Log Instrumentation Specifications and Low Overhead Logging

Roy Shea

Department of Computer Science
University of California, Los Angeles

roy@cs.ucla.edu

July 17, 2009

With Mani Srivastava, Young Cho, and endless input from folks in NESL.

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Problems with the Vision

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Problems with the Vision

- Lions live in the savanna

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Problems with the Vision

- Lions live in the savanna
- Still paying $139.00 for a sensor node (TelosB with sensor suite)

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Problems with the Vision

- Lions live in the savanna
- Still paying $139.00 for a sensor node (TelosB with sensor suite)
- “Thousands” is a big number

Observing the Unobservable

The Vision

In-network algorithms directing a sensor network made up of thousands (or more) of sub-dollar wireless sensor nodes to localize a lion roaming through the jungle.

Problems with the Vision

- Lions live in the savanna
- Still paying $139.00 for a sensor node (TelosB with sensor suite)
- "Thousands" is a big number
- Easy to create suboptimal, fragile, and buggy localized algorithms

After the Dream is Gone

Picking Up Pieces

After the Dream is Gone

Picking Up Pieces

- Lions live in the savanna
  - Embrace breadth of expertise while seeking out researchers in other fields

After the Dream is Gone

Picking Up Pieces

- Lions live in the savanna
  - Embrace breadth of expertise while seeking out researchers in other fields
- Still paying $139.00 for a sensor node (TelosB with sensor suite)
  - Count on the power of economy of scale *if* the demand is present

Picking Up Pieces

- Lions live in the savanna
  - Embrace breadth of expertise while seeking out researchers in other fields
- Still paying $139.00 for a sensor node (TelosB with sensor suite)
  - Count on the power of economy of scale *if* the demand is present
- "Thousands" is a big number
  - Ends up that you can learn a lot about the world with one node, or two nodes, or ten nodes...

After the Dream is Gone

Picking Up Pieces

- Lions live in the savanna
  - Embrace breadth of expertise while seeking out researchers in other fields
- Still paying $139.00 for a sensor node (TelosB with sensor suite)
  - Count on the power of economy of scale *if* the demand is present
- “Thousands” is a big number
  - Ends up that you can learn a lot about the world with one node, or two nodes, or ten nodes...
- Easy to create suboptimal, fragile, and buggy localized algorithms
  - Provide tools to help developers write efficient, robust, and functioning code for wireless embedded systems

My Vision

Develop new techniques and tools that provide developers with the insight needed to create applications for and understand behavior within deployed networks of bottom tier sensing devices.

My Vision

Develop new techniques and tools that provide developers with the insight needed to create applications for and understand behavior within deployed networks of bottom tier sensing devices.

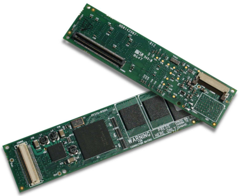
LIS

Framework for describing and implementing logging tasks.

LowLog

Optimizing to token name spaces to reduce the overhead of collecting runtime call traces.

Machine Class

Desktop / Server	Embedded Computer	Microcontroller
		

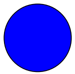
High End

- Native hardware support for debugging
- Ample resources that debugging infrastructure can use
- Rich I/O capabilities

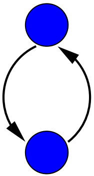
Bottom Tier

- Real time constraints limit interactive or CPU intensive debugging techniques
- Minimal resources require very small footprint debugging utilities

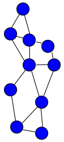
Number of Machines (or Processes)



- Basic debugging setup
- Ability to suspend program without impacting other processes

