TinyOS: An Open Platform for Wireless Sensor Networks

Part II: Sensor Network Architecture

Philip Levis
Stanford University
10.v.2006
11.v.2006

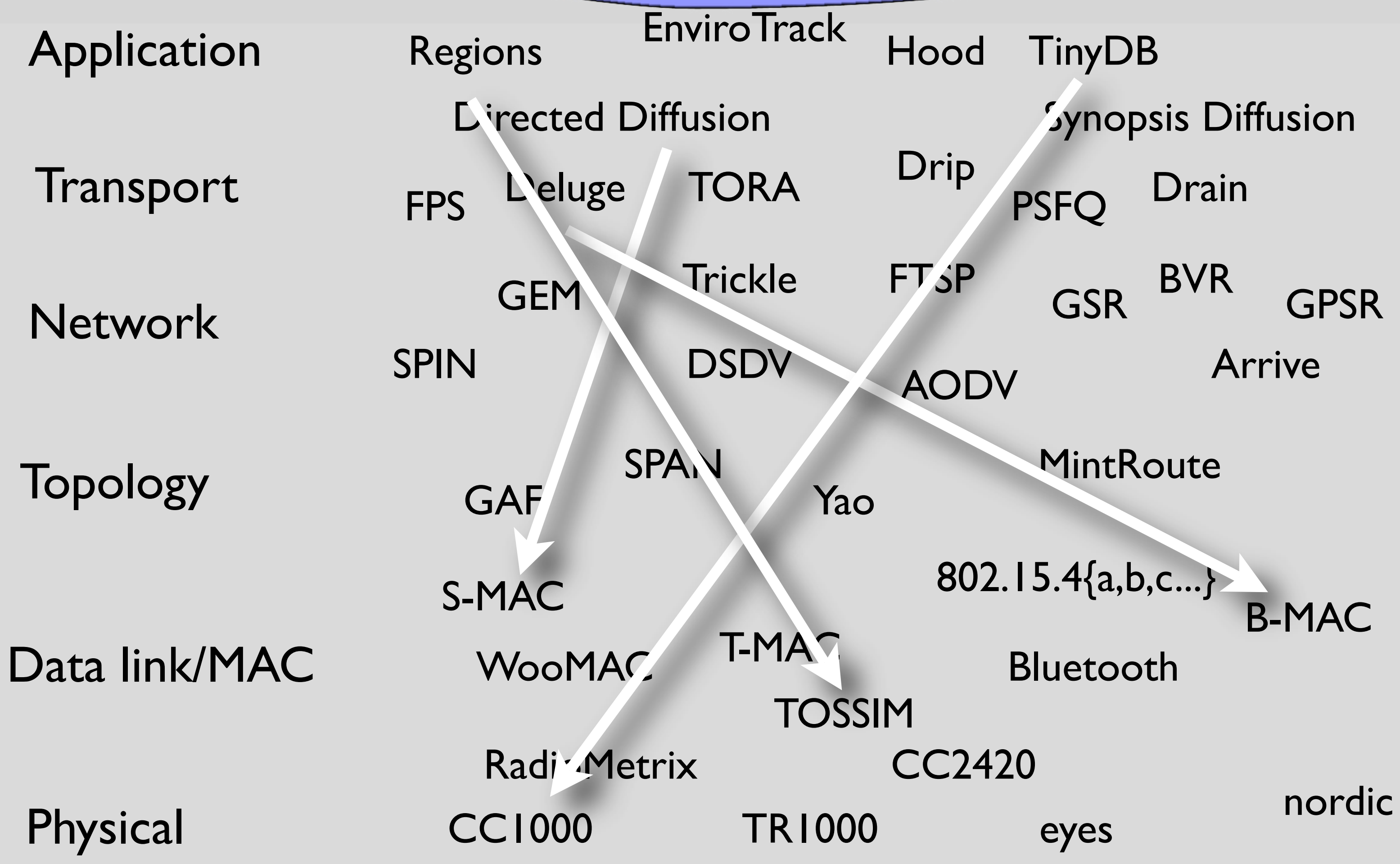
In the Beginning

- National Research Council thesis: “embedded sensor networks are different.”
  - Embedment, energy limitations, data-centric operation
  - They’re not just a new set of IP devices
- But if not IP, what are they?
  - What are the critical services and mechanisms?
  - What does a sensornet protocol stack look like?
  - Maybe it is just IP...

Testing the Hypothesis

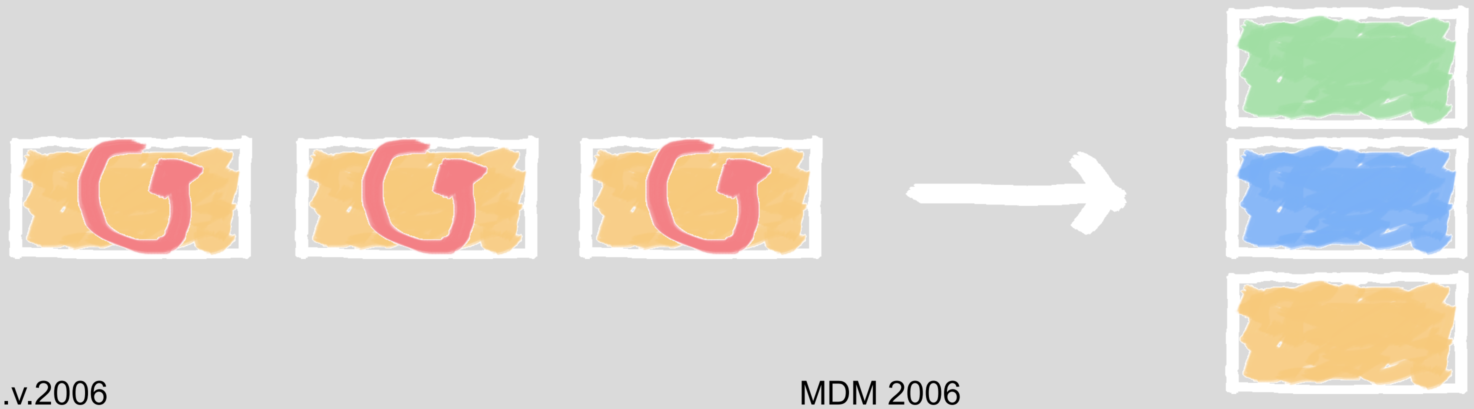
- We don’t know what they should look like, so let’s build a framework so everyone can figure it out.
- TinyOS: component-based OS
  - Developers can change any part of the system
  - No real division between application and OS code
- A lot of people start using TinyOS, and a few years later...

Sensor Network Systems



Defining an Architecture

- While there are many different protocols, systems, and abstractions, there are commonalities
  - Time stamps, per-hop retransmissions, neighborhoods
- From these commonalities, we can begin to define a sensor network architecture
  - Provide a structure in which to place systems and protocols, separating concerns and promoting composition



Outline

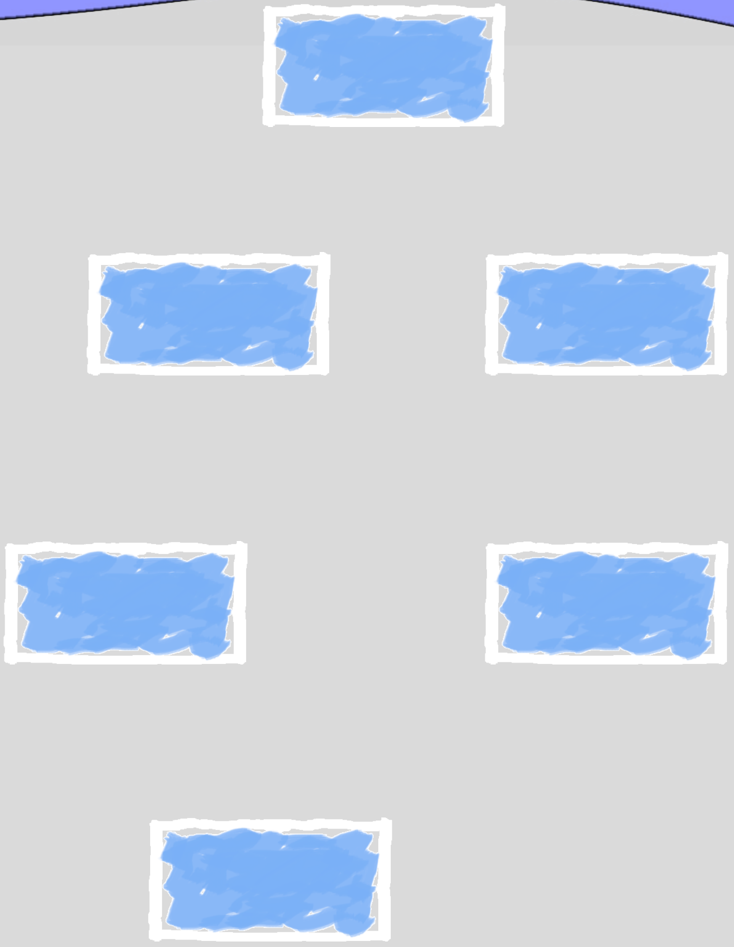
- The case for an architecture
- Why a new architecture?
- What does it look like?
- What’s the OS support?
- Conclusion

Why a New Architecture

- Short answer: we haven’t seen IP take over
- Long answer: the Internet assumes a usage model
  - Independent end-to-end flows
  - Host-centric communication
  - Edge networks with a shared infrastructure
- Sensor networks do not follow this model
  - Dissemination and collection are the common protocols
  - Data-centric communication
  - Sensing removes distinction between edge and core

Protocol 1: Dissemination

- Reliably deliver a piece of data to every node
  - Commands, reconfiguration, installing new binaries
  - Basic control building block
- Use suppression and local broadcasts
  - Collaborative protocol



