Remote Reprogramming of Wireless Sensor Networks

Rajesh Krishna Panta

PhD Final Examination

Advisor : Prof. Saurabh Bagchi
Committee Members : Profs. Samuel P. Midkiff, James V. Krogmeier, Vijay Raghunathan

Dependable Computing Systems Laboratory (DCSL)
Electrical and Computer Engineering Department, Purdue University

PURDUE UNIVERSITY

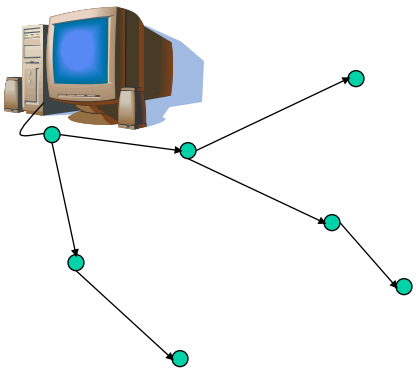
Thesis Statement

- Develop novel techniques for building robust, energy-efficient, and practical middleware services for networks of low power resource-constrained embedded devices
- Contributions
  - Reprogramming
  - Time Synchronization
  - Security
  - MAC

PURDUE UNIVERSITY

Introduction: Sensor Network Reprogramming

- Uploading new software while the nodes are *in situ*, embedded in their sensing environment



- Fix software bugs
- Adapt to changing user needs and environmental conditions in which the network is deployed
- Shorten software development phase
- Make software robust
- Fine-tune algorithms
- Complete application replacement

PURDUE UNIVERSITY

Requirements of Network Reprogramming

- For correctness, all nodes in the network should receive the code completely
  - Reliable dissemination using unreliable wireless channels is challenging
- For performance, code upload should minimize
  - *reprogramming time* so that sensor nodes can quickly resume their normal function
  - *reprogramming energy* spent in disseminating code through the network since sensor nodes have limited energy
- Solution should fit within computation, memory, and bandwidth constraints of sensor nodes

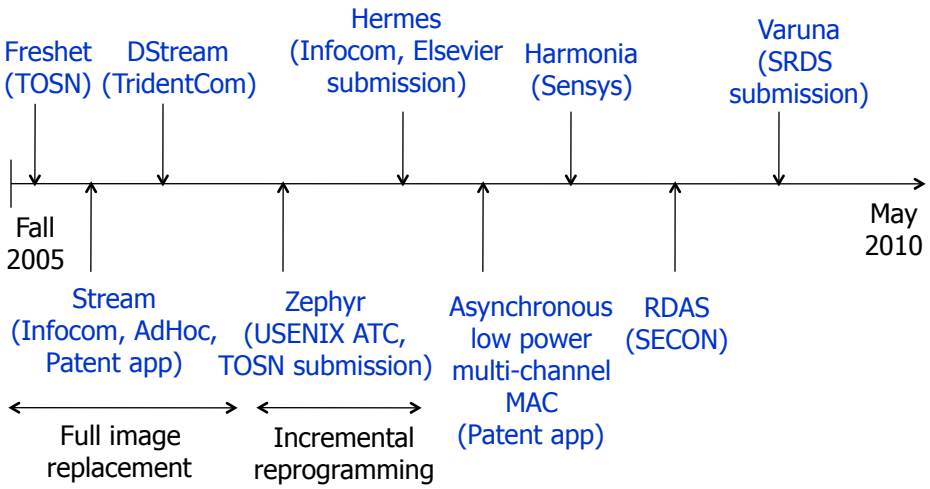
PURDUE UNIVERSITY

Outline of the Talk

- Research contributions
- Hermes: Incremental reprogramming system
  - Byte level comparison
  - Application level modifications
- Varuna: Steady state maintenance protocol
  - Fixed steady state energy cost
- Conclusion

PURDUE UNIVERSITY

Research Contributions



PURDUE UNIVERSITY

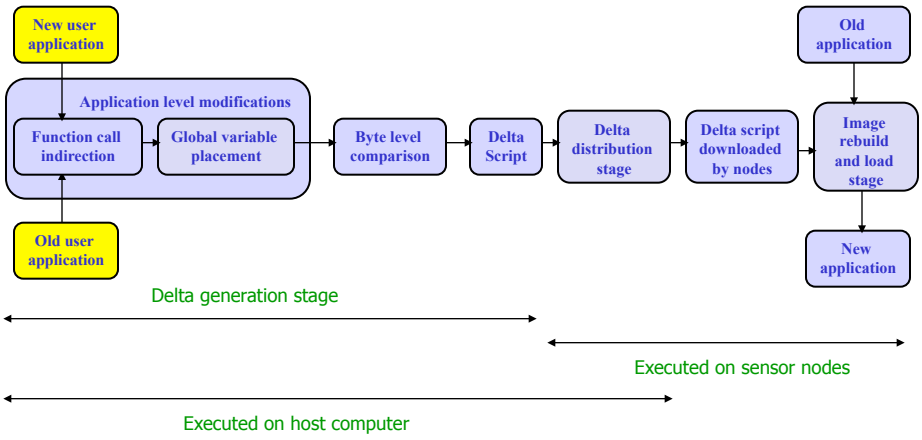
Hermes: Motivation

- In practice, software running on the sensor nodes evolves with incremental changes to its functionality
- TinyOS [Berkeley] does not support dynamic linking on the sensor nodes
  - Cannot transfer just the components that have changed and link them in at the node
- SOS [Han05] and Contiki [Dunkels04] support dynamic linking on the nodes
  - Limitations of position independent code in SOS
  - Wireless transfer of symbol and relocation tables in Contiki is costly
- Instead of transferring the entire image, Hermes transfers the *difference* between the old and new versions of the software