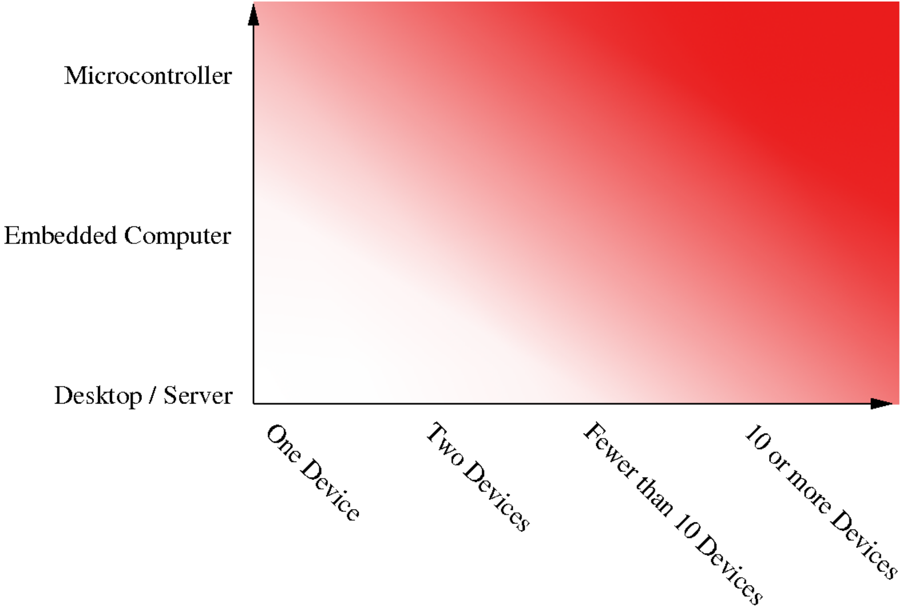


- Suspension of multiple processes tricky
- Interactive debugging tracks execution on multiple processes to understand interactions
- Logs capturing consistent system state are a powerful debugging aid

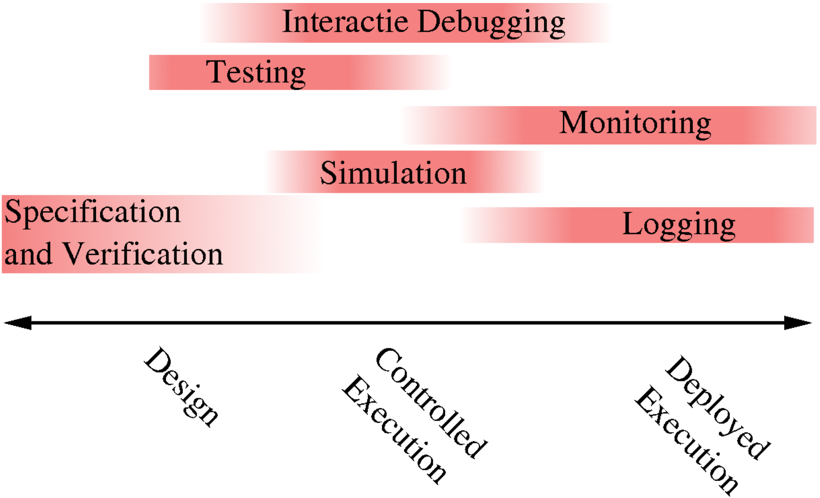


- Suspending execution becomes very difficult
- Challenging to capture consistent system state
- Use detailed logs used to understand aggregate system behavior