The Real Dissemination Cost

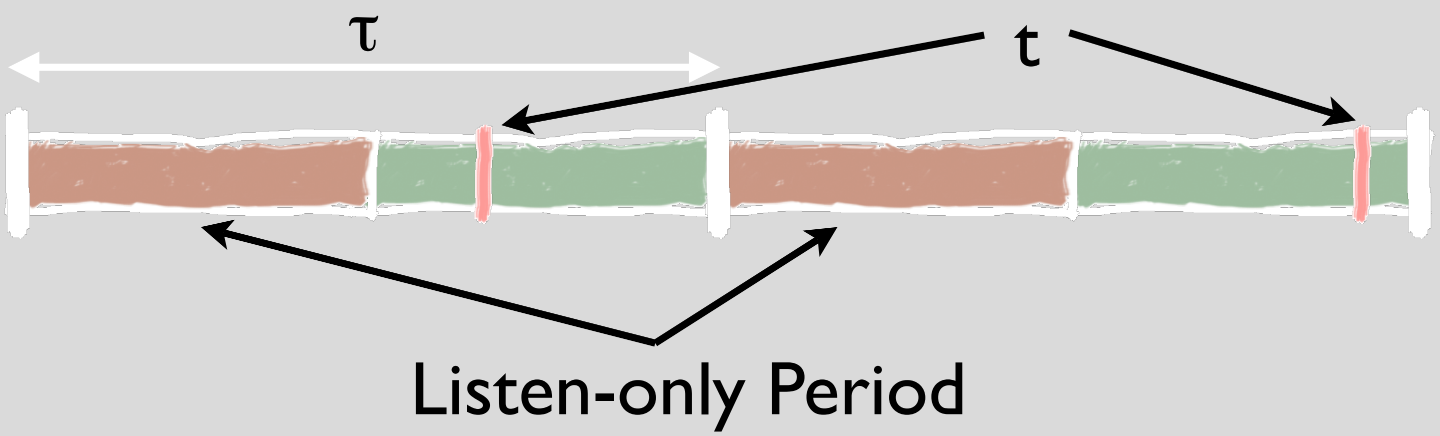
- 100% reliability requires continuous maintenance
  - Transient disconnection, repopulation, loss, etc.
- Propagation is costly
  - Must deliver to every node in a large, multihop network
- Maintenance is more so (e.g, 1 transmission/min)
  - >1 minute, more expensive than a parameter (4-20 bytes)
  - >10 minutes, more than a small program (200 bytes)
- <u>Maintaning that every node has the data costs more than propagating the data itself.</u>

Solution: Trickle

- “Every once in a while, broadcast what data you have, unless you’ve heard some other nodes say the same thing recently."
- Behavior:
  - Maintenance: a few sends per hour
  - Propagation: less than a minute
  - Scalability: thousand-fold density changes
- Polite gossip maintains a message rate *<u>over space</u>, taking advantage of the wireless medium.*

Trickle Overview

- Keep a time window of size τ.
  - Maximum discovery latency
- Pick a time t in the range $(\tau/2, \tau]$.
- Broadcast at time t if you haven't heard a few broadcasts in the time window.



Trickle Scalability

- When a node transmits, it suppresses the set of nodes around it that hear the message.
  - Some nodes may not hear it (packet loss); the number of redundant transmissions scales with log(density).
  - Establishes a message rate over space
- Simple, local decisions leading to global behavior
  - Requires only a few bytes of state: t, count
  - Random selection of t spreads transmission load

Interval Size Tradeoff

- Large interval τ
  - Lower transmission rate (lower maintenance cost)
  - Higher latency to discovery (slower propagation)
- Small interval τ
  - Higher transmission rate (higher maintenance cost)
  - Lower latency to discovery (faster propagation)
- Examples
  - τ = 10 seconds: 6 transmits/min, discovery of 5 sec/hop
  - τ = 1 hour: 1 transmit/hour, discovery of 30 min/hop

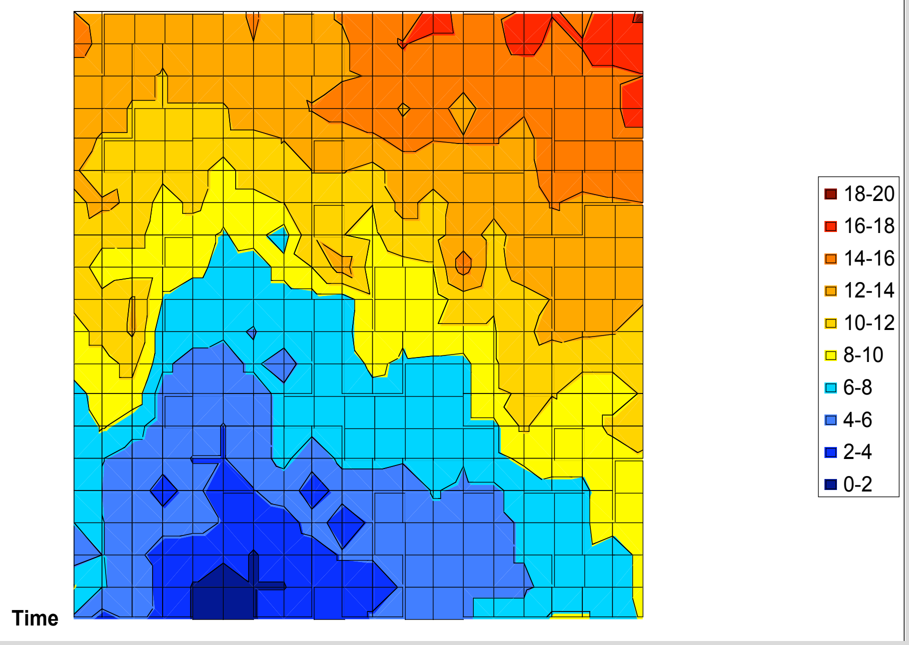
Best of Both Worlds

- Dynamically adjust τ: τ_l, τ_h
- When expires, double τ up to τ_h
- When you hear newer metadata, shrink τ to τ_l
- When you hear newer data, shrink τ to τ_l
- When you hear older data, send update

Propagation

- 16 hop network
- τ_l of 1 second
- τ_h of 1 minute
- 20 seconds to disseminate across the network
- Wave of activity

Time To Reprogram, Tau, 10 Foot Spacing (seconds)

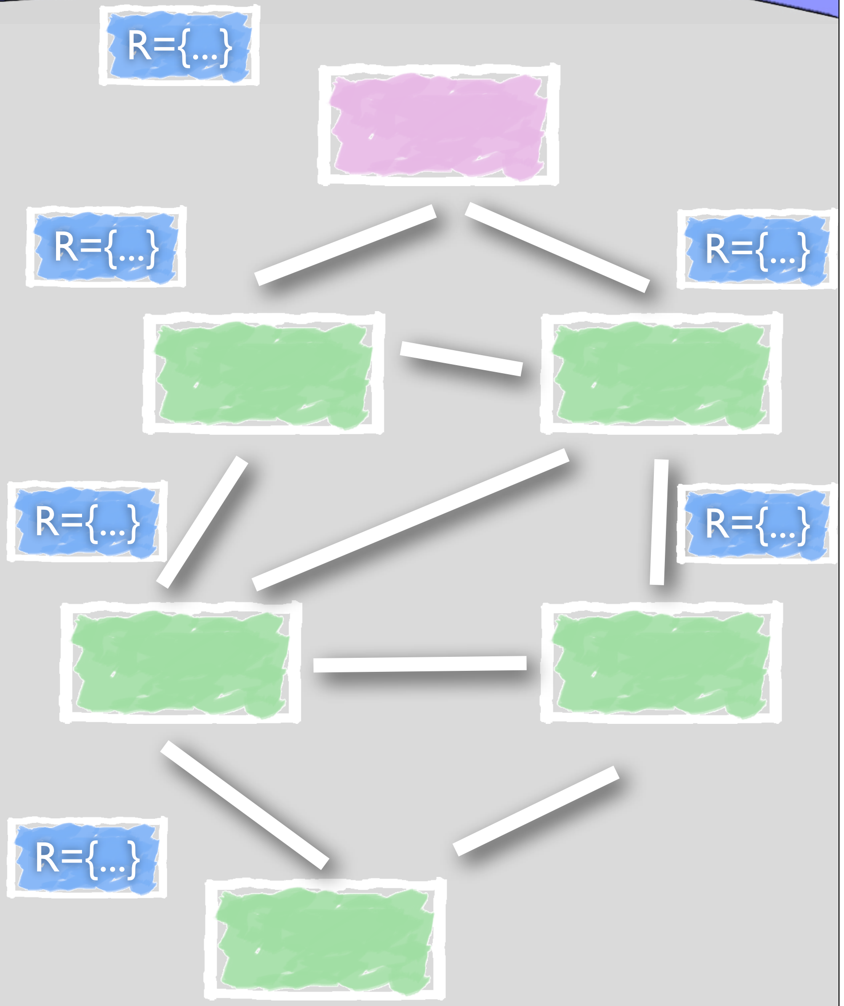


Trickle Overview

- Relies only on single-hop broadcasts
    - Address-free protocol
    - Uses wireless connectivity as an implicit naming scheme
- Can obtain rapid propagation with low overhead
    - Deployments commonly use a > 1 hour
    - Propagation of under a minute
- Simple, scalable, operates in a wide range of network conditions
- Flexibility in timing

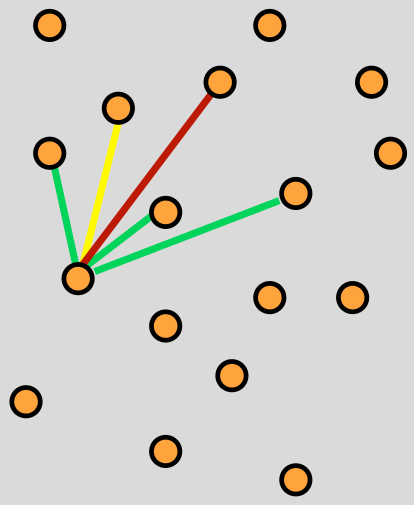
Protocol 2: Collection

- Stream up a collection DAG
  - Sketches, TAG, collect
  - Adapt to network dynamics
- Maybe aggregate
  - Merge data to save energy
  - O(n), not O(n$\sqrt{n}$) packets
- Link estimation to select a parent