[Berkeley] www.tinyos.net
[Dunkels04] Dunkels, A., Gronvall, B. and Voigt, T., "Contiki-a lightweight and flexible operating system for tiny networked sensors", Proceedings of the 29th Annual IEEE Conference on Local Computer Networks.
[Han05] Han, C.C., Kumar, R., Shea, R., Kohler, E. and Srivastava, M., "A Dynamic Operating System for Sensor Nodes", Proceedings of the 3rd Conference on Mobile Systems, applications and services.

PURDUE UNIVERSITY

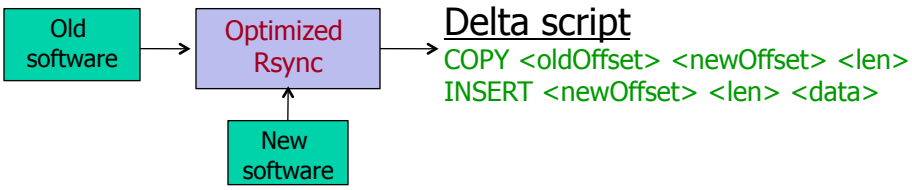
Overview of Hermes



PURDUE UNIVERSITY

Byte Level Comparison: Rsync

- Rsync[Tridgell99] divides the images to be compared into fixed size blocks
- Rsync finds the matching blocks between the two images using comparison at two levels – checksum and MD4
- Optimized Rsync finds the *maximal superblock*



[Tridgell99] Trigdell, A. , "Efficient algorithms for Sorting and Synchronization", Ph.D. Thesis, Australian University, 1999.

PURDUE UNIVERSITY

Byte Level Comparison Alone is not Sufficient (1)

- To see the drawback of using optimized Rsync alone, consider the following cases of software changes:
  - Case I (Changing Blink application)
    - Changing an application from blinking a green LED every second to blinking every 2 seconds
    - A single parameter change (very small change)
    - Delta script produced by optimized Rsync (byte level comparison) is 23 bytes - proportional to the amount of the actual change made in the software
  - Case II (Adding few lines of code to Blink application)
    - This is also a small change.
    - But delta script is 2183 bytes - disproportionately larger than the amount of actual change made in the software
  - Case III (Adding one global variable to Blink application)
    - Again a small change
    - The delta script is 6090 bytes.

PURDUE UNIVERSITY

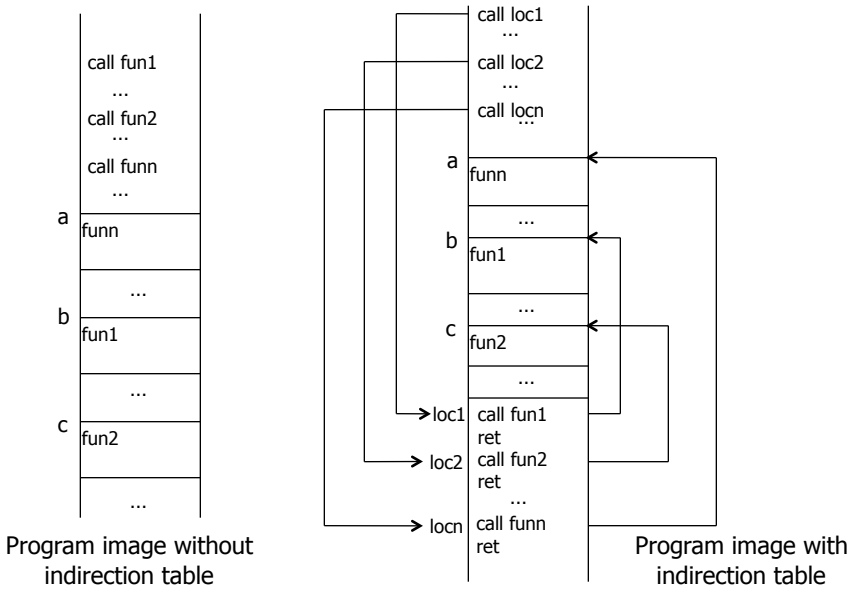
Byte Level Comparison Alone is not Sufficient (2)

- None of the functions shift in Case I
- Functions following the added lines get shifted in Case II, causing all the function call statements referring to the shifted functions to change
- In Case III, many global variables are shifted in memory due to addition of a new variable, causing all the instructions that refer to the shifted variables to change

Size of the delta script produced by byte level comparison alone may be huge even if the actual amount of change is small. So application level modifications are necessary *before* performing byte level comparison

