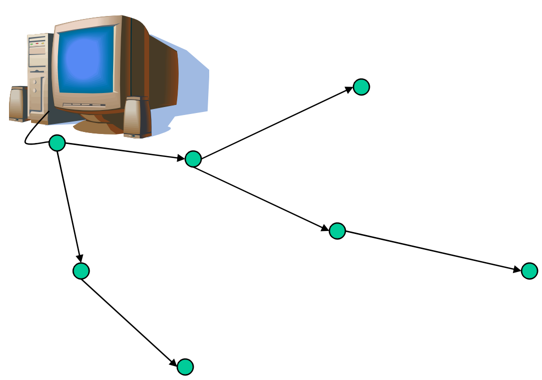
Hermes: Fast and Energy Efficient Incremental Code Updates for Wireless Sensor Networks

Rajesh Krishna Panta
Saurabh Bagchi

Dependable Computing Systems Laboratory (DCSL)
Purdue University

Introduction: What is Wireless Sensor Network Reprogramming?

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



Requirements of Network Reprogramming

- For correctness, all nodes in the network should receive the code completely
- 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

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

[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.

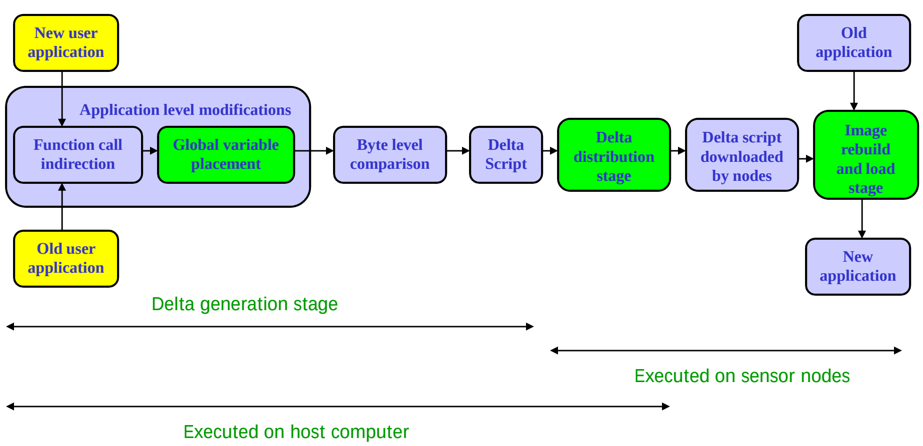
Hermes: Approach

- Instead of transferring the entire image, Hermes transfers the *difference* between the old and new versions of the software
- The size of the difference is reduced by mitigating the effects of the shifts of various software components
- Sensor nodes build the new image from the difference and the old image
- Hermes achieves low communication overhead – reduces reprogramming time and energy

PURDUE UNIVERSITY

5

Overview of Hermes



Delta script: COPY <OldOffset> <NewOffset> <Len>
INSERT <NewOffset> <Len> <NewString>

