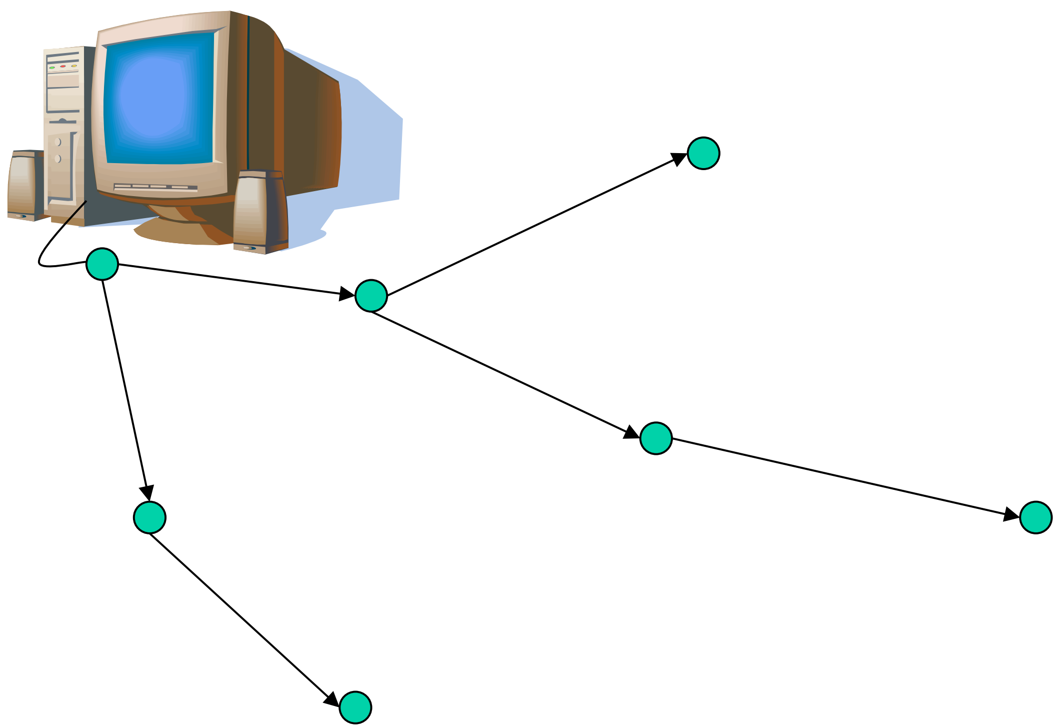
Zephyr: Efficient Incremental Reprogramming of Sensor Nodes using Function Call Indirections and Difference Computation