PURDUE UNIVERSITY

Function Call Indirections

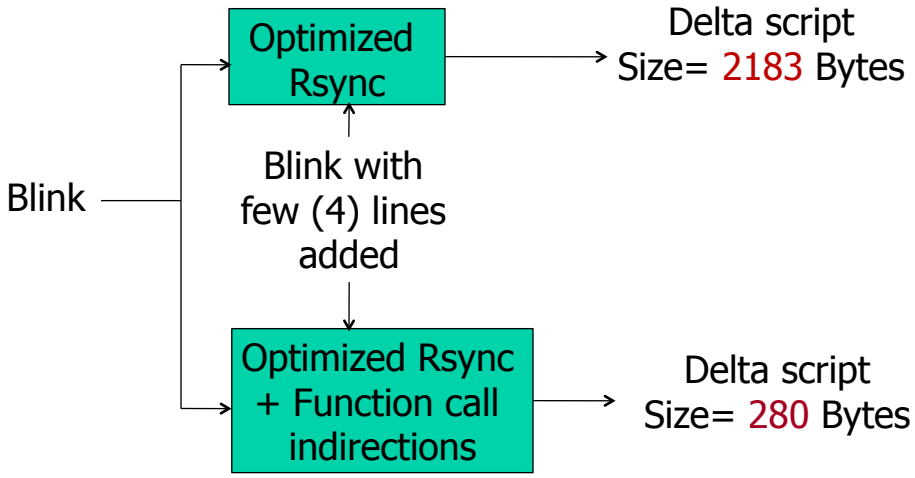


Program image without indirection table

Program image with indirection table

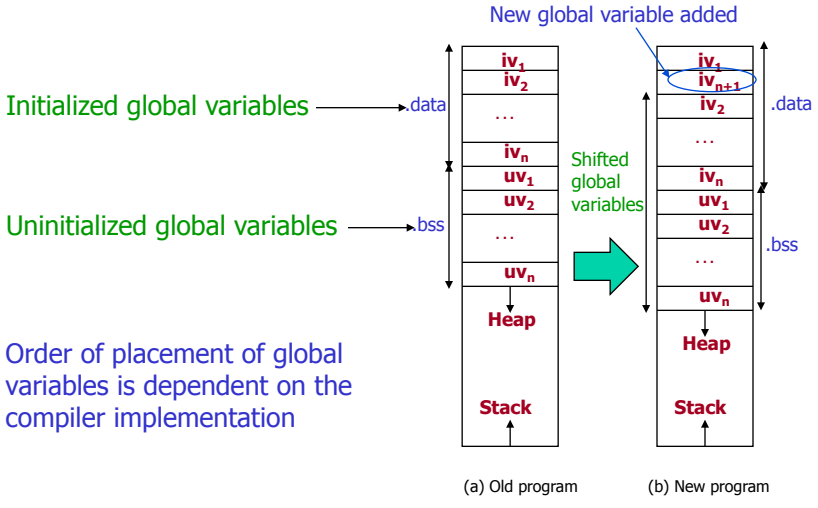
PURDUE UNIVERSITY

Effect of Function Call Indirections



PURDUE UNIVERSITY

Placement of Global Variables in RAM



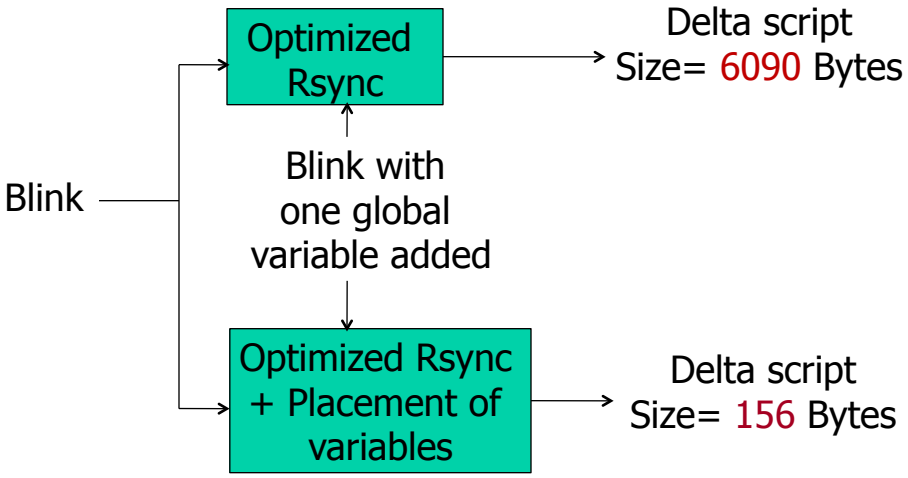
PURDUE UNIVERSITY

Hermes Solution Approach

- Members of a structure are placed in memory in the same order as they are defined in the structure
- Hermes scans through all the source files of the application and transforms the initialized and un-initialized global variables into members of two structures, *iglobStruct* and *uglobStruct* respectively
  - Hermes places a global variable in the new version of the software at the same location in the structure, and hence in RAM as in the old version