Debugging Complexity of Different Systems



Handling Bugs Through the Entire System Design Cycle

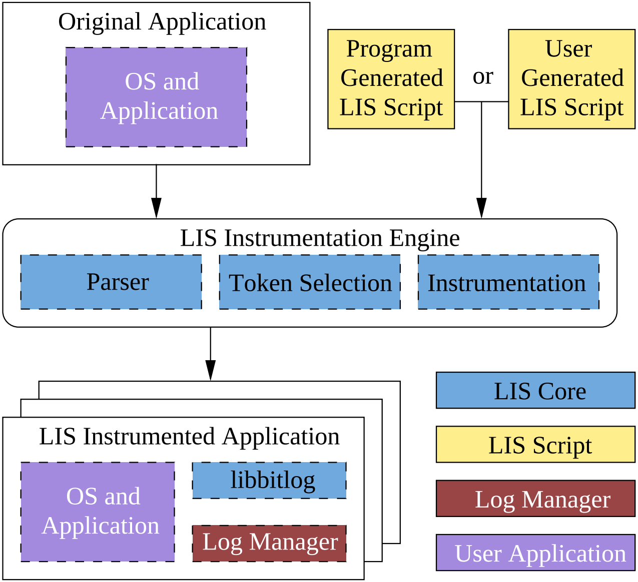


Outline

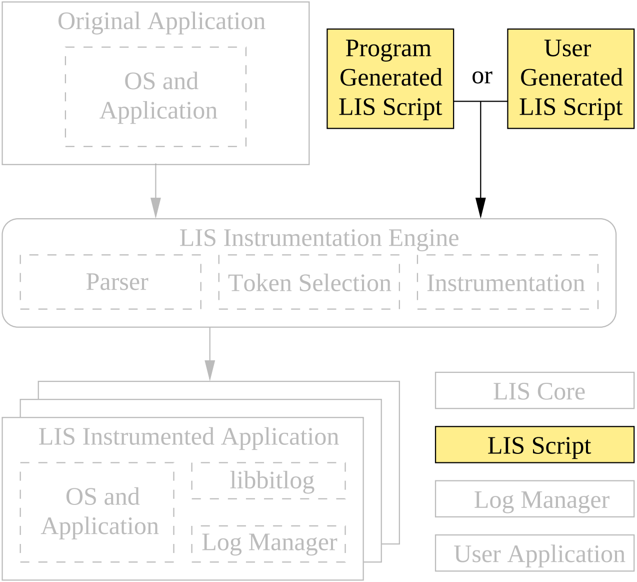
Logging with Log Instrumentation Specifications (LIS)

- Observe demand for `printf` style logging in distributed embedded systems
  - Low learning curve and direct semantics make it easy to use
  - But resulting logs are typically verbose and ad-hoc logging is difficult to maintain
- Want to provide developers an alternate logging solution
  - Separation of logging specification from underlying code base
  - Encourages design of optimized logging tasks
  - Easy to pickup and use
- Logging streams of tokens via LIS is our answer

Architecture



Architecture



Structure of a LIS Statement

Statement Types

- Function header
- Function footer
- Function call
- Control flow
- Variable watchpoint

Statement Location

- Union of log type and a function name
- Type specific directives bind log to specific called function, control flow type, or variable name

Structure of a LIS Statement

Statement Types

- Function header
- Function footer
- Function call
- Control flow
- Variable watchpoint

Statement Location

- Union of log type and a function name
- Type specific directives bind log to specific called function, control flow type, or variable name

Token Scope

Global Token value unique throughout the program
Local Token value is unique throughout the function
Point Token value is a singled fixed value

LIS Specification

$$
\begin{aligned}
\mathit{Start} &\rightarrow \mathit{Statements} \mid \epsilon \\
\mathit{Statements} &\rightarrow \mathit{Stmt}\ \mathit{Statements} \mid \mathit{Stmt} \\
\mathit{Stmt} &\rightarrow \mathit{Header} \mid \mathit{Footer} \mid \mathit{Call} \mid \mathit{ControlFlow} \mid \mathit{Watch} \\
\mathit{Header} &\rightarrow \textbf{header}\ \mathit{Placement}\ \mathit{Scope} \\
\mathit{Footer} &\rightarrow \textbf{footer}\ \mathit{Placement}\ \mathit{Scope} \\
\mathit{Call} &\rightarrow \textbf{call}\ \mathit{Placement}\ \mathit{Scope}\ \mathit{Target} \\
\mathit{ControlFlow} &\rightarrow \textbf{controlflow}\ \mathit{Placement}\ \mathit{Scope}\ \mathit{Flag}\ \mathit{Var} \\
\mathit{Watch} &\rightarrow \textbf{watch}\ \mathit{Placement}\ \mathit{Scope}\ \mathit{Var} \\
\mathit{Placement} &\rightarrow \mathit{F} \\
\mathit{Scope} &\rightarrow \textbf{global} \mid \textbf{local} \mid \textbf{point} \\
\mathit{Target} &\rightarrow \mathit{F} \mid \textbf{\_\_PTR\_\_} \\
\mathit{Flag} &\rightarrow \textbf{if} \mid \textbf{switch} \mid \textbf{loop} \mid \textbf{if} - \textbf{switch} \mid \textbf{if} - \textbf{loop} \\
&\quad \mid \textbf{switch} - \textbf{loop} \mid \textbf{if} - \textbf{switch} - \textbf{loop} \\
\mathit{Var} &\rightarrow \langle \text{Varible name from program} \rangle \mid \textbf{\_\_ANY\_\_} \\
\mathit{F} &\rightarrow \langle \text{Function name from program} \rangle
\end{aligned}
$$

LIS Script

Script and Target Program

```
header read_done global
controlflow read_done local if send_busy
footer read_done point
```

```
/* Pre-LIS */
void read_done(error_t result, uint16_t data) {



    if (send_busy == TRUE) {


        return;
    }


    /* Rest of function body elided... */


    return;
}
```

