Sensor Networks

Abhishek Sreenath & Shannon Chen

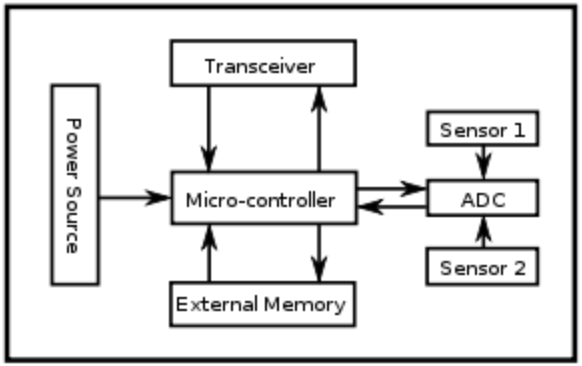
February 21, 2013

Sensor Networks

- A sensor network is a group of specialized transducers with a communications infrastructure intended to monitor and record conditions at diverse locations

Sensor Node/Motes

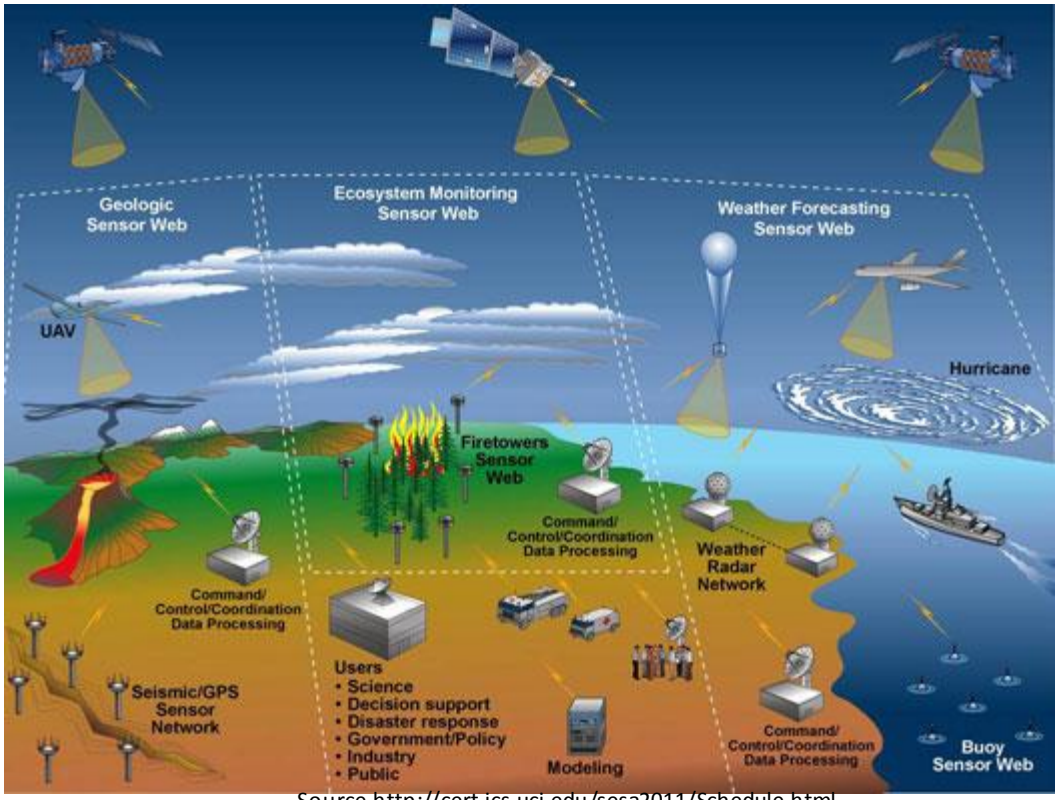
- Advances in processor , memory and radio technologies have enabled small nodes that can perform communication and computation.
- When the computational power is coupled with transducers, they can be used for sensing physical phenomena



The typical architecture of the sensor node.[1]

[1] Source : http://en.wikipedia.org/wiki/Sensor_node

Distributed Sensor Networks



Source http://cert.ics.uci.edu/sesa2011/Schedule.html

– Challenges in Sensor Networks :
  - Traditional networking models are not adaptable for sensor networks.
  - Motes have low resources. It is necessary to design operating systems specially for sensors. Tiny OS is a popular OS for sensors.

"Directed Diffusion : A Scalable and Robust Communication Paradigm for Sensor Networks"

Chalermek Intanagonwiwat,
Ramesh Govindan and
Deborah Estrin

MobiCom 2000

Energy Conservation is the key…

- 2 Types of Typical Sensor Networks
  - Large , complex sensors deployed far away from the phenomenon to be sensed.
  - Network of sensors having low processing power transmit time series of sensed phenomenon to a central node(s)

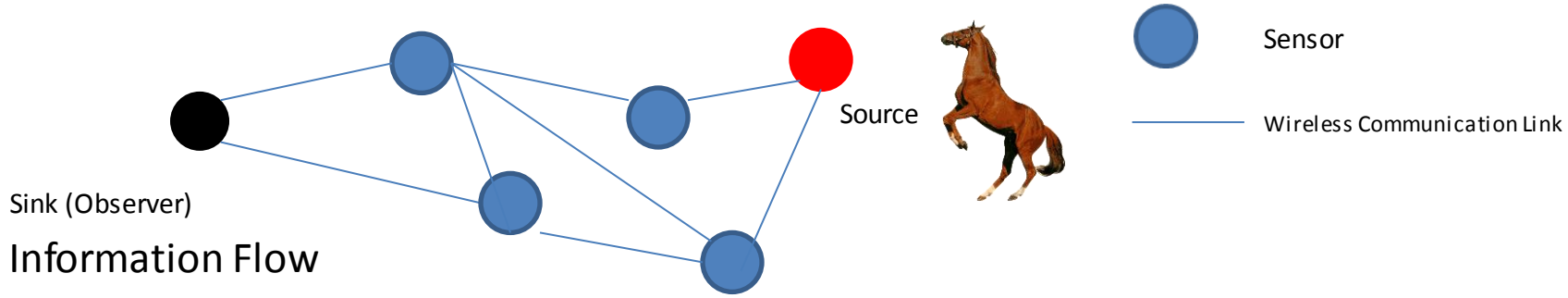
- Sensors expected to have lifetimes of several days – but mostly run on batteries.

