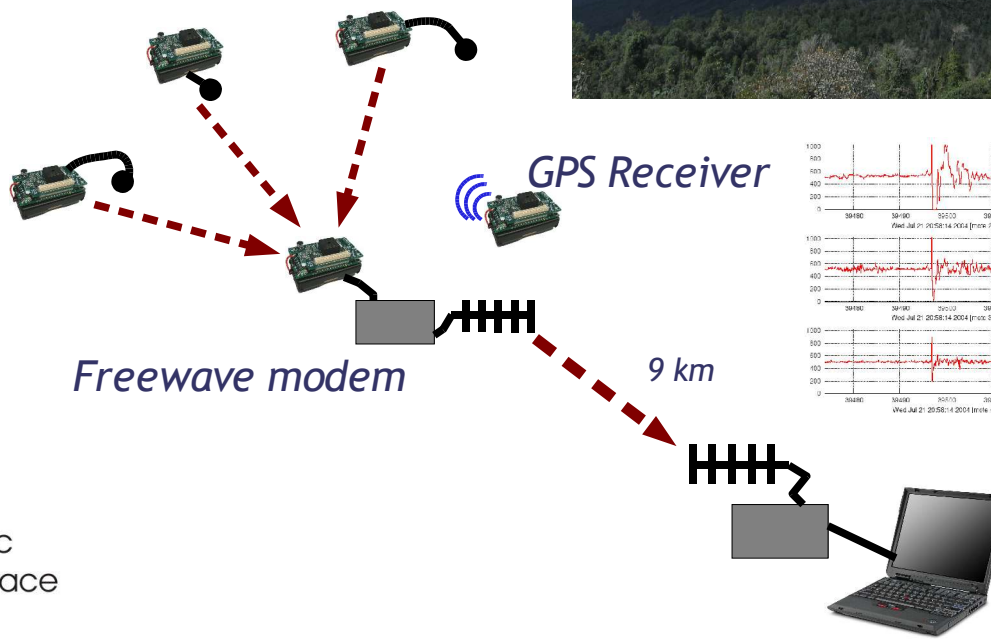
Infrasonic microphone array to monitor volcanic activity

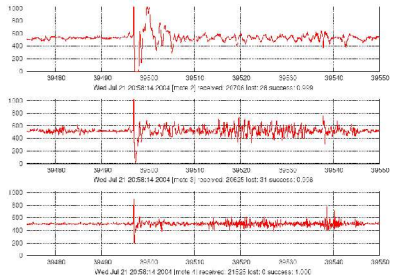
- Deployed at Volcán Tungurahua, Ecuador, July '04

Future plans call for much larger array

- Need tools to manage energy, bandwidth across multiple competing applications/users







Our Inspiration: Markets

Can we use ideas from economics to control the operation of a sensor network?

Why might this be attractive?

- Markets operate in a (mostly) decentralized fashion
- Individual agents make locally greedy (profit-making) decisions
- Markets capture the notion of globally efficient resource allocations

Inspiration: Wellman's work on market-oriented programming

- Demonstrated idea of allocating resources in a distributed system using prices

M. P. Wellman, Market-oriented programming: Some early lessons.
In S. Clearwater, editor, *Market-based Control: A Paradigm for Distributed Resource Allocation.*
World Scientific, 1996.

Market-Based Macroprogramming

Basic model:

- Nodes act as agents that sell *goods* (such as sensor readings or routed msgs)
- Each good is produced by an associated *action* that produces it
- Nodes attempt to *maximize their profit,* subject to energy constraints