PURDUE UNIVERSITY

Improvement due to Elimination of Global Variable Shifts



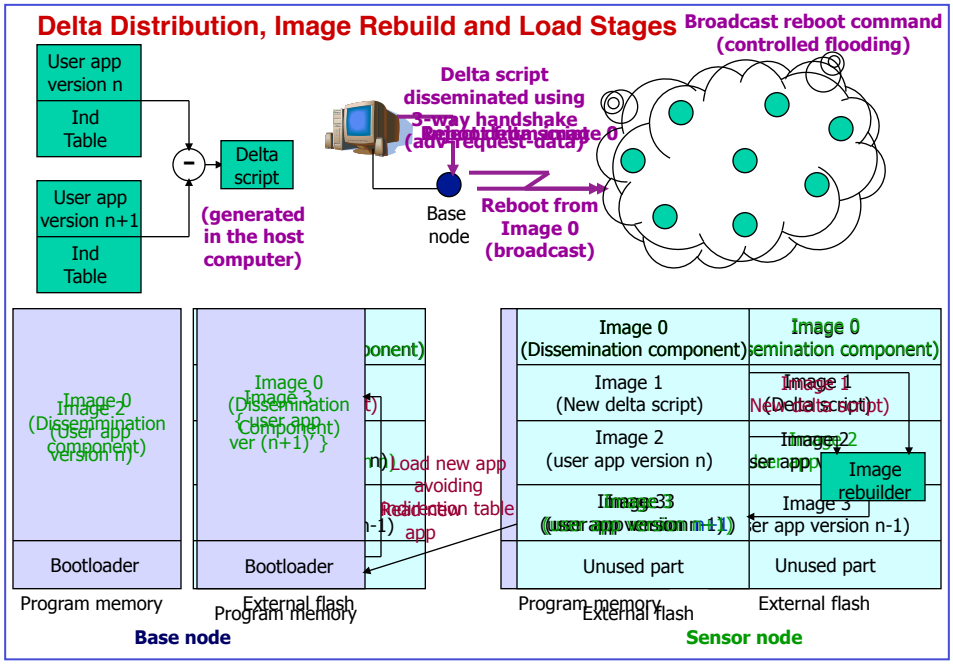
PURDUE UNIVERSITY

Latency due to Function Call Indirection

- One level of indirection increases the latency of the user program by few CPU cycles (e.g. 8 cycles in AVR platform) for each function call
- The increase in latency accumulates over time
  - Sensor network applications run in a continuous loop
- Hermes avoids this latency by using the exact function address in call statements while loading the latest software image from the external flash to the program memory

PURDUE UNIVERSITY

Delta Distribution, Image Rebuild and Load Stages



PURDUE UNIVERSITY

Experiments : Evaluation of Function Call Indirections

Case 1	Blink application blinking green LED every second to blinking every 2 seconds.	Small change (SC)
Case 2	Few lines added to the Blink application	Moderate change (MC)
Case 3	Blink application to CntToLedsAndRfm	Very large change (VLC)
Case 4	CntToLeds to CntToLedsAndRfm	Very large change (VLC)
Case 5	Blink to CntToLeds	Large change (LC)
Case 6	Blink to Surge	Very large change (VLC)
Case 7	CntToRfm to CntToLedsAndRfm	Large change (LC)
Case A	An application that samples battery voltage and temperature from MTS310 sensor board to one where few functions are added to sample the photo sensor also.	Large change (LC)
Case B	Few functions were deleted to remove the light sampling features.	Large change (LC)
Case C	Added the features for sampling all the sensors on the MTS310 board except light (e.g. magnetometer, accelerometer, microphone). Collected mean and mean square values of the samples taken during a user specified window size.	Very large change (VLC)
Case D	Same as Case C but with addition of few lines of code to get microphone peak value over the user specified window size.	Moderate change (MC)
Case E	Removed the feature of sensing and wirelessly transmitting to the base node the microphone mean value.	Moderate change (MC)
Case F	Added the feature of allowing the user to put the nodes to sleep for the user specified duration.	Very large change (VLC)
Case G	Changed the microphone gain parameter.	Small change (SC)

Standard TinyOS applications

eStadium applications

PURDUE UNIVERSITY

Testbed Experiments

- Topology: 2x2, 3x3, and 4x4 grid networks; Linear network with 2, 3, …, 10 nodes (mica2 motes)
- A node at one corner of the grid or the end of the line acts as a base node.
  - Base node generates delta for the various software change cases discussed above and injects the delta in the network
- Compare delta script size, network reprogramming time and energy of **Zephyr** (same as Hermes except it does not eliminate variable shifts) with Deluge[Hui], Stream[Panta], Rsync[Jeong], and Optimized Rsync
  - Use number of packets transmitted in the network as a measure of reprogramming energy

[Hui]J.W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale." SenSys 2004.
[Panta] R.K.Panta, I. Khalil, S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," Infocom 2007.
[Jeong]J. Jeong, D. Culler, "Incremental network programming for wireless sensors," SECON 2004.

PURDUE UNIVERSITY

Size of Delta Script

	Very large change	Large change	Moderate change	Small change
	Case 4	**Case 7**	**Case E**	**Case G**
Deluge : Zephyr	4.29	29.76	243.25	1987.2
Stream : Zephyr	2.65	18.42	168.40	1324.8
Rsync : Zephyr	1.96	8.34	42.03	49.6
OptRsync : Zephyr	1.57	3.87	9.016	1.4

Deluge needs to transfer up to 1987 times more bytes than Zephyr.
Optimized Rsync generates delta script of size up to 9.01 times more than Zephyr.

PURDUE UNIVERSITY

Reprogramming Time

	Class 1 (SC)			Class 2 (MC)			Class 3 (LC)			Class 4 (VLC)		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Deluge:Zephyr	22.39	48.9	32.25	25.04	48.7	30.79	14.89	33.24	17.42	1.92	3.08	2.1
Stream:Zephyr	14.06	27.84	22.13	16.77	40.1	22.92	10.26	20.86	10.88	1.46	2.23	1.54
Optimized Rsync:Zephyr	1.01	1.1	1.03	2.01	4.09	2.71	2.05	3.55	2.54	1.27	1.55	1.35

Zephyr is up to 48.9, 40.1 and 4.09 times faster than Deluge, Stream, and optimized Rsync without application level modifications, respectively.