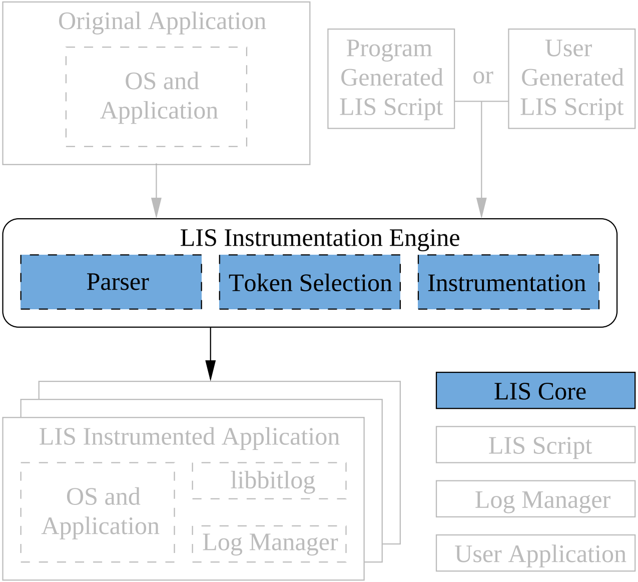
LIS Script

Instrumented Program

```
header read_done global
controlflow read_done local if send_busy
footer read_done point
```

```
/* Post-LIS */
void read_done(error_t result, uint16_t data) {

    bitlog_write(4, 3); /* Header LIS statement */

    if (send_busy == TRUE) {
        bitlog_write(5, 3); /* Control flow LIS statement */
        bitlog_write(0, 1); /* Footer LIS statement */
        return;
    }
    bitlog_write(6, 3); /* Control flow LIS statement */

    /* Rest of function body elided... */

    bitlog_write(0, 1); /* Footer LIS statement */
    return;
}
```

Architecture



Instantiation of LIS

Instrumentation Engine

- Built using using C Intermediary Language (CIL) framework
  - Input a LIS script and source program
  - Outputs instrumented program
- Functional ports for x86, ATMega128, and MSP430 chip sets
- Patch available to integrate LIS into the TinyOS build system

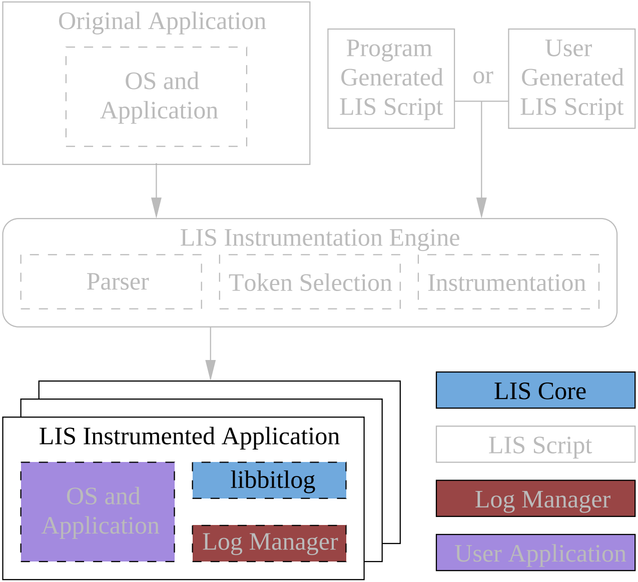
Instantiation of LIS

Instrumentation Engine

- Built using using C Intermediary Language (CIL) framework
  - Input a LIS script and source program
  - Outputs instrumented program
- Functional ports for x86, ATMega128, and MSP430 chip sets
- Patch available to integrate LIS into the TinyOS build system

	Mica2		MicaZ		TelosB	
	Program Memory	RAM	Program Memory	RAM	Program Memory	RAM
TinyOS Radio Stack	7178	137	9264	210	8456	229
TinyOS CTP	9692	1230	10284	1360	11126	1295
LogTap (CTP)	1384	303	1412	351	2228	353
LogTap (broadcast)	108	113	74	128	388	131
Bitlog Library	386	43	386	43	362	43
Call to `bitlog_write`	14	0	14	0	12	0

