TECHNISCHE UNIVERSITÄT DRESDEN

ComNets
Deutsche Telekom Chair of Communication Networks

Patrick Ziegler

Design and Evaluation of a Network Softwarization Platform for Edge and In-Network Computing

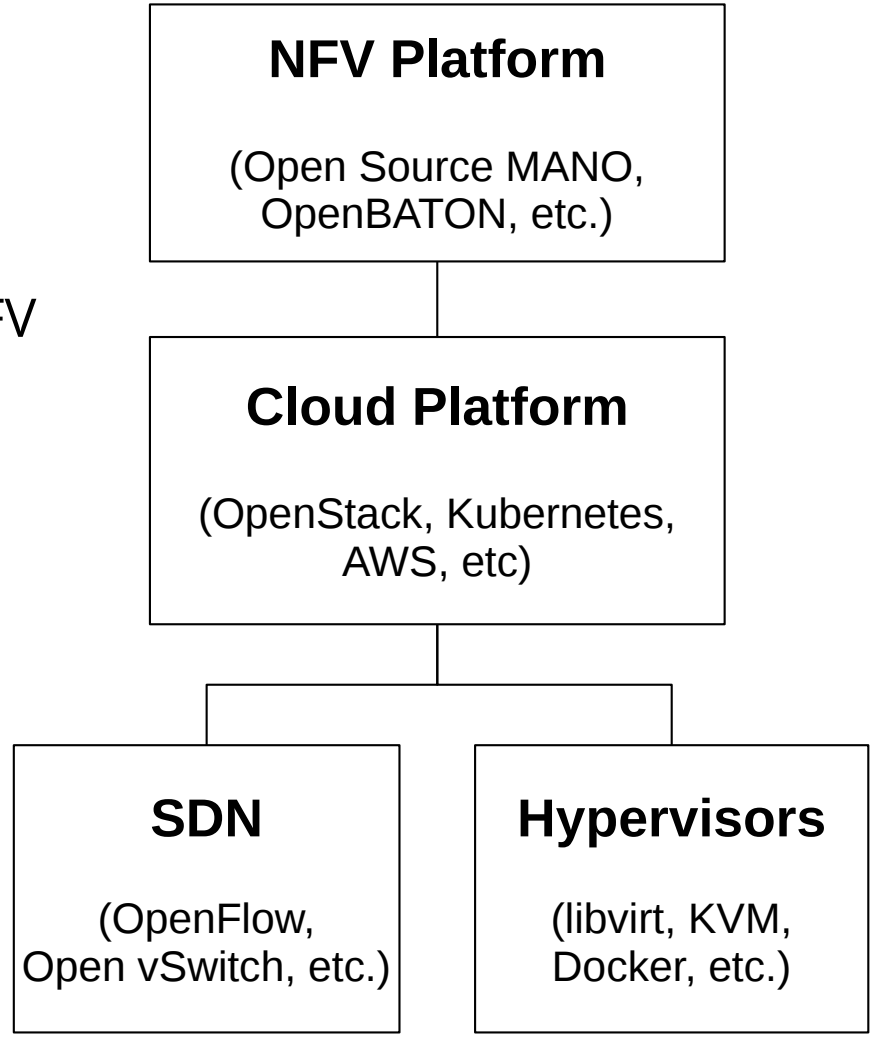
Diploma Thesis Defense, 24.10.2019
Deutsche Telekom Chair for Communication Networks

Motivation

- Motivation
  - **Dynamically deploy functions at the network edge**
  - Enable new use cases for IoT, Mobile Edge Cloud etc.
  - Offloading computation tasks (also for cooperation between clients)
- This is supported by **Network Function Virtualization (NFV)**
- State of the Art NFV platforms are
  - Based on **complex** cloud infrastructure platforms like OpenStack
  - Mainly focused on **business related problems**
  - Restricted to **fixed deployments**