Each good has an associated *price*

- Network is “programmed” by setting prices for each good

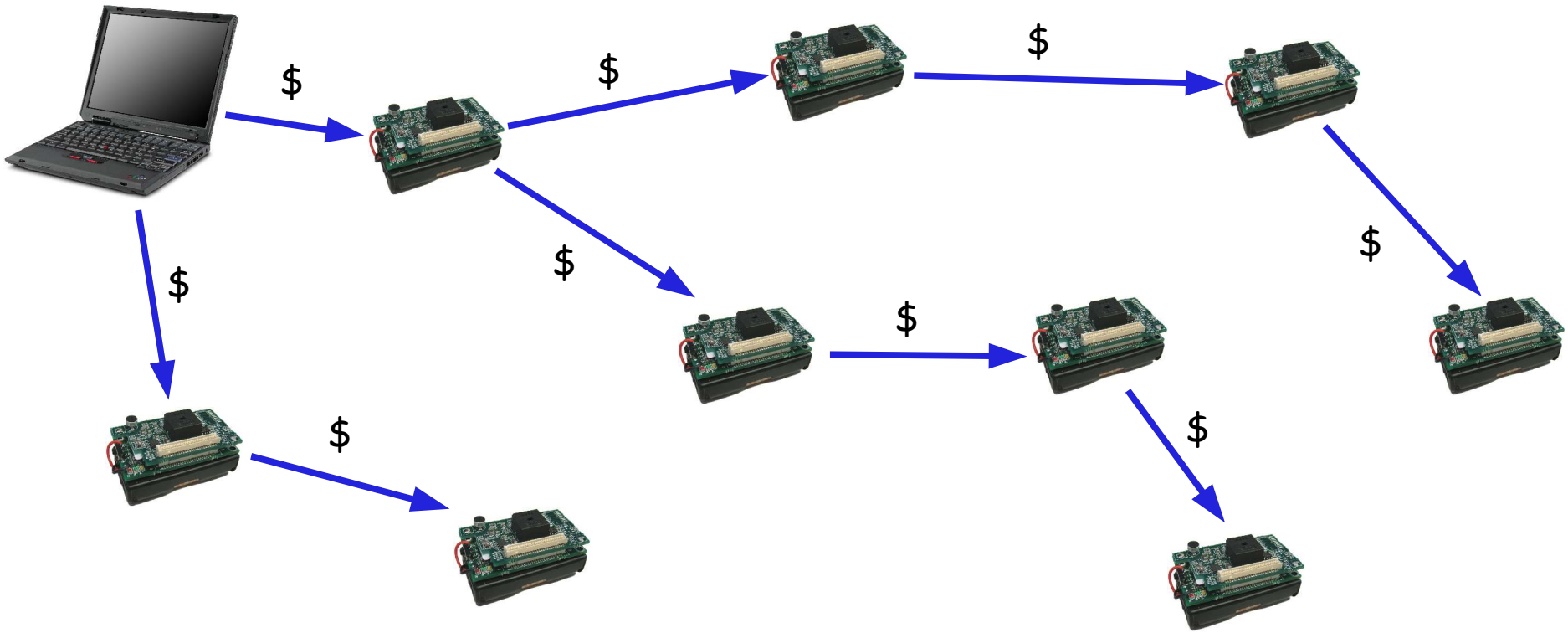
Each action has an associated energy cost

- e.g., Cost to sample a sensor << Cost to transmit a radio message

Price Propagation

First step: Flood prices for each good to the network

- Several efficient protocols for this (e.g., Trickle)
- Can readily update prices on the fly



Utility Function

Nodes continuously select the action with the greatest *utility*

The utility for an action is a function of:

Price

- Advertised by base station

Energy availability

- Taking an action must stay within energy budget

Other dependencies

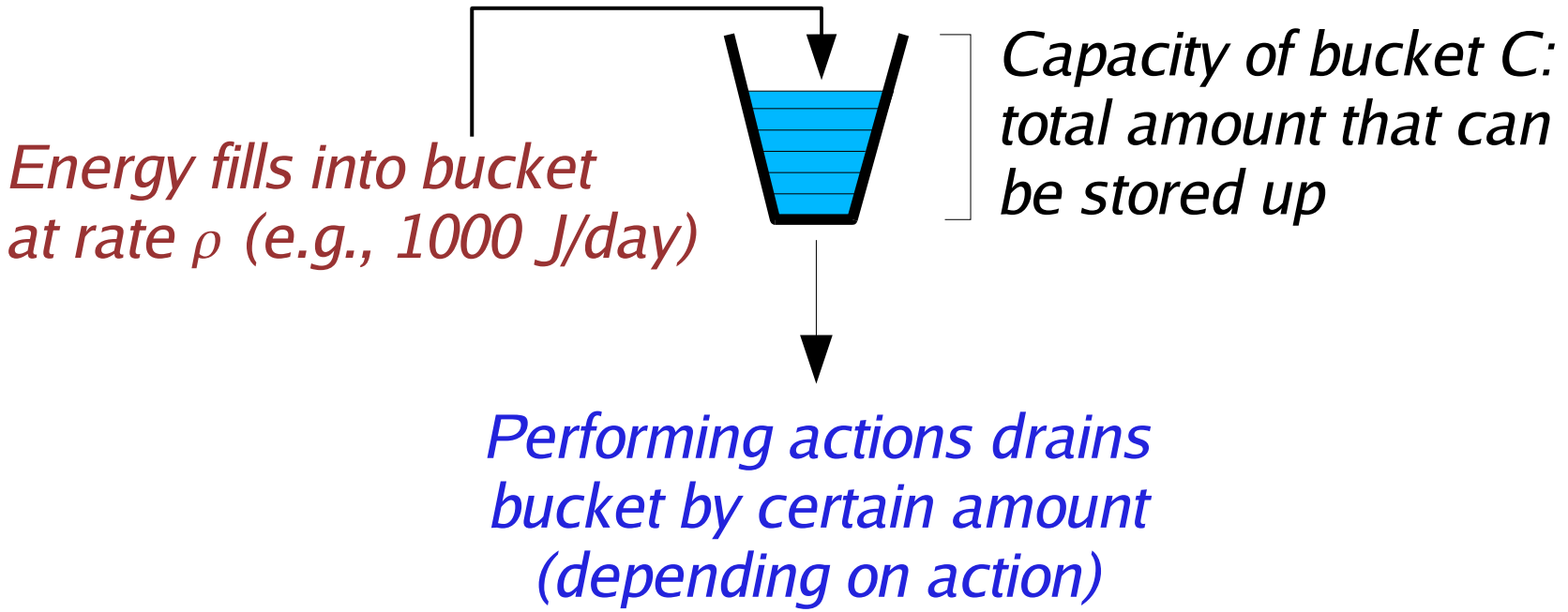
- Cannot aggregate data until multiple samples have been received
- Cannot transmit if nothing in local buffer

