A path to the Future Internet: Networks controlled by software and softwarized networks

Holger Karl
hkarl@ieee.org

https://sfb901.uni-paderborn.de
http://sonata-nfv.eu
Content

- Motivation for NFV, SDN
- SDN: Some basics
- NFV: Some basics
- NFV meets DCC and microservices
- NFV reference architecture
- Templates and applications
- NFV: Algorithmic issues
- NFV: Testing services on your laptop
NFV & SDN

• What is Network Function Virtualization?

• What is Software-Defined Networking?
NFV: Ingredients

- Functions
  - Physical
  - Virtual
  - Typically, with management interface
  - In an executable form

- Services
  - Consist of functions, traversed in certain order
  - Service chain, forwarding graph, ...
  - Typically, with management interface

- Infrastructure
  - Virtualized by a hypervisor

- Orchestration

Virtual Network Function
Virtual Deployment Unit
NFV: A definition attempt

- The ability to run functions inside a network
- Typical functions process data flows
- The idea to restructure code running inside a network
  - From monolithic blobs to individual components
  - Flexibly composed into services
SDN: Ingredients

• Switches
  • With flow tables

• Flows

• Control
SDN: A definition attempt

• Making flow tables controllable from the outside via a standardized API

• Concentrating control
  • From individual switches to few / a single controller
    • Logically centralized; implementation might be distributed

• A programming model
  • Or: a family of programming models, imposed by the particular controller framework
SDN is *not* ...

- Revolutionary: MPLS
- OpenFlow
- Central control
- A guaranteed market success
- Limited to specific networks (e.g., data centres, cloud networks)
- White-box switching
  - White-box switch: Hardware without a default operating system
Why?

- Why NFV? Why SDN?

- Business models?
Content

• Motivation for NFV, SDN
• **SDN: Some basics**
• NFV: Some basics
• NFV meets DCC and microservices
• NFV reference architecture
• Templates and applications
• NFV: Algorithmic issues
• NFV: Testing services on your laptop

Slides mostly by N. McKeown; Stallings
Step 1:
Separate intelligence from datapath

Operators, users, 3rd party developers, researchers, …

New function!
Traditional vs. SDN networking

(a) Traditional network architecture

(b) SDN approach

FIGURE 3.2 Control and Data Planes

From Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud by William Stallings (0134175395)
Copyright © 2016 Pearson Education, Inc. All rights reserved.
Step 2: **Cache decisions in minimal flow-based datapath**

“**If header = x, send to port 4**”
“**If header = y, overwrite header with z, send to ports 5,6**”
“**If header = ?, send to me**”
What is a flow?
- Application flow
- All http
- Jim’s traffic
- All packets to Canada
- ...

Types of action
- Allow/deny flow
- Route & re-route flow
- Isolate flow
- Make flow private
- Remove flow
Packet-switching substrate

Collection of bits to plumb flows (of different granularities) between end points
Properties of a flow-based substrate

We need flexible definitions of a flow
- Unicast, multicast, waypoints, load-balancing
- Different aggregations

We need direct control over flows
- Flow as an entity we program: To route, to make private, to move, …

Exploit the benefits of packet switching
- It works and is universally deployed
- Its efficient (when kept simple)
Substrate: “Flowspace”

Collection of bits to plumb flows (of different granularities) between end points

Payload

Header
User-defined flowspace
Flowspace: Simple example

- Single flow
- All flows from A
- All flows between two subnets
Flowspace: Generalization

Field 1

Field 2

Field n

Single flow

Set of flows
And the result: The SDN architecture

**FIGURE 3.3 Software-Defined Architecture**

From *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud* by William Stallings (0134175395)  
Copyright © 2016 Pearson Education, Inc. All rights reserved.
The SDN Stack

Monitoring/debugging tools

Applications

Controller

Slicing Software

Commercial Switches

OpenFlow Switches

From [2]
OpenFlow pipeline processing

- Packets arriving at switch traverse through a sequence of ingress and egress flow tables
  - Simplest case: only a single ingress table
- Processing always starts at table 0
  - Later pipeline stages can be skipped depending on packet

Figure 2: Packet flow through the processing pipeline.