- Use hop-by-hop communication – Radio consumes a lot of energy

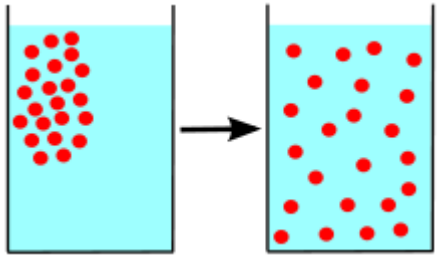
- Transmitting time series of sensed phenomena is inefficient

Directed Diffusion

- Nodes in sensor networks are anonymous. No way to address individual nodes
- Directed Diffusion – Dissemination mechanism for tasks and events
  - Ex : Animal Tracking



- Information Flow
  - Monitoring Task information is propagated to sensors.
  - Sensors collect the requested information and reply to sender
  - Replies routed back to source on reverse path
  - Nodes perform aggregation of information while forwarding replies



Traditional Networking v/s Directed Diffusion – A Comparison

Traditional Networking	Directed Diffusion
End – to – end communication	Neighbor-to-neighbor communication
Use routing tables	No routing tables – Paths chosen empirically
Global view of network	No global view of network. Nodes only know of neighbors – Enables plug and play operation.
No application related semantics	Application semantics built into communication model

The directed diffusion paradigm is similar to communication mechanisms in nature , for ex, ant colonies. Hence, this paradigm is highly scalable and robust .

Data Naming

- Task descriptions & responses are named
  - Use attribute-value pairs.
  - Task Description specifies an interest for data matching the parameters specified

Task Description

```
type =  four-legged animal    //detect animal location
interval = 20 ms              //send back every 20 ms
duration = 10 secs            // for the next 10 secs
rect = [100, -100, 200,400]   //from sensors within rectangle
```

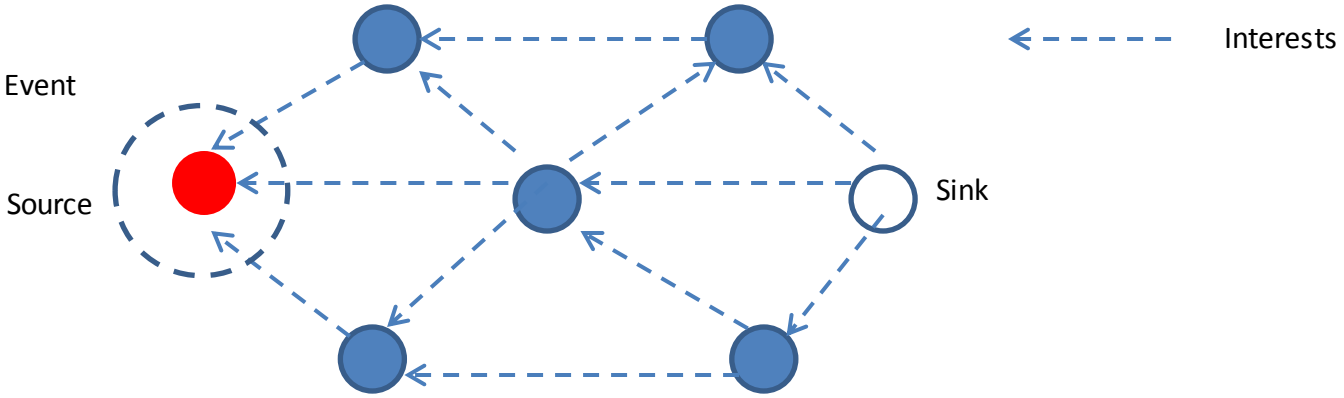
Response

```
type =  four-legged animal    //detect animal location
instance = elephant           // instance of this type
location = [125, 220]         //node location
intensity = 0.6               // signal amplitude measure
confidence = 0.85             //confidence in the match
timestamp =  01:20:40         // event generation time
```

Interests

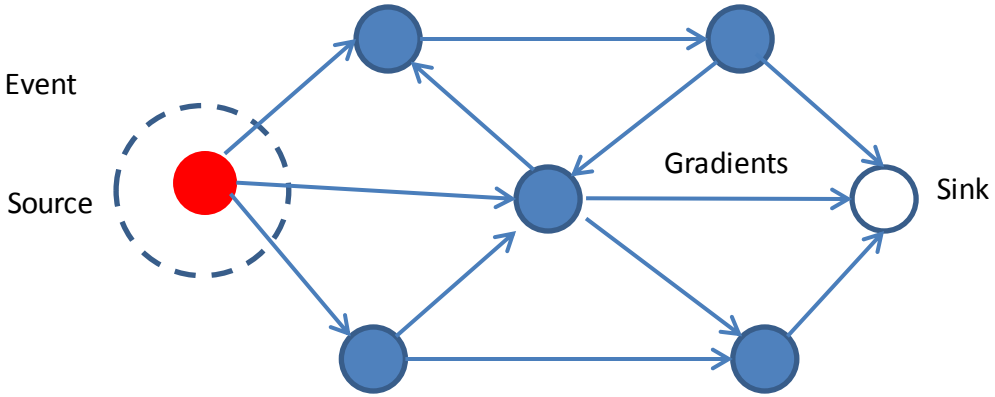
- Interests "injected" at some node – Sink
- Sink periodically transmits active interests to neighbors
  - Interests are periodically refreshed
  - Initial Interests are exploratory (use larger intervals)
- Different options for propagating interests (flooding, geographic routing etc)

Sample Interest
type = four-legged animal
interval = 1 s
rect = [100, -100, 200,400]
timestamp = 01:20:40
expiresAt = 01:30:40



Gradients

- Each node contains an interest cache
- Each Cache entry corresponds to a unique Interest
- Interest fields can contain several gradients, up to 1 per neighbor



Cache Entry

Interest 1			
Timestamp	**Gradient**		
01:20:40	**Neighbor**	**Rate**	**Duration**
	A	10/sec	10 mins
	B	100/sec	20 mins

Interest 1
type = four-legged animal
interval = 1 s
timestamp = 01:20:40
expiresAt = 01:30:40

Data Propagation

- Detection of event triggers collection of samples
- Event description sent to neighbors for which gradient is present
- Nodes drop, filter or re-transmit messages by looking into data cache
- Loop prevention and down-conversion – Enabled by coupling application semantics with communication mechanism.

Reinforcement

- Low rate interests are used for probing – Conserves energy
- When an event of interest is detected at the source, this event descriptor is communicated to the source.
- The source then “reinforces” one particular neighbor to pull higher data rate events.

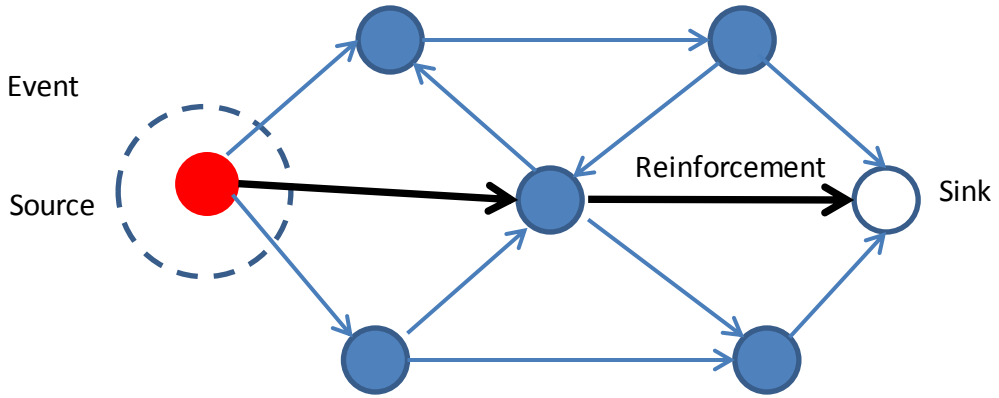
Initial Interest
type = four-legged animal
interval = 1 s
rect = [100, -100, 200,400]
timestamp = 01:20:40
expiresAt = 01:30:40

Reinforced Interest
type = four-legged animal
interval = 10 ms
rect = [100, -100, 200,400]
timestamp = 01:20:40
expiresAt = 01:30:40

- On receiving an interest with a lower interval, the node reinforces one of it’s neighbors similarly.

