Applying Operating System Principles to SDN Controller Design

Matthew Monaco          Oliver Michel          Eric Keller
University of Colorado, Boulder

HotNets-XII
November 21, 2013
What we clearly need is an “operating system” for networks...
Monolithic
Introduction

- Event handling
- Core services
- Access control
Traditional Operating Systems
Distributions

- Technically sound
Distributions

- Technically sound
- Distributions
Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users
Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users
- Networks
INTRODUCTION

Distributions

- Technically sound
- Distributions
  - Core decisions
  - Different Audiences
  - Hackers vs. End Users
- Networks
  - Datacenters, Universities, Enterprises
  - Small business, Home Users, Hobbyists
Software Projects
How do administrators and developers interact with traditional OSes?
How do administrators and developers interact with traditional OSes?

$ grep $ find $ sed $ sort $ cat
How do administrators and developers interact with traditional OSes?

$ grep
$ find
$ sed
$ sort
$ cat
Detective Work

$ cd /var/log
$ grep some_problem syslog
Performance Analysis

$ ps -A | sort -k3nr | head -10
$ kill -TERM 12345
#!/bin/bash

procs=(
    ps -A | sort -k3nr | head -$1 | tr -s ' ' \ |
        cut -d' ' -f9
)

for p in "${procs[@]}"; do
    printf "%d is misbehaving\n" "$p" >&2
    kill -TERM "$p"
    sleep 3
    kill -KILL "$p"
done
Large Applications

```c
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
    puts("Hello, World!");
    return EXIT_SUCCESS;
}
```

```
$ ./configure
$ make
$ sudo make install
```
What can we learn from a traditional operating system?
What can we use from a traditional operating system?
INTRODUCTION

Integrate more tightly with Linux
Use off-the-shelf technologies
Encourage active ecosystem
Integrate more tightly with Linux
Integrate more tightly with Linux

Use off-the-shelf technologies
Integrate more tightly with Linux

Use off-the-shelf technologies

Encourage active ecosystem
Yanc: Yet Another Network Controller
Obligatory XKCD
Yanc: Yet Another Network Controller
Yanc: Yet Another Network Controller

Everything is a file

App    App    App    App

Filesystem

Linux

University of Colorado Boulder
Yanc: Yet Another Network Controller

Everything is a file

Procsfs, Sysfs
Why a filesystem?
Introduction

Logically Distinct Applications

- firewall
- learning switch
- logging
- slicing

Filesystem

Linux
Independent Development
Independent Development

# apt-get install yanc-learning-switch

# apt-get install yanc-router

$ git clone git@github.com/mmonaco/yanc-firewall && cd yanc-firewall
$ make
# make install
Any Programming Language
Other Technologies

- Inotify
- File permissions and ACLs
- Namespaces and CGroups
- Layered Filesystems
Decouple from Hardware
The Yanc Filesystem
The Yanc Filesystem

Root Directory

/net
  ├── hosts/
  │    ├── switches/
  │    │    ├── sw1/
  │    │    └── sw2/
  │    └── views/
  │         └── http/
  └── management-net
       └── hosts/
            └── switches/
                └── views/
Root Directory

/net
  └── hosts/
    └── switches/
      └── sw1/
      └── sw2/
  └── views/
    └── http/
    └── management-net
      └── hosts/
      └── switches/
      └── views/
Switch Directory

/net
  ├── switches
  │    └── 01:02:03:04:05:06
  │         ├── flows/
  │         ├── packet_in/
  │         └── packet_out/
  │             └── ports/
  │             └── actions
  │                 └── capabilities
  │                     └── id
  │                         └── n_buffers
Switch Directory

/net
  └── switches
      ├── 01:02:03:04:05:06
      │    └── flows/
      │          └── packet_in/
      │                  └── packet_out/
      │                      └── ports/
      │                              └── actions
      │                                           └── capabilities
      │                                               └── id
      │                                                   └── nbuffers
Port Directory

```
/net
  └── switches
    └── 01:02:03:04:05:06
  └── ports
    └── LOCAL
          └── config.port_down
          └── hw_addr
               └── peer -> /dev/null
          └── port_no
          └── stats.rx_bytes
               └── stats.rx_packets
```
Flow Entry Directory

/net
  ─── switches
      ── 01:02:03:04:05:06
          ── flows
              ── arp_flow
                  ── counters
                  ── action.out
                  ── match.dl_src
                  ── match.dl_type
                  ── priority
                  ── timeout
                  ── version
THE YANC FILESYSTEM