Rajesh Krishna Panta
Saurabh Bagchi
Samuel P. Midkiff

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

Zephyr: Motivation

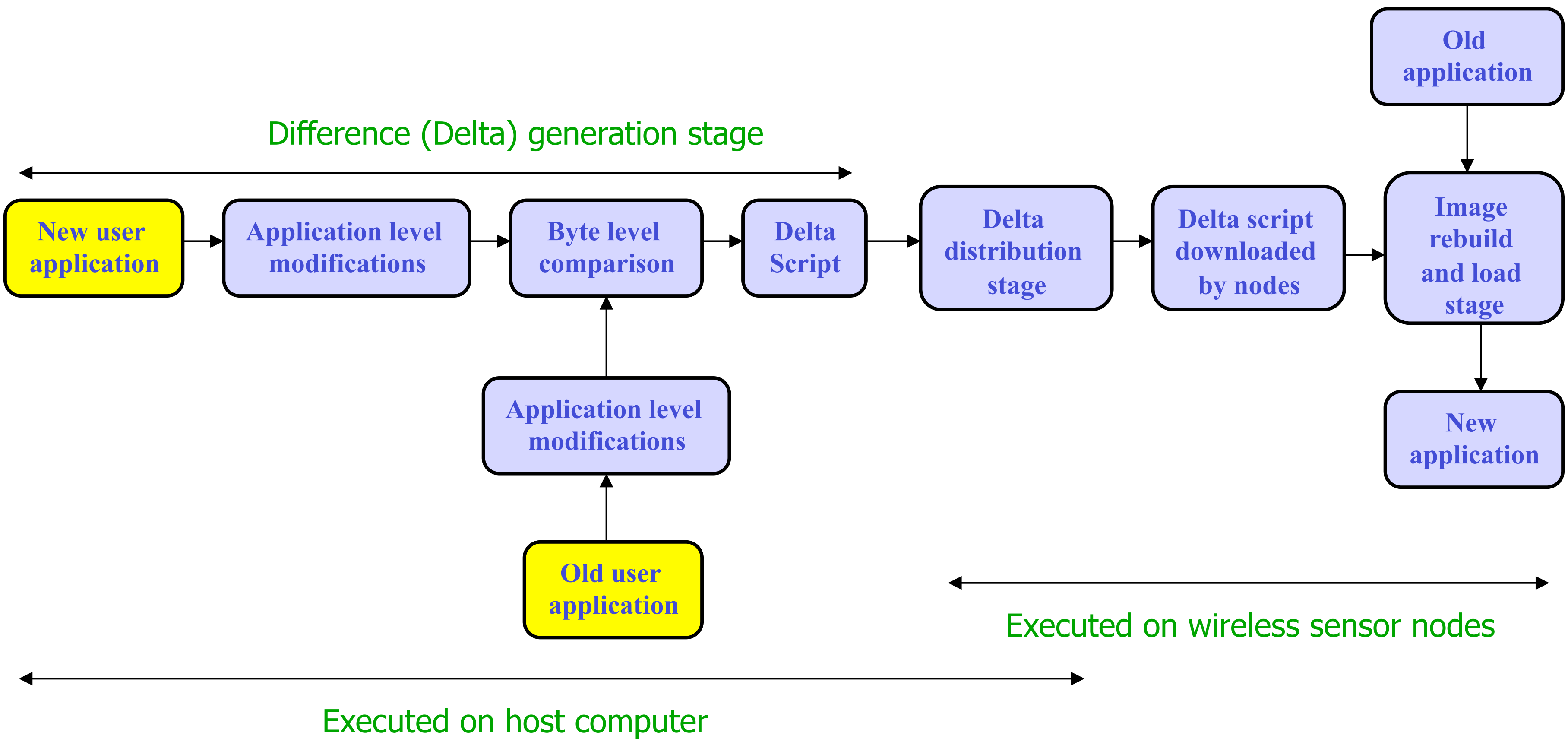
- In practice, software running on the sensor nodes evolves with incremental changes to its functionality
- TinyOS [Berkeley] does not support dynamic linking on the sensor nodes
  - Cannot transfer just the components that have changed and link them in at the node
- SOS [Han05] and Contiki [Dunkels04] 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.

Zephyr: Approach

- Instead of transferring the entire image, Zephyr transfers the *difference* between the old and new versions of the software
- The size of the difference is reduced by using
  - *application level modifications* to mitigate the effect of software component shifts
  - efficient *byte level comparison* that compares the binary images to produce a small difference
- Sensor nodes build the new image from the difference and the old image
- Zephyr transfers relatively small amount of data – reduces reprogramming time and energy