Reinforcement (Contd…)

- How does the node decide which neighbor to reinforce ?
  - Multiple choices
    - Choose the node from which it first received the latest event matching the interest.
    - Choose all neighbors from which the new event was received.



- The first choice establishes a low delay path from sink to source - Interesting outcome. Without explicitly using any routing tables, it is possible to establish a low delay path between the source and the sink.

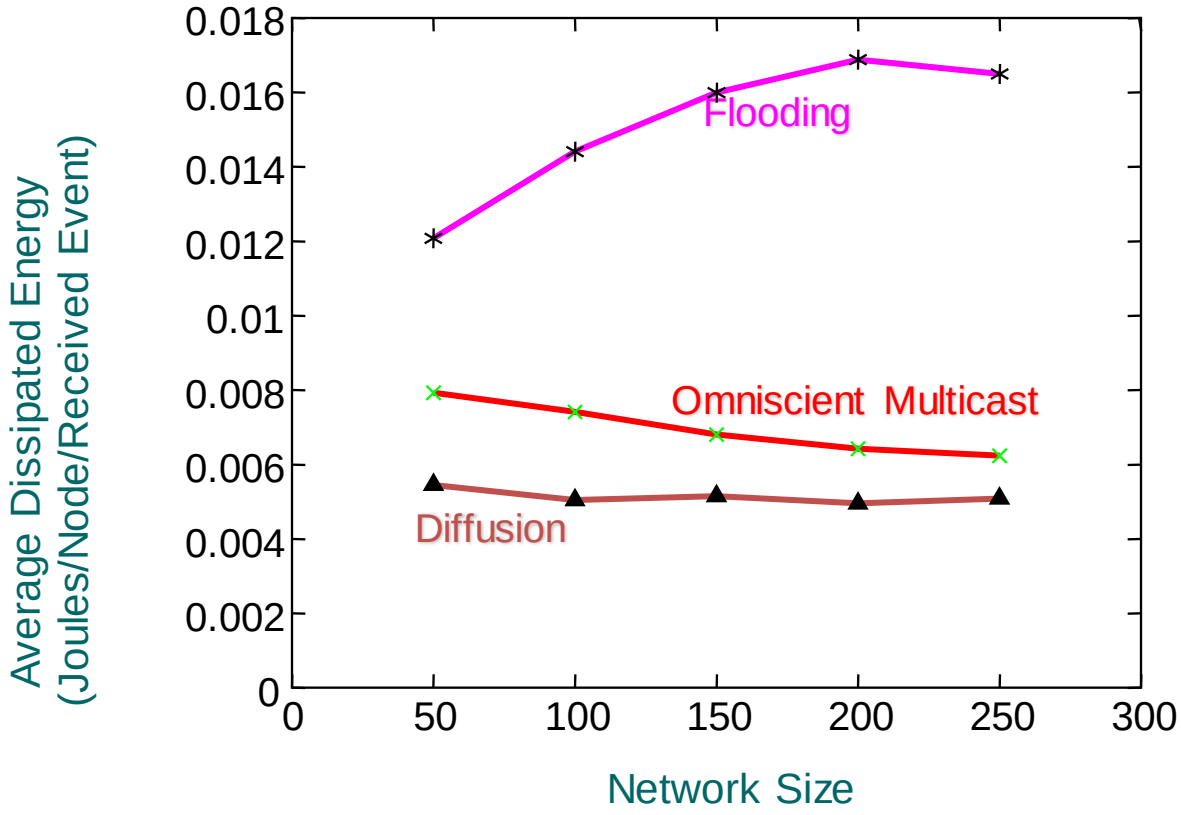
Reinforcement (Contd…)

- Negative Reinforcements
  - A previously reinforced node may superseded by another - Need to negatively reinforce older neighbors
    - Passive reinforcement – Implicit . Use timeouts
    - Explicit reinforcement – Re send interests with lower data rate.

- Using Reinforcements for Repair
  - Intermediate nodes could also initiate reinforcement if they detect link failure with neighbors – enables self healing