Energy Budget

Most important constraint on node operation is energy

We model the energy budget for each node as a *token bucket*

- Rate of bucket fill determines average rate of energy use
- Capacity of bucket bounds "burst size"



Learning Expected Profits

The utility function is the *expected profit* for taking an action.

$$u(a) = \begin{cases} \beta(a)\ price(a) & \text{if the action's dependencies have been met} \\ 0 & \text{otherwise} \end{cases}$$

$\beta(a)$ is the estimated probability of payment for action a

- An action is only profitable if it is "useful"
- e.g., Only get paid to transmit if another node is listening

$\beta(a)$ is learned using an exponentially weighted moving average

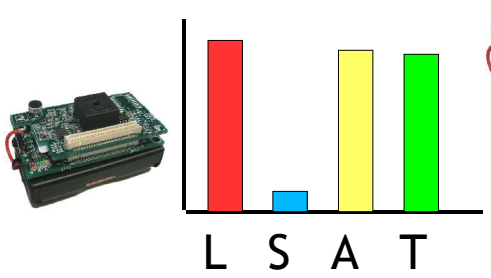
Action selection uses ϵ-greedy reinforcement learning

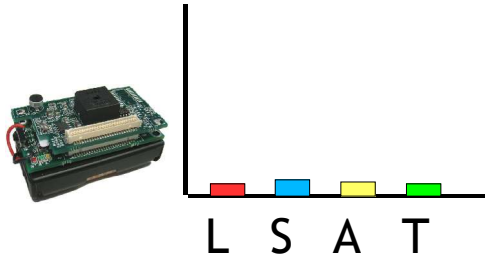
- Node usually takes action with highest expected profit
- However, node takes random action with (small) probability ϵ

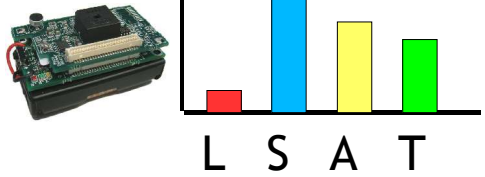
Utility Function Example

Utility functions vary depending on node's position in network

- Nodes near target have high utility for sampling
- Nodes along routing path have high utilities for listening and sending







Design Issues

Nodes operate using a very simple program

- Uses only local knowledge: energy state, samples, etc.

Network automatically adapts to changing conditions

- Nodes take actions that they individually find to be profitable
- Learning strategy for utility function allows runtime adaptivity

Payments lead to a natural equilibrium

- Example: Fraction of nodes transmitting and listening for messages is *balanced*

Prices yield control over network-wide behavior

- But, some question about how general this is...