Overview of Zephyr



PURDUE
UNIVERSITY

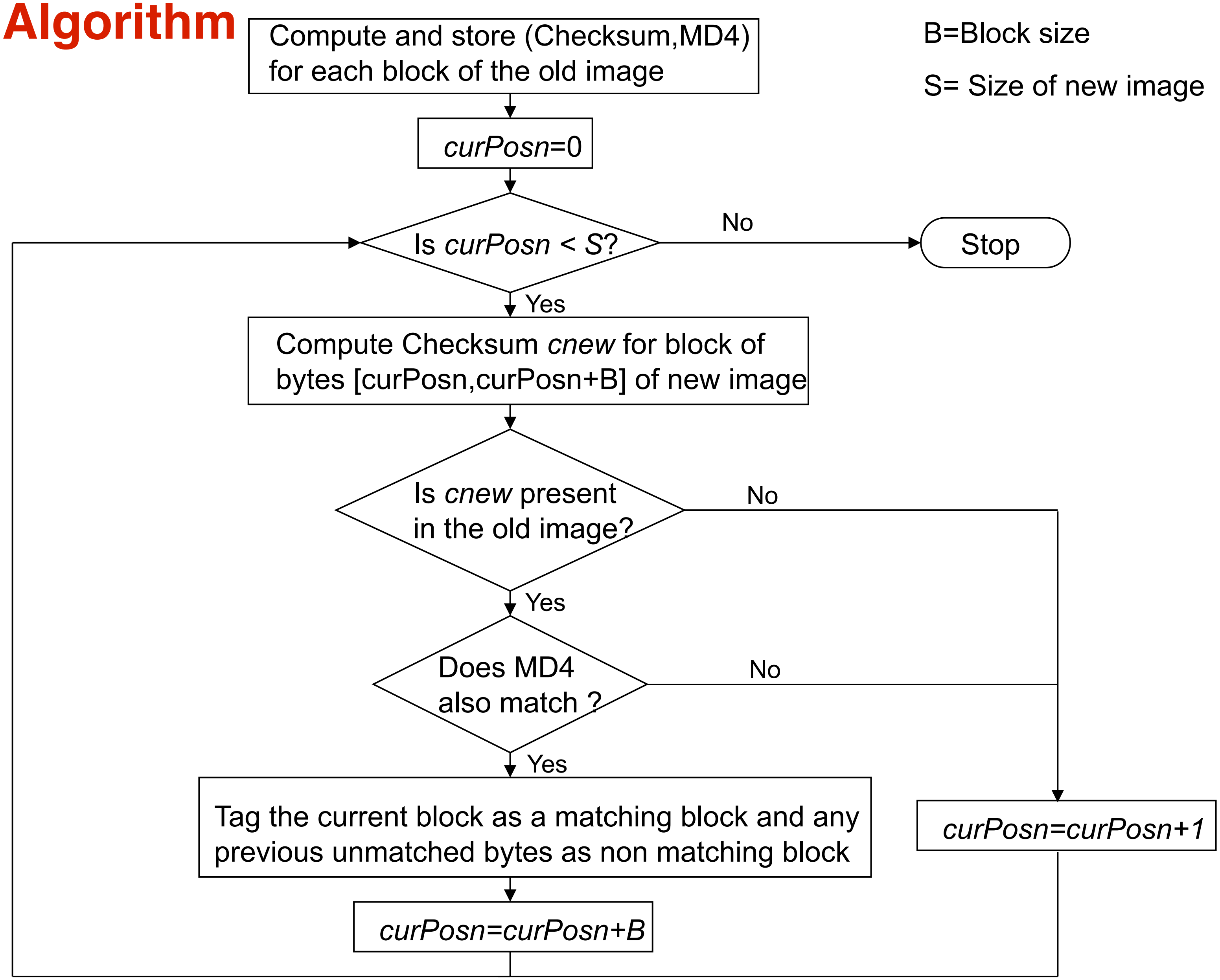
Rsync

- Rsync[Tridgell99] algorithm was originally developed to update binary data between computers over a low bandwidth network
- It divides the binary data into fixed size blocks
- Both sender and receiver compute the pair (Checksum, MD4) over each block
  - Sensor nodes cannot afford to perform expensive MD4 computation
  - We modify Rsync so that all the expensive operations for delta computations are performed on the host computer

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

PURDUE UNIVERSITY

Rsync Algorithm



Delta Script

- After running Rsync algorithm, Zephyr generates a list of COPY and INSERT commands for matching and non matching blocks respectively

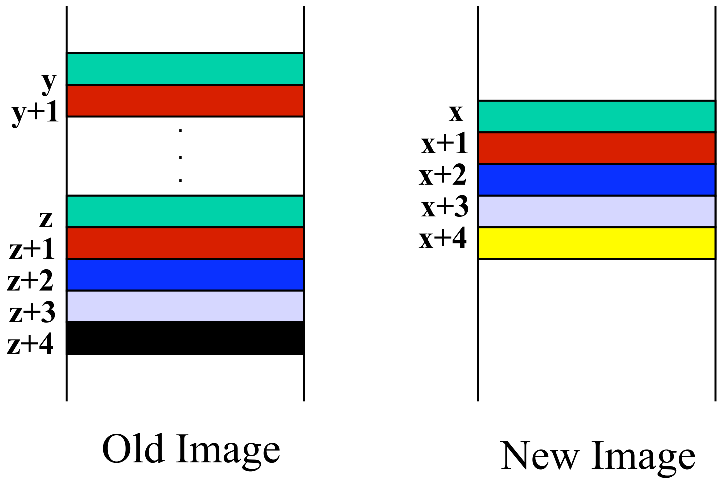
  COPY <oldOffset> <newOffset> <len>

  INSERT <newOffset> <len> <data>

- Goal : Minimize the size of the delta script that has to be wirelessly transmitted to all the sensor nodes in the network