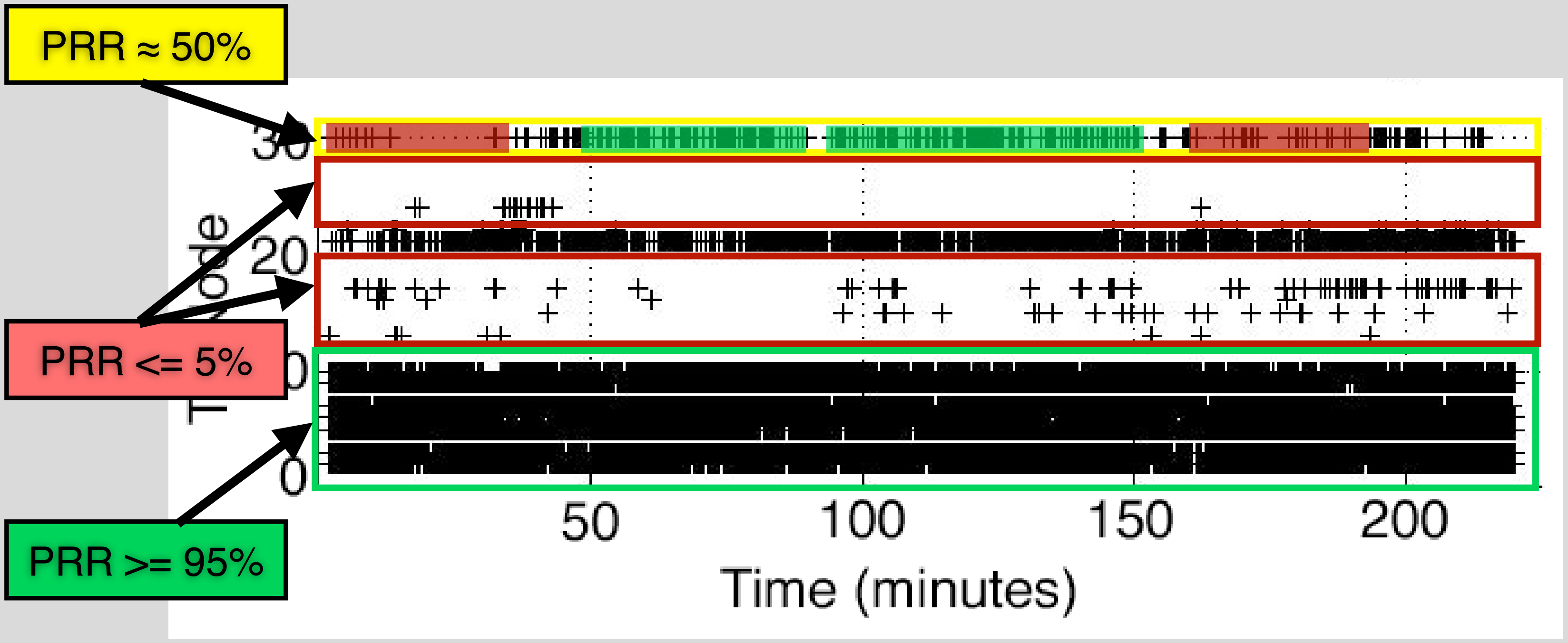
Link Estimation

- Estimating the probability that a neighbor will receive a packet
- Fundamental to almost all routing protocols
    - A basic OS primitive
- Can primitive to compute ETX to destination, a common route selection mechanism



PRR over Time

- Receptions observed at a single node over time[2]
- 30 nodes, office environment, 1500m



Link Estimation Approaches

- Use received signal strength indicator (RSSI)
  - Strengths: easy to measure, quick to adapt, stable
  - Weakness: packet reception depends on signal + noise
- Use bit error rate indicators
  - Strengths: can be easy to measure, true measure of PRR
  - Weaknesses: highly variable, not always available
- Use per-node sequence numbers
  - Strengths: most accurate, link independent
  - Weaknesses: expensive, slow to adapt

Neighbor Table

- Each collection protocol maintains a table of candidate parents
  - Many different table eviction and insertion policies
  - Must be stable when there are 100s of possible parents
- Each node advertises its ETX to the root
  - For node n, cost of a route through p is calculated as

$$\text{cost} = \text{etx}(n, p) + \text{etx}(p, \text{root})$$

  - Node picks parent with lowest cost

Sensornet Protocols

- The dominant metric is energy
  - Low duty cycles lead to low data rates
  - Wasted energy is worse than spent energy
  - Often happy to trade latency for reliability or energy
- Collaborative operation
  - Nodes work together as a patch
  - A little effort can help another node a lot
- Simple, local rules that lead to global behavior

Why Not IP

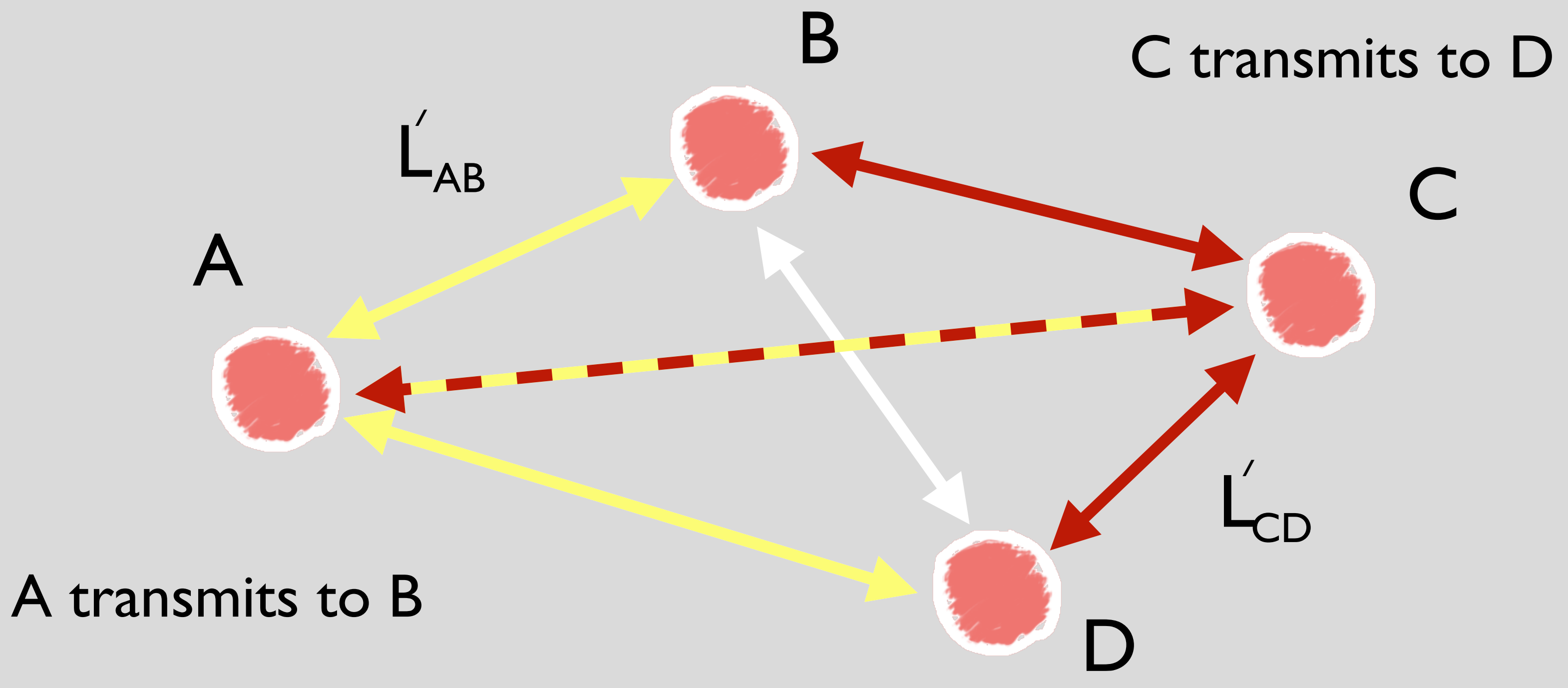
- Arbitrary node-to-node routing isn’t common
  - And it requires a lot of state
- Protocols aren’t always end-to-end
  - There are *many network protocols, not just one*
- *We could give nodes unique 32-bit identifiers...*

How IP Approaches Fall

- Assumptions of behavior
  - Hari Balakrishnan: The problem with any-to-any routing is your bandwidth goes to zero as the network size increases. Robert has a paper on this in Mobicom.
  - Phil: But GDI uses less than 0.2% of the network bandwidth….
- Assumptions of environment (control)
  - Stability (can it hose my network?) vs. optimality (can it go faster?)
  - Sensor nets are a walled garden, but you're outside….
- Wireless networks aren't a graph
  - There is wireless IP, but…

Not a Graph

- Graphs assume link independence
- In wireless, transmissions can affect distant nodes
    - This is not a binary relationship



Outline

- The case for an architecture
- Why a new architecture?
- What does it look like?
- What’s the OS support?
- Conclusion

Usage Patterns and Assumptions

Internet	Sensornet
Independent hosts	Collaborative use
End to end flows	Collect, disseminate, etc.
Two-tier architecture	Ad-hoc (more homogenous)
Wired (generally)	Low power wireless
Latency	Wake time
Throughput	Very low utilization
Bandwidth is cheap	Bandwidth is expensive
Limited inner state	RAM is a limiting resource

There are a lot of differences...

Network Design Goals

Internet	Sensornet
Interconnect separate networks	Dense real world monitoring
Resilience to loss and failure	Resilience to loss, failure and noise
Support many protocols	Support many applications
Accommodate variety	Scale to large, small, and long
Distributed management	Cost effective
Cost effective	Evolvable in resources
Low effort attachment	Composable
Resource accountability	Security

... but many of the same architectural principles apply

Sensornet Commonalities

- Packet-level timestamps needed by a wide range of protocols, applications, and algorithms
  - TDMA, event detection, localization
- Singe-hop level acks/retransmissions
  - Allows immediate reclamation of RAM
- Neighborhood management
  - Maintaining an estimation of other nearby nodes

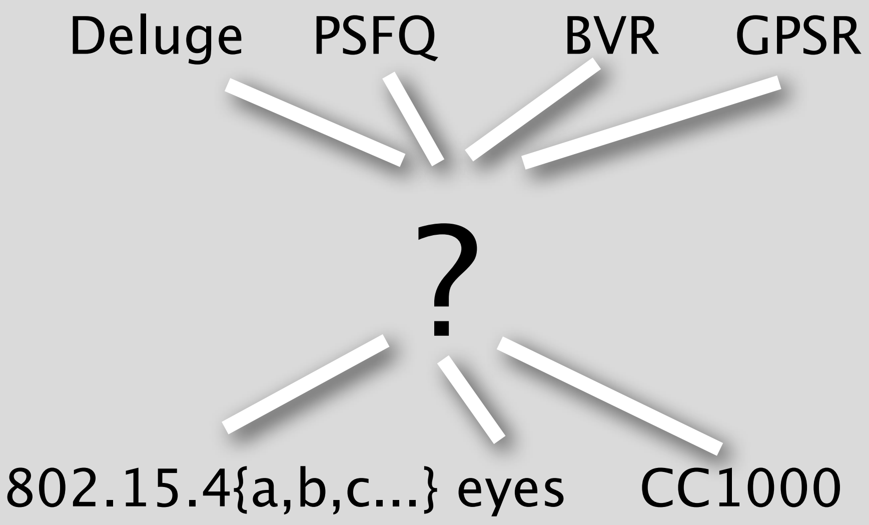
We have a diverse set of layer 3 protocols, and have commonalities at layer 2...