Flow Entry Directory

/net
  └── switches
      └── 01:02:03:04:05:06
          └── flows
              └── arp_flow
                  ├── counters
                  │     └── action.out
                  │                 └── match.dl_src
                  │                                  └── match.dl_type
                  │                                      └── priority
                  │                                           └── timeout
                  │                                                └── version
THE YANC FILESYSTEM

Packet In Directory

/net
   I switches
      I 01:02:03:04:05:06
         I packet_ins
            I in1
               I buffer_id
               I data
               I data_len
               I in_port
               I reason
Packet In Directory

```
/net
  └── switches
      └── 01:02:03:04:05:06
            └── packet_ins
                  └── in1
                          └── buffer_id
                          └── data
                              └── data_len
                                  └── in_port
                                      └── reason
```
Packet Out Directory

```
/net
├── switches
│   ├── 01:02:03:04:05:06
│   │   └── packet_outs
│   │       └── out1
│   │           └── action.out_port
│   │           └── buffer_id
│   │               └── data
│   │                   └── state
```
Packet Out Directory

```
/net
  └── switches
      └── 01:02:03:04:05:06
          └── packet_outs
              └── out1
                  └── action.out_port
                      └── buffer_id
                          └── data
                              └── state
```
Using the Filesystem
$ echo 1 > port_1.port_down
$ echo 1 > port_1.port_down

$ cd switches/00:01:02:0a:0b:0c/flows
$ mkdir my_flow_entry
$ ls -l my_flow_entry
  counters
  priority
  timeout
  version
$ find /net -name tp.dst -exec grep 22 {} +
$ find /net -name tp.dst -exec grep 22 {} +

#!/bin/bash
flowdir=/net/switches/"$1"/flows/"$2"
mkdir "$flowdir"
echo ff:ff:ff:ff:ff:ff:ff > "$flowdir"/match.dl_dst
echo 0x0806 > "$flowdir"/match.dl_type
echo FLOOD > "$flowdir"/action.out
#!/usr/bin/env python3

def new_switch(id, n_tables=1):
    pass

def write_flow(switch, matches=[], actions=[]):
    pass

#ifndef _YANC_H_
def _YANC_H_
#endif/*_YANC_H_*/

int new_switch(uint64_t, uint8_t);
int write_flow(const char* path, match_t*, action_t*);
Applications and Implementation

- Filesystem - C, FUSE
- OpenFlow - C++
- Discovery - Python
- Static Flow Pusher - Bash
- Filesystem - C, FUSE
- OpenFlow - C++
- Filesystem - C, FUSE
- OpenFlow - C++
- Discovery - Python
- Filesystem - C, FUSE
- OpenFlow - C++
- Discovery - Python
- Static Flow Pusher - Bash
Ongoing Work
FUTURE WORK

Composition
Distribution
Applications
Composition

- Independent development
Composition

- Independent development
- Event handling
Composition

- Independent development
- Event handling
- Synthesize configuration
Composition

- Independent development
- Event handling
- Synthesize configuration
- /usr/share/yanc.d/
- /etc/yanc.d/
Future Work

Composition

- Independent development
- Event handling
- Synthesize configuration
- /usr/share/yanc.d/
- /etc/yanc.d/
- Ship with configuration
Composition

- Independent development
- Event handling
- Synthesize configuration
- `/usr/share/yanc.d/`
- `/etc/yanc.d/`
- Ship with configuration
- Administrator controlled
Composition

# /etc/yanc.d/

<event> slice
<event> firewall
<event> router
Distribution

Latency

Fault Tolerance

Administration

Different Requirements
Distribution

filewall
learning switch
logging
slicing

Filesystem

Linux
Distribution

Distributed Filesystem
Yanc Filesystem
Linux
Distribution
Distribution
Controller A

Controller B

Controller C
FUTURE WORK
FUTURE WORK

Controller A

Controller B

Controller C
Defined a Universal Interface for Network Control
Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes
Defined a Universal Interface for Network Control

Allow Control Applications as Separate Processes

Implemented Interface
Defined a Universal Interface for Network Control
Allow Control Applications as Separate Processes
Implemented Interface
Leveraging Existing OS Technologies
Defined a Universal Interface for Network Control
Allow Control Applications as Separate Processes
Implemented Interface
Leveraging Existing OS Technologies
Built Functional Applications on Top of Interface
Thank You!