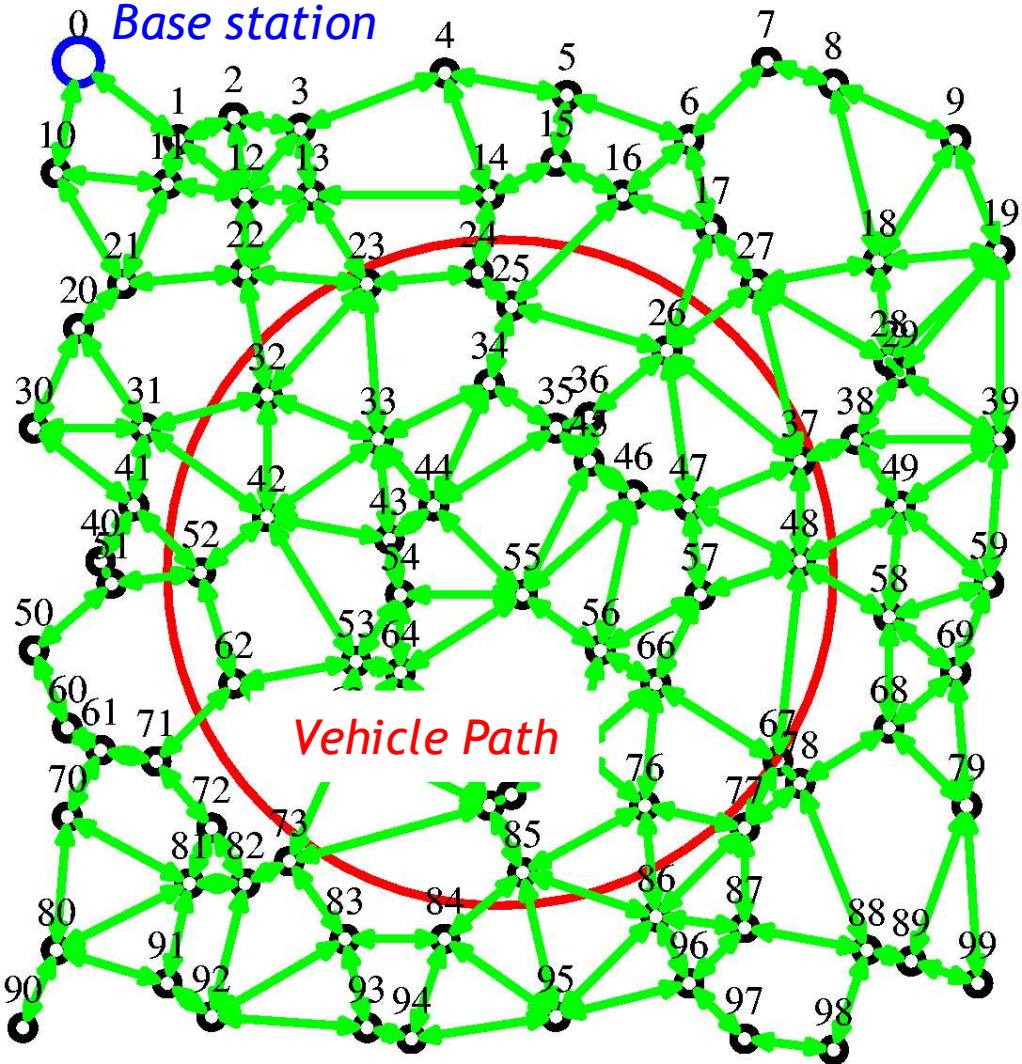
Evaluation Methodology

Simulation based on TinyOS environment

- Captures realistic hardware-level effects
- Target travels in circular path
- Routing using GPSR to base station

Evaluation goals:

- Can markets effectively control the network?
- How efficient is the resulting resource allocation?
- How well do nodes adapt to changing conditions?



Comparison to Alternatives

Implemented three alternative tracking systems:

Static Scheduling

- Nodes periodically sample, listen, aggregate, transmit, and sleep
- All nodes operate at the same rate, calculated offline to meet energy budget
  - *This is conservative: Nodes may not use entire energy budget*