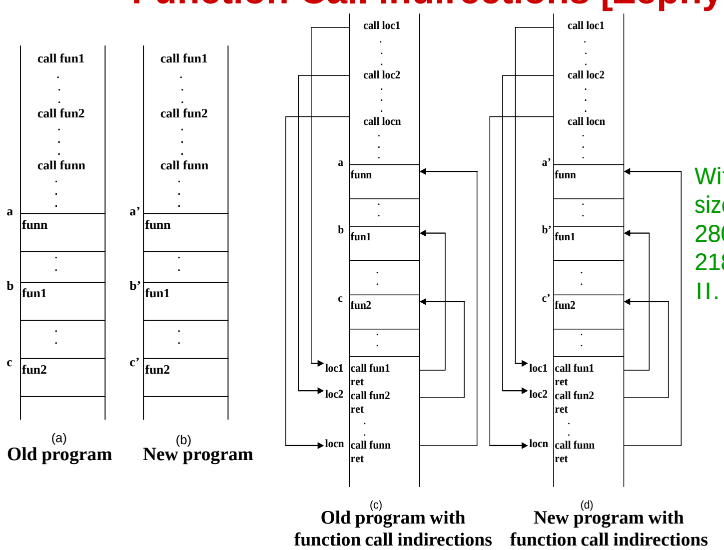
PURDUE UNIVERSITY

6

Byte Level Comparison Alone is not Sufficient

- No matter how good the byte level comparison is, the delta script will be large in many cases if we do not use application level modifications to mitigate the effects of
  - **Function shifts**
    - e.g. 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 (congruent with the amount of the actual change made in the software)
    - e.g. Case II (Adding few lines of code to Blink application)
      - This is also a small change.
      - But delta script is 2183 bytes (very large)
  - **Global variable shifts**
    - e.g. Case III (Adding one global variable to Blink application)
      - Again a small change
      - The delta script is 6090 bytes.

Function Call Indirections [Zephyr09]



With this technique, size of delta script is 280 bytes instead of 2183 bytes for case II.

[Zephyr09] R.K.Panta, S. Bagchi, S. Midkiff , "Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation," USENIX 2009.

Drawback of 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 loop – sample the sensor, process the sensed data, send data to some sink node, and repeat the same process

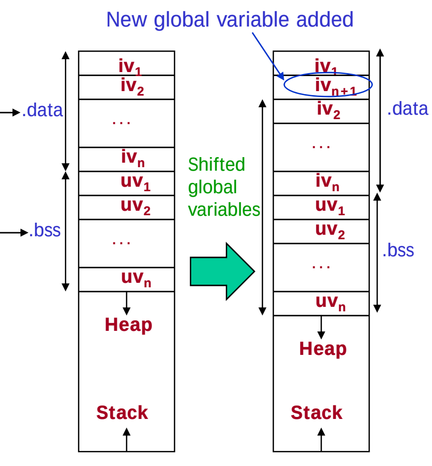
PURDUE UNIVERSITY

Placement of Global Variables in RAM

Initialized global variables

Uninitialized global variables

Order of placement of global variables is dependent on the compiler implementation



(a) Old program (b) New program

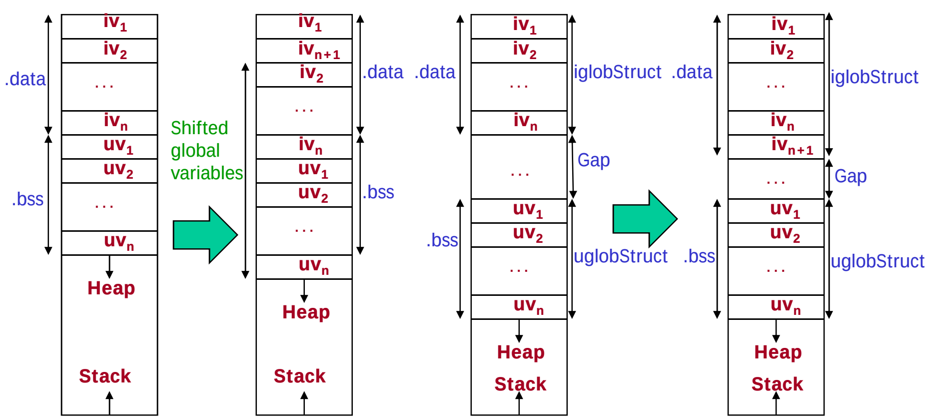
PURDUE UNIVERSITY

Hermes Solution Approach

- Compilers place the members of a structure 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

11

Hermes Solution: Eliminating Effect of Global Variable Shift



(a) Baseline old program (b) Baseline new program (a) Old program with Hermes (a) New program with Hermes