Key Architectural Principle

- IP: the Internet Protocol
- The narrow waist
  - Separate above (transport) from below (data link)
  - Allow parallel and independent development
- Specify the interface between the two

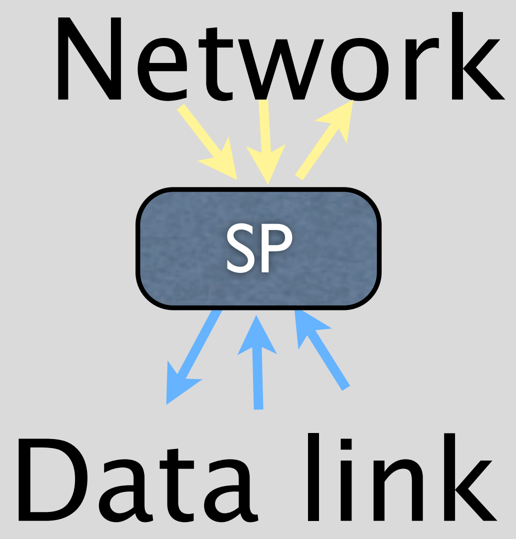
The Narrow Waist

- Layer 2 is the narrow waist of sensor networks
- But L2 is heterogeneous (tied to PHY, regulation)
- Need a unified L2 abstraction
- Would separate multihop protocols from specific data link layers...



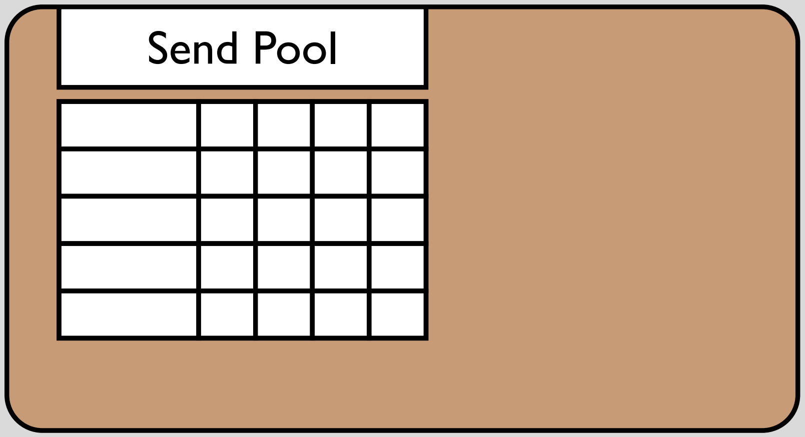
Richer than Send

- Support a range of network protocols, each with their own timing, reliability, and other requirements
- Provide feedback from the data link layer to network layers and vice versa
  - Link estimation
  - Reliability
  - Packets outstanding
- Allow network layers to opaquely cooperate, improving efficiency

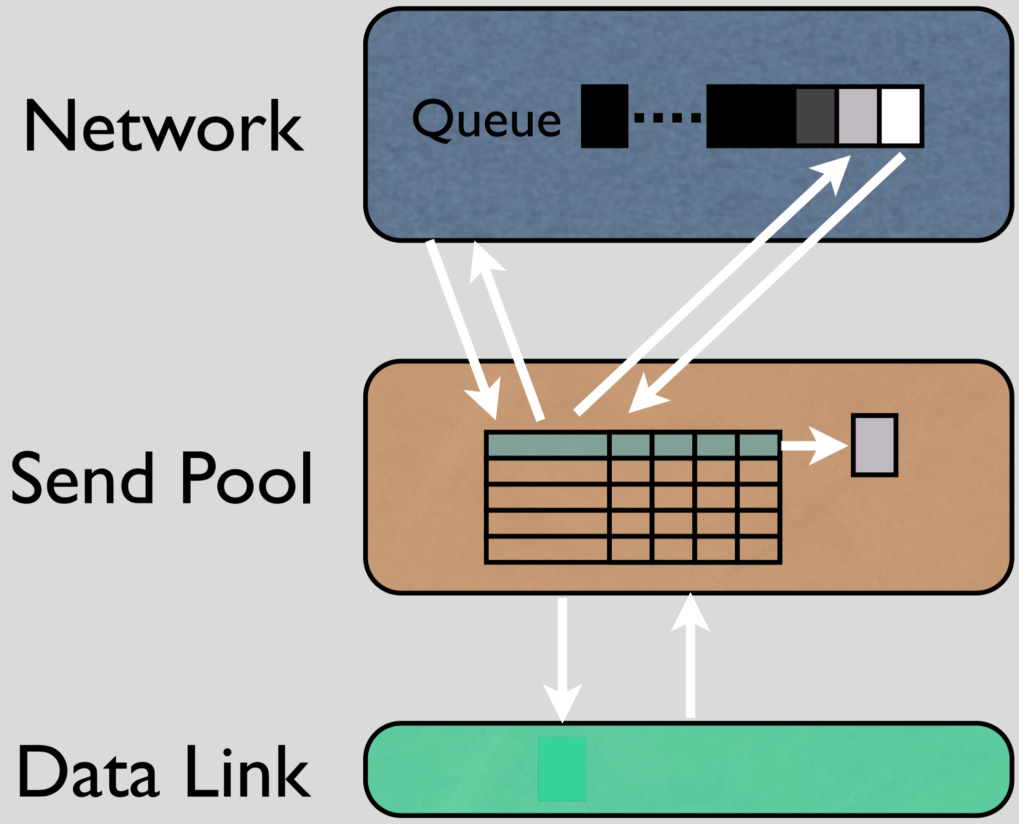


A Send Pool

- Multiple protocols may have outstanding packets
- Queues define an ordering, but L2 does as well
- Use a send pool
- An active entry is a promise to send packets in the future
  - Future count
  - Destination address
  - Urgent bit
  - Reliable bit



Message Futures



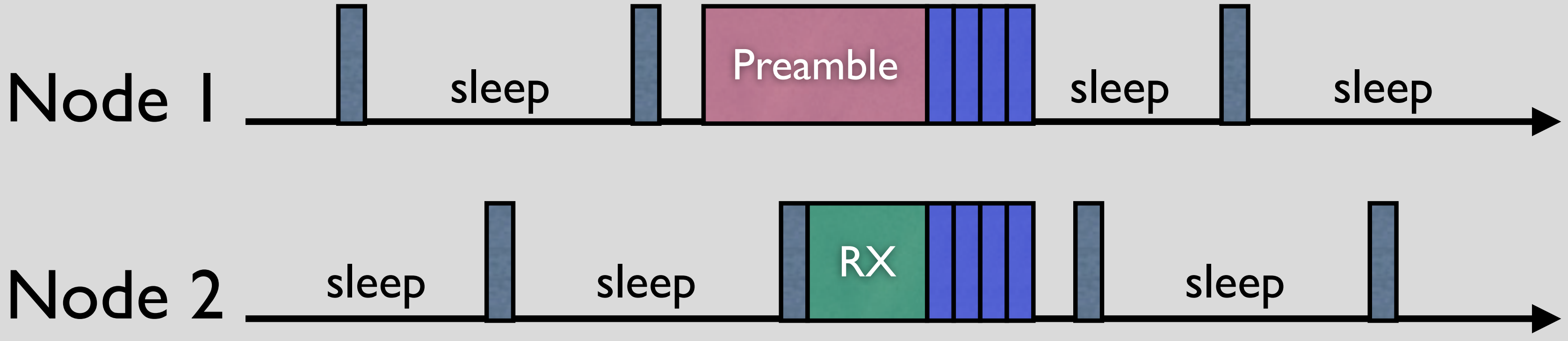
0) Configure pool entry
1) Call start() with first packet
2) Pool schedules transmit
3) Pool gives packet to data link
4) Link notifies pool send done
5) Pool signals next(),
   gets next packet,
   decrements future count

...