Architecture



Instantiation of LIS

Runtime Support

Bitlog Library

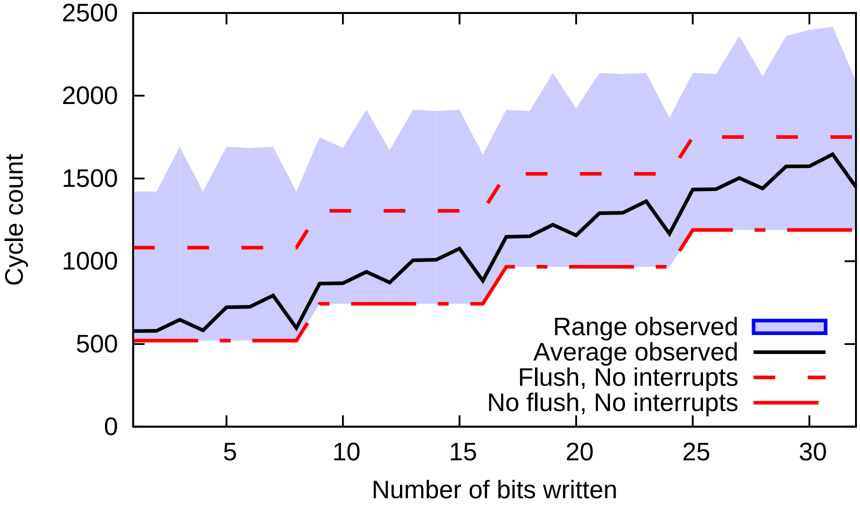
- Writing of tokens into a buffer managed by a logging library
- Includes the Bitlog library that provides low overhead bit aligned logging

LogTap

- Storag / transfer of logs managed by log management library
- Writing log buffers out to stderr for use on desktop machines
- Broadcasting logs using the TinyOS `AMSend` interface
- Routing logs using the TinyOS collection tree protocol (CTP)

Instantiation of LIS

Runtime Support

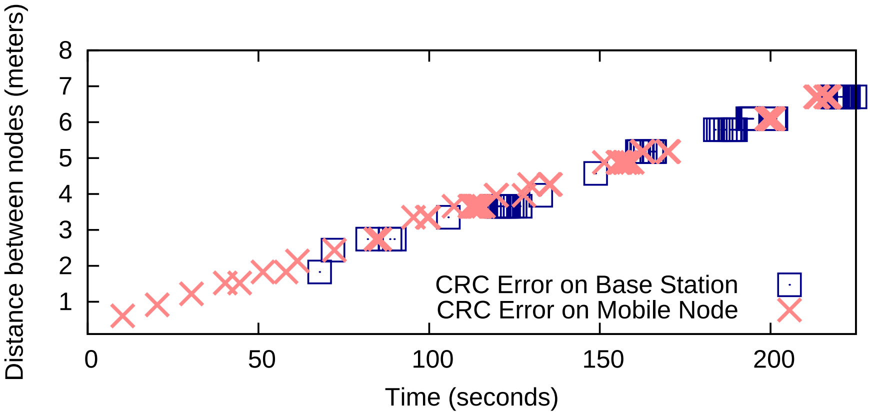


Using LIS: Observing CRC Errors

```
controlflow CC2420ReceiveP.RXFIFO.readDone global if buf
```

Using LIS: Observing CRC Errors

```
controlflow CC2420ReceiveP.RXFIFO.readDone global if buf
```



Using LIS: Learning about CTP

```
header CtpForwardingEngineP.0.sendTask.runTask point
controlflow CtpForwardingEngineP.0.sendTask.runTask \
        local if-switch-loop __ANY__
```

Using LIS: Learning about CTP

```
header CtpForwardingEngineP.0.sendTask.runTask point
controlflow CtpForwardingEngineP.0.sendTask.runTask \
        local if-switch-loop __ANY__
```

LIS Log Token	Comments
`<--`	Enter `runTask`
`BID: 1`	CTP is not busy
`BID: 3`	Send queue is nonempty
`BID: 32`	Node is not root so must route message
`BID: 33`	Found a route to the root
`BID: 4`	Prepare to send data
`BID: 9`	Neighbor is not congested
`BID: 16`	Message has not already been sent
`BID: 19`	Node is not the root so must route message
`BID: 21`	Found current path quality metric
`BID: 23`	Not congested
`BID: 24`	Succeeded in sending message
`BID: 25`	Note that the client sent a packet

