

Static Analysis of Bottom Tier Sensor Networks

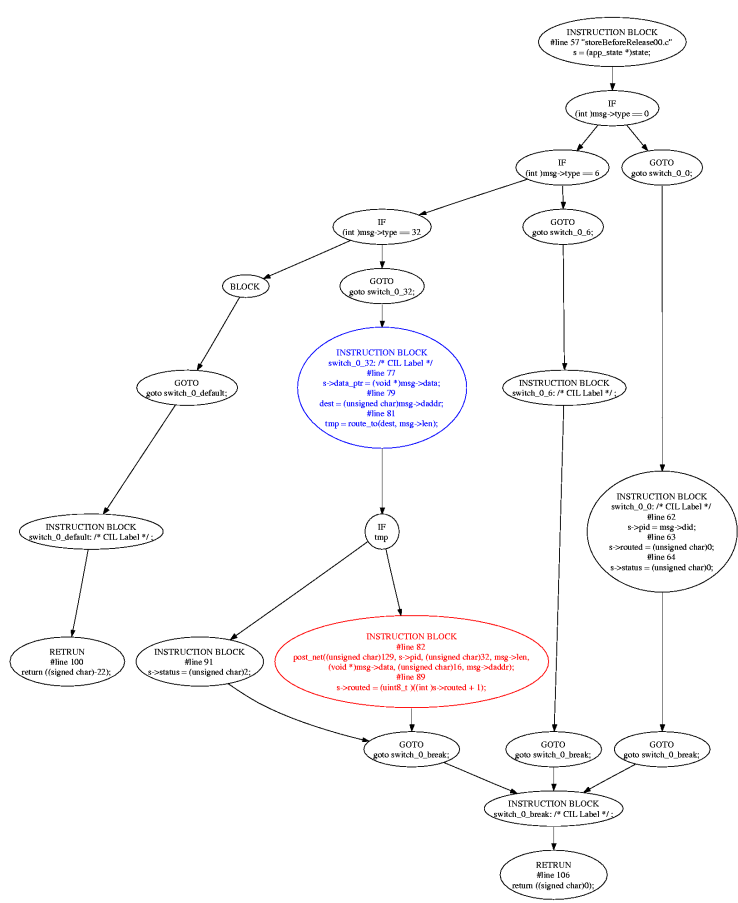
Roy Shea, Todd Millstein, Rupak Majumdar, and Mani Srivastava

Software Systems Laboratory and Networked and Embedded Systems Laboratory

Motivation: Bringing Accessible Static Analysis To Sensor Network Development

Static Analysis is Cool

- Has the potential to find bugs before they arise
- Streamlines the development process
- Lower learning curve for new developers