PURDUE UNIVERSITY

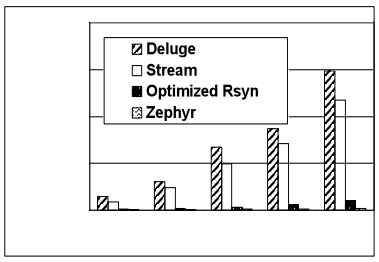
Reprogramming Energy

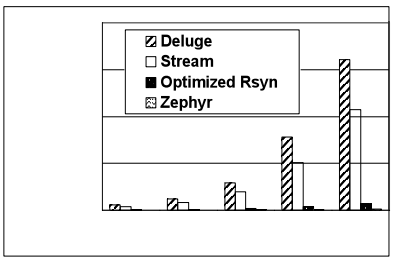
	Class 1 (SC)			Class 2 (MC)			Class 3 (LC)			Class 4 (VLC)		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Deluge:Zephyr	90.01	215.3	162.5	40	204.3	101.1	12.27	55.46	25.65	2.51	2.9	2.35
Stream:Zephyr	53.76	117.9	74.63	28.16	146.1	82.57	8.6	36.19	15.97	1.62	2.17	1.7
Optimized Rsync:Zephyr	1.13	1.69	1.3	4.38	22.97	9.47	2.72	10.58	3.95	1.38	1.64	1.49

Deluge, Stream, and optimized Rsync without application level modifications transfer up to 215, 146 and 22 times more bytes than Zephyr, respectively.

PURDUE UNIVERSITY

TOSSIM Simulation Results





Zephyr is up to 92.9, 73.4, and 6.3 times faster than Deluge, Stream, and optimized Rsync without application level modifications, respectively. Deluge, Stream, and optimized Rsync transmit up to 146.4, 97.9 and 6.4 times more number of packets than Zephyr, respectively.

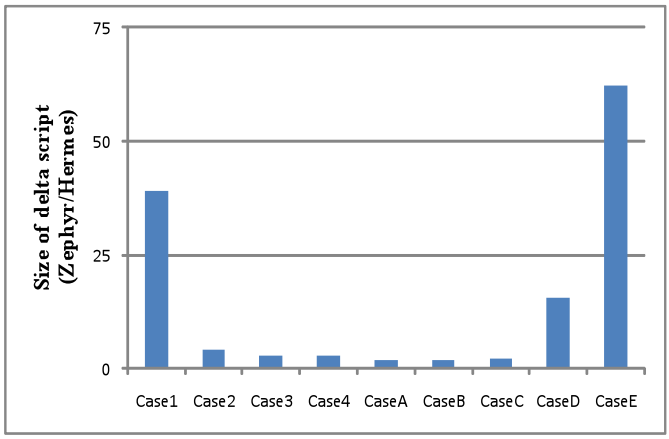
PURDUE UNIVERSITY

Experiments: Evaluation of Elimination of Variable Shifts

Case 1	Blink to Blink with a global variable added	Standard TinyOS applications
Case 2	Blink to CntToLeds	
Case 3	Blink to CntToLedsAndRfm	
Case 4	CntToLeds to CntToLedsAndRfm	
Case A	An application that samples battery voltage and temperature from MTS310 sensor board to one where few functions are added to sample the photo sensor also.	
Case B	Few functions were deleted to remove the light sampling features.	
Case C	Added the features for sampling all the sensors on the MTS310 board except light (e.g. magnetometer, accelerometer, microphone).	eStadium applications
Case D	Same as Case C but with the addition of a feature to reduce the frequency of sampling battery voltage.	
Case E	Same as Case D but with the addition of a feature to filter out microphone samples (considering them as noise) if they are greater than some threshold value.	

PURDUE UNIVERSITY

Size of Delta Script



- Delta script generated by Deluge and Zephyr are up to 201 and 62 times larger than Hermes, respectively
- Deluge and Zephyr transfer up to 150 and 46 times more bytes than Hermes, respectively.

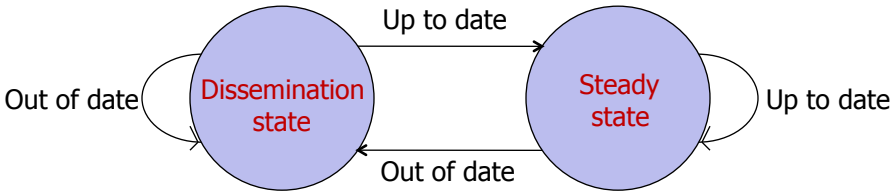
PURDUE UNIVERSITY

Outline of the Talk

- Research contributions
- Hermes: Incremental reprogramming system
  - Byte level comparison
  - Application level modifications (Main focus)
- **Varuna: Steady state maintenance protocol**
  - **Fixed steady state energy cost**
- Conclusion

PURDUE UNIVERSITY

Steady State Maintenance: Motivation



Energy expenditure for code dissemination

Energy expenditure even when there is no code dissemination

- Why steady state energy cost?
  - Dynamic network topology caused by transient link failures, node mobility, incremental node deployment, etc
  - Nodes may remain disconnected from the network for some time and may miss the code dissemination
  - After they come out of disconnection, they must detect the inconsistency
  - To ensure that all nodes are up to date all the time, existing systems periodically broadcast advertisement message containing metadata – e.g. version number of the code

