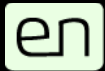


# SWITCHD

An OpenFlow implementation for OpenBSD – BSDCan 2016  
Reyk Flöter (reyk@openbsd.org) – ESDENERA NETWORKS GmbH



This presentation introduces switchd(8) and switch(4), a simple OpenFlow controller and virtual switch for **OpenBSD**. After vxlan(4), this presentation is the second part about the

# CLOUD NETWORKING STACK – PART II



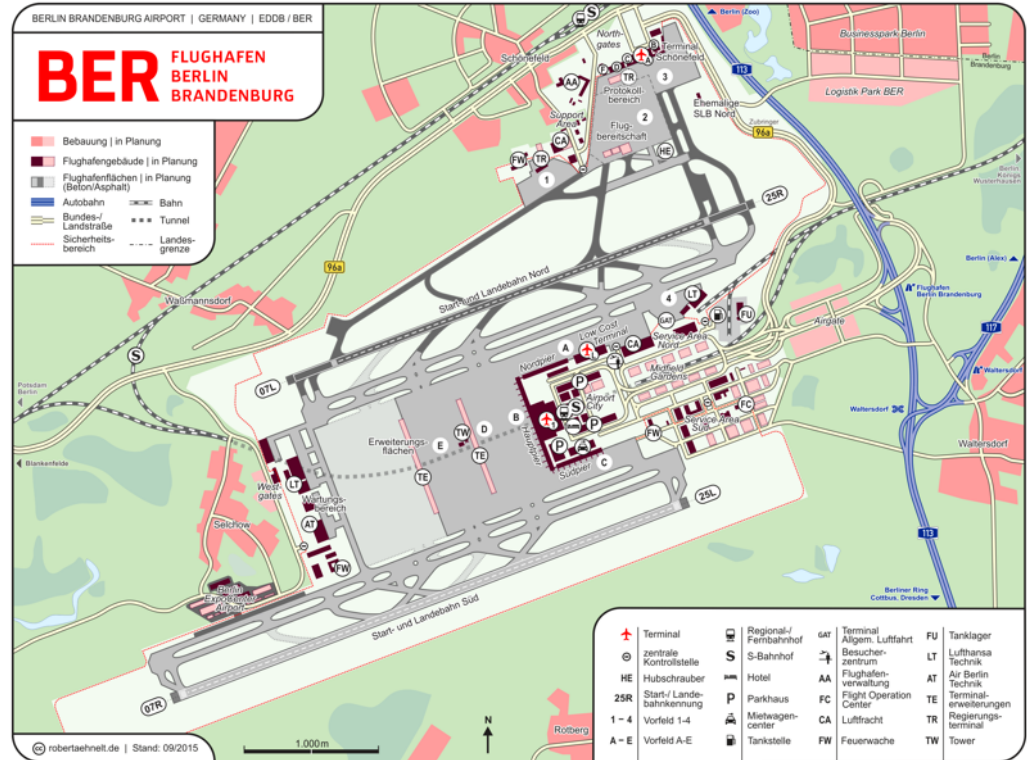
# CLOUD NETWORKING STACK

APPLICATION LAYER	relayd, httpd		...	
TCP/IP	Routing Domains		pf (Packet Filter)	
VIRTUAL NETWORKS	vxlan(4)	<div>NVGRE gre(4)</div>	vlan(4) svlan(4)	VPN ...
VIRTUAL ETHERNET	OpenFlow, SDN		switch(4) and switchd(8)	
VIRTUAL DEVICES	Virtual I/O: vic(4), vio(4), vmx(4), xnf(4), hvn(4)			

# ROME WASN'T BUILT IN A DAY

## Disclaimer

- switchd(8) and switch(4) haven't been released yet
- The code exists and will (hopefully) show up in –current soon
- It will not be enabled soon and there is still a lot of work to do