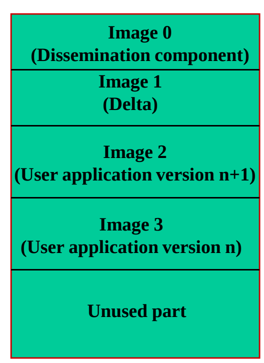
12

Improvement due to Hermes

- With this technique to eliminate the effect of global variable shifts, the size of the delta script for case III is reduced from 6090 bytes to 156 bytes
- Gap between .data and .bss sections to accommodate addition of variables
  - Can be eliminated by moving .bss variables to program memory (Von-Neumann architecture)
  - Can be reduced by moving only those .bss variables straddled by the expansion of .data variables to the end of the .bss section (Harvard architecture)

Delta Distribution

- Let all nodes in the network be running image 2 (old program). At the host computer, delta script is generated between the old image and the new image
- The user gives the command to the base node to reboot all nodes in the network from image 0
- The base node broadcasts the reboot command and itself reboots from the dissemination component. The nodes receiving the reboot command rebroadcast the reboot command and themselves reboot from the dissemination component
- All nodes do the same and finally all nodes in the network run the dissemination component image 0



Structure of external flash

- The user then injects the delta script to the base node. It is wirelessly transmitted to all nodes in the network by the dissemination protocol
- All nodes receive the delta script and store it as image 1

Image Rebuild and Load Stage

- After the nodes download the delta script, they rebuild the new image using the delta (stored as image 1 in the external flash) and the old image (stored as image 2). The new image is stored as image 3
- We modify the image-load stage to avoid latency due to function call indirection
  - When the bootloader loads the new image (image 3) from the external flash to the program memory, it eliminates the indirection by using the exact function address from the indirection table
- In the next round of reprogramming, image 3 becomes the old image and the newly rebuilt image is stored as image 2

Experiments

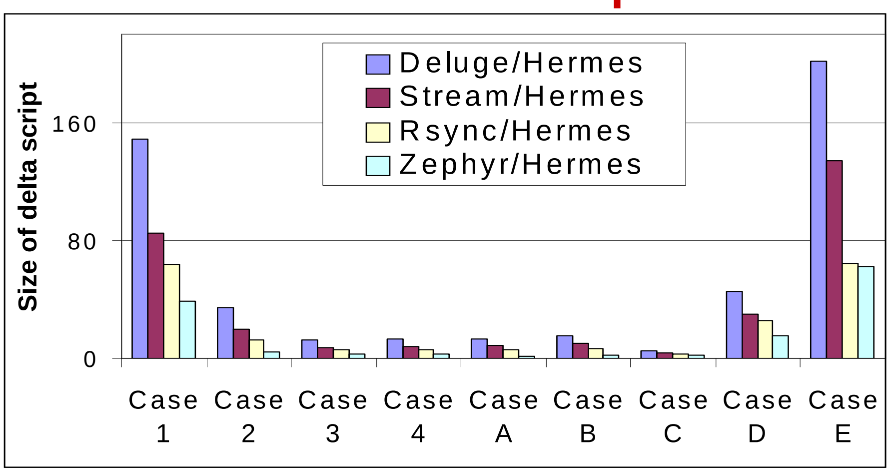
Case 1	Blink to Blink with a global variable added	Standard TinyOS applications
Case 2	Blink to CntToLeds	
Case 3	Blink to CntToLedsAndRfm	
Case 4	CntToLeds to CntToLedsAndRfm	
Case A	An application that samples battery voltage and temperature from MTS310 sensor board to one where few functions are added to sample the photo sensor also.	eStadium applications
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).	
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.	