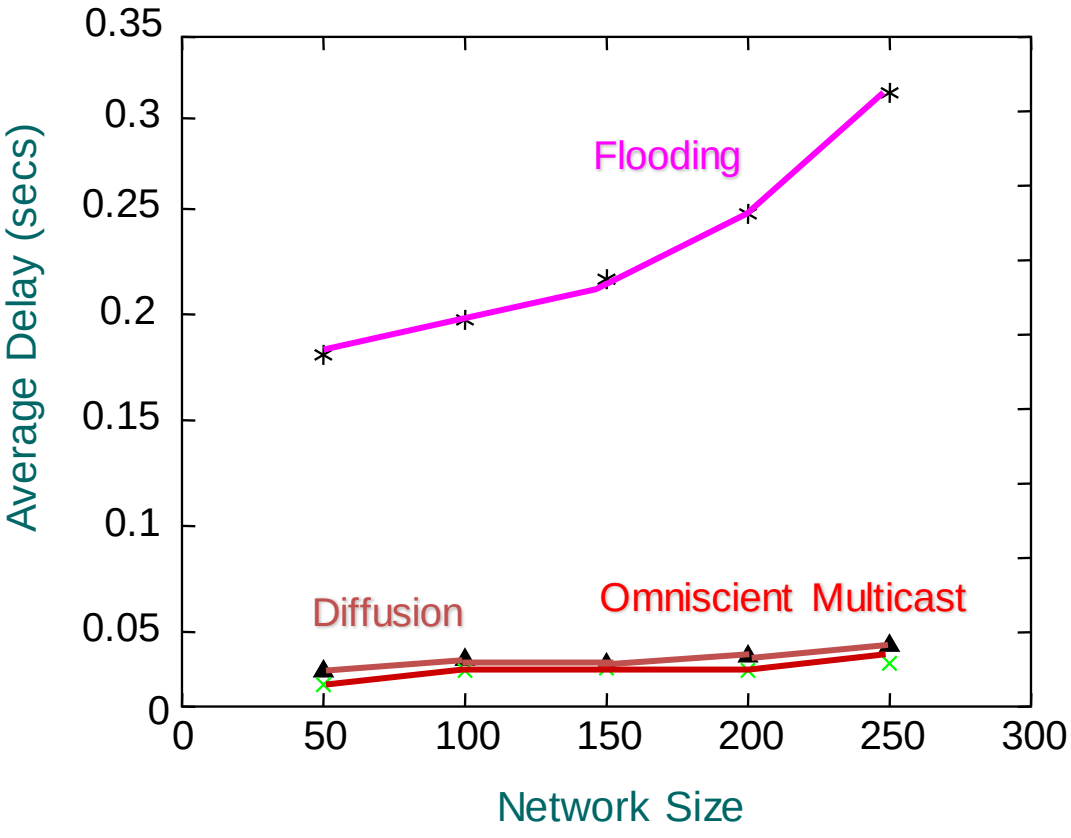
Experimental Evaluation

- ns-2 simulator used to simulate MAC layer
- Sensor fields vary from 50 – 250 nodes in increments of 50 nodes
- 50 node sensor field generated by randomly placing the nodes in a 160m x 160m square.
- Benchmarks :
  - 1) Flooding
  - 2) Omniscient multicast