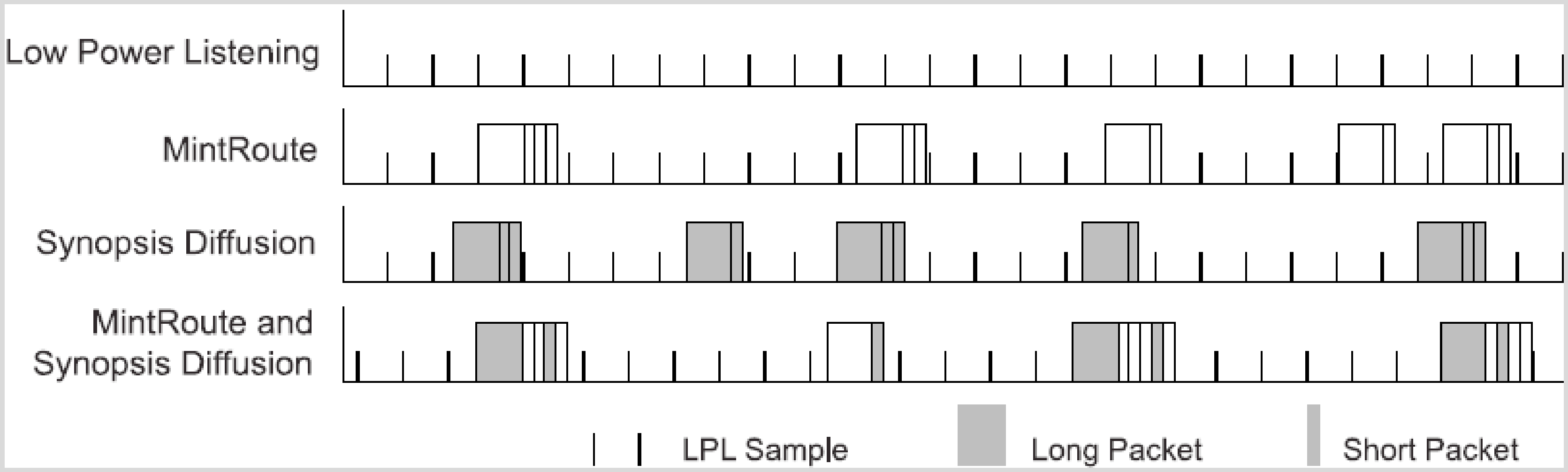
N) Pool signals stopped()

Example Data Link: LPL+CSMA

- Periodically sample the channel for activity
  - Spend most of the time asleep
  - Long packets wake up receivers
- Can piggyback short packets after a long packet



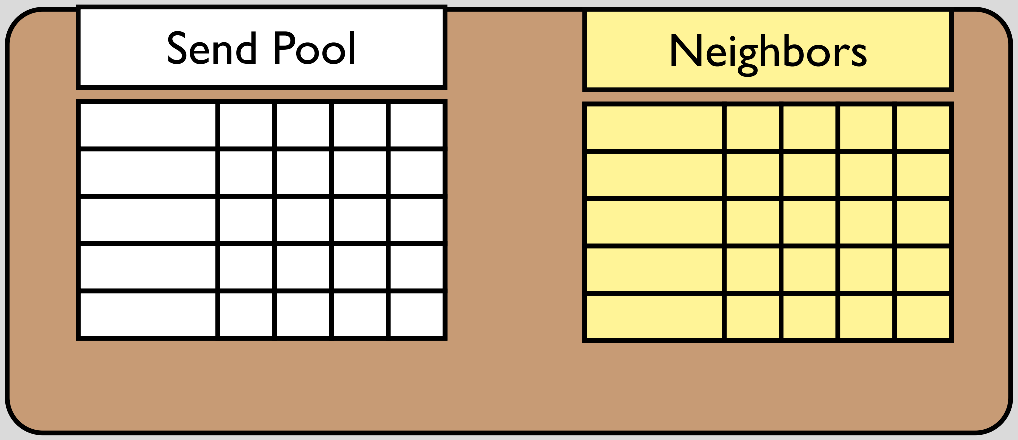
Pooling Sends



- Send pool allows the system to make multiple network protocols unknowingly cooperate (SenSys 2005)

A Neighbor Table

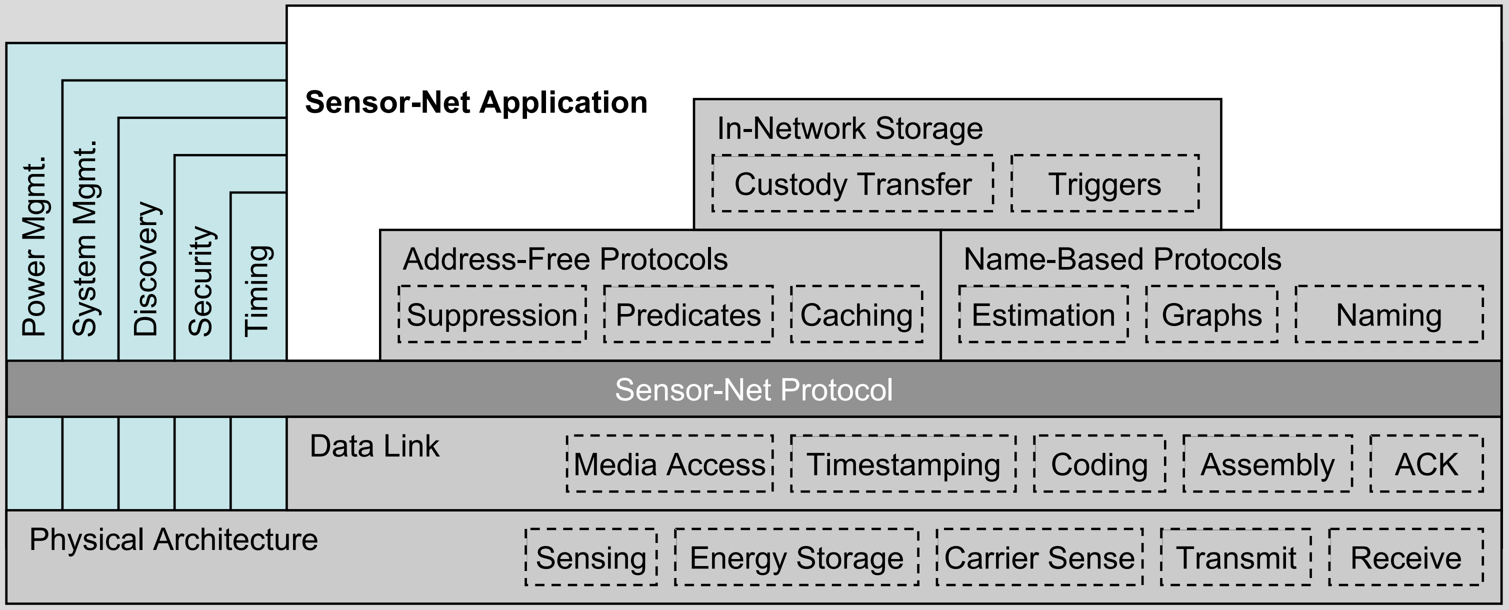
- Cooperation between data link and network layers
    - Scheduling: network (FTSP) vs. data link (TDMA)
    - Link estimation: L3 (MintRoute) vs. L2 (802.15.4)
- Shared table mechanism for many network layers
    - Table arbiter above manages policy



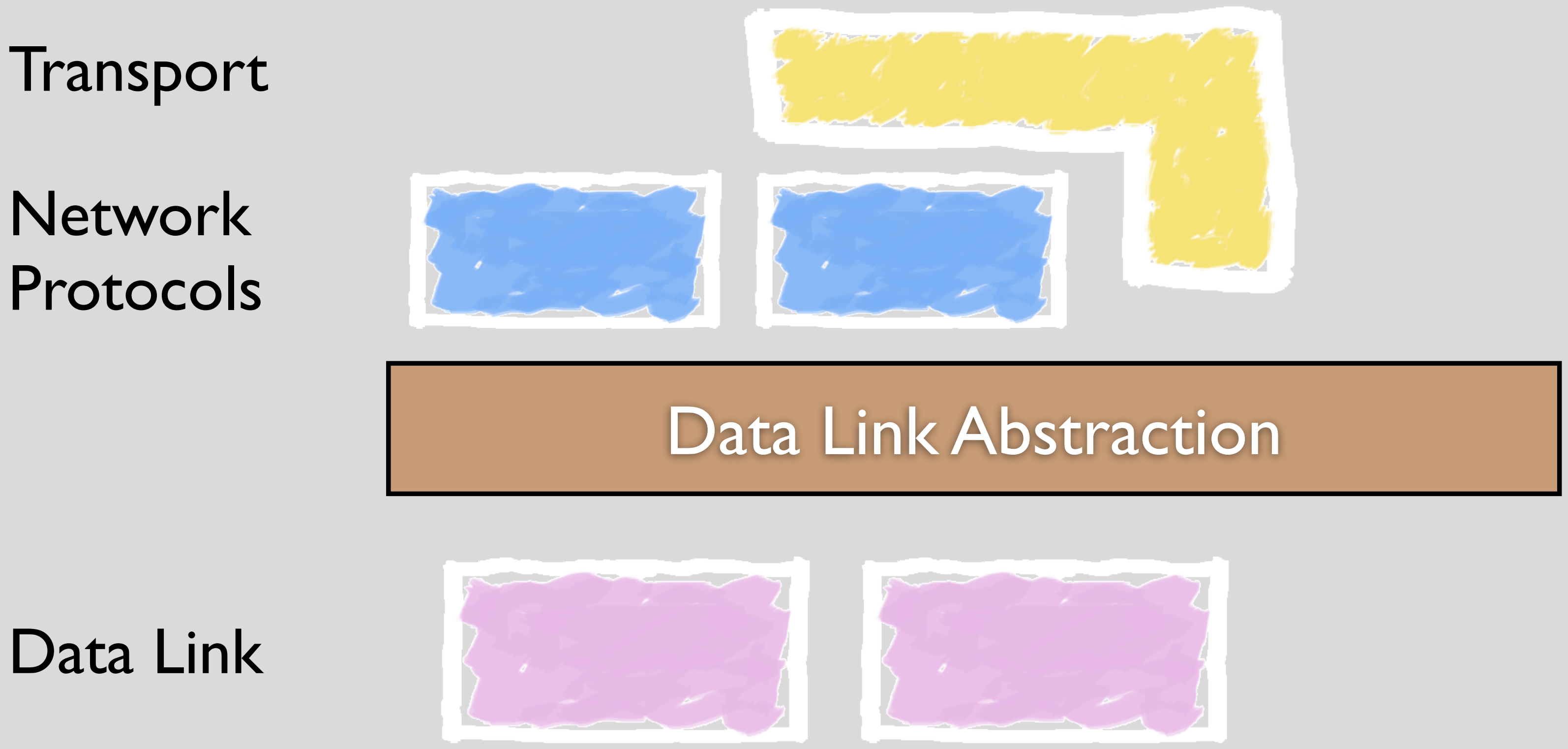
Refining and Revisiting

- SP is a first step towards a more flexible, portable and efficient data-link abstraction for wireless sensor networks
  - Requires further evaluation, use, and experience before standardization
- Next task of the TinyOS Alliance net2 Working Group (net2 WG) is to explore, design, and develop the single-hop protocol and abstraction for TinyOS 2.0.

Proposed Architecture



A More Traditional View



A Working Proposal

- Evaluate and adapt designs based on use
- Deployment model makes iteration acceptable
  - Sensor nets are often autonomous domains: cross-system compatibility is not a major requirement
- The free-for-all of the past few years has shed light on where the boundaries should lie: let’s start to try to codify them.