OF 1.5.1 specification
Figure 4: Matching and Instruction execution in a flow table.
Content

- Motivation for NFV, SDN
- SDN: Some basics
- **NFV: Some basics**
- NFV meets DCC and microservices
- NFV reference architecture
- Templates and applications
- NFV: Algorithmic issues
- NFV: Testing services on your laptop
Look at many functions: services

- Isolated network functions the exception
- Rather: data flows via several network functions
- Simplest case: a chain
  - E.g.: A firewall, then a DPI, then a CDN
- More sophisticated: arbitrary acyclic graph
  - Network function forwarding graph
Placement

• Given a forwarding graph: On which actual nodes to execute which function?
  • Dealing with many graphs? Reuse functions? ...?
Deployment

• Suppose placement is solved
• How to instantiate VMs, set up paths?
  • Software-Defined Networking!
Scaling

• Suppose monitoring tells us that KPIs not met
  • Throughput low, latency high, ...
• Scale: Add virtual network functions, re-route, ...
Content

• Motivation for NFV, SDN
• SDN: Some basics
• NFV: Some basics
• NFV meets DCC and microservices
• NFV reference architecture
• Templates and applications
• NFV: Algorithmic issues
• NFV: Testing services on your laptop
So far: Network functions

• Network functions: Working on a flow of packets

• Other functions?
  • Examples?
  • Deployment?
Broadening the scope: Distributing data centers

• Common rationale: Economy of scale
  • Build huge data centers to save money
  • But results in only a few centers

• Consequence: data centers far away from users
  • Fine for many applications
  • Critical for some: interactive, gaming, content distribution, streaming, ...

• Have some local functionality close to users?
  • Smaller centers, widely distributed
  • More expensive, more suitable?
Distributed Cloud Computing

- DCC pioneered in late 2000s, under different names
  - Distributed Cloud Computing
  - Carrier cloud
  - In-network cloud
  - Nano data centers
  - Mobile edge computing
  - Fog & mist computing ….

- Many commonalities with NFV
  - Different emphasis: end-user applications vs. network-oriented features
  - Convergence? Open question …
Expectations for services?

- Scalability
- Dependable
- Cost-efficient
- Maintainable
Common pattern: Service out of services, loosely coupled

- Construct a service out of smaller services
  - Independently runnable
  - Focused on a single functionality
- **Loosely coupled**, well defined, simple interfaces

- **Microservices**
  - Recursive pattern: Chop up services into smaller services
    - E.g.: data analysis
Scalable microservices: Example

- Simple example: Service consists of three microservices, in a linear chain
- Each microservice: stateless, can process each request in isolation
- Computation per request on a standard machine:
  - MS1, MS3: 10 milliseconds
  - MS2: 100 milliseconds
  - Maximum throughput: 10 requests/second
Scalable microservices: Dealing with bottlenecks

- Options for MS2?
  - Buy a faster machine – only helps up to a point
    - “Scaling up”
  - Run multiple instances of MS2
    - “Scaling out”

How to partition requests?

How to unify requests?
Microservices: Challenges

• When to run how many instances of which service doing which work?
  • Buzzword: Stateless, autoscaling, load balancing

• How to deal with crashing services?
  • Buzzword: Automatic restart

• How to organize dependable, efficient message exchange between individual services?
  • Buzzword: Message queueing

• Stateful services?
  • Buzzword: Database
Same, same?

• Differences and commonalities?
  • NFV vs. DCC?
  • Microservice as pattern common to both?

• Is there a difference between “network services” and “IT services”?
  • An inherent one?
  • One due to current architectures?
Hard-core VNF interpretation

• VNFs are transparent on an IP level
  • Transparent for source and destination
  • Should not modify Layer 3 addresses
  • Implies: only use Layer 2 interfaces
  • Cannot use L3 forwarding
    • Why? Really?
  • Need to move traffic towards VNFs/put VNFs in the path

• Why? Consequences?
  • Eases reuse of existing code bases for functions
  • Makes shunting of traffic between VNFs more difficult
    • SDN to the rescue
  • Introduces a difference between “ordinary” and “network” services
Mid-term goal: Merge NFV, DCC, SDN
Content

• Motivation for NFV, SDN
• SDN: Some basics
• NFV: Some basics
• NFV meets DCC and microservices
• **NFV reference architecture**
• Templates and applications
• NFV: Algorithmic issues
• NFV: Testing services on your laptop
NFV reference architecture

- And for completeness: the ETSI NFV architecture

- IETF: Service function chaining
  - Substantially quite similar, just different terminology
ETSI reference architecture

Mano framework

VIM (e.g., OpenStack, VMWare, …)
Network Services as Forwarding Graphs of VNFs

- In the end, we want to build services meaningful for a user
  - In the network, end-to-end
  - Examples: virtual private network, mobile voice, …
  - This goes beyond the purview of a single VNF

- Goal: combine multiple VNFs into a network service
  - Executed by the compute domain, interconnected by the network infrastructure domain

- Typically: Order matters!
  - A service needs several functions, working on a data flow in a certain order
  - Expressed as a forwarding graph
Forwarding graphs: Recursion allowed

- A VNF inside a forwarding graph can be constructed using a forwarding graph!
Goal: Deploy an NFV service

• Decide:
  • Which instances of which VNF are needed
  • Where to run which instance
  • How to connect instances together

• Required knowledge?
  • Structure of service
  • Offered load and infrastructure capabilities
  • VNF quantification: How many resources to deal with how much traffic, at which quality
  • Hence: quantitatively annotated templates
Content

• Motivation for NFV, SDN
• SDN: Some basics
• NFV: Some basics
• NFV meets DCC and microservices
• NFV reference architecture
• *Templates and applications*
• NFV: Algorithmic issues
• NFV: Testing services on your laptop
Annotated templates

- Simple example: Annotate edges in a service by *multiplicity*:
  VNF instance can handle so many instances of the previous stage

- Better: State tuples of
  (offered load, used resources, quality, produced load)
  - Can be used by ManO framework to scale and place
  - ... and manage lifecycles (shutdown!)