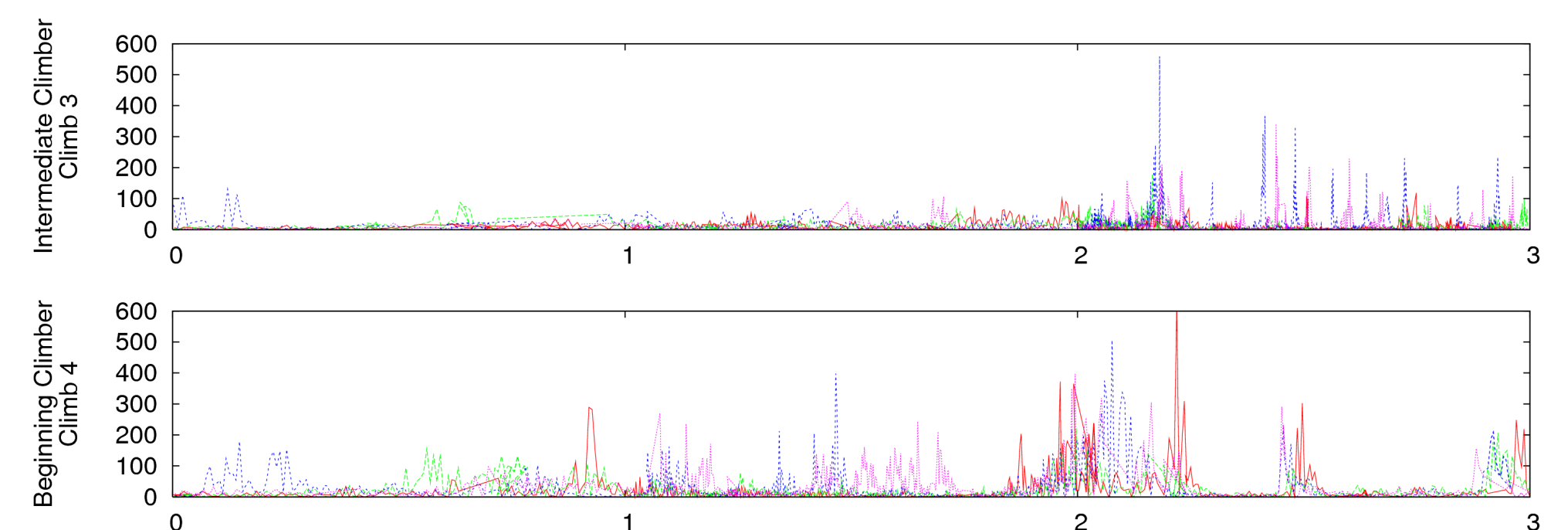


Combination of These?

- Difficulties developing sensor network applications
 - Limited insight into a running system
 - Common failure mode is simply “not working”
 - Failures are not well contained
- Gain insight by applying static analysis to bottom tier sensing devices

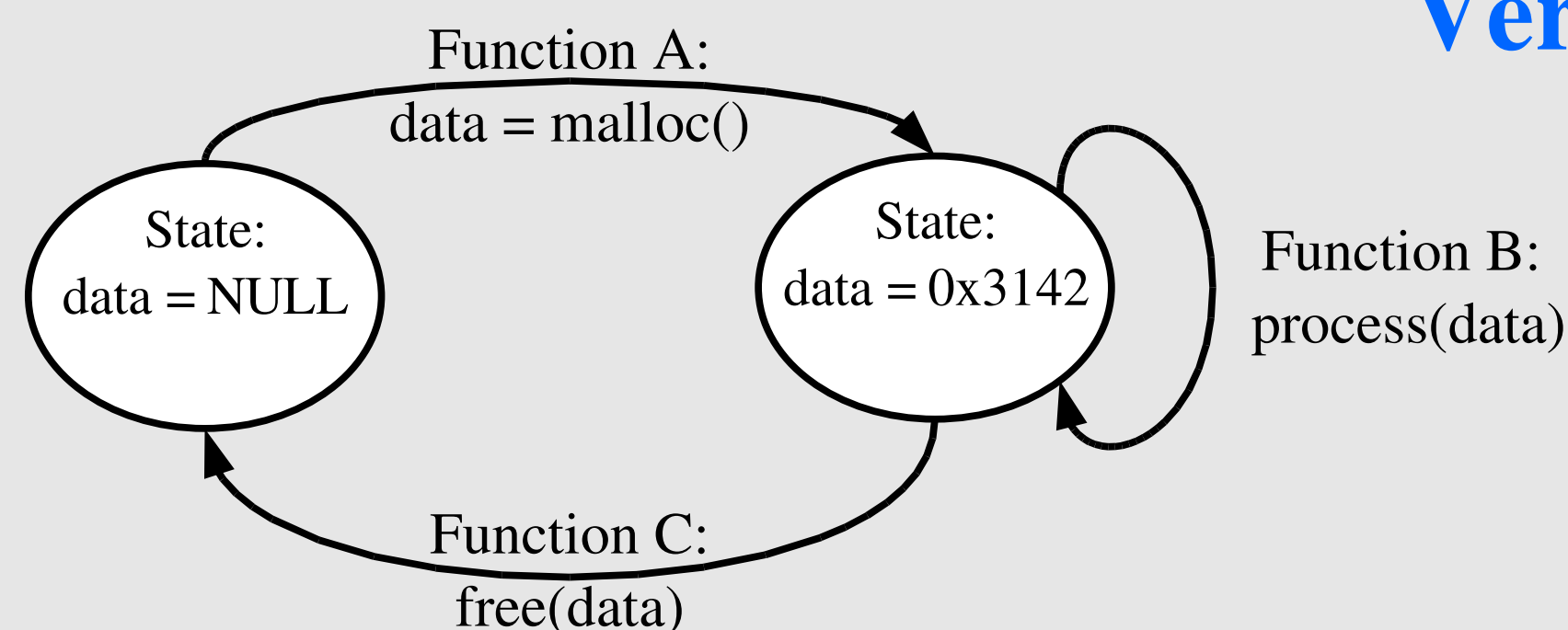
Bottom Tier Sensing Devices are Cool

- Developer can look at the entire system
- All in one hardware is great for hobbyist and students
- Show you amazing things about the world we live in