Outline

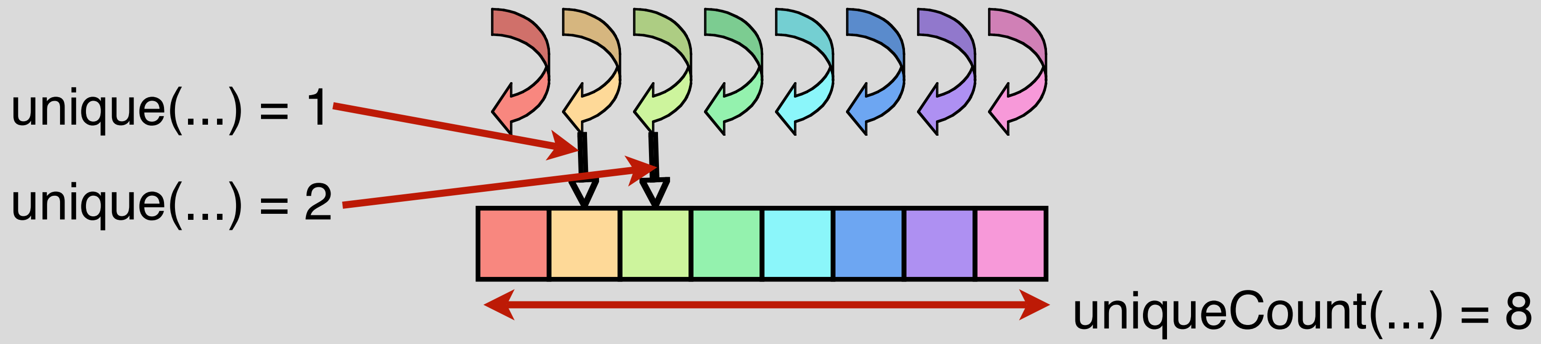
- The case for an architecture
- Why a new architecture?
- What does it look like?
- What's the OS support?
- Conclusion

OS Support

- The OSI model of architecture
  - Services
  - Interfaces
  - Protocols
- *Services* are the building blocks of the architecture
- *Interfaces* are the OS mechanisms and abstractions
- *Protocols* are the actual implementations
- TinyOS is a clean-slate OS design for embedded wireless sensor networks: how can it support needed interfaces?

The Power of Counting, Revisited

- Basic language mechanism that TinyOS provides
- Ability to count elements in an application at compile time
  - unique(key): for each key, returns a unique number starting at 0
  - uniqueCount(key): returns number of calls to unique(key)
- Each needed service or abstraction can use its own key
  - Tasks: unique("TinySchedulerC.BasicTask"), etc.

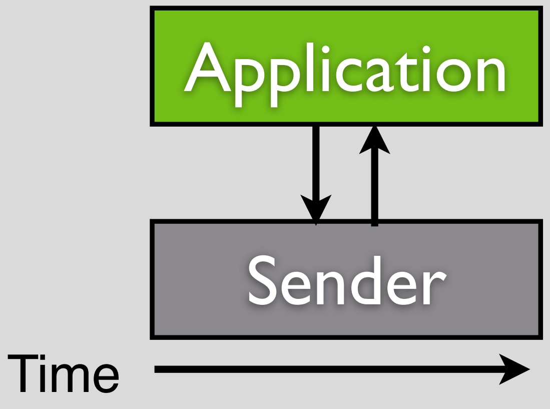


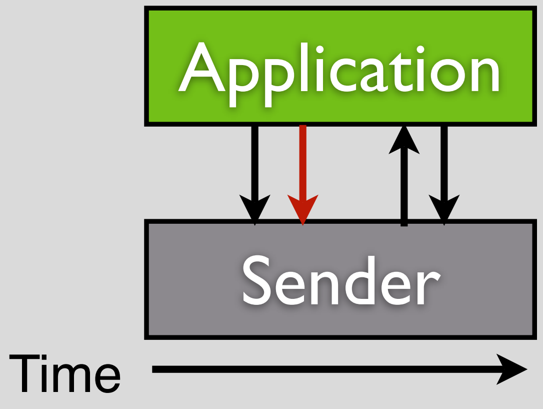
Basic OS Requirement: QoS

- Observed flaw in many protocols: under load, routing fails
  - Data packets overflow queues
  - Control packets are lost, routes disintegrate
- Priority scheduling is difficult, as it can require breaking promises
  - I’ve agreed to forward this data packet, but have to drop it now...
  - Defining priorities across many protocols can be difficult
- Want to be able to promise a *minimum* quality of service
  - Control traffic receives at least k/n of the available bandwidth
  - A control packet has to wait for at most x packets

QoS Through an OS Interface

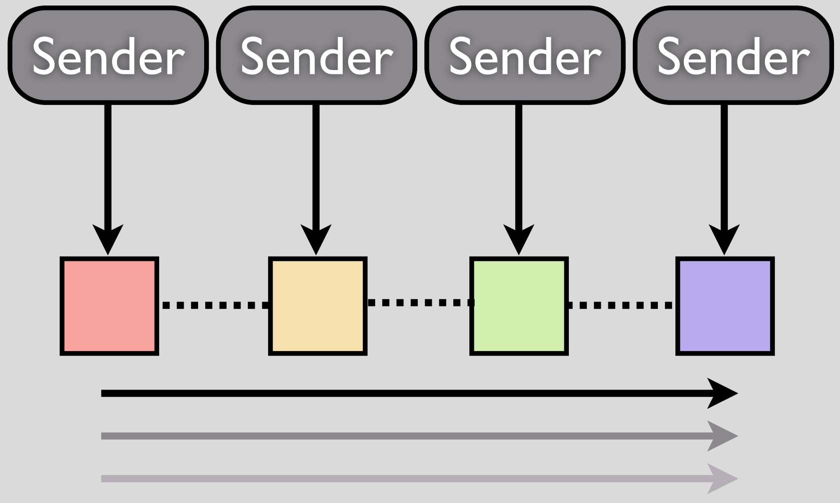
- Every component that needs to send a packet instantiates an instance of a packet sending service
  - Broadcast, collection, unicast, etc.
- Each instance of the service can have at most *one* outstanding packet at any time
- Like tasks, send fails if and only if a packet is already pending





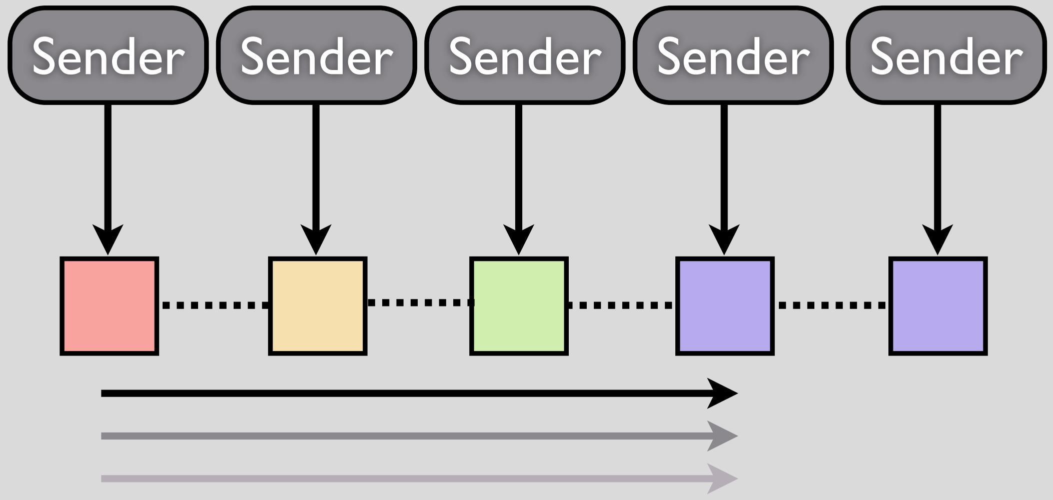
QoS Through Counting

- Each instance allocates a queue entry with unique(...)
- The service has a queue of length uniqueCount(...)
- Implementation scans through the queue for pending packets



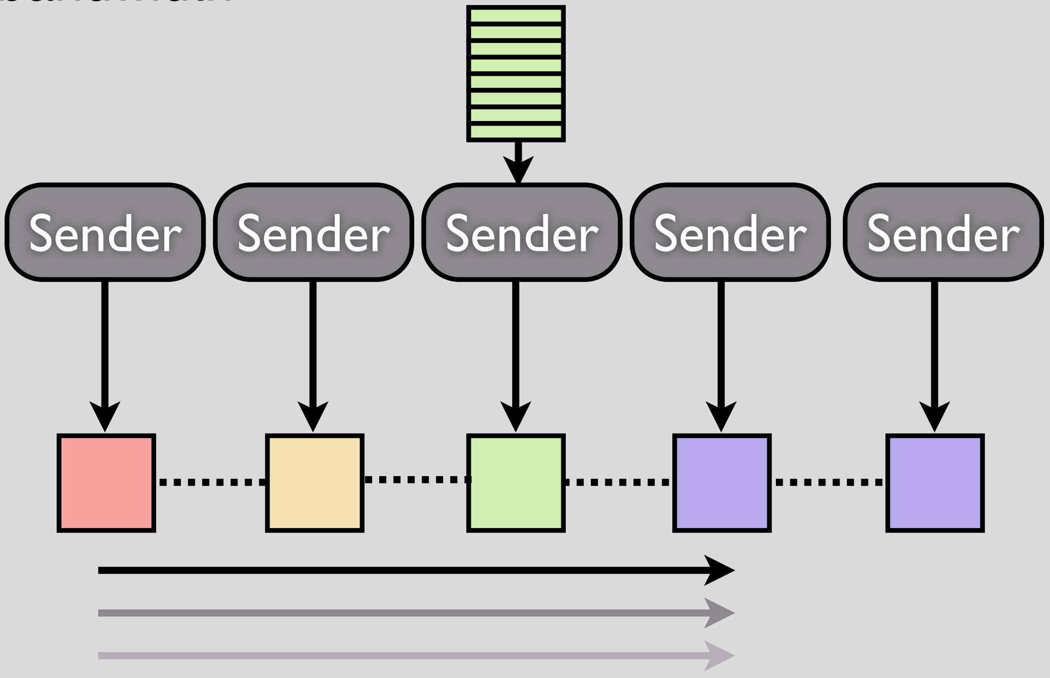
Extending the Model

- A protocol can allocate more than one sender for a greater share



Extending the Model

- A protocol can allocate more than one sender for a greater share
- A protocol can introduce its own queue
- Still uses k/n bandwidth