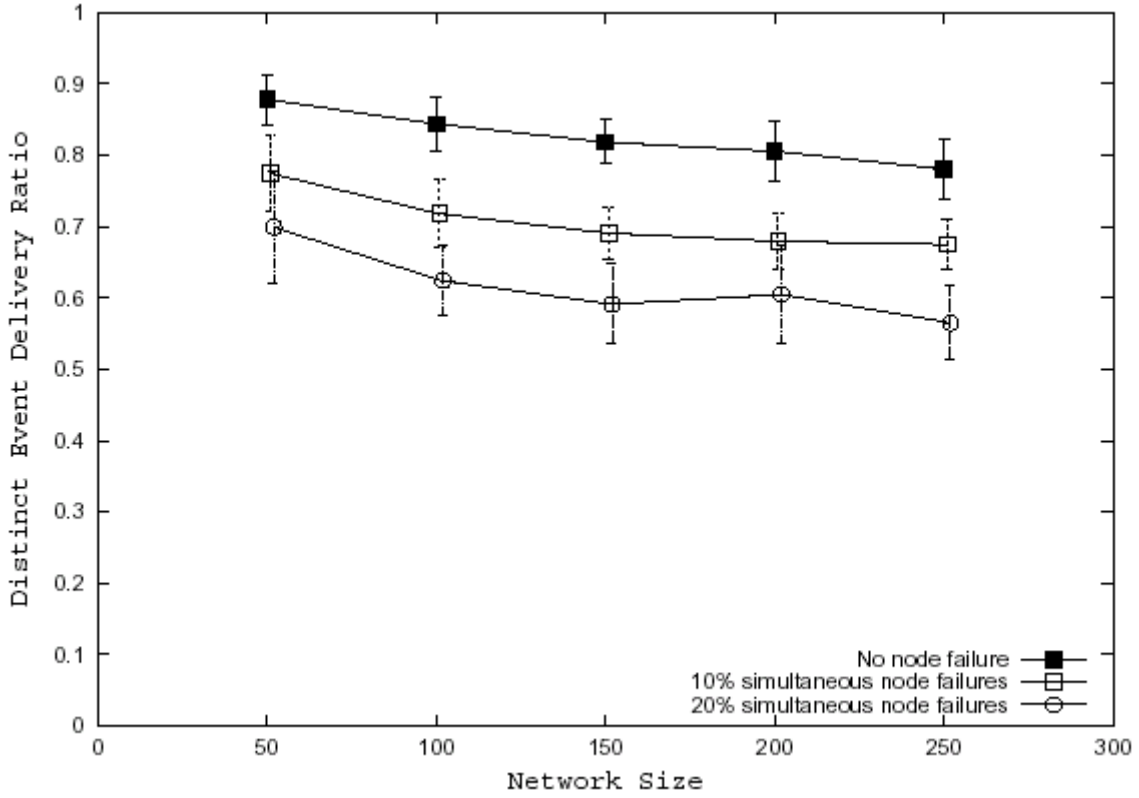
Average Dissipated Energy

Experimental Evaluation



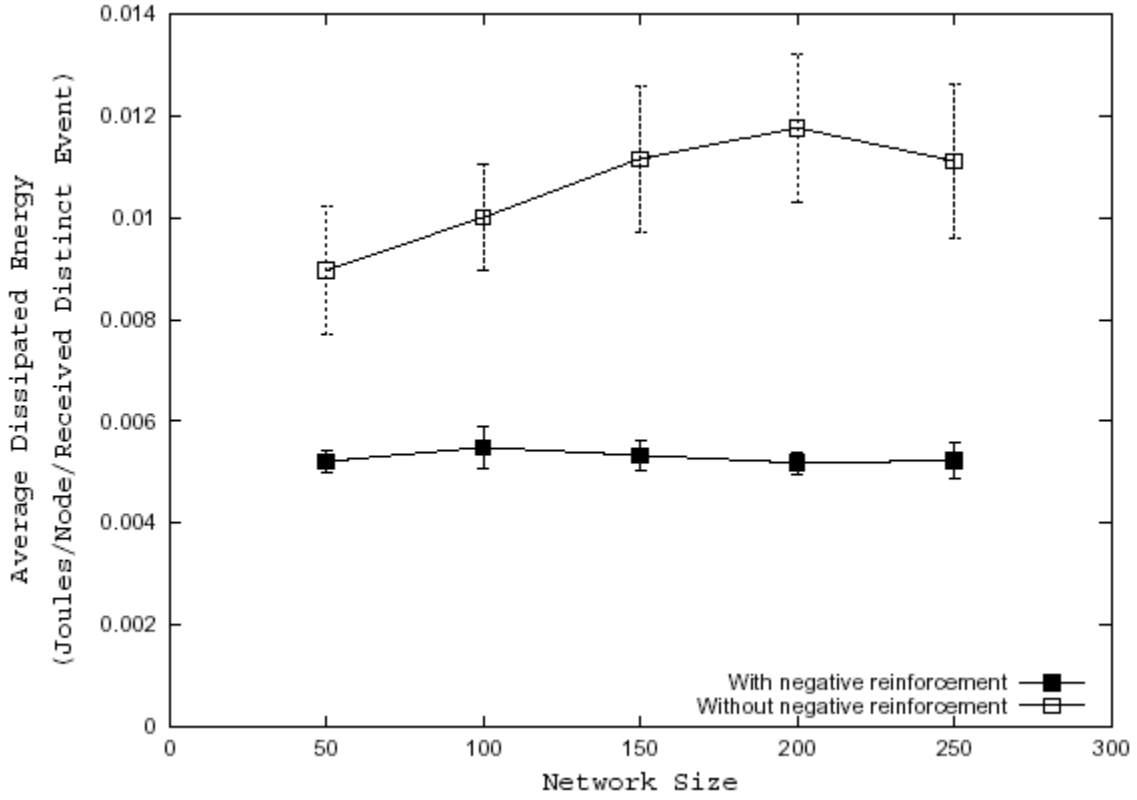
Average delay

Experimental Evaluation
Impact of node failures



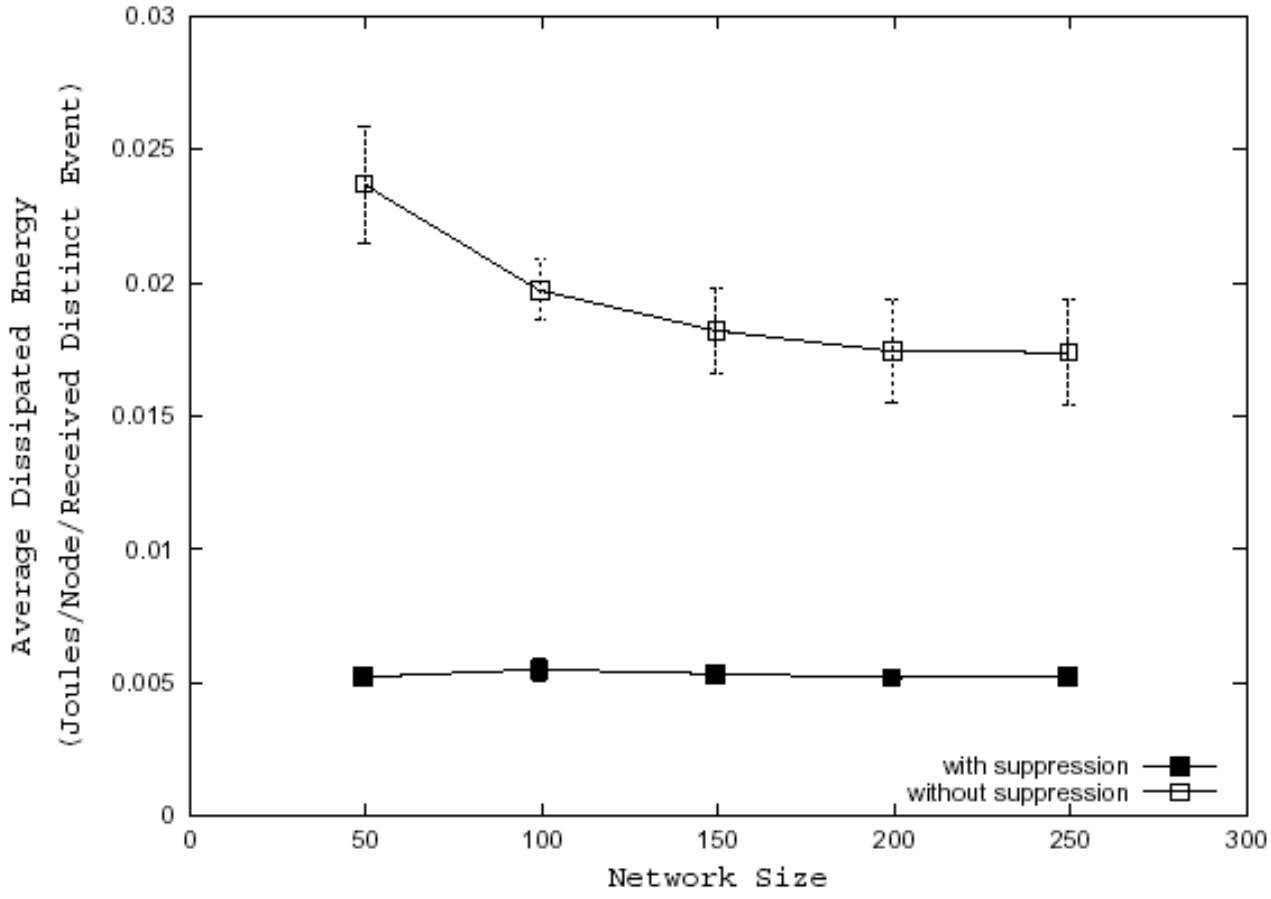
Distinct Event Delivery Ratio

Experimental Evaluation
Impact of negative reinforcement



Average dissipated energy

Experimental Evaluation
Impact of duplicate suppression



Average Dissipated Energy

Discussion

- Is the ISO/OSI model for networking over-relied on for even ad-hoc cases ?
  - ISO/OSI model was designed for connecting multiple LAN segments over long distances. But, the huge popularity of TCP/IP means that it is being used everywhere.
- Named Data : Can the concept of Named Data used here be used in traditional networking as well ?
- Method of reinforcement – Event description has to be sent to the source before reinforcement can occur – Could we miss events ?
- How generic is the approach in Directed Diffusion for other applications ?
- <u>Discussions from Piazza</u>
  - Since paths are chosen without a global view of the network, they could be inefficient globally.
  - Congestion is not considered in evaluation
  - Memory requirements of sensor nodes & interest look up expense

"A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks"

Elizabeth M.Royer, Chai-Keoing Toh

Wireless Networks

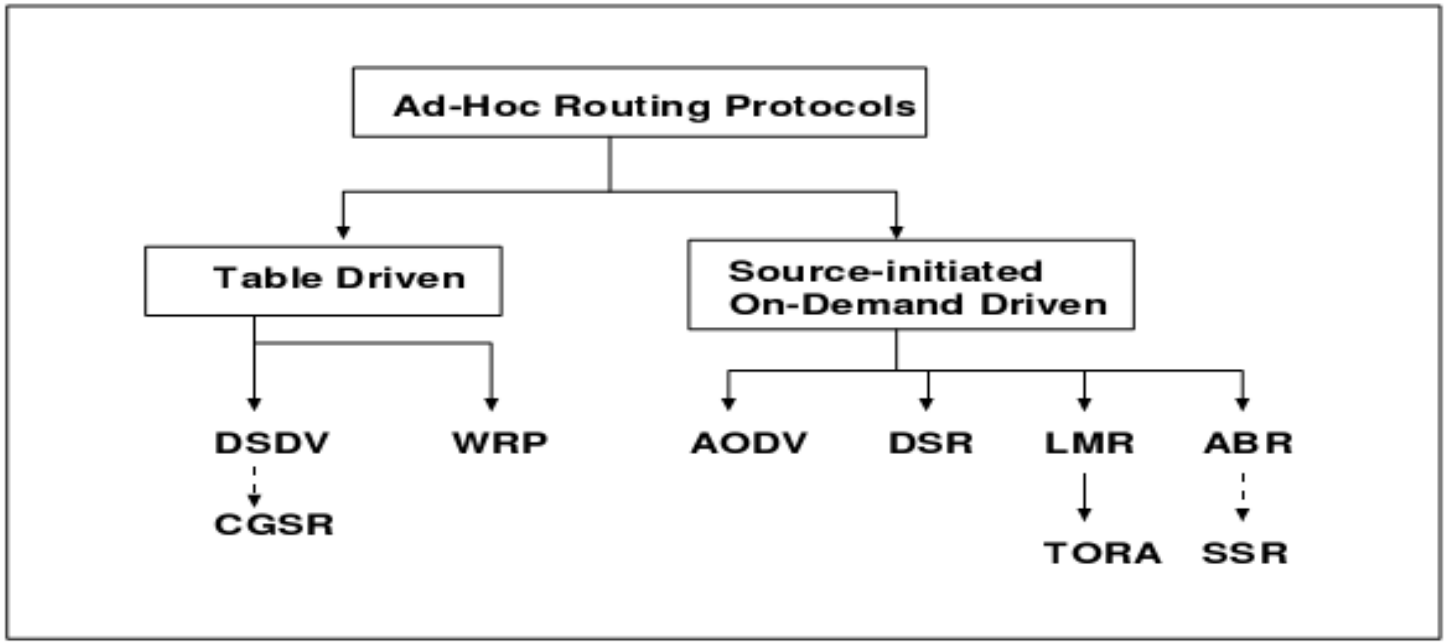
- Wireless networks are extremely popular today. Every smartphone today has multiple wireless capabilities. (Wi-fi, NFC etc)