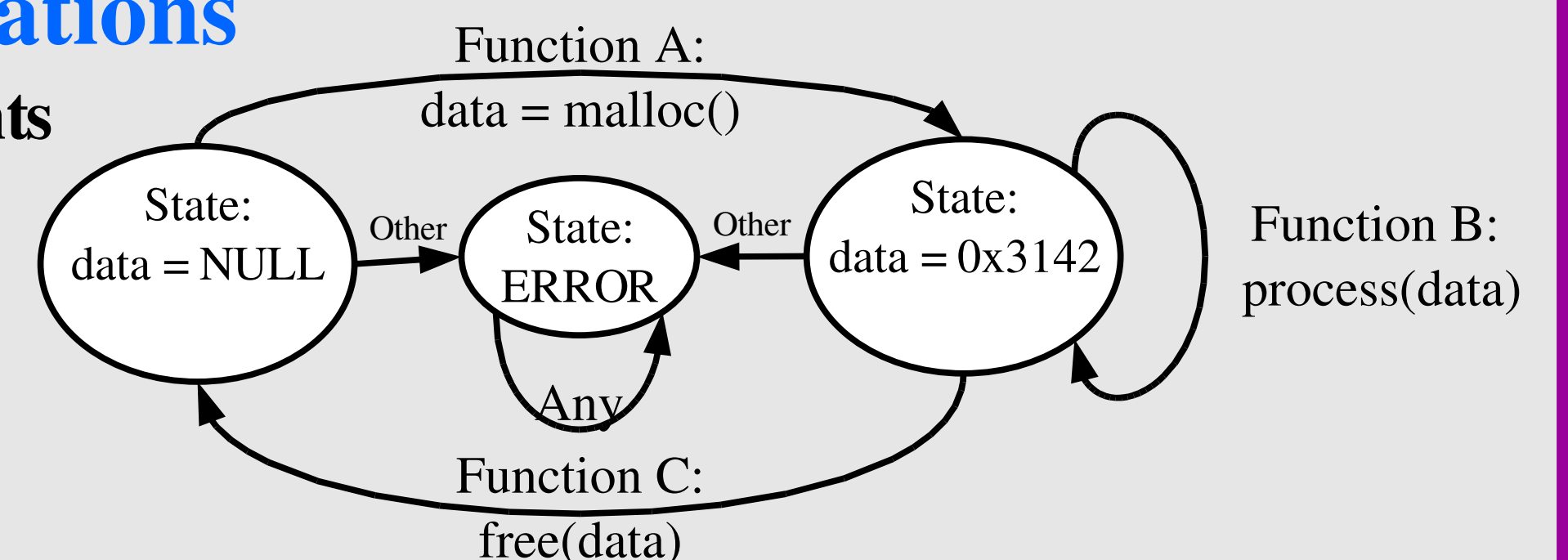


Problem: Mapping Traditional Analysis Onto Very Reactive Systems

Verifying Sensor Network Applications



- Event handlers react to external events
 - Sensor detects an event of significance
 - Message is received over the radio
 - Timer injects event into the system
- Naïve model assumes any event can occur at any time



Proposed Solution: Drive Static Analysis with Ordering Specifications

Lighthouse

- Simple exclusive resource ownership model
 - All heap memory is owned by exactly one module
 - Owner is responsible for releasing the memory or transferring it to another module
 - Modules may not free or transfer ownership of heap data that isn't owned
- Uses standard dataflow techniques to verify each function
 - May / must pointer analysis used to resolve aliases
 - Function summaries used to handle function calls

Verified Memory Leaks	8
False Memory Leaks	8
Verified Dangling Pointers	0
False Dangling Pointers	9