Open Questions

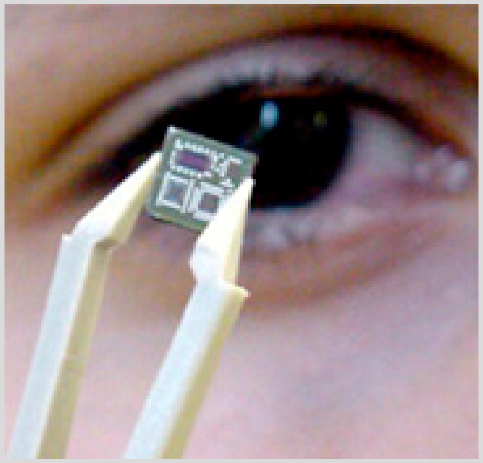
- Fairness vs. optimization
- Transparency vs. simplicity
- Cross-layer concerns

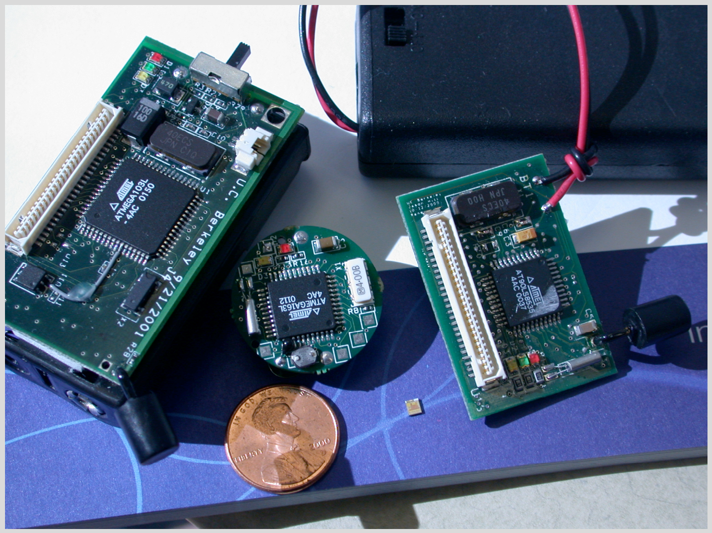
Outline

- The case for an architecture
- Why a new architecture?
- What does it look like?
- What’s the OS support?
- Conclusion

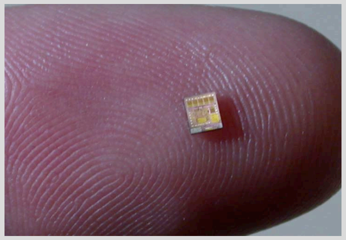
Energy Defines Everything

- Embedment and scale preclude wires
  - Wires are expensive to install and maintain
  - Energy reserves define form factor





=



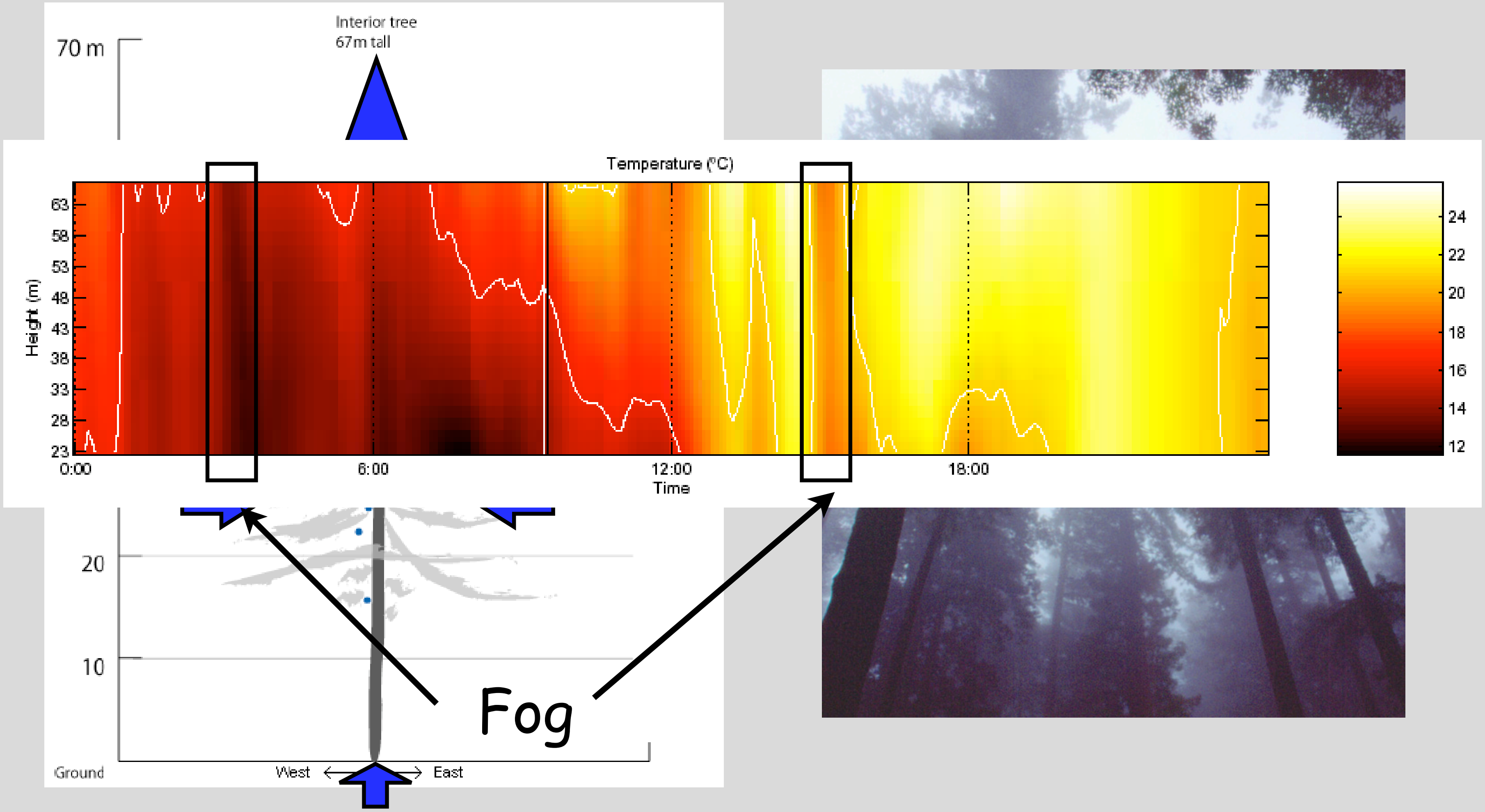
Revisiting Old Assumptions

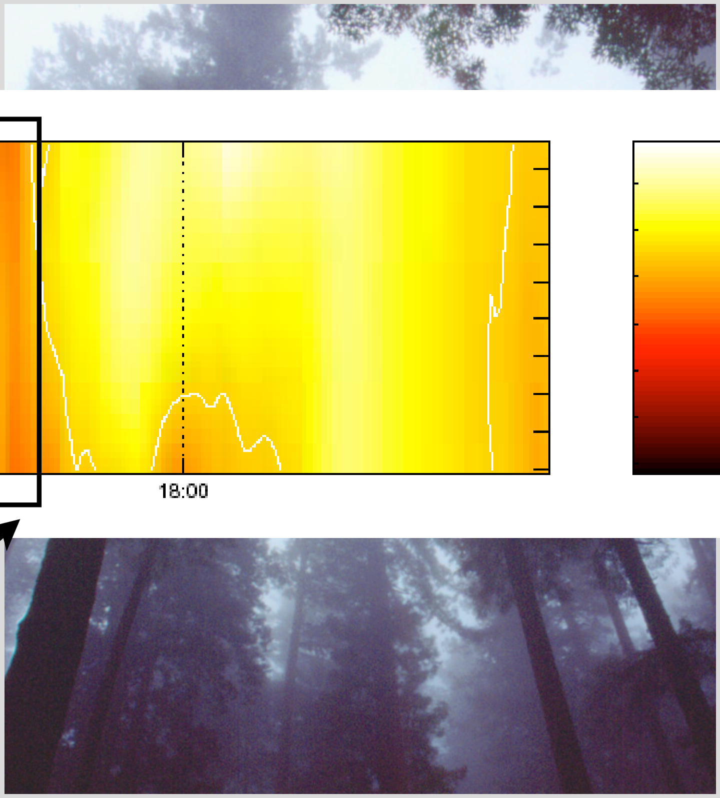
- The requirements and use cases of wireless sensor networks are *qualitatively different* that in other domains
  - Energy
  - Embedment
  - Scale
- These departures deserve a re-examination of many long-held systems and networking approaches
  - Concurrency
  - OS structure
  - Network architecture
- Re-examining these questions may lead to the same conclusions, but in many cases so far they have not....