Problem statement

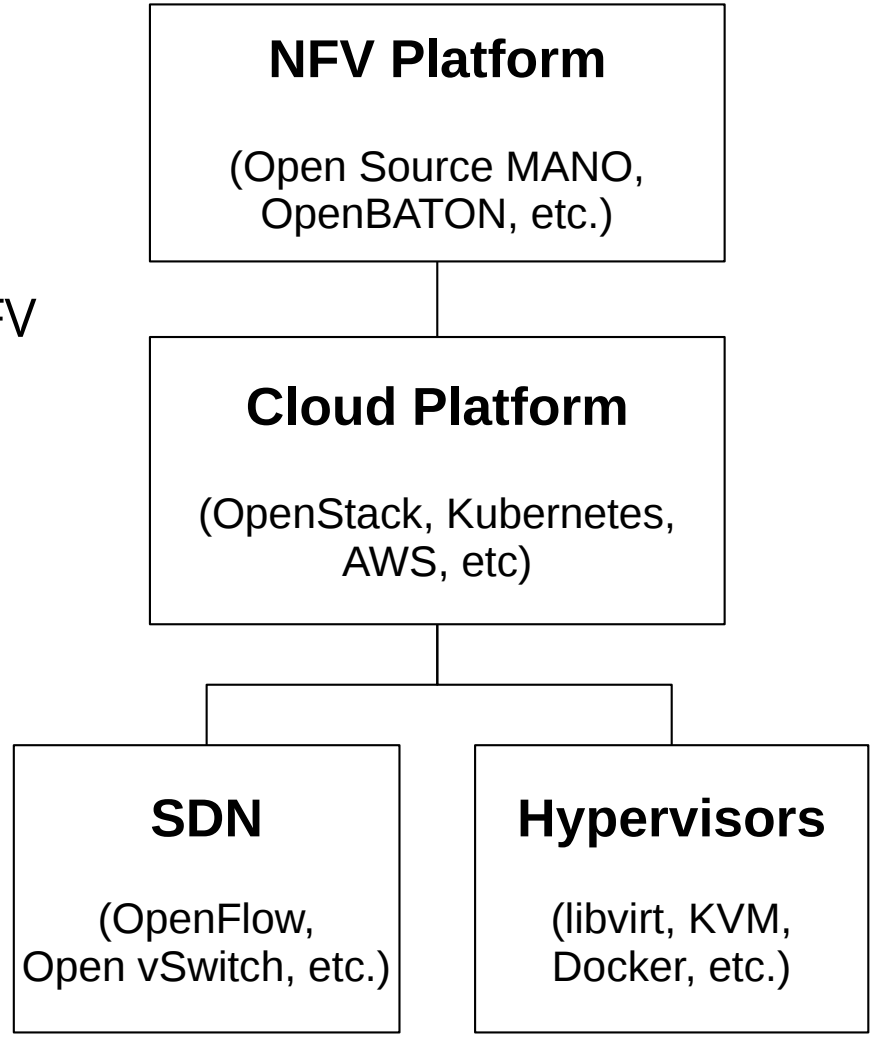
- Cloud platforms provide **abstracted view** of infrastructure
  - This **hides network related information** from NFV platforms
- Edge deployments should provide minimal latency
  - NFV needs **latency and topology awareness**
- Manual optimization is too complex
  - NFV should provide **autonomous optimization and deployment**



Problem statement

- Cloud platforms provide **abstracted view** of infrastructure
  - This **hides network related information** from NFV platforms
- Edge deployments should provide minimal latency
  - NFV needs **latency and topology awareness**
- Manual optimization is too complex
  - NFV should provide **autonomous optimization and deployment**

