

## Multihop Code Distribution for Sensor Networks

Thanos Stathopoulos, John Heidemann and Deborah Estrin

Laboratory for Embedded Collaborative Systems

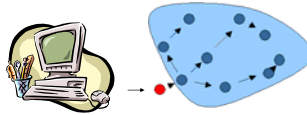
<http://lecs.cs.ucla.edu>

### Multihop Over the Air Programming: Supporting in-situ code updates for motes

#### Useful for users...

Users need over-the-air code distribution to:

- Add new **functionality**
- Facilitate **debugging**
- Extend **usefulness** of the network
- Program nodes that are not **physically reached**
- Automate the process to support **large network sizes**



#### ...and for researchers

- Special case of **data dissemination**
  - Large **volume** of data
  - All nodes in the network **must be reached**
- Strict **reliability requirements**
  - **Everything** must be received
- Limited **resources**
  - **Low-power radios**, limited **memory** and **storage**
- Helps explore sensor net design space for **reliable communications**

### Goals and Design Questions

#### Goals: Resource prioritization

- **Energy: most important** resource
  - Directly related to **radio transmission** and **stable storage (EEPROM) access**
  - Motes must **stay alive** for as long as possible
- **Memory usage:** secondary importance
  - Must limit usage to **less than 1K of RAM**, to leave enough for the **real** application
- **Latency:** the **least important**.
  - Since there is **no real-time** requirement for this application, it can be **traded off for energy**.

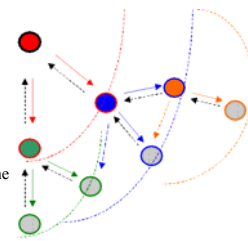
#### Design questions

- **Transfer protocol:** How is data propagated?
  - Stream data to all nodes at the same time (**flooding**)
  - Neighborhood-by-neighborhood dissemination (**ripple-like**)
- **Segment management on the receiver:** How to store, retrieve, keep track of segments?
  - Treat RAM + EEPROM as a **hierarchical data structure**
  - Use a SACK-like **sliding window**
- **Retransmission policy:** How are requests sent, how are replies generated?
  - Requests: **Unicast** vs **broadcast**
  - Suppression mechanisms

### Design and implementation details

#### Approach and analysis

- **Ripple transport protocol:** One source per neighborhood
  - Nodes **periodically advertise** their versions
  - Interested nodes (not already attached to a source) **subscribe**
  - Sources without subscribers are **silent**
  - **Single-hop propagation** from the source to all receivers.
  - **Local** repairs
  - Once a node has the **complete image** it sends **publish** messages and the process repeats itself
  - Significant expected **traffic reduction** compared to flooding at the expense of **latency**



#### Design Alternatives

- **Segment mapping:** SACK-like sliding window
  - Problem: how does the node find **which segments are missing**?

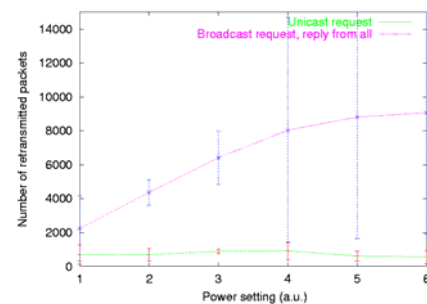
Segment mapping	RAM (bytes)	TX Cost	RX Cost	Gap detection Cost	Out-of-order tolerance
None	0	R	W always	(# of segments)*R	Complete
Full (RAM map)	512	R	W when segment missing, else 0	0	Complete
Partial (k packets per map bit)	512/k	R	KR+W when segment missing, else 0	Up to kR, Minimum R	Complete
Hierarchical full (RAM + EEPROM)	4	R	R+2W when segment missing, else R	R	Complete
Sliding window	M (usually 2.4 bytes)	R	W always	0	Up to map size

- **Retransmission policy:** Energy-latency-complexity tradeoffs

Policy	Expected number of replies	Latency	Complexity
Broadcast request, all nodes reply	$(1-p)^k(k+m)$	0	$O(1)$
Broadcast request, suppressible replies	$(1-p)^k \left\{ 1 + \frac{(k+m-2)}{C} \right\}$	Up to C	$O(\text{neighborhood size})$ for a good estimation of C. Several timers
Broadcast request, all nodes reply with a static probability	$(1-p)^k(a_1 k + a_2 m)$	Depends on selection of $a_1, a_2$	$O(1)$
Broadcast request, all nodes reply with a dynamic probability	$(1-p)^k(a_1 k + a_2 m)$	Depends on selection of $a_1, a_2$	$O(\text{neighborhood size})$ for a good estimation of $a_1, a_2$
Unicast request, only publisher replies	$(1-p)^k$	Considerable if link to publisher fails, else 0	$O(1)$ . 2 extra bytes required on request packet

#### Preliminary results

- Comparison between two different retransmission policies



#### Conclusions and future work

- Design choices for the current implementation
  - **Ripple data transfer**, with a **publish-subscribe** interface and late-joiner support via periodic advertisement
  - **SACK-like sliding window** for energy-efficient segment management and gap (loss) detection.
  - **Unicast** repair requests and replies from the **original source only** provide a large (up to 20x) reduction in the number of duplicate replies at a very low complexity cost
  - 950 Bytes RAM **footprint**
  - The most reasonable selection for a **low-complexity, energy efficient** mechanism, when **loss probability is low**
  - Experimental results **needed** for qualitative comparisons: Ripple vs Flooding, Hierarchical segment mapping vs Sliding Window
- Several more to choose from!
  - Choosing the right segment management scheme or retransmission policy depends on the **resource prioritization** and the **expected loss rate**
  - As in many systems, there is no 'one size fits all'
- Next step: **Deployment at James Reserve**, as part of ESS