- From where to get annotations?
  - Developers, testing while onboarding, monitor during operation
Templates, as they should be

- **Quantitative** annotations
  - From profiling
- Scaling options
- ...

![Diagram of Front-end to Back-end system with labels on edges: Firewall (1000), Load-balancer (200), Front-end, Backend (5000), Database.](image)
Services?

• “Ordinary” application boxes?

• “Network” functions?
Applications become more complex

- Application = Actual application + networking

Diagram:

- Application components/microservices
- Networking components (SDN or NFV)
- Quantitative templates
- Front-end
- Back-end
- Firewall
- Load-balancer
Developer knows best!

- **Application** = Orchestration + actual application + networking
  - UNIFY, SONATA, ...

![Diagram showing application components: Front-end, Back-end, Placement, Life-cycle, Load-balancer, Firewall, and Bespoke orchestration.](image)
Sonata Mano framework

- Sonata 5G-PPP [http://sonata-nfv.eu](http://sonata-nfv.eu)
- Claim to fame: Unparalleled flexibility
  - Functions and services can bring along their own management functionality
  - Executed by ManO framework
Reality check: Networking

- Networking: SDN controller applications
- Create network topology for application
- Developer support? Legacy?

Application Diagram:
- Front-end
- Back-end
- Load-balancer
- Firewall
- Placement
- Lifecycle

Bespoke networking!
High-level “architecture”
Content

- Motivation for NFV, SDN
- SDN: Some basics
- NFV: Some basics
- NFV meets DCC and microservices
- NFV reference architecture
- Templates and applications
- **NFV: Algorithmic issues**
- NFV: Testing services on your laptop
• Time permitting: Some discussions on algorithmic aspects in NFV orchestration
Content

• Motivation for NFV, SDN
• SDN: Some basics
• NFV: Some basics
• NFV meets DCC and microservices
• NFV reference architecture
• Templates and applications
• NFV: Algorithmic issues
• **NFV: Testing services on your laptop**
Background: OpenVswitch and mininet

- Initial question: How do you connect virtual machines’ network interfaces to the host’s network interface?
  - Answer: OpenVswitch (OVS)

- Follow-up question: supposing you have this ability – can’t we turn this into a nice experimental tool for SDN research?
  - Answer: Sure – Mininet

- Follow-up question: Can we extend that do an NFV research tool?
  - Answer: Yes – Containernet
Hands-on!

- How to experiment with NFV, orchestration, …?

- Typical requirements
  - Reproducible experiments
  - Easy to use
  - Captures relevant parts of real-life infrastructure
  - Cheap (!)

- Compare: Mininet for SDN

- Concrete requirements?
  - Run executables at multiple locations, patch them into chains
A Mininet-based testbed for NFV: Containernet/Medicine

  - https://github.com/containernet/containernet


https://goo.gl/uGjR3r
Extended version: Containernet under OSM control

- Paper: A Flexible Multi-PoP Infrastructure Emulator for Carrier-grade MANO Systems
  - Best Demo award at NetSoft, 2017
  - Ready-made VM to download (about 17 GB!)
- Video: [https://www.youtube.com/watch?v=pFL9wDNOBho](https://www.youtube.com/watch?v=pFL9wDNOBho)
A Flexible Multi-PoP Infrastructure Emulator for Carrier-grade MANO Systems

Manuel Peuster  Sevil Dräxler  Hadi Razzaghi Kouchaksaraei  Stéven Van Rossem  Wouter Tavernier  Holger Karl

Multi-PoP NFVI Emulator

- Mininet/Containernet-based network emulation
- Compute instances (VNFs) deployed as Docker containers
- Single SDN switch per PoP to abstract data-center-internal details
- Arbitrary user-defined multi-PoP topologies
- OpenStack-like northbound interfaces to control the emulated PoPs
- Built-in monitoring of VNFs
- Apache 2.0 license

Single-VM Sandbox Environment

Demonstration Scenario

1. Define topology and start emulation
2. Connect emulated PoPs as VIMs to OSM or SONATA
3. Define vCDN service using OpenStack HEAT templates, OSM or SONATA descriptors
4. On-board and instantiate vCDN service using OSM or SONATA MANO systems
5. Stream a video through the deployed service
6. Monitor service components using the emulator’s monitoring functionalities

Give it a try!

https://goo.gl/ordJDu

Who are we?