Goals of this work

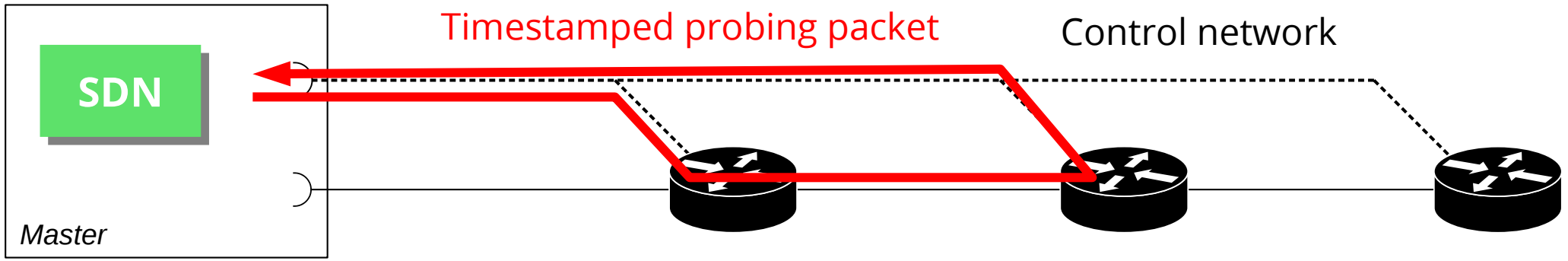


Scope of this work

- Design and implement a lightweight NFV platform from scratch
  - Serves as **proof of concept** for new ideas
  - May arise **new insights** on NFV platforms („Wissen kommt von machen“)
  - „**Instant NFV on your laptop**“ → good for research
- Three problem fields identified
  - **Monitoring** (Latency, Topology, Resources)
  - **Management and Orchestration**
  - **Placement** (Algorithms)
- These combined constitute a **new adaptive approach to NFV**