Outline

Tracing Calls Within a Region of Interest with LowLog

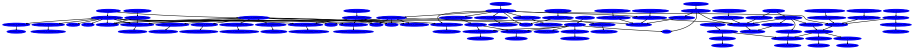
- Function call traces
  - Describe the sequence of function calls made at runtime
  - Already used by developers to understand system behavior
  - Often provides enough information in itself to diagnose bugs or jump start more aggressive debugging efforts
- Region of interest (ROI)
  - Subsystem of interest to a developer
  - For example there is (usually) no reason to trace execution through the kernel when debugging a user space application
  - Provides a focused view of a system
- LowLog provides optimized region of interest call tracing
- LowLog sits as a higher level analysis that outputs efficient LIS scripts

Partitioning a Program into an ROI

- *Entry function* is reachable from a function not within the ROI
- *Body function* only reachable from functions within the ROI
- *Return of interest* when any function from the ROI returns

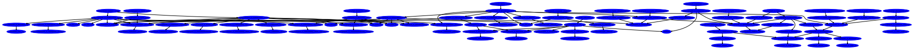
Partitioning a Program into an ROI

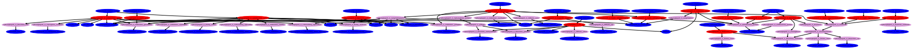
- *Entry function* is reachable from a function not within the ROI
- *Body function* only reachable from functions within the ROI
- *Return of interest* when any function from the ROI returns



Partitioning a Program into an ROI

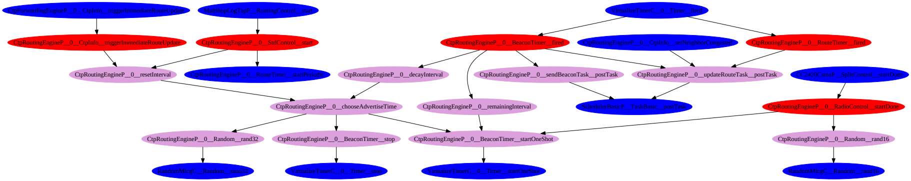
- *Entry function* is reachable from a function not within the ROI
- *Body function* only reachable from functions within the ROI
- *Return of interest* when any function from the ROI returns





Region of Interest

Detail



Create Log of Entry, Body, and Return Tokens

Optimizing Call Trace Bandwidth

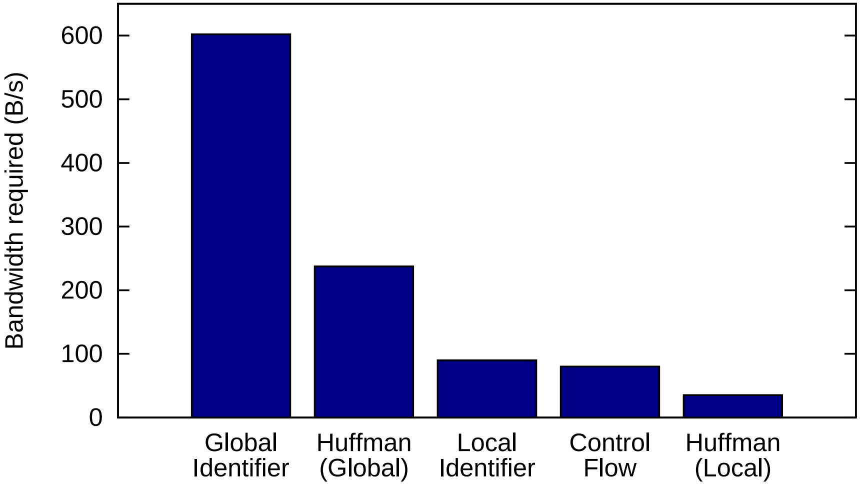
- Standard call tracing uses *global identifier logging* that lumps entry and body tokens into a single name space and use simple fixed bit width token encoding
- LowLog proposes to alternate schemes
  - *Local identifier logging* creates multiple caller specific token name spaces
  - *Control flow logging* tracks runtime control flow decisions rather than body calls
- All three logging techniques can apply more powerful encoding techniques if more information is available