- 2 types of mobile networks :
  - Infra-structure based : Use a centralised co-ordinator
  - Ad-hoc networks: Consists of equal – peers (Although one among them may be elected as head).

Ad-hoc wireless networks have multiple use cases. For ex, It could be use in search and rescue operations when mobile networks are down.

Ad Hoc Routing Protocols

- Ad-Hoc Routing protocols are classified into the following types :
  - Table Driven
  - Source initiated(demand driven)



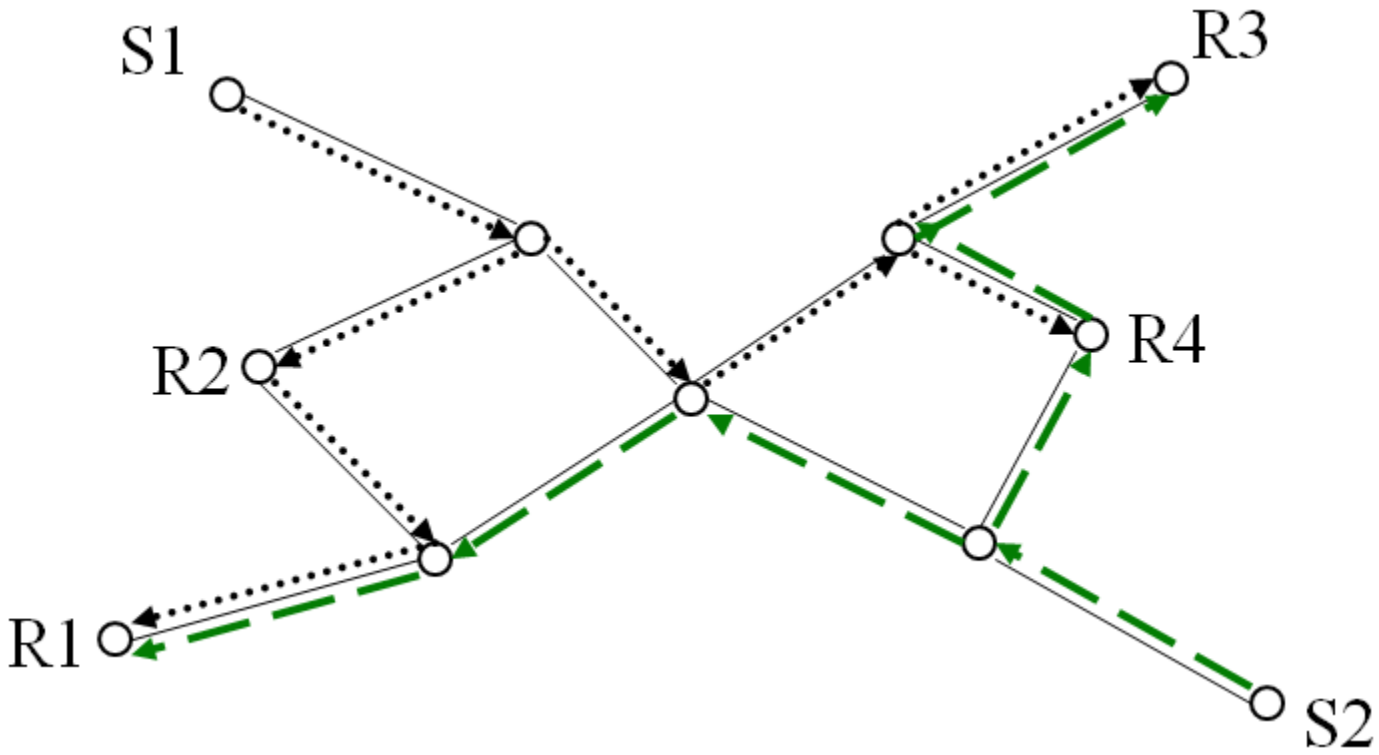
- Table Driven
  - Maintain consistent, up-to date routing information from each node to every other node in the network.
  - Changes in link status propagated to all users to maintain a consistent view

- Source – Initiated On –Demand Routing
  - Routes created only when desired by the source node.
  - When a packet needs to be routed, the node initiates a route discovery process.
  - The established route is maintained by a route maintenance procedure.

Backup Slides

- Experimental Setup
  - Sensor fields vary from 50 – 250 nodes in increments of 50 nodes
  - 50 node sensor field generated by randomly placing the nodes in a 160m x 160m
  - ns-2 simulator implements a 1.6 Mbps 802.11 MAC layer
  - Idle Power Dissipation - 35 mW
  - Receive Power Dissipation - 395 mW
  - Transmit Power Dissipation - 660 mW

Multicast Trees



Data Propagation

- On detecting a target, a sensor searches its interest cache for matching interest entry & schedules collection of data samples at the highest requested event rate among its gradients.
- An event description is sent to each neighbor in the interest cache for which a gradient is present.
- On receiving a data message from a neighbor, a node matches it's attributes with the interest entries.
  - Message is dropped if no matching interest entry is found.
  - There is a data cache associated with each interest entry that keeps track of recently sent items. If a match is found, the message is silently dropped – Prevents loops
  - If no matching data cache entry is found, the message is added to the data cache and re-sent to neighbors.
  - Data cache is also used to determine the rate of incoming data – Allows down-conversion to appropriate gradient by filtering and aggregation of event messages

- Loop prevention and down-conversion – Enabled by coupling application semantics with communication mechanism.