Monitoring

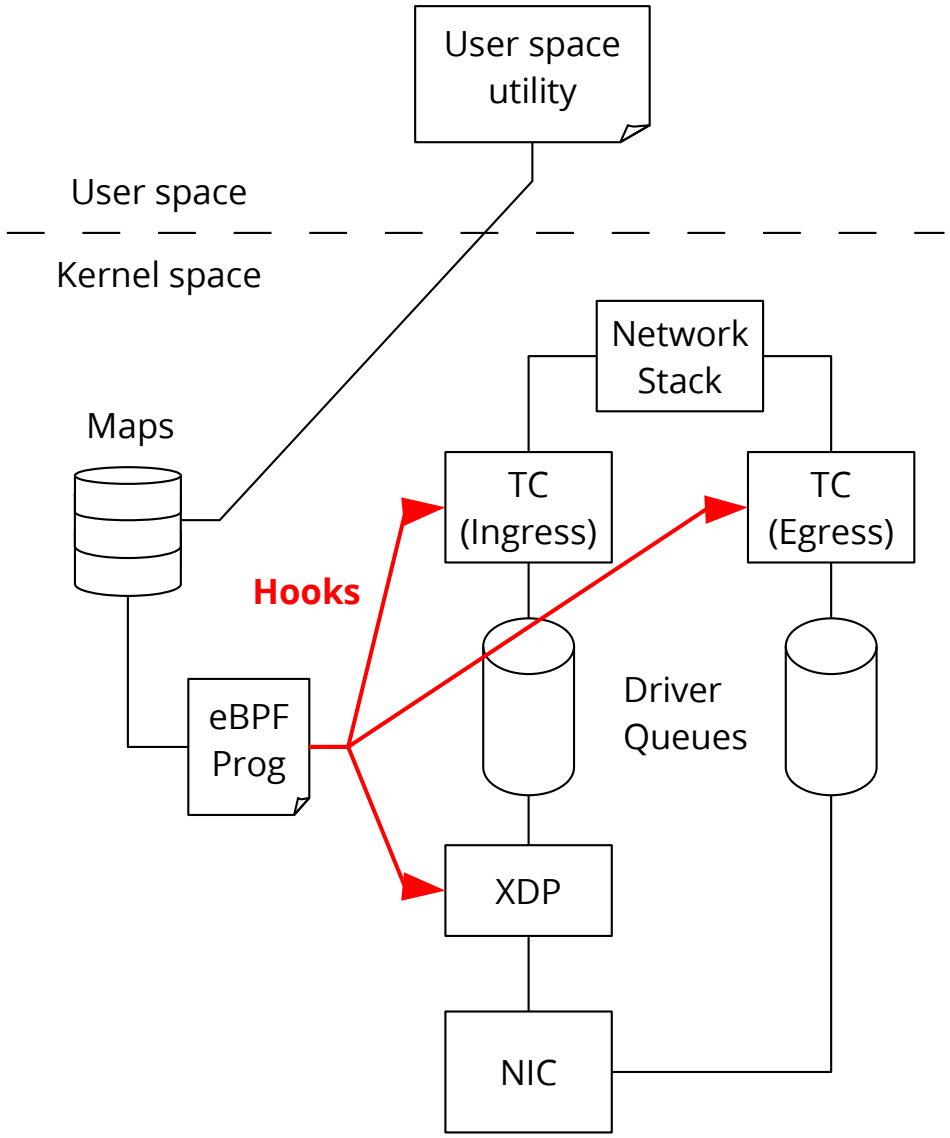
Problem setting



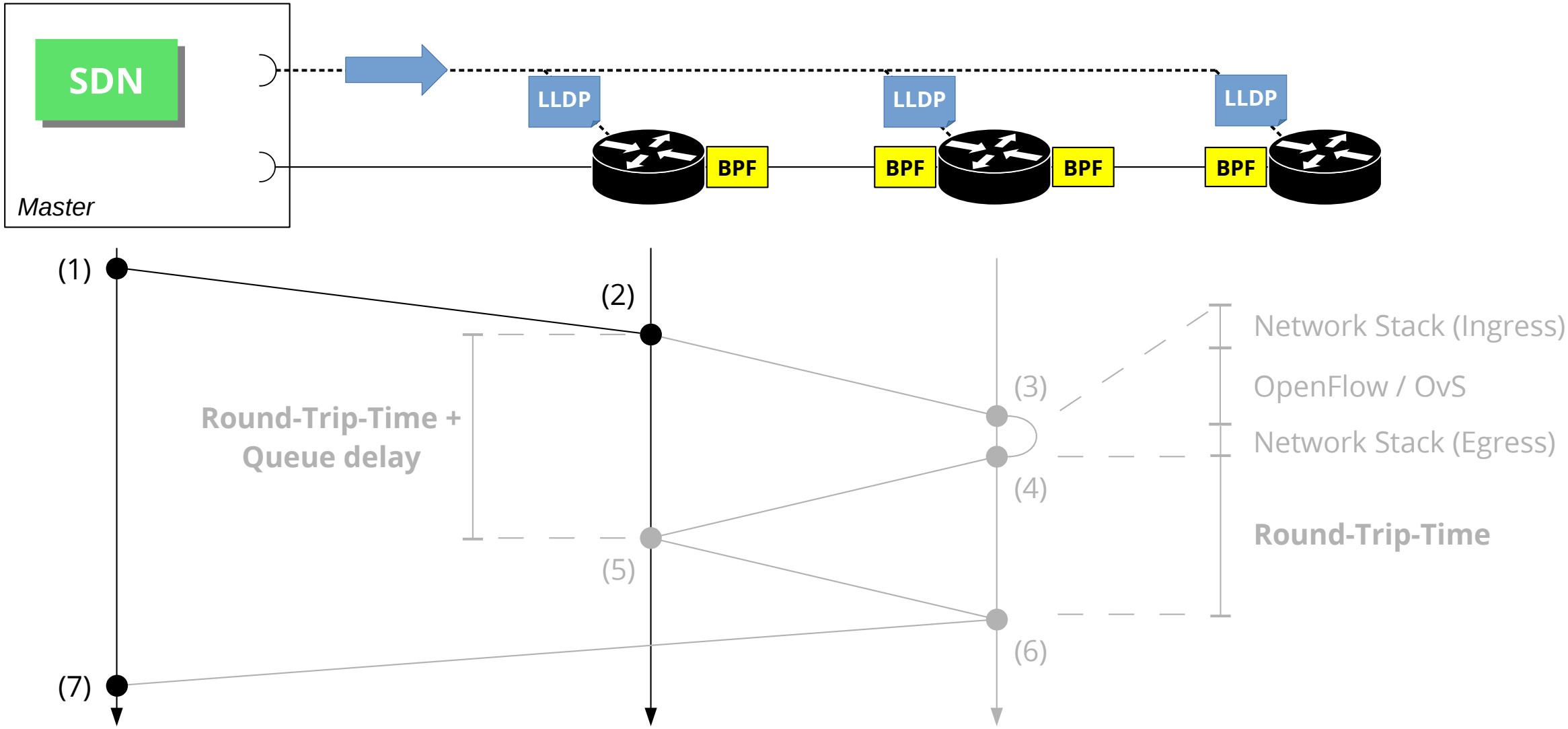
- How to measure link latency in OpenFlow networks?
- OpenFlow only supports **predefined set of operations**
  - Not (easily) possible to implement new functionality
- State of the art monitoring solutions use **timestamped probing packets** injected by SDN controller
  - Compensation of **delay introduced in control network** necessary
  - How to handle links between switches and **compute nodes?**

