Center for Embedded Networked Sensing

Compile Time Checking of Runtime Resource Management

Roy Shea, Shane Markstrum, Todd Millstein, Rupak Majumdar, and Mani Srivastava
NESL and TERTL

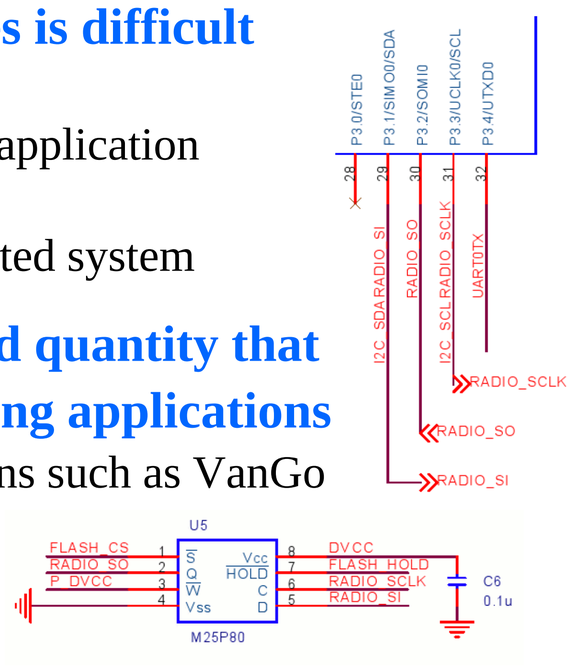
Introduction: Resource Usage in Sensor Networks

Correct usage of resources is difficult

- Low layer access to hardware
- Interactions between distributed application components
- Complex control flow in distributed system

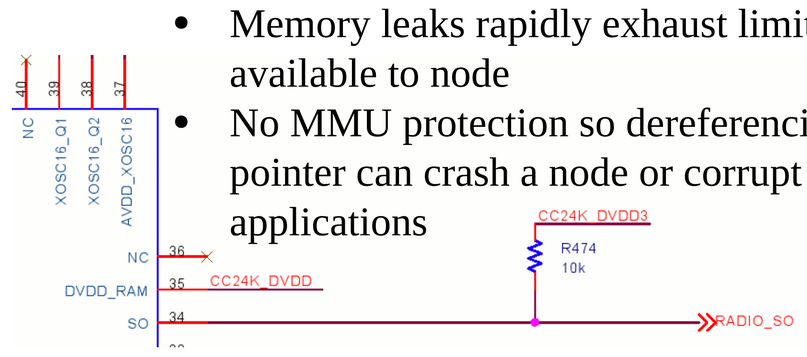
Resource is an item of limited quantity that used by one or more consuming applications

- Buffer pools in TinyOS extensions such as VanGo
- Pointer swapping in TinyOS
- Dynamic memory in SOS
- Access to underlying hardware



Focus on locating memory mismanagement in sensor networks

- Models used for memory apply to other resource management problems
- Mismanaged memory leads to significant problems
  - Memory leaks rapidly exhaust limited memory available to node
  - No MMU protection so dereferencing a dangling pointer can crash a node or corrupt state in other applications



Problem Description: Formulating a Model of Correct Memory Usage

Intuitive model for correct memory usage

- Memory should only be under the control of one program at any given time
- A program controlling memory must either keep track of the data, free the data, or transfer ownership to another program

Formal model for correct memory usage

- No memory leaks
  - $\Box\,((\text{alloc}\ x) \rightarrow \Diamond\,((\text{free}\ y \vee \text{store}\ y) \wedge \text{alias}(x, y)))$
- No dangling pointers
  - $\Box(\text{free}\ x \rightarrow \bigcirc(\Box(\neg(\text{access}(y) \wedge \text{alias}(x, y)))))$

Memory management API used within the SOS operating system for sensor networks

- Transferring ownership of memory
  - `int8_t sys_post(..., void* data, SOS_MSG_RELEASE)`
  - `void* sys_msg_take_data(void* data)`
- Allocating and freeing memory
  - `void* sys_malloc(uint16_t size)`
  - `void sys_free(void* data)`

Proposed Solution: Examining Memory Usage and Compile Time

Automated localization of memory mismanagement

- Built using C Intermediate Language (CIL)
- Combines dataflow and alias analysis
- Configured using annotation templates

Compile time feed back for application developers

- Incorrect memory usages displayed as compiler warnings
- Checker integrated with build process

Evaluation on the SOS operating system for sensor networks

Function	Total uses out of 46371 SLOC
sys_malloc	135
sys_msg_take_data	25
sys_free	124
SOS_MSG_RELEASE	116
Other common memory operations	54

Memory usage in SOS operating system

Memory usage in SOS user applications

Function	Total uses out of 12990 SLOC
sys_malloc	68
sys_msg_take_data	23
sys_free	46
SOS_MSG_RELEASE	66
Other common memory operations	8

Errors found in SOS kernel

Actual memory leaks	1
Actual dangling pointer errors	1
Missing annotations	10
Free within a loop	5
False positives	26

```
mod_op = (sos_module_op_t*) sys_msg_take_data(msg);
if(mod_op == NULL) return -ENOMEM;
if(mod_op->op == MODULE_OP_INSMOD) {
   existing_module = sys_get_module(mod_op->mod_id);
   if(existing_module != NULL) {
      uint8_t ver = sos_read_header_byte(
            existing_module->header,
            offsetof(mod_header_t,
            version));
      if (ver < mod_op->version) {
         sys_unload_module(existing_module->pid,
               sos_read_header_byte(existing_module->header,
               offsetof(mod_header_t, version)));
      } else {
         return SOS_OK;
      }
   }
   ret = fetcher_request(sys_DFT_LOADER_PID, mod_op->mod_id,
         mod_op->version, entohs(mod_op->size), msg->saddr);
         s->pend = mod_op;
   sys_led(LED_RED_TOGGLE);
   return SOS_OK;
}
return SOS_OK;
```

Errors found in SOS user applications

Actual memory leaks	16
Missing annotations	153
Free within a loop	66
False positives	72

Limitations

- Analysis requires some annotations
  - Annotations kept outside of code
  - Small set provides significant benefits
- Limited accuracy in alias analysis
  - Checker can use any external alias analysis engine
- Analysis has no concept of state

Future work

- Extend verification with a model of the application
- Formal type system for inference
- Integration with link time and run time checking