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 Hermes with Deluge[1], Stream[2], Rsync[3], and Zephyr[4]
  - Use number of packets transmitted in the network as a measure of reprogramming energy

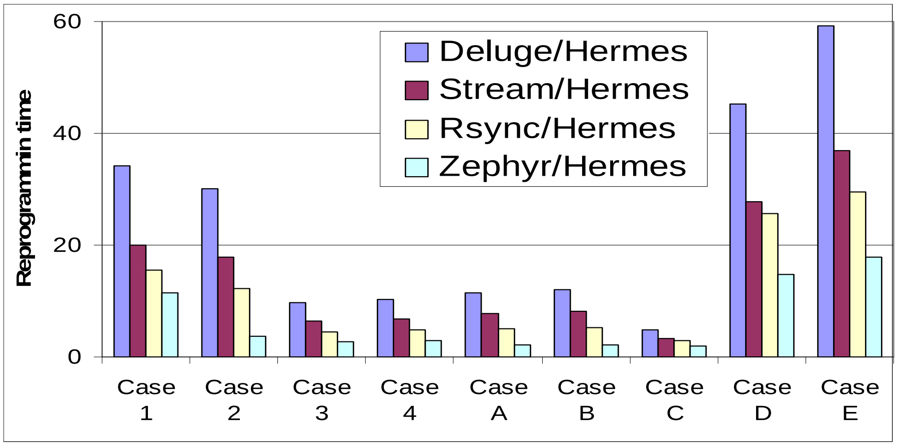
[1] J.W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale." SenSys 2004.
[2] R.K.Panta, I. Khalil, S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," Infocom 2007.
[3]J. Jeong, D. Culler, "Incremental network programming for wireless sensors," SECON 2004.
[4] R.K.Panta, S. Bagchi, S. Midkiff , "Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation," USENIX 2009.

Size of Delta Script



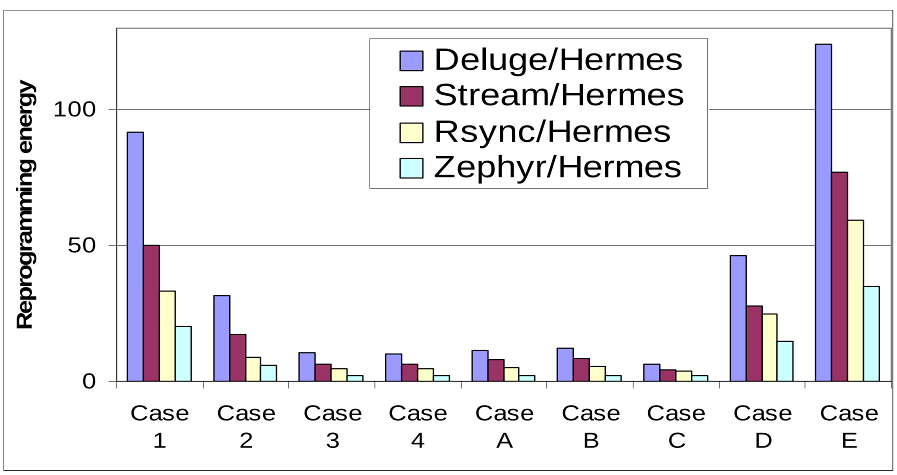
Deluge needs to transfer up to 201 times more bytes than Hermes.
Zephyr generates delta script of size up to 62 times more than Hermes.

Reprogramming Time



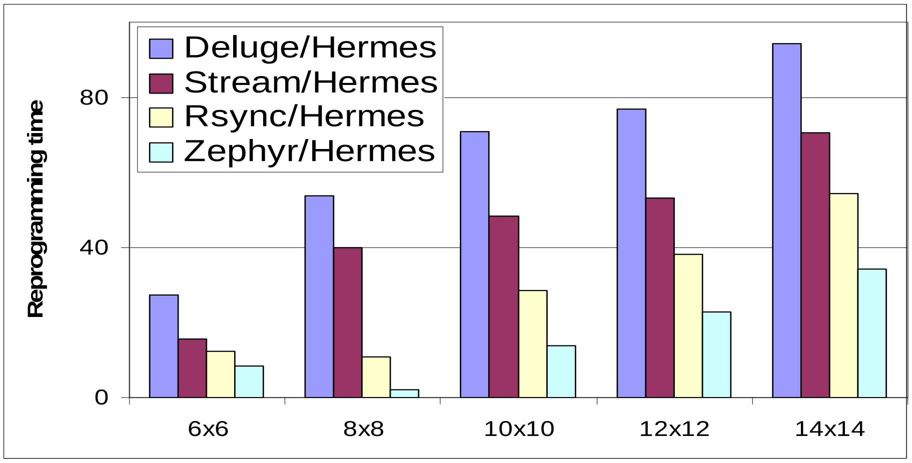
Hermes is up to 59, 36, 29, and 17 times faster than Deluge, Stream, Rsync and Zephyr, respectively.