Experience from a Decade of TinyOS Development

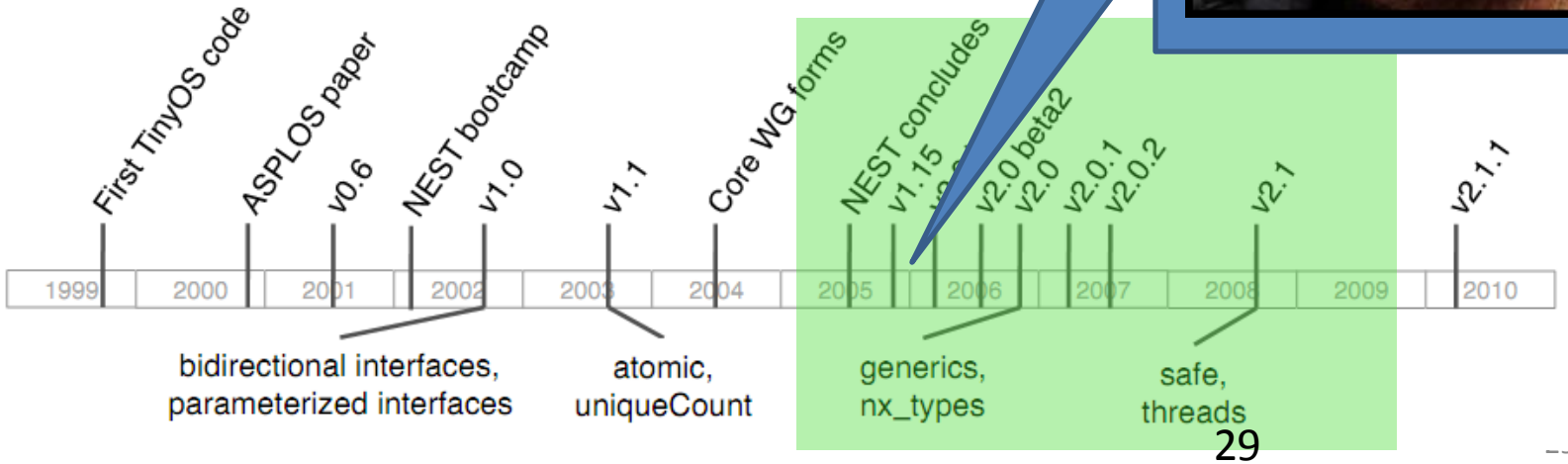
Philip Levis

OSDI 2012

(presenter: Shannon Chen)

The Author

- Philip Levis
- Associate Prof @ Stanford
- Got involved in TinyOS project @ Berkeley in 2005 (PhD)
- TinyOS 1.x -> TinyOS 2.x



The Paper

- The evolvement/lessons learned of TinyOS
- TinyOS targets WSN
  (limited RAM, power, CPU; event-driven)
- 1999
  - A few Perl scripts that generate C code
  - Used in a few internal projects (SmartDust) in UCB
- 2012
  - nesC: a langue developed along with TinyOS
  - Broadly used across industry and academia
  - 25,000 downloads per year; hundreds for papers

The Design Goal

1. Minimize resource use

Model	ROM	RAM	Sleep	Price
F2002	1kB	128B	1.3μA	\$0.94
F1232	8kB	256B	1.6μA	\$2.73
F155	16kB	512B	2.0μA	\$6.54
F168	48kB	2048B	2.0μA	\$9.11
F1611	48kB	10240B	2.0μA	\$12.86

(a) TI MSP430 Microcontrollers

- Requiring little state (RAM)
- Tight code (ROM)

2. Bug prevention
   - "Debugging is notoriously hard in sensor networks"
   - Programing -> deployment <-> remote debug

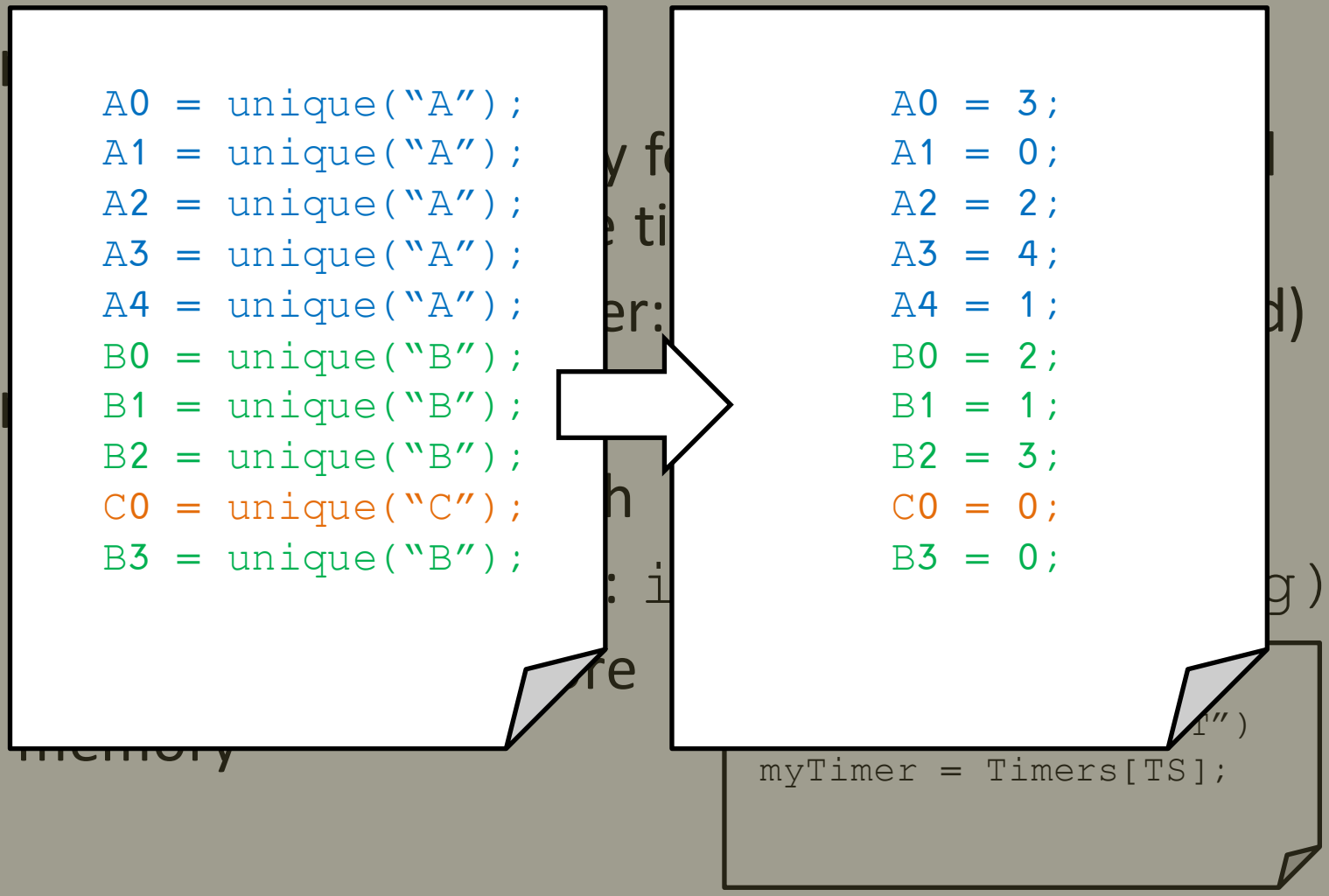
The Approaches

- RAM Allocation
- Language

Goals
1. Minimize resource use
2. Bug prevention

- Isolation
- Components

RAM Allocation



RAM Allocation

- TinyOS 1.1 (nesC)
  - Fixed array, fixed length, decided on compile
  - `int unique(string)`
  - `int uCount(string)`