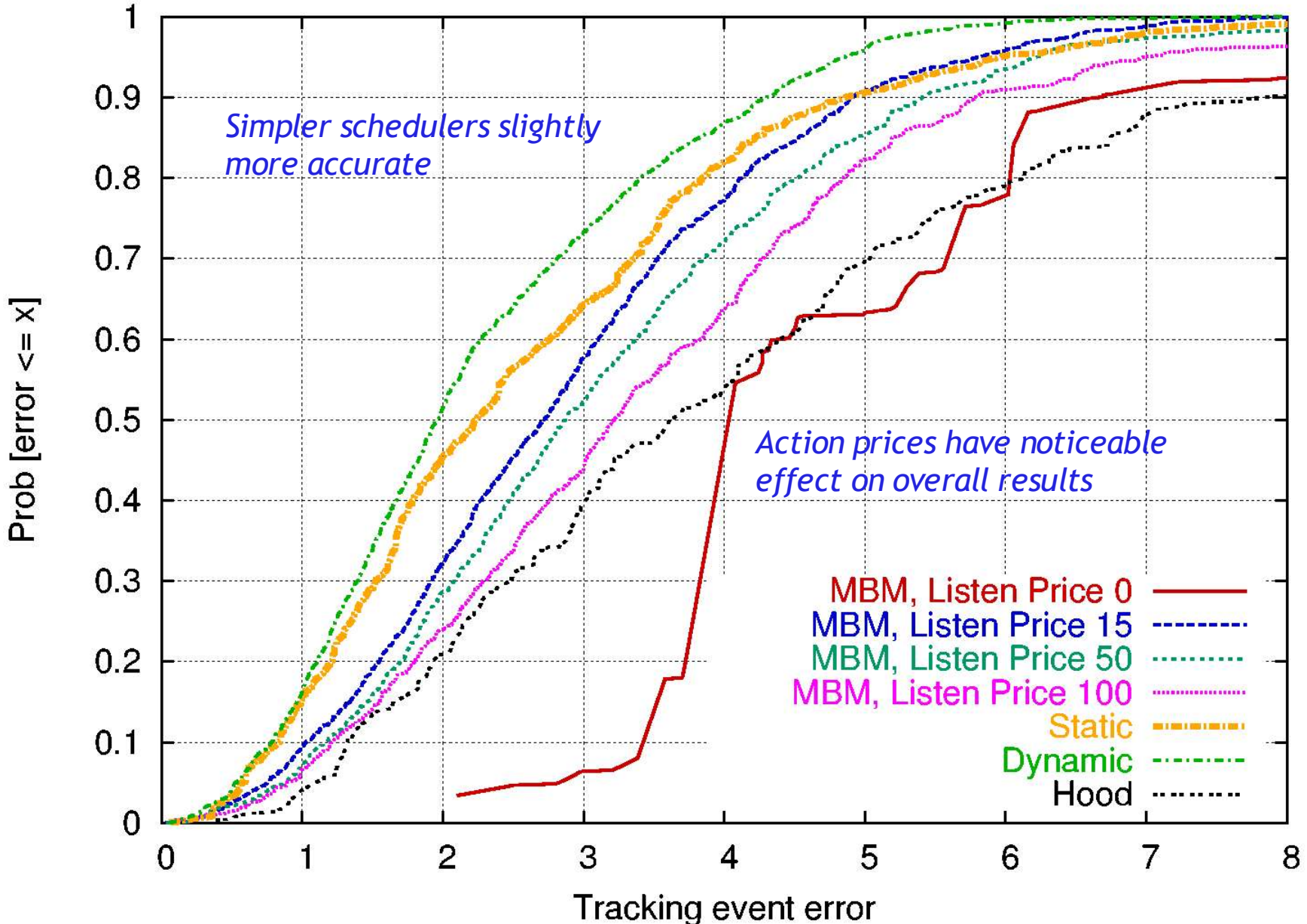
Dynamic Scheduling

- Nodes dynamically tune processing rate based on available energy

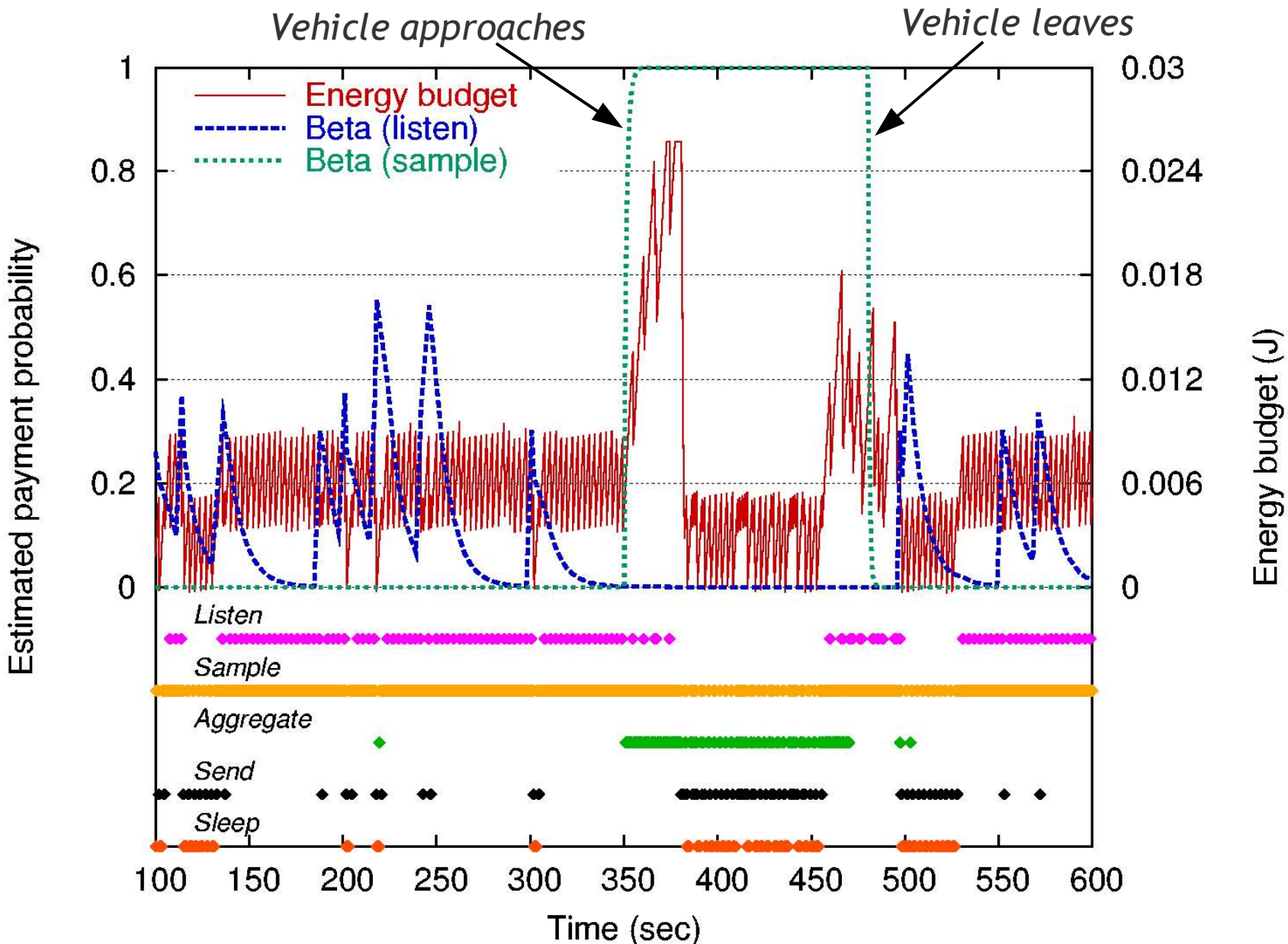
Hoods [Whitehouse et al., MobiSys'04]