# THE OPENFLOW PROTOCOL

| TCP or TLS

## 64 bit OpenFlow Header:

Version	Type	Length
---------	------	--------

Transaction ID	Account	Amount	Date	Description
1	John Doe	1000	2023-01-01	Initial deposit
2	John Doe	500	2023-01-15	Withdrawal
3	John Doe	250	2023-02-01	Withdrawal
4	John Doe	750	2023-02-15	Deposit
5	John Doe	100	2023-03-01	Withdrawal
6	John Doe	300	2023-03-15	Withdrawal
7	John Doe	400	2023-04-01	Withdrawal
8	John Doe	150	2023-04-15	Withdrawal
9	John Doe	200	2023-05-01	Withdrawal
10	John Doe	100	2023-05-15	Withdrawal
11	John Doe	500	2023-06-01	Deposit
12	John Doe	200	2023-06-15	Withdrawal
13	John Doe	150	2023-07-01	Withdrawal
14	John Doe	100	2023-07-15	Withdrawal
15	John Doe	500	2023-08-01	Deposit
16	John Doe	250	2023-08-15	Withdrawal
17	John Doe	100	2023-09-01	Withdrawal
18	John Doe	150	2023-09-15	Withdrawal
19	John Doe	200	2023-10-01	Withdrawal
20	John Doe	100	2023-10-15	Withdrawal
21	John Doe	500	2023-11-01	Deposit
22	John Doe	250	2023-11-15	Withdrawal
23	John Doe	100	2023-12-01	Withdrawal
24	John Doe	150	2023-12-15	Withdrawal
25	John Doe	200	2024-01-01	Withdrawal
26	John Doe	100	2024-01-15	Withdrawal
27	John Doe	500	2024-02-01	Deposit
28	John Doe	250	2024-02-15	Withdrawal
29	John Doe	100	2024-03-01	Withdrawal
30	John Doe	150	2024-03-15	Withdrawal
31	John Doe	200	2024-04-01	Withdrawal
32	John Doe	100	2024-04-15	Withdrawal
33	John Doe	500	2024-05-01	Deposit
34	John Doe	250	2024-05-15	Withdrawal
35	John Doe	100	2024-06-01	Withdrawal
36	John Doe	150	2024-06-15	Withdrawal
37	John Doe	200	2024-07-01	Withdrawal
38	John Doe	100	2024-07-15	Withdrawal
39	John Doe	500	2024-08-01	Deposit
40	John Doe	250	2024-08-15	Withdrawal
41	John Doe	100	2024-09-01	Withdrawal
42	John Doe	150	2024-09-15	Withdrawal
43	John Doe	200	2024-10-01	Withdrawal
44	John Doe	100	2024-10-15	Withdrawal
45	John Doe	500	2024-11-01	Deposit
46	John Doe	250	2024-11-15	Withdrawal
47	John Doe	100	2024-12-01	Withdrawal
48	John Doe	150	2024-12-15	Withdrawal
49	John Doe	200	2025-01-01	Withdrawal
50	John Doe	100	2025-01-15	Withdrawal
51	John Doe	500	2025-02-01	Deposit
52	John Doe	250	2025-02-15	Withdrawal
53	John Doe	100	2025-03-01	Withdrawal
54	John Doe	150	2025-03-15	Withdrawal
55	John Doe	200	2025-04-01	Withdrawal
56	John Doe	100	2025-04-15	Withdrawal
57	John Doe	500	2025-05-01	Deposit
58	John Doe	250	2025-05-15	Withdrawal
59	John Doe	100	2025-06-01	Withdrawal
60	John Doe	150	2025-06-15	Withdrawal
61	John Doe	200	2025-07-01	Withdrawal
62	John Doe	100	2025-07-15	Withdrawal
63	John Doe	500	2025-08-01	Deposit
64	John Doe	250	2025-08-15	Withdrawal
65	John Doe	100	2025-09-01	Withdrawal
66	John Doe	150	2025-09-15	Withdrawal
67	John Doe	200	2025-10-01	Withdrawal
68	John Doe	100	2025-10-15	Withdrawal
69	John Doe	500	2025-11-01	Deposit
70	John Doe	250	2025-11-15	Withdrawal
71	John Doe	100	2025-12-01	Withdrawal
72	John Doe	150	2025-12-15	Withdrawal
73	John Doe	200		

```
| Type-specific header, packet data ...
```

# THE OPENFLOW PROTOCOL

- A method to decouple the switch data and control plane
- A switch can ask a remote controller to make forwarding decisions
- OpenFlow is a TCP-based protocol between switch and controller
- Protocol message types:
  - HELLO: connection setup
  - PACKET-IN: switch-controller message with full Ethernet packet
  - PACKET-OUT: controller-switch response with packet or buffer ID
  - FLOW-MOD: controller installs a “flow” in the switch
  - That is enough to implement a “learning switch” on the controller
- But the complexity is in the details, sub-types and classifiers

# THE OPENFLOW PROTOCOL

- Evolution of the OpenFlow Protocol
  - openflow-spec-v1.0.0.pdf 42 pages – simple and nice
  - openflow-switch-v1.1.0.pdf 56 pages – MPLS & VLAN, TTL
  - openflow-switch-v1.2.pdf 85 pages – IPv6, Extensible Match
  - openflow-switch-v1.3.5.pdf 177 pages – VXLAN, ...
  - openflow-switch-v1.4.pdf 206 pages – ...
  - openflow-switch-v1.5.1.pdf 283 pages – ...