Rsync Optimization

- If there are *n* contiguous blocks in the new image that match *n* contiguous blocks in the old image, Rsync generates *n* number of COPY commands
- Zephyr optimizes Rsync to find the *maximal super block* (i.e. largest contiguous matching block)



Rsync:

COPY y x B

COPY y+1 x+1 B

Semi optimized Rsync:

COPY y x 2*B

(Super block)

Optimized Rsync:

COPY z x 4*B

(Maximal super block)

Byte Level Comparison Alone is Not Sufficient

- To see the drawback of using optimized Rsync alone, consider the following two cases of software changes:
  - Case 1 (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 with optimized Rsync is 23 bytes - proportional to the amount of the actual change made in the software
  - Case 2 (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
- None of the functions shift in Case 1. Functions following the added lines get shifted in Case 2 causing all the call statements referring to the shifted functions 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

Possible Solutions

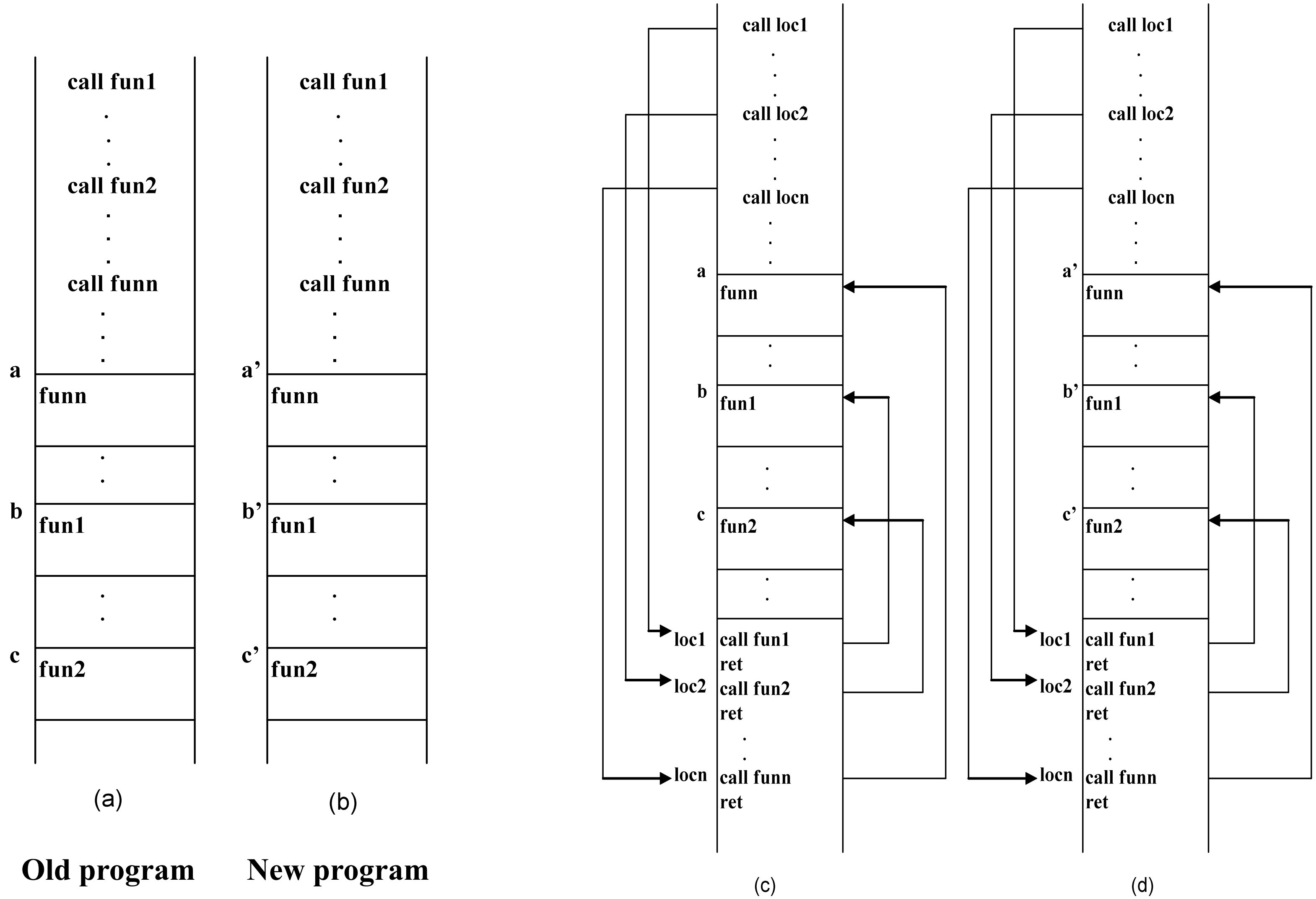
- [Koshy05] leaves empty space (slop region) after each function
  - Waste of program memory
  - How to decide the size of the slop region?