A Platform for Innovation

- Since its introduction in 1999, TinyOS has gone through four major revisions: 0.43, 0.6, 1.0, 1.1
- Seven years and a half million nodes later, there is an effort to distill the many experiences into a more stable and standardized system
- Standardizing well-understood services will enable reduce academic balkanization and provide a commercially viable platform
    - Innovation *within* the services
    - Innovation through *new* services
- TinyOS 2.0 standardization mechanism: TinyOS Enhancement Proposals (TEPs), akin to IETF RFCs
    - Examples: TEP 102: Timers, TEP 106: Schedulers and Tasks, TEP 116: Packet Protocols, TEP 118: Dissemination

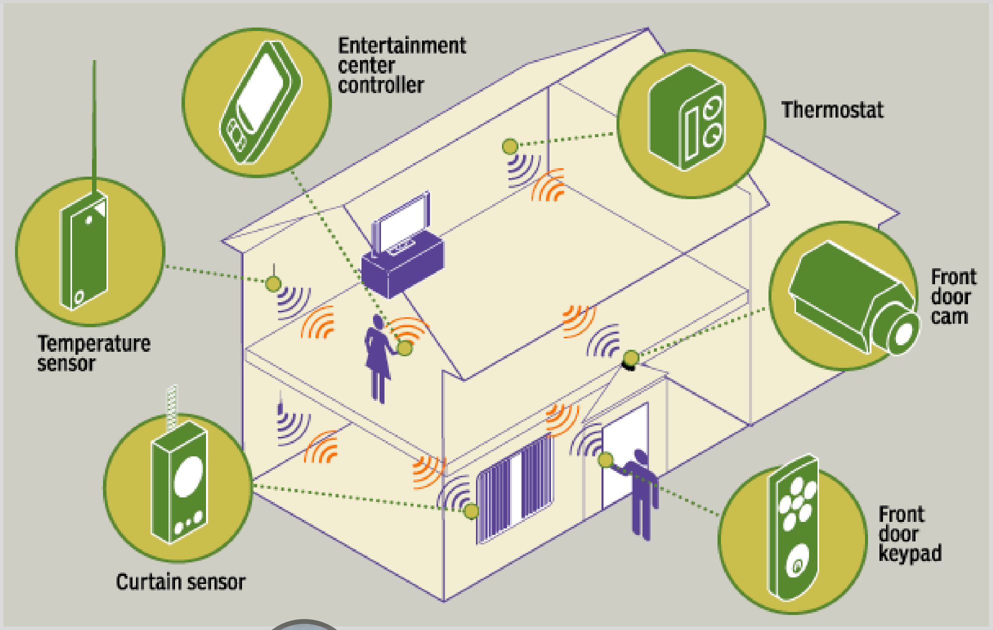
Instrumenting the World

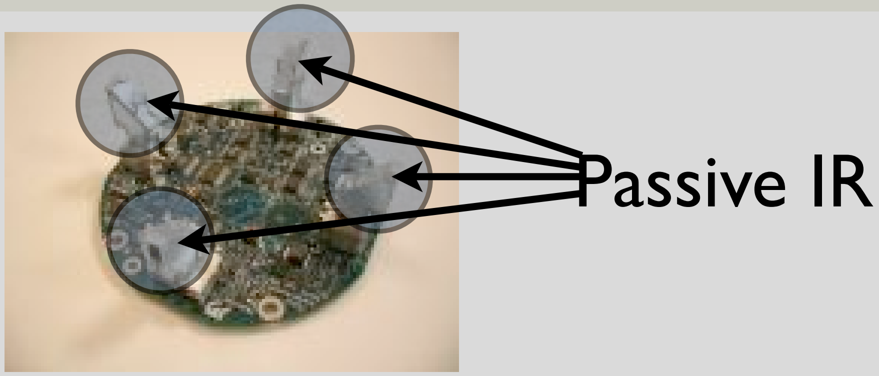




Instrumenting Our Homes

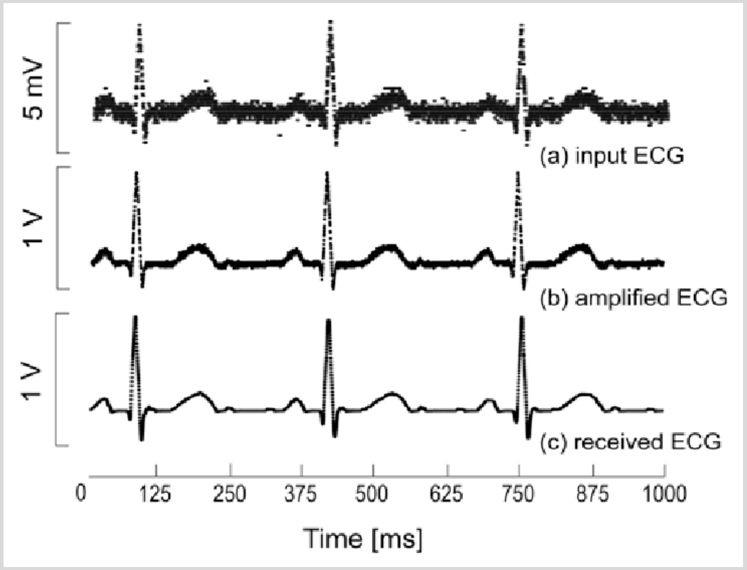
- Instrument every
  - Outlet
  - Faucet
  - Light switch
  - Air duct
  - Door
  - Window
  - Appliance



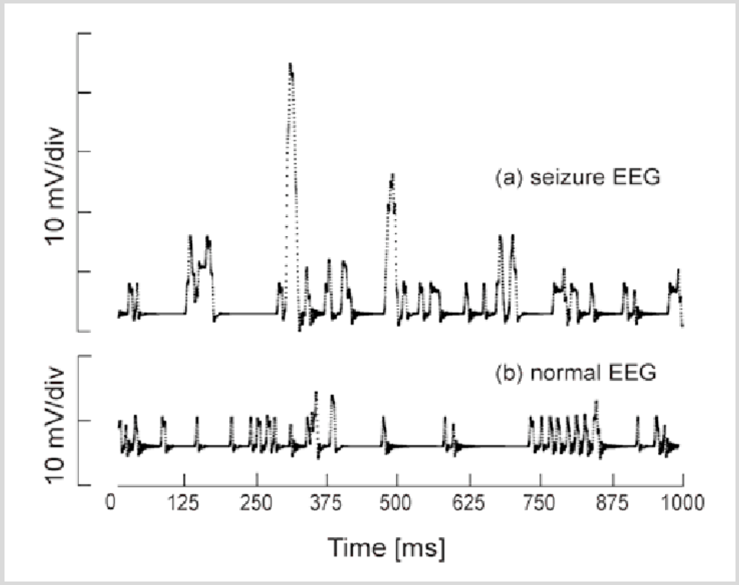


Instrumenting Ourselves

- Biomedical implants and devices
- Micro-stimulation of affected cortical areas can suppress seizures

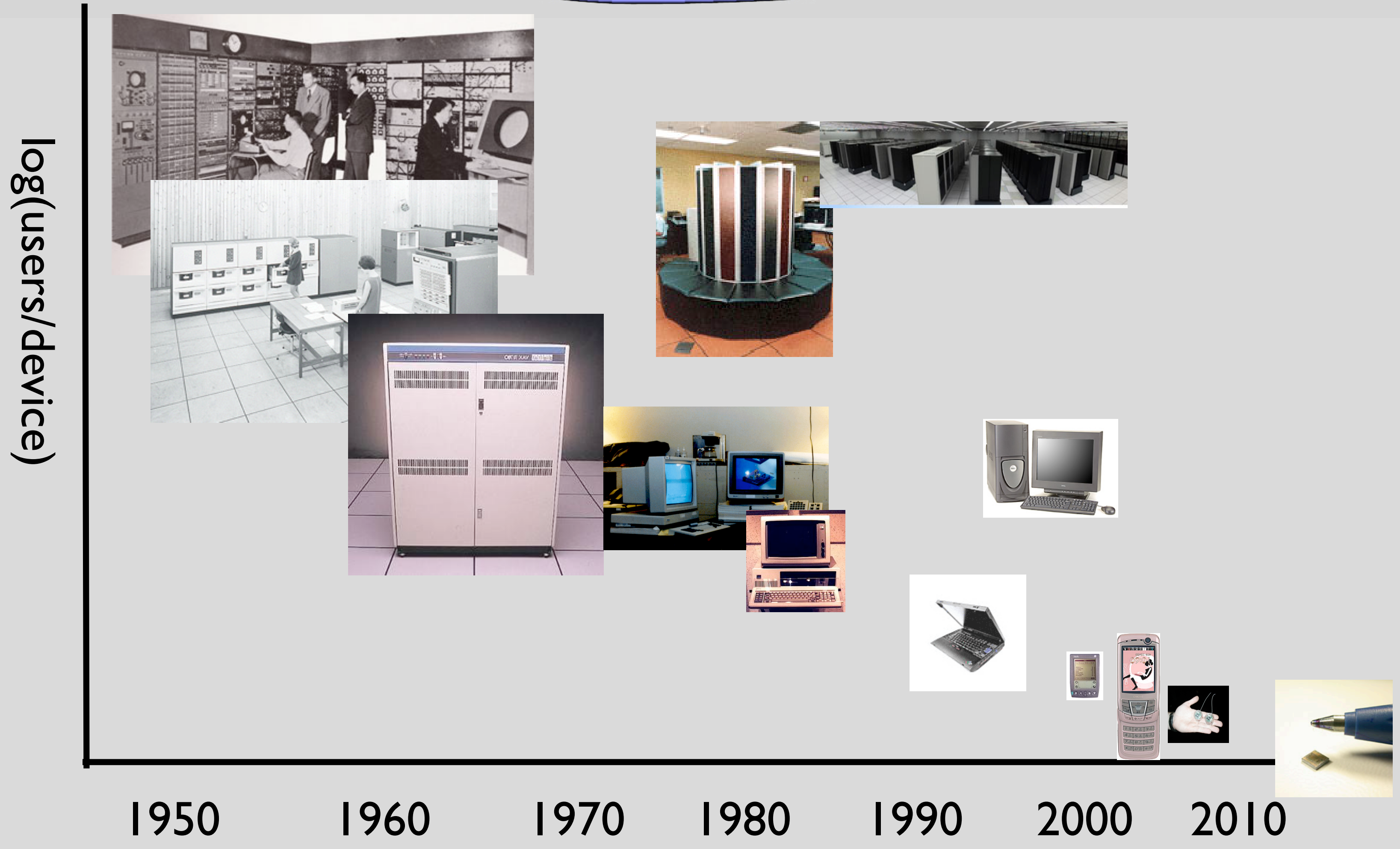


Simulated ECG



Real EEG

Bell’s Law



Conclusion

- TinyOS is a community-driven, open-source operating system
  - Seven years, thousands of users, half a million nodes
  - BSD license: no strings attached (except not using developers)
- Realizing the full potential of wireless sensor networks requires an open and transparent collaboration between academia and industry
  - Academia can afford to take risks and explore novel ideas
  - Industry can provide guidance on requirements and needs
- The TinyOS Alliance provides a forum for this collaboration
  - Anyone and everyone can contribute: code, resources, time...
  - All discussions are transcribed and made available
- Learn, participate, and use: http://www.tinyos.net

Questions