PURDUE UNIVERSITY

Drawback of Periodic Advertisement

- Steady state energy cost increases linearly with the steady state period – the most dominant phase in a node's lifetime
  - In one day, default steady state advertisement rate of Deluge (1 adv/2 mins) incurs the same number of radio transmissions as disseminating a 25KB program code
- Radio transmissions are the most energy expensive operations

• Learning when to disseminate code is overwhelmingly more expensive than disseminating the code itself
• Goal: Make steady state energy cost independent of steady state duration

PURDUE UNIVERSITY

Trickle

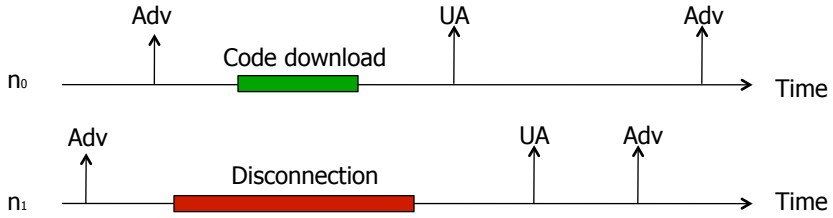
- Each node broadcasts an advertisement message at a time instant chosen randomly from an interval T if it has not heard more than k identical advertisements in that interval
- $T_L \leq T \leq T_H$
- When a node hears a different advertisement, $T = T_L$
- When a node does not hear a different advertisement, T is doubled in successive intervals till $T = T_H$, and kept constant thereafter
- Advertisement suppression is necessary to make the scheme scalable with high node density
  - Without loss, collision, and with perfect time synchronization, the number of advertisements in any time interval within a single hop is bounded by k
  - With these practical conditions, the number of advertisements in a given time interval is $O(logN)$ where N is the number of nodes within a single hop *[Levis]*
  - Number of advertisements during steady state period Ts ($Ts >> T$) is $O(Ts)$

[Levis] P. Levis, N. Patel, D. Culler, and S, Shanker, "Ttickle: A Self Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks" NSDI 2004.

PURDUE UNIVERSITY

Increasing Advertisement Interval is not a Solution

- Steady state energy cost can be reduced by increasing the advertisement interval
- Increase in advertisement interval also increases *detection latency* – time taken by a node to realize that it is out of date
- Steady state energy cost still increases linearly with time
- If advertisement interval is greater than the time required by a node to download the code, communication between inconsistent nodes is possible



PURDUE UNIVERSITY

Insight 1: Unnecessary Advertisements

- A node does not need to broadcast an advertisement message if its metadata and neighborhood topology have not changed since its last advertisement transmission
- Most of Trickle's advertisements in the steady state are unnecessary
- A node can determine trivially whether its metadata has changed – through a local lookup
- Determining if the neighborhood topology has changed is difficult – requires wireless communication among the neighboring nodes

PURDUE UNIVERSITY

Insight 2 : Monitor User Application Traffic

- Relaxed requirement: It is generally sufficient for a node to verify the freshness of its metadata not with all neighbors, but only with the nodes with which it communicates
- Instead of periodic advertisements, listen to User Application (UA) traffic to check if the neighborhood has changed
  - UA traffic – all messages that are not part of the dissemination protocol

PURDUE UNIVERSITY

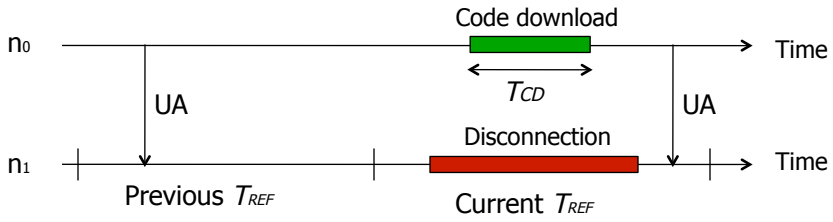
Intuitive Approach

- Time is divided into intervals of length T_{REF}
- Each node monitors the UA traffic to see if it receives a UA packet from a node from which it did not receive any packet in the previous interval
- If so, the node assumes that the neighborhood topology might have changed since the last interval
  - Exchange the advertisement messages to check if the nodes are consistent
- Significant reduction of energy consumption

PURDUE UNIVERSITY

How to Choose T_{REF}? (1)

- T_{REF} should be sufficiently large so that with a high probability, a node hears UA packets from all its neighbors within each T_{REF}
  - In the worst case when the neighbor table changes in every T_{REF} interval, this scheme is equivalent to Trickle with advertisement period equal to T_{REF}
- But T_{REF} cannot be increased arbitrarily

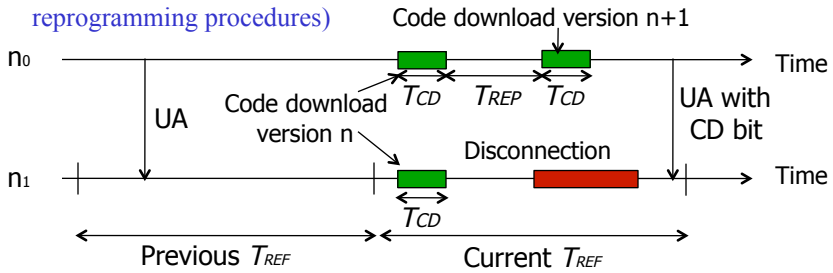


Necessary condition: $T_{REF} \leq T_{CD}$ (T_{CD}: Time to download the code by a node)

