Poster Abstract: Application-Specific Trace Compression for Low Bandwidth Trace Logging [*]

Roy S. Shea
Dept. of Computer Science
Univ. of Los Angeles, California
roy@cs.ucla.edu

Young H. Cho
Dept. of Electrical Engineering
Univ. of Los Angeles, California
young@ee.ucla.edu

Mani B. Srivastava
Dept. of Computer Science
Univ. of Los Angeles, California
mani@cs.ucla.edu

ABSTRACT

This poster introduces an application-specific trace log compression mechanism targeted for execution on wireless sensor network nodes. Trace logs capture sequences of significant events executed on a node to provide visibility into the system. The application-specific compression mechanism exploits static program control flow knowledge to automate insertion of trace statements that capture trace data in a concise form. Initial evaluation reveals that these compressed trace logs, when generated, consume just over a fifth of the space required by standard trace logging techniques.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*debugging aids, diagnostics, distributed debugging*

General Terms

Reliability, Performance, Design

Keywords

logging, debugging, wireless sensor networks

1. INTRODUCTION

Understanding behavior in a wireless sensor network remains extremely challenging. Sensor network application designers often observe misbehavior in a deployed network, but lack the means to systematically reconstruct and understand the underlying problem.

One tool currently available to system designers are runtime logs. Logs are currently used within the sensor network community in projects including SNMS [4], NodeMD [1], and EnviroLog [2]. Although the idea of logging continues to garner interest from the sensor network community, scarce device storage and limited network bandwidth continue to hinder its use.

[*]This material is supported in part by the NSF under award CNS-0520006, and by the Center for Embedded Networked Sensing at UCLA. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the listed funding agencies.

2. COMPRESSING TRACE LOGS

Our goal is to produce detailed traces that give developers and users insight into the execution of their wireless and embedded sensing systems, while minimizing log sizes and without requiring developers to make manual program modifications. We compare our optimized technique to similar logging techniques in current use.

2.1 Trace Logging Through Global Event IDs

Call traces can be gathered by adding to each function or event handler a brief preamble that records a unique ID. Such preambles may be added either manually, via a custom tool, or through generic call interception techniques.

2.2 Compressing Traces Using Control Flow Graph Knowledge

The purpose of call trace logging is to provide a detailed view of runtime program flow. It is trivial to reconstruct a call trace using a log of global IDs. However, much of the information provided within such a log is redundant.

Our contribution in designing an efficient logging infrastructure is reducing this redundancy. Figure 1 illustrates the control flow graph of a block allocation routine. A sample execution through this function may enter the `for` loop at line 16, call `MergeBlocksQuick` once before exiting the loop, pass over the `if` statements at lines 25 and 30, and call `Unlink` before returning from the function. This would result in a call trace recording: the call to and ID for `sos_blk_mem_alloc`, the call to and ID for `MergeBlocksQuick`, the trace resulting from the `MergeBlocksQuick` call, the call to and ID for `Unlink`, and the trace resulting from the `Unlink` call. Each ID must be unique across the system.

Note that from the node containing line 30 in the sample control flow graph (CFG), one of two paths must be taken: the path leading to only `Unlink`, or the path leading through `SplitBlock`. Knowledge of path taken is sufficient for determining the set and sequence of executed functions. Rather than recording full IDs for each function executed, our logging mechanism records the path taken through functions. Further compression is accomplished by pruning nodes from the CFG that do not effect the execution of functions. Only two types of control flow nodes affect what functions will be called: instruction nodes with calls to functions and branching nodes that lead to more than one of the prior. Our compression algorithm collapses other nodes within the CFG while maintaining the relative position of nodes of interest.

Figure 2 shows the reduced CFG for the sample program. This results in a final trace of: branch from line 16 to line

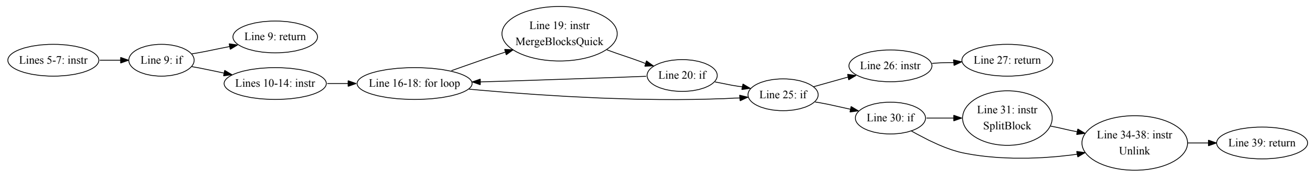

Figure 1: Call graph of the block allocation routine.