- Uses group communication to collect and process sensor readings
- Processing rate same as dynamically scheduled tracker

Overall Tracking Performance

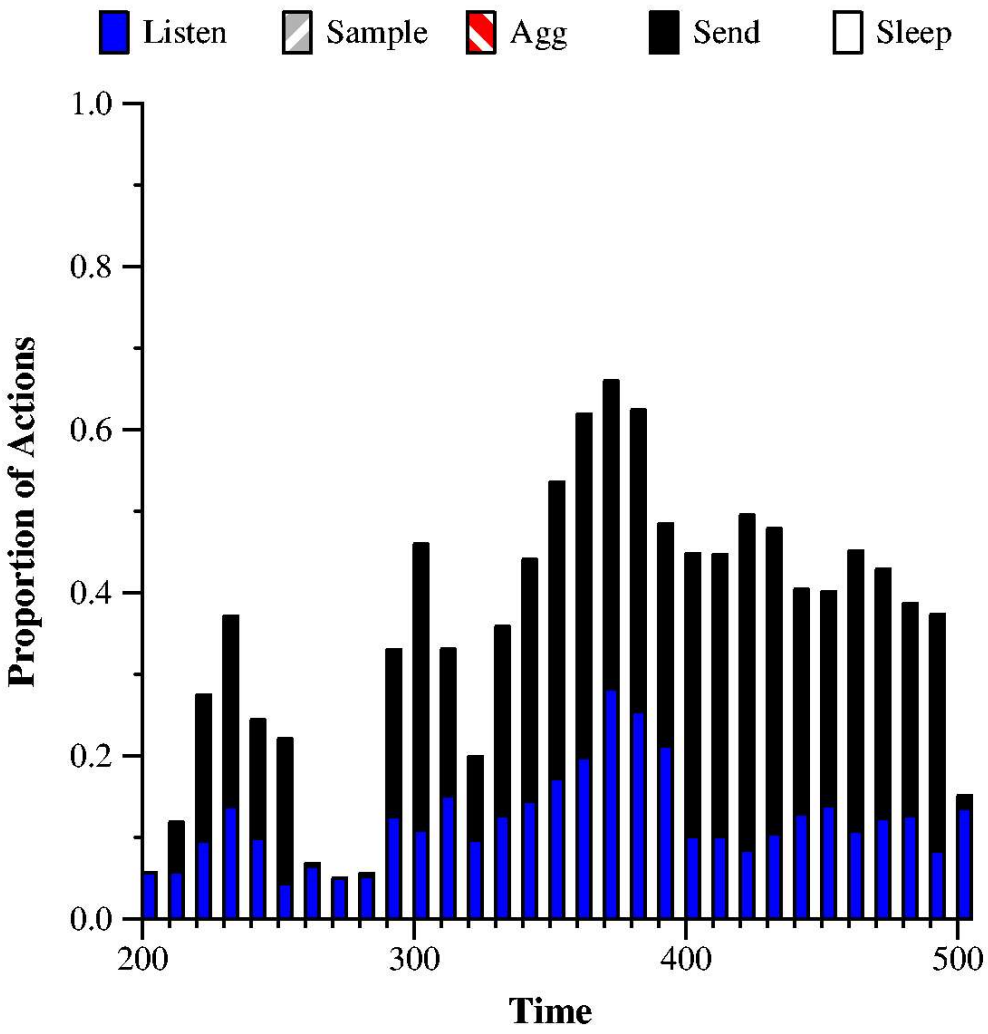


Node Behavior over Time

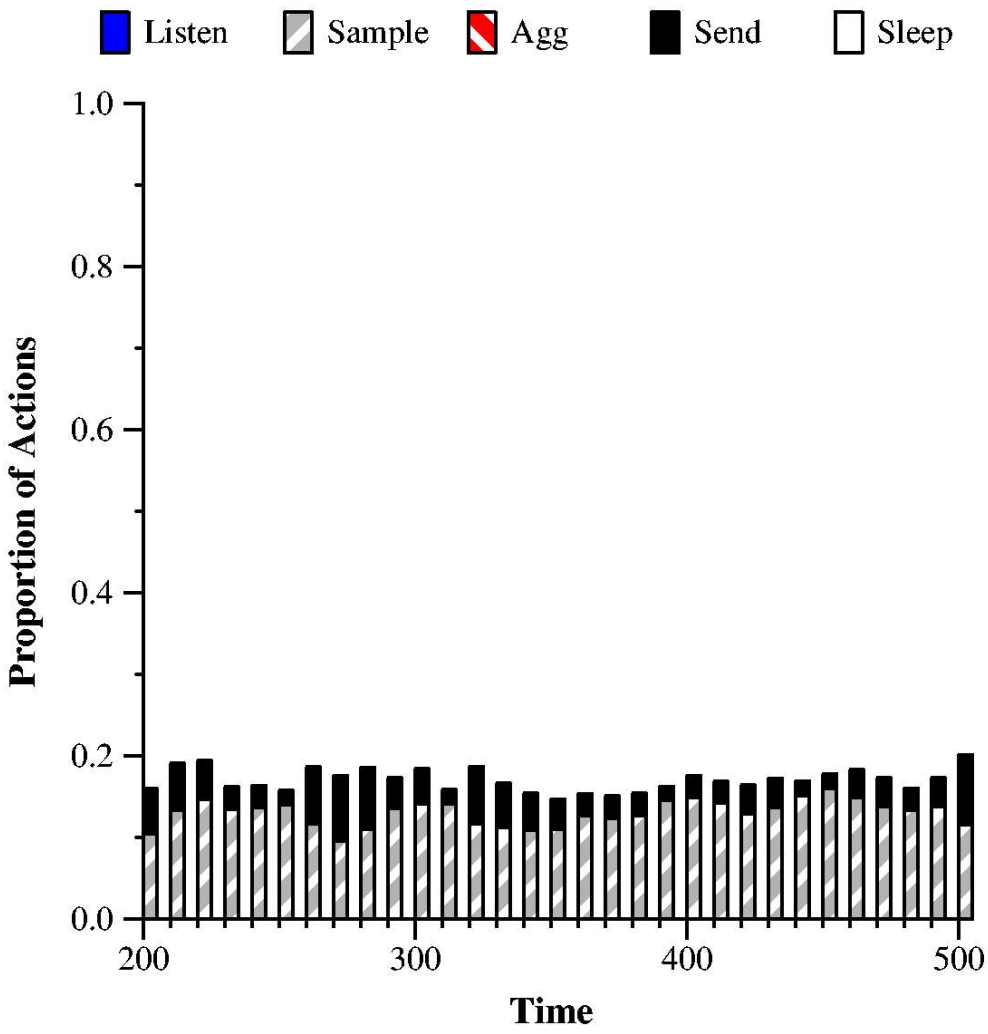


Effect of Prices on Action Choice

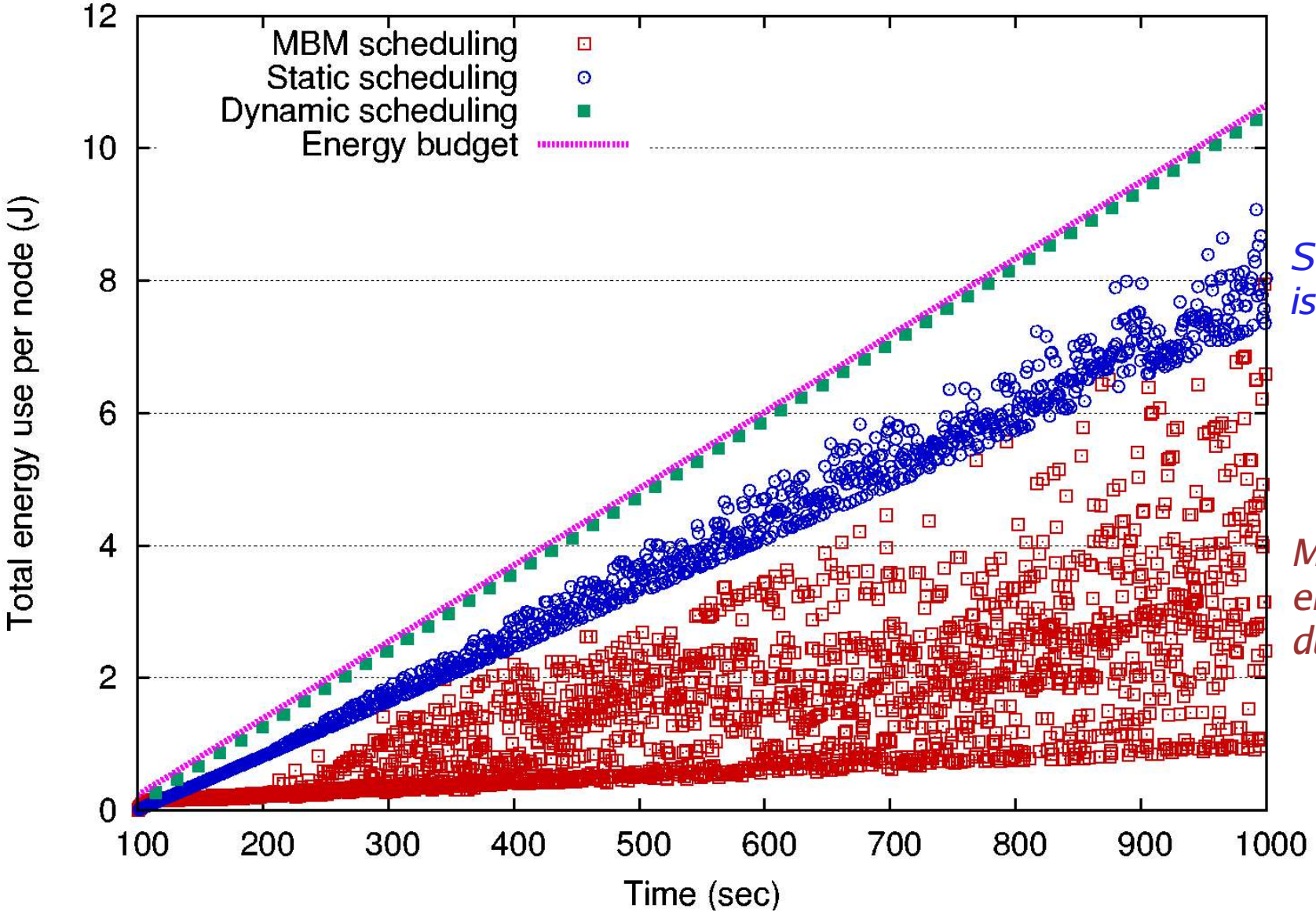
"Routers"



"Sensors"



Energy Efficiency



Static scheduling is conservative

MBM nodes save energy by tuning duty cycle

Energy Efficiency

We calculate the energy efficiency for each scheme as:

$$Efficiency = \frac{Total\ energy\ used\ to\ collect\ and\ route\ sensor\ data}{Total\ energy\ used\ by\ network}$$

Scheduling	*1000 J/day*	*3000 J/day*
Market-based	36%	95%
Static	21%	22%
Dynamic	13%	22%

- Market-based scheduling allows nodes to scale down duty cycles dynamically

High-Level Programming Languages

Ultimate goal: Program the network *globally*

- Automatically compile down to node-level programs
- Save programmer the effort of decomposing distributed tasks

Regiment: Functional Macroprogramming Language