PURDUE UNIVERSITY

How to Choose T_{REF}? (2)

- When $T_{REF} > T_{CD}$, piggyback a *Code Downloaded* (CD) bit in each UA packet transmission for T_{REF} interval, after downloading the new version of the code
- But T_{REF} cannot be larger than T_{REP} (minimum time between successive reprogramming procedures)

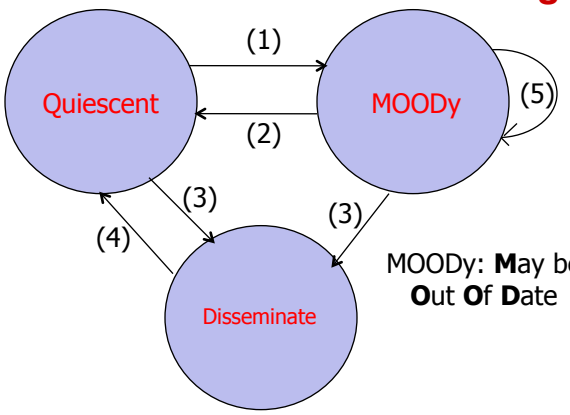


Necessary condition $T_{REF} \leq T_{REP}$

- Fundamental problem with any scheme that uses refresh interval – since UA can be arbitrary, no matter how well a T_{REF} is chosen, in the worst case, a neighbor can be such that it sends a UA packet at every other T_{REF} interval (equivalent to Trickle with advertisement period equal to T_{REF})

PURDUE UNIVERSITY

Varuna Design



(1) UA msg received from a *new* neighbor

(2) Similar Adv msg received from dest or no Adv Msg is received for T_{MOODy}

(3) ReqToDisseminate msg received or broadcast

(4) Code update complete

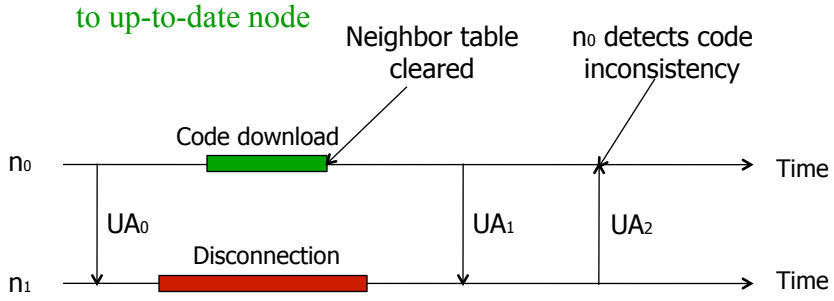
(5) No Adv Msg received for $t \leq T_{MOODy}$

- No refresh interval
- Neighbor table is cleared when the node boots or its metadata changes
- When the neighbor table is full, the least recently used (LRU) neighbor is replaced by a new node

PURDUE UNIVERSITY

Eventual Consistency

- Varuna's invariant: If a node receives a packet from another node with a lower version of the metadata than its own, the metadata inconsistency is detected by the receiving node
  - Communication from up-to-date node to out-of-date is possible
  - Eventual consistency when out-of-date node sends a UA packet to up-to-date node



PURDUE UNIVERSITY

Fixed Steady State Cost

- After a node downloads a new version of the code, it verifies its metadata with each of its neighbor only once
- Steady state energy cost is independent of the steady state period
  - Sufficient memory
  - Relatively good link reliability

PURDUE UNIVERSITY

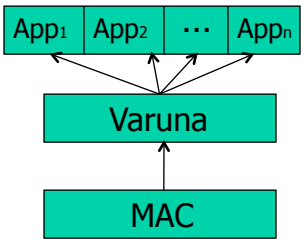
State Maintenance Cost

- No state maintenance in Trickle
- Varuna needs to maintain a neighbor table
- Trend towards increasing RAM for sensor nodes
  - 512 bytes in Rene mote to 256KB in IMote2
- Reasonable RAM requirement
  - Size of neighbor table increases with density, not the total number of nodes in the network
  - For most practical cases, neighbor table requires less than 200 bytes
  - For *very dense networks,* less than 600 bytes are sufficient
- Neighbor table is a fundamental data structure
  - Used by many MAC protocols, routing protocols, 6LoWPAN standard, ZigBee, and many applications

PURDUE UNIVERSITY

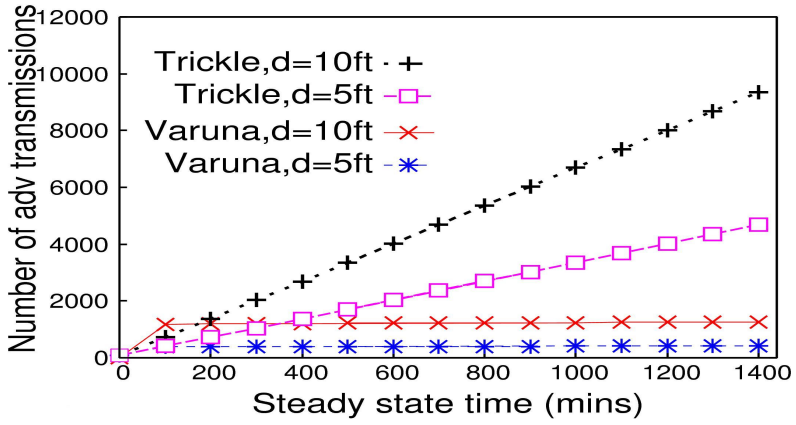
Experiments and Evaluation

- 5x6 grid network of TelosB nodes
- UA packets → U[0,60sec]
- T_L = 2 sec, T_H = 120 sec, k = 2
- T_{MOODy} = 60 sec
- Use number of packets transmitted in the network by the Trickle and Varuna as a measure of energy cost