Sonata: Agile Service Development and Orchestration in 5G Virtualized Networks
http://sonata-nfv.eu

Contact person
Manuel Peuster  +49 5251 60-4341  manuel.peuster@upb.de

Computer Networks Group
Prof. Dr. Holger Karl  +49 5251 60-5375  holger.karl@upb.de

https://github.com/sonata-nfv
https://twitter.com/sonataNFV
http://www.upb.de/cs/cn/
Walkthrough steps

1. Start containers in a Mininet network
   1. containernet as such
2. Multi-pop version: Start multiple containers at a single location
   1. son-emu
   2. Interface similar to OpenStack Nova
3. Build chains among containers
4. Define a service package
   1. Using Sonata’s service definitions
5. Add a (highly simplified) Sonata ManO
6. Upload a service
7. Test the running service
Step 0: Start the virtual machine

  - about 5 GB

- Or use the provided instances

- Login
  - Username and password: sonata

- Open terminal
Containers in Mininet?

- Mininet host: Basically just a process
  - Limits what you can do as VNF experiments

- Expectation?

- Hands-on!
Start containers in Mininet

- Similar to adding Mininet hosts to topology
- See ~/demo/topologies/containernet_example1.py

```python
# add containers to topology
d1 = net.addDocker('d1', ip='10.0.0.251',
                    dimage="ubuntu:trusty")
d2 = net.addDocker('d2', ip='10.0.0.252',
                    dimage="ubuntu:trusty")

# connect containers to switches
net.addLink(d1, s1)
net.addLink(d2, s2)
```

- Start this file:
  sudo python ~/demo/topologies/containernet_example1.py
Use containers in Mininet/containernet

- Play around with topology:

```
containernet> dump
containernet> d1 ifconfig
containernet> d2 ifconfig
containernet> d1 ping -c3 d2
containernet> d1 ping -c3 h1
containernet> h1 ls
containernet> d1 ls
```

- Play with the containers (second terminal!!):

```
docker ps
docker attach mn.d2
<enter>
<enter>
root@d2> top
```
Why Multi-PoP experiments?

• Represent and simplify data centres
  • Inside data centre: Big-switch abstractions

• Provide control over data centres
  • Akin to OpenStack Nova, e.g.

• Expectation?

• Hands-on: Get it to work!
Chaining?

• So far: VNFs running individually, not connection yet
  • No automatic learning switch in place
• Set up chaining: Put entries into flow tables

• Expectation?

• Hands-on!
  • Try to hook up the VNFs
From individual, chained containers to packages

- Actual deployment in an NFV context: service package
  - Describes: which functions, how connected, how scaled, ...
  - Unit of deployment, passed to a MANO framework

- Expectation?

- Hands-on!
  - Try it: Start a SONATA gatekeeper in the emulator, create a package, upload and startup package in gatekeeper
Thoroughly missing so far…

- Agreed-upon benchmark scenarios!
- Load generators!
  - At (at least) two abstraction levels
    - Generating new service request with service duration
    - Generating load for individual services
- Collaborators welcome!
Conclusion

• NFV still an active research and innovation area
  • Lot’s of conceptual/architectural, algorithmic/optimization, systems work still necessary

• Ability to test in practice so far limited
  • Some steps to overcome that: Containernet/Medicine
  • Down the road: For developers and operators?

• Missing: Benchmarks!
Open source activity: OpNFV

- Initiative for an open-source NFVI/NFVI-Manager (including Nf-Vi interface)
- Public release available

https://www.opnfv.org/software/technical-overview
OpenSourceMano (OSM)

- Open-source Management / Orchestration framework
  - https://osm.etsi.org
  - Aligned with ETSI reference architecture
  - UPB: member!
OSM high-level architecture

https://osm.etsi.org/wikipub/images/d/d4/OSM%2816%29000018r1_MWC16_architecture_overview.pdf
OSM high-level architecture

Riftware (NSO)

Juju Server (CM)

OpenMANO (RO)

NSO: Network Service Orchestrator
CM: Configuration Manager
RO: Resource Orchestrator

https://osm.etsi.org/wikipub/images/d/d4/OSM%2816%29000018r1_MWC16_architecture_overview.pdf
References, further reading

  - With various important white papers on reference architecture, use cases, terminology (see “Specifications” tab on that web page)
- Vendor-specific white papers:
- Service chaining surveys
  - http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6702549
- Papers for this chapter: Two mendeley groups
  - https://www.mendeley.com/groups/6764011/distributedcloudcomputing/
  - https://www.mendeley.com/groups/6763981/networkfunctionvirtualization/