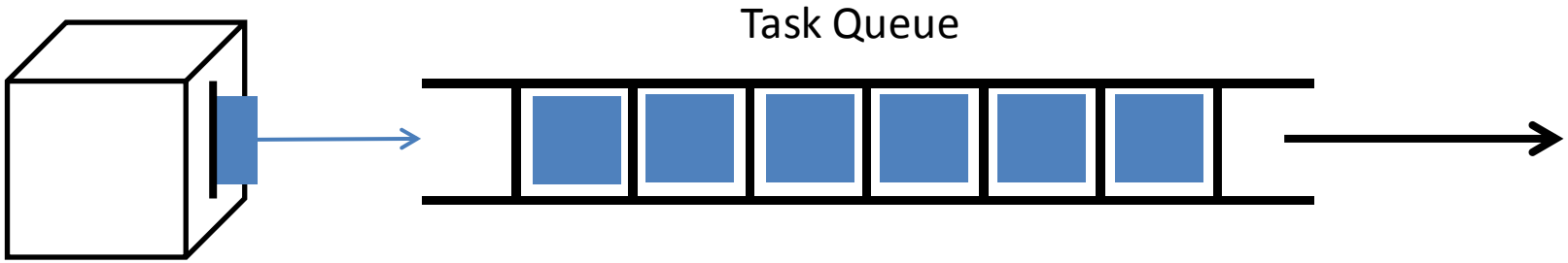
```
#define TS unique("T")
myTimer = Timers[TS];
```

```
#define TN uCount("T")
Timer_state Timers[TN];
```

  - Allocate the minimum amount of memory needed

Isolation

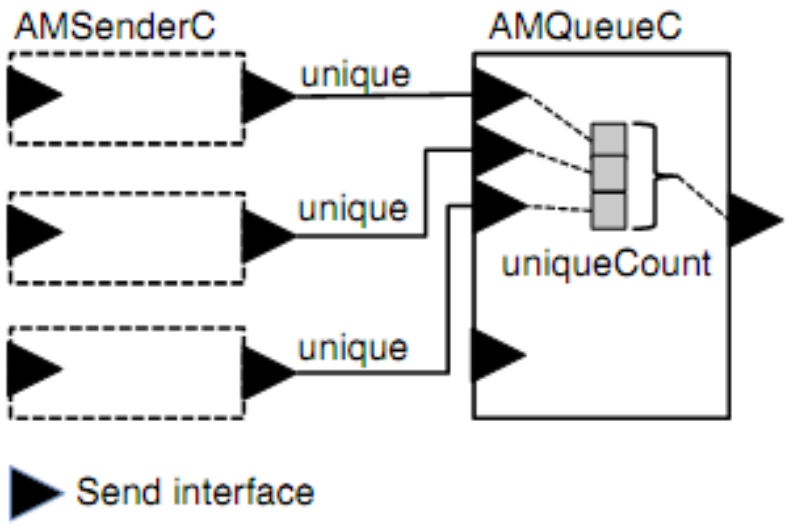
- TinyOS 1.x



  - Fixed queue length
  - Queue full -> re-invoke after some time
    -> need timer (task)
  - Error handling -> need RAM
- Conclusion: shared global memory pool is bug-prone and a waste of RAM.

Isolation

- TinyOS 2.x: Static Virtualization
  - No more shared global memory (isolation)




```
AMSenderC:

#define QS unique("Q")
mySlot = Queue[QS];
```

```
AMQueueC:

#define QN uCount("Q")
task Queue[QN];
While(1)
   for i=0..QN-1
      check(Queue[i]);
```

  - The behavior of the API is solely based on the caller

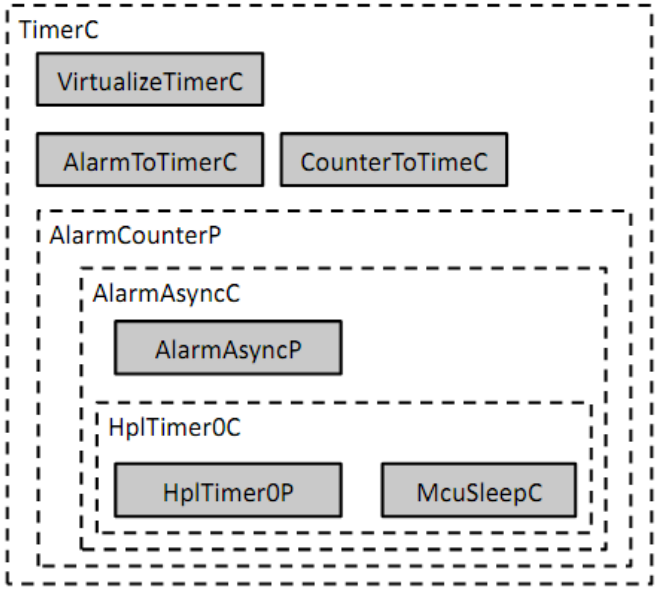
Language

- nesC
  - Generic code
  - Compile time memory allocation
  - Bug prevention
  - Move away from C -> raise the bar to entry

> “Making it harder to write buggy code had the unfortunate result of making it just harder to write code.”

Components

- Fine-grained components
- Code re-use, privacy, security, etc.
- Intend to make lower level modification easier
- Commercial use
- Steep learning curve
- Drives users away (industry and academia)



The Island Syndrome

Missed being a platform for simple sensing apps

Missed being a platform for Internet of Things

- Arduino

Hobbyist

- Contiki
  - Pure C
  - Traditional OS structure

Academia

Industry

Conclusion

- “We should have avoid the island syndrome”
- Easy entry
- Should not focus only on academic use

- Public interactive documentation: wiki
- Late industrial involvement

Discussion

- Is popularity that important?
  - Easy entry
    vs.
    well-defined, well-structured, re-use enabled, bug-prevented code
- Contradicting purposes
  - Industrial vs. academic
  - Application level production
    vs. lower level investigation

Discussion (From Piazza)

- What effort has TinyOS made on energy efficiency?
- Will debuggers help?
  (e.g., powerful debugger + C)
- "CS graduate students have very little motivation to actively support users": publish or perish
- Future plans? (e.g., splitting the system)