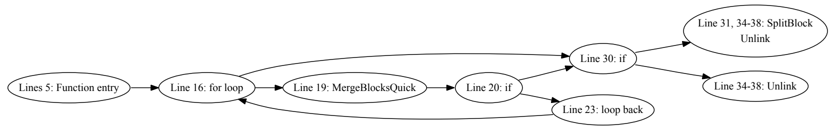

Figure 2: Reduced control flow graph providing a minimal encoding needed to identify called functions.

Kernel Image	Program Text Size (B)
Unmodified OS	52552
Globally Unique IDs	68824 (+31%)
CFG Based Compression	87406 (+66%)

Table 1: Program text expansion.

19, the trace resulting from the `MergeBlocksQuick` call, branch from line 20 to line 30, branch from line 30 to line 34, and the trace resulting from the `Unlink` call. Note that, rather than recording unique IDs, the trace only records the branch taken from each conditional. Whereas unique IDs for programs on small embedded systems often require eight or more bits, most branches require only a single bit.

Complications arise from function pointers and hardware interrupts that result in unpredictable changes in program locality. Explicit use of a jump symbol combined with the unique ID of the interrupt handler can be used to unambiguously set program locality into and after an interrupt. A similar approach is applied to function pointers.

3. EVALUATION

We evaluated the tracing mechanism proposed above by measuring the size of logs generated within a simple sensor network using tree routing to transfer sensor data out of the network. The presented preliminary test runs were simulated using Avrora [3]. Bandwidth consumption within this evaluation was measured by recording the number of times program points of interest were passed within a simulated sensor network application. Points of interest for call tracing using unique IDs are function or interrupt entries only. Points of interest for traces using the CFG based compression method as presented on this poster are: entry to calls executed via function pointers, entry to interrupt handlers, edge trace statements, and return statements.

3.1 Effects on Program Text Size

Both inserting statements to record the unique ID of each executed function and instrumenting code to record the path traversed over the CFG have nontrivial impact on the underlying code base. Compressed logs created using CFG information have greater code size impact since the technique embeds traces throughout the code base, rather than a single logging statement at the head of each function call. Table 1 shows the respective resulting program text size increases.

3.2 Runtime Bandwidth Consumption

The two logging approaches were evaluated using runtime counters. The log size resulting from recording unique IDs is the aggregate number of functions called (including interrupts) multiplied by the bit width of the unique identifiers. The evaluated system required 9 bits to provide a unique identifier to each of the 362 functions. The log size required for traces using our CFG compression technique depends on the token size to record and number of: branches taken (average of 2.5 bits per token), interrupts encountered (8 bits), function pointer dispatches (8 bits), and return statements encountered (3 bits). Counts are provided for all user code and nearly all kernel code.

Each second, the evaluated system experienced an average of: 1.34 function pointer dispatches, 10.2 interrupts, the traversal of 113 significant control flow branches, and 322 function calls. CFG driven trace collection generates 685 bits of log data each second, compared to 2902 bits generated through the logging of unique IDs.

4. DISCUSSION

This poster introduces a technique to compress trace logs using static CFG information. A prototype implementing these ideas and automating the insertion of traces into programs is being used within our lab. We look forward to carrying these ideas further in our continued efforts to expose state currently hidden in deployed sensor networks.

5. REFERENCES

[1] V. Krunic, E. Trumpler, and R. Han. Nodemd: diagnosing node-level faults in remote wireless sensor systems. In *MobiSys*. ACM Press, 2007.

[2] L. Luo, T. He, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic. Achieving repeatability of asynchronous events in wireless sensor networks with envirolog. In *INFOCOM*. IEEE, 2006.

[3] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN*. IEEE Press, 2005.

[4] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *EWSN*. IEEE, 2005.