Center for Embedded Networked Sensing

Program Analysis for Reliable Sensor Network Software

Roy Shea, Shane Markstrum, Mani Srivastava, Todd Millstein, Rupak Majumdar
NESL - http://nesl.ee.ucla.edu

Introduction: Early Identification of Errors in Embedded Sensor Network Systems

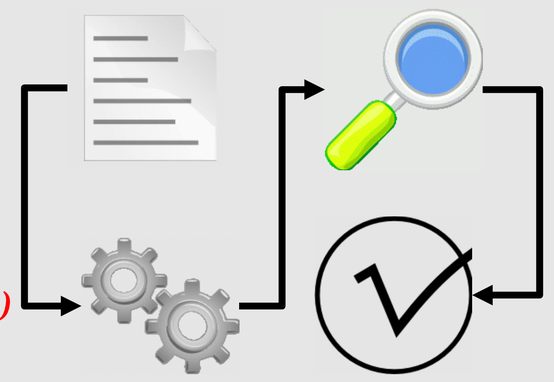
Using Staged Checks to Identify Errors

- **Staged checks can detect many classes of errors in sensor networks**
  - ***Static checkers*** verify that code base is compliant with system policies and conventions at compile time
  - ***Load-time checkers*** examine hardware and software of deployed nodes during module insertion to ensure safe module operation
  - ***Run-time checkers*** monitor node execution to detect entry into error states and provide runtime permutations to system state
- **Program analysis *complements debugging techniques* ranging from JTAG to network oriented tools such as Sympathy**
- **Analysis improves software development**
  - ***Early error identification*** eliminates many problems before deployment
  - Provides ***meaningful feedback*** to systems designers
- **This work focuses on static checkers**
  - Builds upon foundation created by established analysis techniques
  - Static analysis can be applied with minimal changes to the target system
  - Static checkers provide base work that load-time checkers and run-time checkers can stand upon

Approaching the Problem: Static Analysis of Program Source Code

Overview of Statically Analysing Sensor Code

- **(1) Begin with program code**
  - Analysis is applied directly to SOS ***module code***
  - Same framework should work for other C based systems such as EmStar and TinyOS
- **(2) Transform code into workable format**
  - Currently using the ***C Intermediate Language (CIL)***
  - Transforms C to an unambiguous low level form
  - Maintains types and form of original program



- **(3) Apply domain specific checks to transformed code**
  - Examine ***stylistic properties*** specific to target OS
  - Verify that ***protocols and resource*** models are followed
- **(4) Fix any noted errors and compile code**
  - Modules generated from verified code can be used without having to worry about classes of common errors

Current Status: Checking Stylistic Conventions and Memory Models

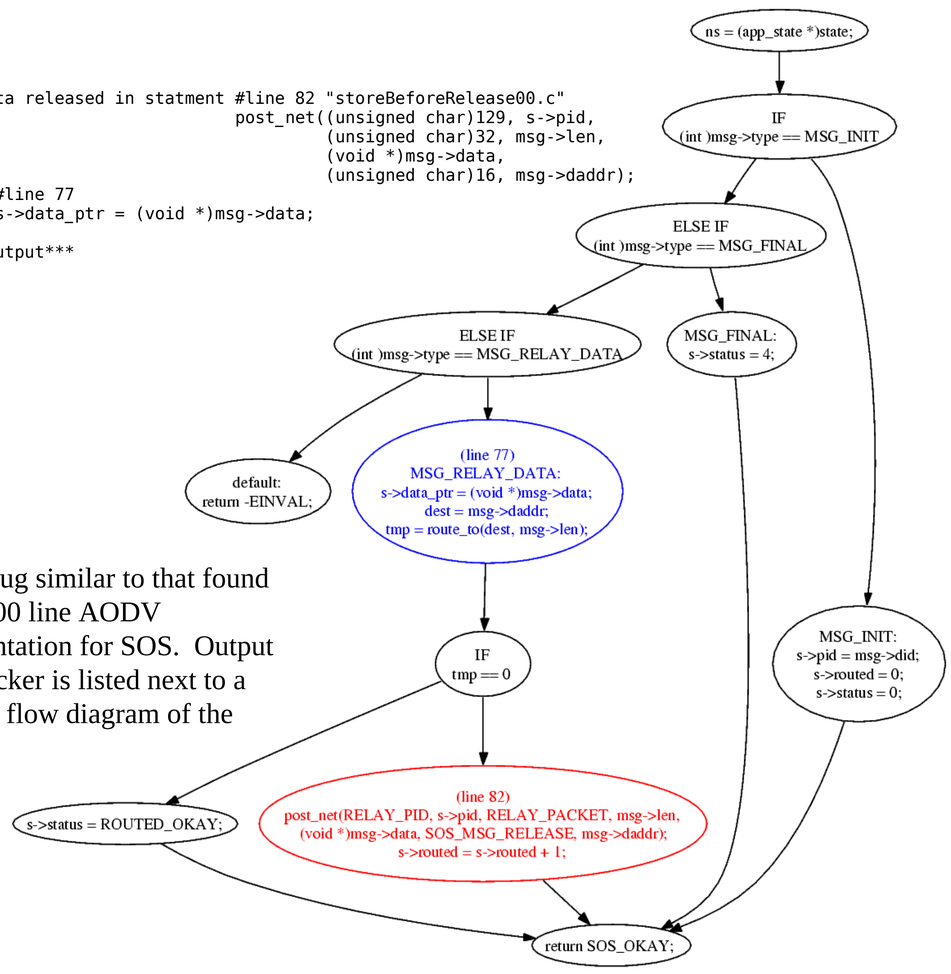
Convention and Style Checking

- **Statically check system specific conventions important to safety**
  - Verify that modules handle events critical to module insertion and removal
  - Warn users about unsafe use of global variables in modules
  - Direct users towards proper checking of return values from system critical kernel calls
  - Prevent accidental use of black listed functions

Sample Run of Static Checker

```
Warning: Data released in statment #line 82 "storeBeforeRelease00.c"
                                post_net((unsigned char)129, s->pid,
                                         (unsigned char)32, msg->len,
                                         (void *)msg->data,
                                         (unsigned char)16, msg->daddr);
escapes at #line 77
           s->data_ptr = (void *)msg->data;

***End of output***
```



Sample bug similar to that found in the 1700 line AODV implementation for SOS. Output from checker is listed next to a graphical flow diagram of the program.

Memory Ownership Verification

- **Mote class devices often have scarce memory (4k – 10k bytes)**
  - SOS provides a simple fixed block memory allocation mechanism that applications can use to allocate memory at run time
  - Ownership of data can change as data is passed between SOS modules
  - ***Memory leaks within or between modules can be a serious problem***
- **A simple memory model helps to guide programmers**
  - Each allocated memory block is owned by exactly one module
  - The owning module must either store a persistent reference to the block, release the block to another module, or free the block
  - This model aids modular analysis
- **Memory model is verified for each module loaded onto a node**
  - Control flow information is combined with alias analysis to check that ***released data is treated as dead***
  - Full data flow analysis is combined with alias analysis to check that ***data claimed by a node is properly stored, released, or freed***

Current Challenges and Future Work

- **Event based framework is pushing the limits of static analysis**
  - Many modules are designed to handle events that conform to an established protocol
  - Exploring ways to encode this information in programs without burdening the system developer
- **Developing formal models of the system**
  - Current checkers provide a good intuitive model for memory ownership
  - Formal models provide formal guarantees about the checkers
- **Expanding checker framework to other systems**
  - Systems such as TinyOS and EmStar have properties similar to those of SOS and will benefit from similar tools