Reprogramming Energy



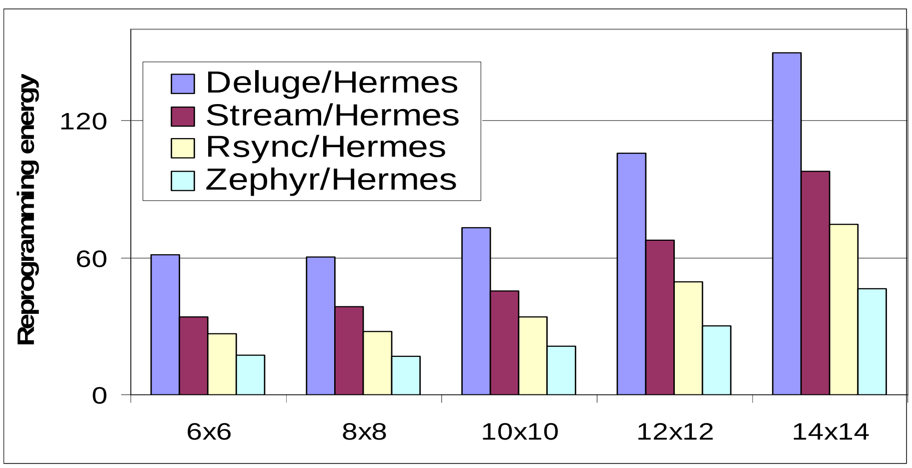
Deluge, Stream, Rsync and Zephyr transfer up to 124, 76, 59, and 35 times more bytes than Hermes, respectively.

TOSSIM Simulation Results (Reprogramming Time)



Hermes is up to 94, 70, 54, and 34 times faster than Deluge, Stream, Rsync, and Zephyr, respectively.

TOSSIM Simulation Results (Reprogramming Energy)



Hermes is up to 149, 97, 74, and 46 times more energy efficient than Deluge, Stream, Rsync, and Zephyr, respectively.

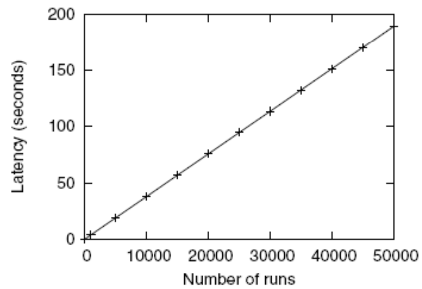
Conclusion

- By eliminating the effect of global variable shifts, Hermes significantly reduces the number of bytes transmitted over the wireless medium for reprogramming
  - For a wide range of software change cases that we experimented with, we found that Hermes takes up to 201, 134, 64 and 62 times less number of bytes than Deluge, Stream, Rsync, and Zephyr, respectively
- Hermes also avoids the latency due to function call indirections
- Future Work:
  - Efficient reprogramming of heterogeneous sensor networks

The End

Credits:
Dimitrios Koutsonikolas as Presenter
Rajesh K. Panta and Saurabh Bagchi, as Authors

Execution Latency Avoided by Hermes



- A sensor network application that operates in a loop with each run of the loop consisting of *work* and *sleep* periods.
- In the work period, the application samples all the sensors on MTS310 sensor board.