Optimizing Control Flow Logging

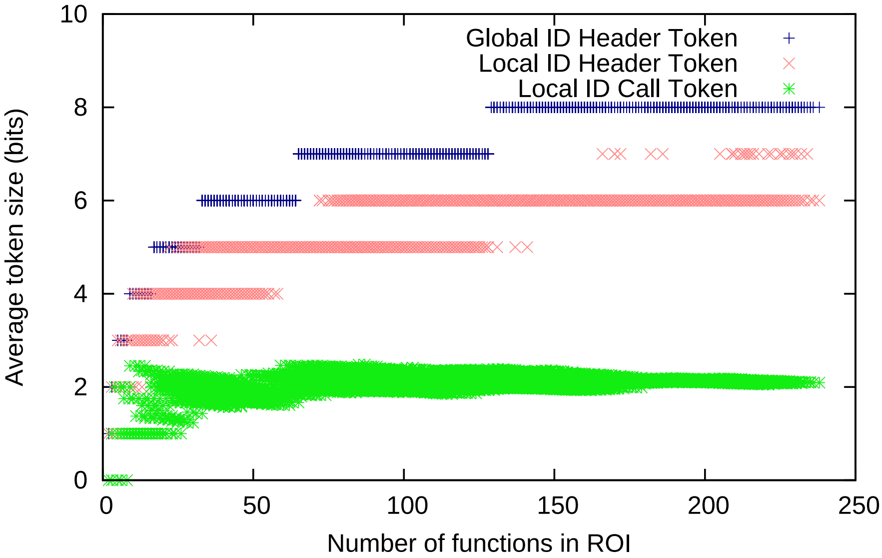
Only Log Control Flow Decisions Effecting Called Functions

- Reverse dataflow analysis to track sets of functions that must be called after reaching a statement
- Join function examines sets of called functions
  - If the sets are different then logging of the control flow taken form the current node is tracked and the empty set is passed into the transfer function
  - If the sets are identical then the set (either since they are identical) is passed into the transfer function
- Transfer function prepends called functions from the current node to the list of called functions

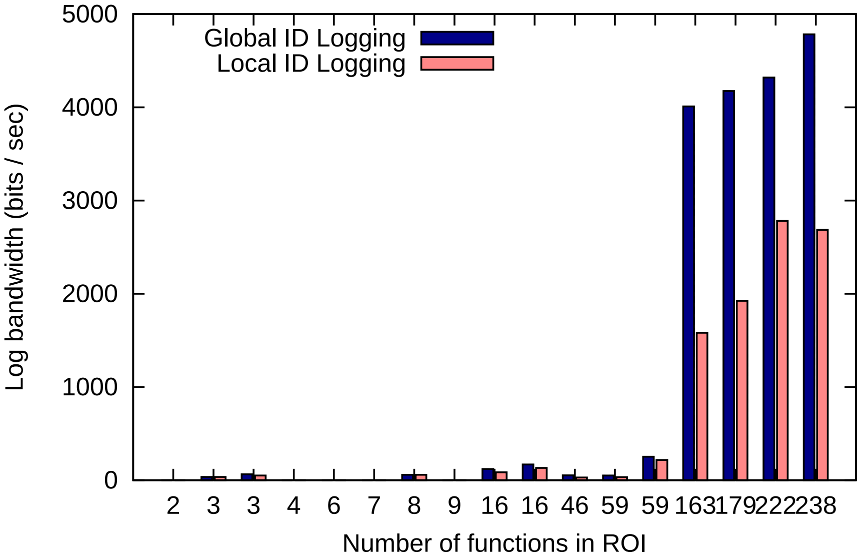
Call Trace Bandwidth



Effect of ROI Size on Token Width



Effect of ROI Size on Bandwidth



Using LowLog: Problems in the Radio Stack

```
CC2420TransmitP
```

Using LowLog: Problems in the Radio Stack

CC2420TransmitP

- TinyOS runtime resource accounting and message time stamping each work in isolation
- Combining the two causes many time stamps to be lost
- Gathered call traces from the TinyOS `CC2420TransmitP` component
  - Traces with both resource accounting and time stamping reveal a frequent number of calls to `CC2420Receive.sfd_dropped`
  - Insight limited code search down to about 20 lines of code
- Timing delays introduced by resource accounting cause an overly conservative block of code to invalidate time stamps

Outline

Summary

- LIS and LowLog help developers understand what their system is doing
  - LIS provides developers with a convenient and powerful framework for describing logging tasks
  - LowLog sits upon LIS and provides optimized call trace logging
- Have not yet found any lions, but am still looking

Want to play with LIS and LowLog?

`http://nesl.ee.ucla.edu/research/lis`

Questions?