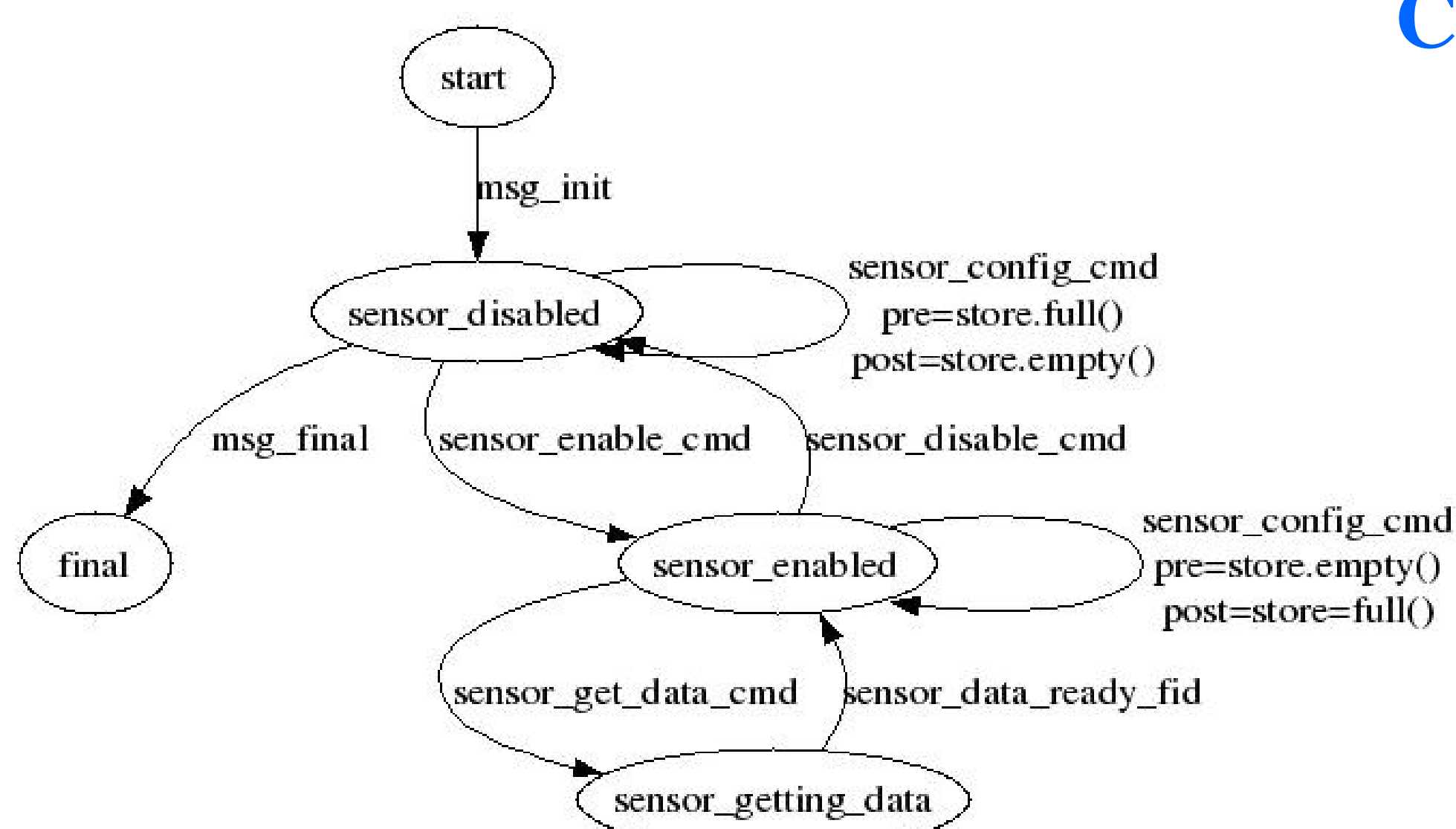
Warnings generated from the analysis of 213 unique versions of SOS modules totalling 28042 SLOC

Verified Memory Leaks	2
False Memory Leaks	22
Verified Dangling Pointers	0
False Dangling Pointers	11

Warnings generated from the analysis of 40 source files making up the SOS kernel totalling 9223 SLOC

```
mod_op = (sos_module_op_t*) ker_msg_take_data(msg);
if(mod_op == NULL) return -ENOMEM;
if(mod_op->op == MODULE_OP_INSMOD) {
    existing_module = ker_get_module(mod_op->mod_id);
    if(existing_module != NULL) {
        uint8_t ver = sos_read_header_byte(...);
        if (ver < mod_op->version) {
            ker_unload_module(...);
        } else {
            return SOS_OK;
        }
    }
    ret = fetcher_request(KER_DFT_LOADER_PID,
        mod_op->mod_id, mod_op->version,
        ntohs(mod_op->size), msg->saddr);
    s->pend = mod_op;
    ker_led(LED_RED_TOGGLE);
    return SOS_OK;
}
return SOS_OK;
```

Error: Expression mod_op is not stored after instruction #line 125
 mod_op = (sos_module_op_t*)
 ker_msg_take_data((unsigned char)18, msg);



Checkpoint

- Lighthouse driven by finite state machine specification
- Specification provided by developer describes
 - Set of arbitrary states associated with specific source code
 - Set of events that cause transitions between states
 - Persistent store pre- and post- conditions for stores when a specific event causes a transition between two states
- Critical for the reactive sensing environments
- Just beginning to evaluate gains of using Checkpoint

Future Work

- Specification generation via inference
 - Of memory manipulation by functions
 - Of interfunction per- / post- state dependencies
- Finite state machine specification
 - What is the best way to describe the FSM?
 - Transitioning from functional specification to slice oriented specification