- Use position independent code (PIC) [Han05]
  - Not all architectures and compilers support this. For example, AVR platforms allow relative jumps within 4KB only and for MSP430, no compiler is known to fully support PIC

[Koshy05] Koshy, J. and Pandey,R., "Remote incremental linking for energy-efficient reprogramming of sensor networks" (EWSN 2005).

[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.

Function Call Indirections



Old program **New program**

Old program with Zephyr function call indirections

New program with Zephyr function call indirections

Other Optimizations

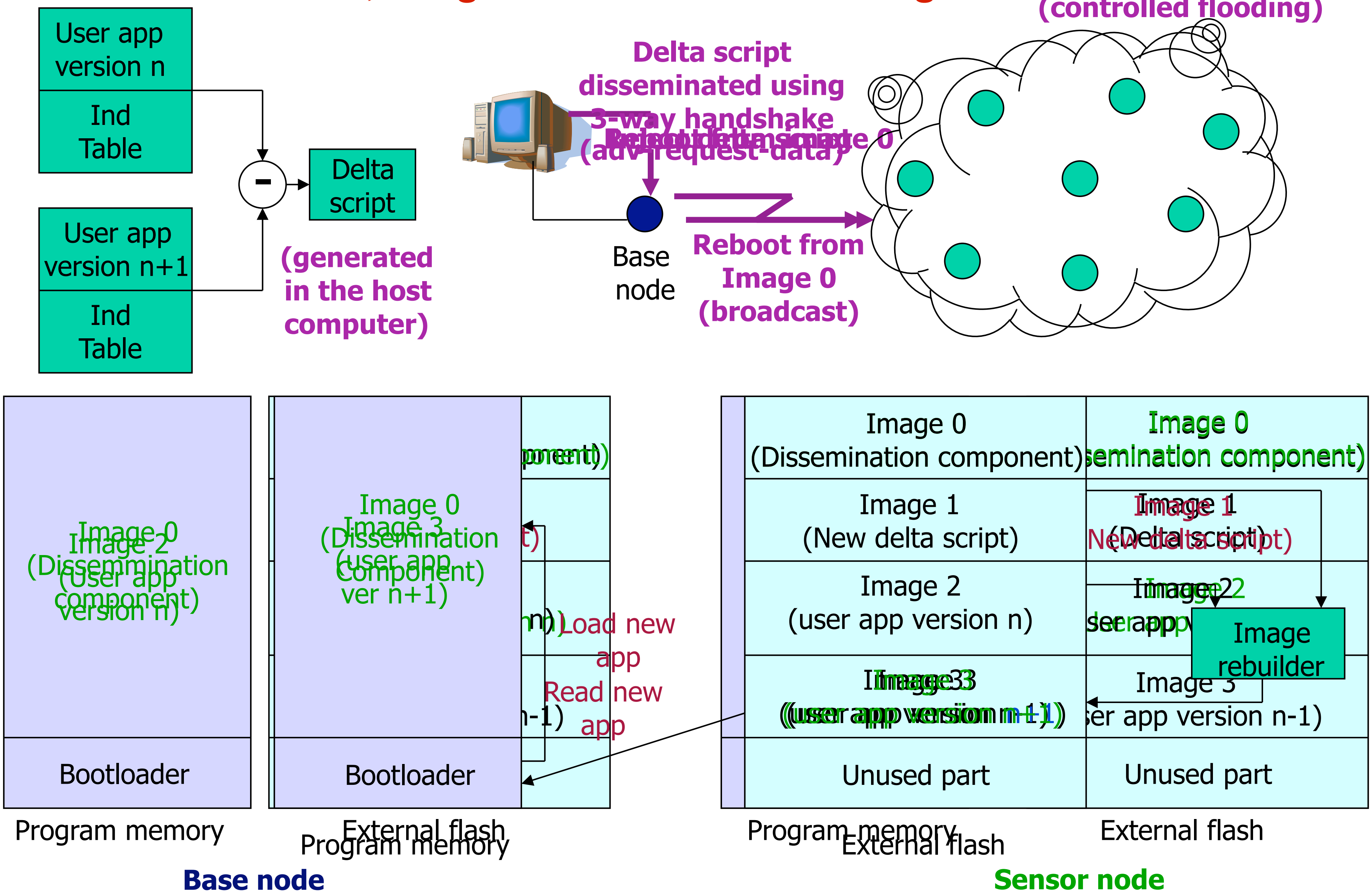
- Zephyr uses *meta commands* – higher level commands that summarize the commonly occurring binary patterns
- Zephyr modifies the linking stage to always put the interrupt service routines at fixed locations in the program memory so that the targets of the calls in the interrupt vector table do not change