# AN OPENFLOW EXPERIMENT

- Around 2013, I experimented with the OpenFlow 1.0 protocol
- When I looked at it, all existing controllers were either really bad or big
  - Written in Java (most popular), Python, Ruby, “insecure C”, ...
- So I implemented a little daemon (ofpd)
  - It provided very basic support for OpenFlow 1.0
- There was no real use case for it and it didn't even have a name
  - ~~openflowd~~ – The OPENFLOW™ trademark is too restrictive
  - ~~ofpd, sdnd, sdnflowd, OpenWolf~~ – Not nice and not funny
- Put it on hold and stopped thinking about it





# THE BRIDGE

- Three main problems:
  1. We are suffering from the aging bridge(4) code in OpenBSD
  2. bridge(4) is in the way of the MP network stack overhaul
  3. The control plane is integrated and not sufficient as a “vswitch”
    - And I promised mlarkin@ to provide one for vmm(4)
- The bridge(4) has many special features
  - bridge rules, blocknonip, VXLAN integration, IPSec bridge, WLAN failover, PF tags, STP ...
  - ... and tentacles everywhere

# THE BRIDGE

- Three possible solutions:
  1. We tried to clean it up and to incrementally improve it
    - Code has been improved, but there are conceptual limitations
  2. We looked at alternatives and experimentally ported Open vSwitch
    - It turned in to a HUGE diff for the kernel code and data path
    - The license is not suitable for OpenBSD's kernel (Apache 2)
  3. Re-implement it as a new driver: switch(4)
    - Using the desin of Open vSwitch would be a massive effort
    - So I had an idea ...

... “Why don’t we use my experimental OpenFlow controller as a vSwitch and talk to it with OpenFlow from the kernel?”

# SWITCHD(8)



# THE ~~BRIDGE~~ SWITCH

Name	Open vSwitch		<b><u>OpenBSD</u></b>
Remote	Controller		<i>Controller</i>
User	ovsdb-server	ovs-vswitchd	Controller or forwarder: switchd(8)
User - Kernel	"dpif" DataPath InterFace		OpenFlow via /dev/switch*
Kernel	Kernel Datapath		switch(4)

# THE SWITCH

- I implemented the userland daemon, a.k.a. switchd(8)
- goda@ and yasuoka@ implemented the kernel switch(4) driver
  - Partially based on OpenBSD's bridge(4):
    - if\_switch.[ch] – the network interface "cloner"
    - switchctl.c – the optional control plane
    - switchofp.c, net/ofp.h – the OpenFlow implementation
    - /dev/switch\* – each switch(4) has a char device
  - It currently shares some code with it:
    - if\_bridge.h – share structures for STP etc.
    - bridgestp.c – the spanning tree implementation





# CONFIGURATION EXAMPLES

## switchd(8) configuration

```
- Currently in /etc/switchd.conf:
listen on 0.0.0.0 port 6633
device "/etc/switch0"
device "/etc/switch1" \
    forward to tcp:192.168.100.1

- Planned:
switch "edge" {
    listen on tcp:0.0.0.0:6633
    connect to device:/dev/switch0
    forward to tls:192.168.100.1
}
```

## switch(4) configuration

```
- Almost like the bridge(4)

# ifconfig switch0 create
# ifconfig switch0 add em0
# ifconfig switch0 add vxlan2
# ifconfig switch0 up

- Unlike bridge, IPs can only be
  assigned to routing "IRB" interfaces

# ifconfig vether0 create 10.1.1.1
# ifconfig switch0 add vether0
```

# FUTURE WORK

- switchd(8)
  - Convert it from OpenFlow 1.0 to 1.3.5
    - Implement all MUST options of the protocol
  - Support multiple independent switch contexts/sections
    - *switch “foo”{ ... }, switch “bar”{ ... }*
  - Support multiple switches per switch context
    - Switch “foo” and “bar” are joined to a “big switch”
  - Enable pledge, turn privsep from “fork” into “fork and execute”



# THE OPENFLOW PROTOCOL

- Evolution of the OpenFlow Protocol
  - openflow-spec-v1.0.0.pdf 42 pages ← switchd(8)
  - openflow-switch-v1.1.0.pdf 56 pages
  - openflow-switch-v1.2.pdf 85 pages
  - openflow-switch-v1.3.5.pdf 177 pages ← switch(4)
  - openflow-switch-v1.4.pdf 206 pages
  - openflow-switch-v1.5.1.pdf 283 pages

# FUTURE WORK

- switch(4)
  - Some cleanup, commit, review, and test
  - Some mallocs have to be replaced with pools
  - Support (old) in-kernel control plane from bridge(4) as a fallback
  - Eventually remove bridge(4)
- Other
  - VXLAN will support IPv6 and OpenFlow-integration
  - NVGE is still not supported

# FUTURE WORK

- vmd(8) integration
  - vmm(4) is OpenBSD's virtual machine monitor
  - Networking support is currently very simple

*# OLD:*

```
vm "openbsd" {  
    interfaces 1  
    ...  
}
```

*# NEW:*

```
vm "openbsd" {  
    kernel "/bsd"  
    memory 512M  
    disk "/home/vm/OpenBSD.img"  
    interface on "vnet1"  
}
```

```
switch "vnet1" {  
    # uplink interface  
    interface em0  
    #controller 10.1.1.1  
}
```

Questions?





...and please keep supporting the OpenBSD project!

<http://www.openbsdoundation.org/campaign2016.html>

