Generating BPF infrared decoders using finite automations

Sean Young <sean@mess.org>

LPC, September 2024

$ whoami

- Sean Young <sean@mess.org>
- Maintainer of Infrared on Linux (spare time)
- Contractor

Introduction

- Infrared Decoding is done either in a few hard coded decoders in kernel space, or user space
- User space decoding requires daemon and has poor latency
- Decoding is a simple state machine with flash/gap as input
- Decoding can be done in BPF programs today - both configurable and low-latency

New tooling

- cir: Consumer InfraRed (not IrDA)
- Parses both `.lirc.conf` files and rc keymaps
- Converts both to IRP Notation
- IRP Notation is converted to BPF for daemon-less decoding
- Single tool that replaces `ir-ctl`, `ir-keytable` and all of lirc tooling
- Written in rust (links to llvm for BPF codegen)
- https://github.com/seanyoung/cir

Simple IRP

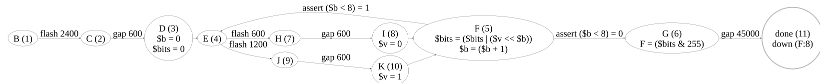
IR protocols can be described in IRP Notation:

```
{40k,600}<1,-1|2,-1>(4,-1,F:8,-45m)[F:0..255]
```

- Usually a single line
- Can also describe complex protocols e.g. air conditioning remotes
- A bit like regular expression
- Let's convert to a state machine

Simple IRP

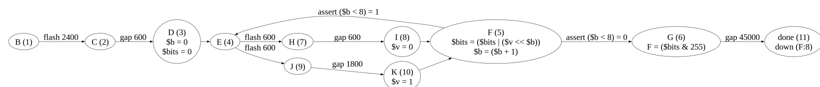
```
{40k,600}<1,-1|2,-1>(4,-1,F:8,-45m)[F:0..255]
```




```
cir decode --irp
'{40k,600}<1,-1|2,-1>(4,-1,F:8,-45m)[F:0..255]'
--save-nfa
```

Non-determinism

```
{40k,600}<1,-1|1,-3>(4,-1,F:8,-45m)[F:0..255]
```

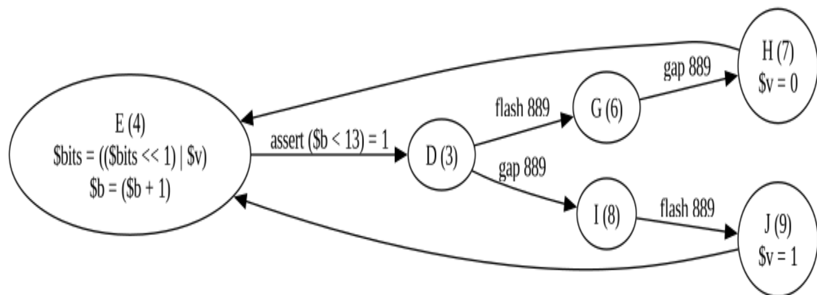


- Decoder can be in multiple states at the same time (increases eBPF program complexity)
- State machine is non-determinstic (NFA)
- Textbook answer: convert to DFA

Determinstic Finite Automation

- NFA to DFA conversion
- Powerset/Subset Construction
- Also removes empty nodes

Merge paths



- flash followed by flash: merge
- gap followed by gap: merge
- Simplifies decoder complexity
- Sometimes reduces node count

Merge paths

flash 1678..1878
$v = 1
$bits = (($bits << 1) | $v)
$b = ($b + 1)
assert ($b < 13) = 1

gap 789..989
$v = 0

gap 1678..1878
$v = 0
$bits = (($bits << 1) | $v)
$b = ($b + 1)
assert ($b < 13) = 1

E (4)

flash 789..989
$bits = (($bits << 1) | $v)
$b = ($b + 1)
assert ($b < 13) = 1

F (5)

flash 789..989
$b = 0
$bits = 0

A (0)

C (2)

flash 789..989
$v = 1

D (3)

gap 789..989
$bits = (($bits << 1) | $v)
$b = ($b + 1)
assert ($b < 13) = 1

Remove duplicate state

- DFA may still have duplicate nodes
- Text book answer: DFA minimization

rc5 decoder

- Generated state machine is simpler than hand coded solution in the kernel today!

Generating BPF #1

- Generate BPF MAP array for all variables (+ state variable)
- Generate LLVM IR
  - Load state variable and switch to code for each state
  - For each state, generate code for each edge
  - If Flash/Gap/Assert does not match, try next edge
  - If edge matches, set state to edge target
  - If no edge matches, reset decoder (state=0)

Generating BPF #2

- Ask LLVM libs to generate optimized object file in memory
- All done in 1240 lines of rust, took two days to write
- Use aya crate to load BPF program and attach to lirc chardev
- Everything done in-memory and in a single binary

Thank you

- Using finite automations allows many optimizations to simplify state machine
- Using llvm to futher optimize code generates very nice BPF code
- Writing this in rust worked out very well
- Perhaps re-usable - maybe be usable for bpfilter/netfilter