With application level modifications, size of the delta script is 280 bytes instead of 2183 bytes for case 2

PURDUE UNIVERSITY

Delta Distribution, Image Rebuild and Load Stages

Broadcast reboot command (controlled flooding)



PURDUE UNIVERSITY

Experiments

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 (Cases 1–7)

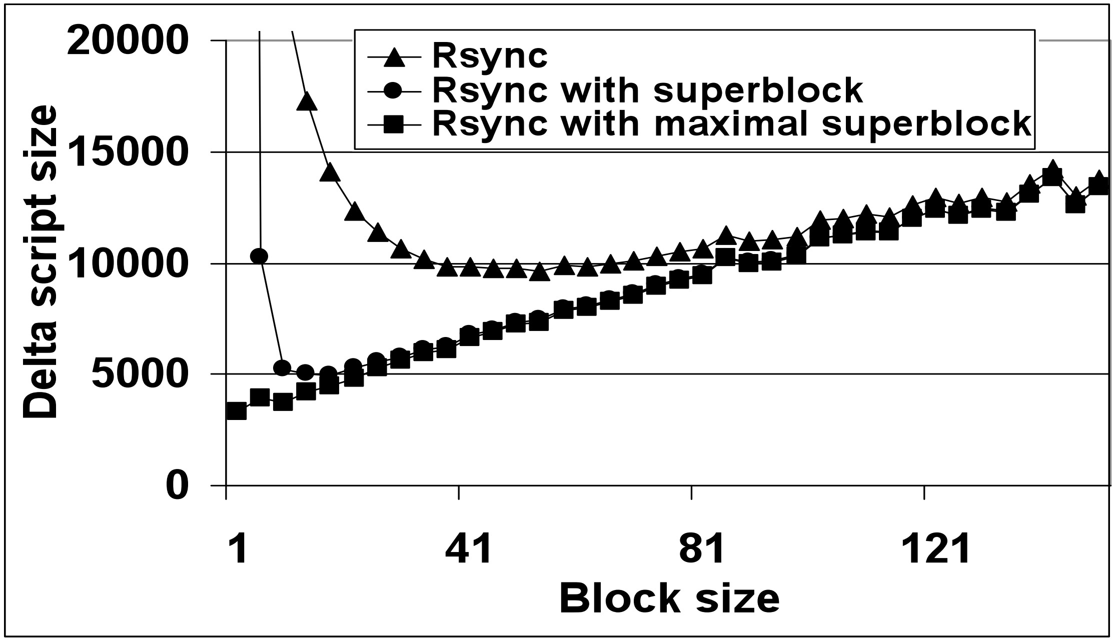
eStadium applications (Cases A–G)

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 with Deluge[1], Stream[2], Rsync[3], and Optimized Rsync
  - 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.