- Global, high-level language for sensor networks
- Based on functional reactive programming concepts

Basic data abstraction: *region streams*

- A time-varying collection of node state
- e.g., “All sensor nodes within area R” form a *region*
- The set of their periodic data samples form a *region stream*

Regiment Example

```
let aboveThresh(p,x) = p > THRESHOLD
    read node = (read_sensor node,
                 get_location node)
in compute_centroid
      (rfilter aboveThresh (smap read world))
```

Vehicle tracking system implemented in a few lines of code

- Simple primitives for declaring, filtering and aggregating sensor streams
- Compiles down to efficient node-level programs

Next steps: Integrate with market-based resource management

Future Directions

Richer pricing models

- Current scheme very myopic: Nodes always pick most profitable action
- Would like to price valuable *sequences* of actions
  - *e.g., Must sample multiple times before aggregating*

Use market to allocate resources to different users

- Each network user can pay for different sets of actions
- Use *equilibrium pricing* to seek Pareto optimal resource allocations

Adjust prices dynamically in response to events in the network

- e.g., Program nodes on edge of network as “sentries”
- When sentry detects arrival of vehicle, sends new prices to neighboring nodes

Conclusions

Sensor networks need new tools for managing resources

- Energy and bandwidth are very constrained
- Manual scheduling and allocation is difficult

Our approach: Market-based macroprogramming

- Use economic ideas to allocate resources in a dynamic, decentralized fashion
- Nodes act as self-interested agents in a virtual “market”
- MBM is effective at controlling the global behavior of the network

Used as tool for resource allocation and as programming model

- Nodes learn ideal behavior to provide value to network
- Retask the network by distributing price vectors, not program code

For more information...

http://www.eecs.harvard.edu/~mdw/proj/mp

mdw@eecs.harvard.edu

Thank you!