PURDUE UNIVERSITY

Testbed Results: Steady State Energy Cost



- **In just 1 day, Trickle consumes 8 and 11 times more energy than Varuna for *d*=10ft and *d*=5ft, respectively.**
- **Steady state cost is independent of the steady state duration in Varuna.**

PURDUE UNIVERSITY

Simulation Results: Neighbor Table Size

- TOSSIM
- 20x20 grid
- Other parameters same as those for testbed experiments

d	Neighbor table size
20 ft	8 slots
10ft	26 slots
5 ft	90 slots

For very dense network with 100x redundancy, less than 100 slots are sufficient for neighbor table

PURDUE UNIVERSITY

Conclusions

- Proposed complete software image dissemination protocols
- Introduced an idea of single vs. multi-hop modes of reprogramming
- Used novel techniques for incremental reprogramming of sensor networks
- Proposed a steady state maintenance protocol
- Significant reduction in time required to reprogram the network as well as dissemination and steady state energy consumption

PURDUE UNIVERSITY

Publications (1)

1. Rajesh K. Panta, Saurabh Bagchi, Samuel P. Midkiff, "Efficient Incremental Code Update for Wireless Sensor Networks," Submitted to ACM Transactions on Sensor Networks(TOSN).
2. Rajesh K. Panta, Saurabh Bagchi, "Mitigating the Effects of Software Component Shifts for Incremental Reprogramming of Wireless Sensor Networks", Submitted to Elsevier AdHoc Networks Journal.
3. Rajesh K. Panta, Madaline Vintila, Saurabh Bagchi, "Fixed Cost Maintenance for Information Dissemination in Wireless Sensor Networks,"Submitted to SRDS 2010.
4. Rajesh K.Panta, Saurabh Bagchi, Issa Khalil, "Efficient wireless reprogramming through reduced band width usage and opportunistic sleeping," Elsevier AdHoc Networks Journal, 2009.
5. Rajesh K. Panta, Saurabh Bagchi, Samuel P. Midkiff, "Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation," USENIX ATC, 2009.
6. Rajesh K. Panta, Saurabh Bagchi, "Hermes: Fast and Energy Efficient Incremental Code Updates for Wireless Sensor Networks," IINFOCOM, 2009.
7. Mark D. Krasniewski, Rajesh K. Panta, Saurabh Bagchi, C-L. Yang,W.J. Chappell, "Energy-efficient, On-demand Reprogramming of Large-scale Sensor Networks," ACM Transactions on Sensor Networks (TOSN), 2008.
8. Rajesh K. Panta, Saurabh Bagchi, Issa Khalil, Luis Montestruque, "Single versus Multi-hop Wireless Reprogramming in Sensor Networks," TridentCom, 2008.
9. Rajesh K. Panta, Issa Khalil, Saurabh Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," INFOCOM, 2007.

PURDUE UNIVERSITY

Publications (2)

10. Carlos Perez-Toro, Rajesh K. Panta, Saurabh Bagchi,"RDAS: Reputation-Based Resilient Data Aggregation in Sensor Network," To appear IEEE SECON, 2010.
11. JinKyu Koo, Rajesh K. Panta, Saurabh Bagchi, Luis Montestruque,"A Tale of Two Synchronizing Clocks," SenSys, 2009.
12. Vinaitheerthan Sundaram, Matthew TanCreti, Rajesh K. Panta, Saurabh Bagchi, "Component Dependency based Micro-Rejuvenation Scheduling," Fast Abstract in DSN, 2008.
13. Keisuke Nakano, Rajesh K. Panta, Masakazu Sengoku, Shoji Shinoda,"On performance of a charging/rewarding scheme in mobile ad-hoc networks," IEEE ISCAS, 2005.
14. Rajesh K. Panta,Keisuke Nakano, Masakazu Sengoku,Shoji Shinoda,"Basic Properties of Charging and Rewarding Scheme in Ad-hoc Networks," IEICE Society Conference, 2004.
15. Rajesh K. Panta, Keisuke Nakano, Masakazu Sengoku, Narumi Umeda, Shoji Shinoda, "Analytic Estimation of Cell Extension with Finite Relaying Capacity of Relay Nodes," IEEE MWSCAS, 2004.
16. Keisuke Nakano, Rajesh K. Panta, Satoshi Narita, Masakazu Sengoku, Narumi Umeda, Shoji Shinoda, "Multi-hop Networking with Relaying Capacity for Cellular Mobile Systems," IEEE VTC, 2004.
17. Rajesh K. Panta, Keisuke Nakano, Masakazu Sengoku, Shoji Shinoda, "Effects of Mobility and Capacity on Cell Extension by Wireless Multi-hop Networking,"IEICE Society Conference, 2003.

PURDUE UNIVERSITY

Thank you

PURDUE
UNIVERSITY