Block Size for Byte Level Comparison



- In all experiments, we use the block size that gives the smallest delta script for corresponding protocol

Size of Delta Script

				Very large change			Large change					Moderate change		Small change
	Case 1	Case 2	Case 3	**Case 4**	Case 5	Case 6	**Case 7**	Case A	Case B	Case C	Case D	**Case E**	Case F	**Case G**
Deluge : Zephyr	1400.82	85.05	4.52	**4.29**	8.47	1.83	**29.76**	7.60	7.76	2.63	203.57	**243.25**	2.75	**1987.2**
Stream : Zephyr	779.29	47.31	2.80	**2.65**	4.84	1.28	**18.42**	5.06	5.17	1.82	140.93	**168.40**	1.83	**1324.8**
Rsync : Zephyr	35.88	20.81	2.06	**1.96**	3.03	1.14	**8.34**	3.35	3.38	1.50	36.03	**42.03**	1.50	**49.6**
SemiOptRsync : Zephyr	6.47	11.75	1.80	**1.72**	2.22	1.11	**5.61**	2.66	2.71	1.39	14.368	**17.66**	1.36	**6.06**
OptRsync : Zephyr	1.35	7.79	1.64	**1.57**	2.08	1.07	**3.87**	2.37	2.37	1.35	7.84	**9.016**	1.33	**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.

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	1.54	2.23	1.46
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

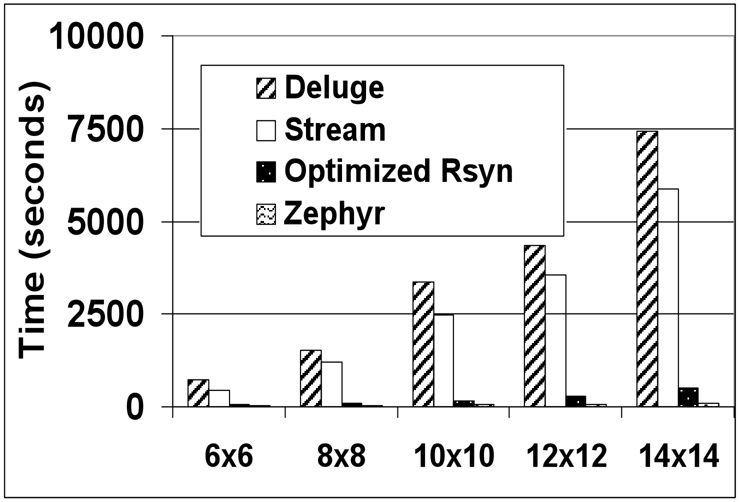
Zephyr is up to 48.9, 40.1 and 4.09 times faster than Deluge, Stream, and optimized Rsync without application level modifications, respectively.

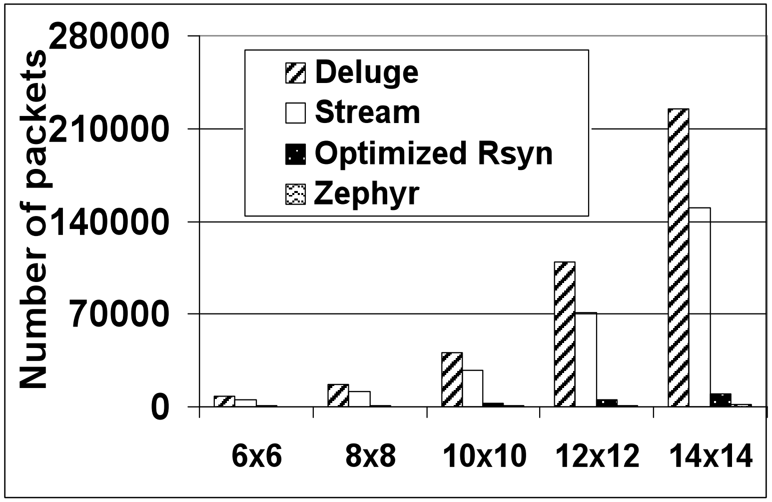
Reprogramming Energy

	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.

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

Conclusion

- Contributions:
  - Application level modifications
  - Efficient byte level comparison
- Achievement : Significant reduction in reprogramming time and energy
- Future work
  - To remove latency due to function call indirection. When the bootloader loads the new image from the external flash to the program memory, it can eliminate the indirection by using the exact function address from the indirection table
  - Efficient reprogramming of heterogeneous sensor networks

Thank you !!!