Background: eBPF / XDP

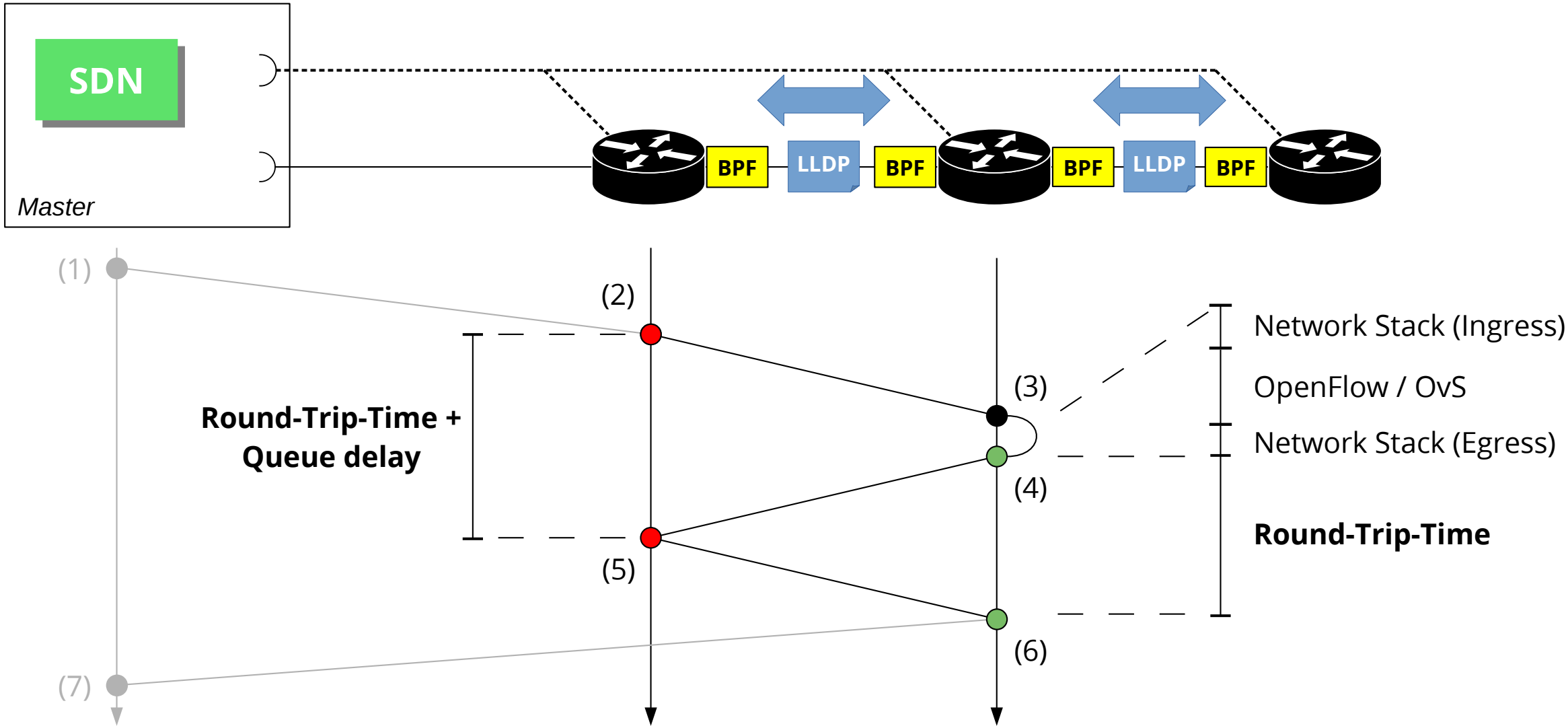
- Extended Berkeley Packet Filter (eBPF)
- A **virtual machine** inside the Linux kernel
  - Resembling the Java Virtual Machine
- eBPF Programs
  - are **attached to provided hooks** and executed when events occur
  - have to **pass a verifier** before being allowed to be attached to a hook
  - can **manipulate in- and outgoing packets**
  - decide over packets further processing (**PASS, DROP, REDIRECT**)



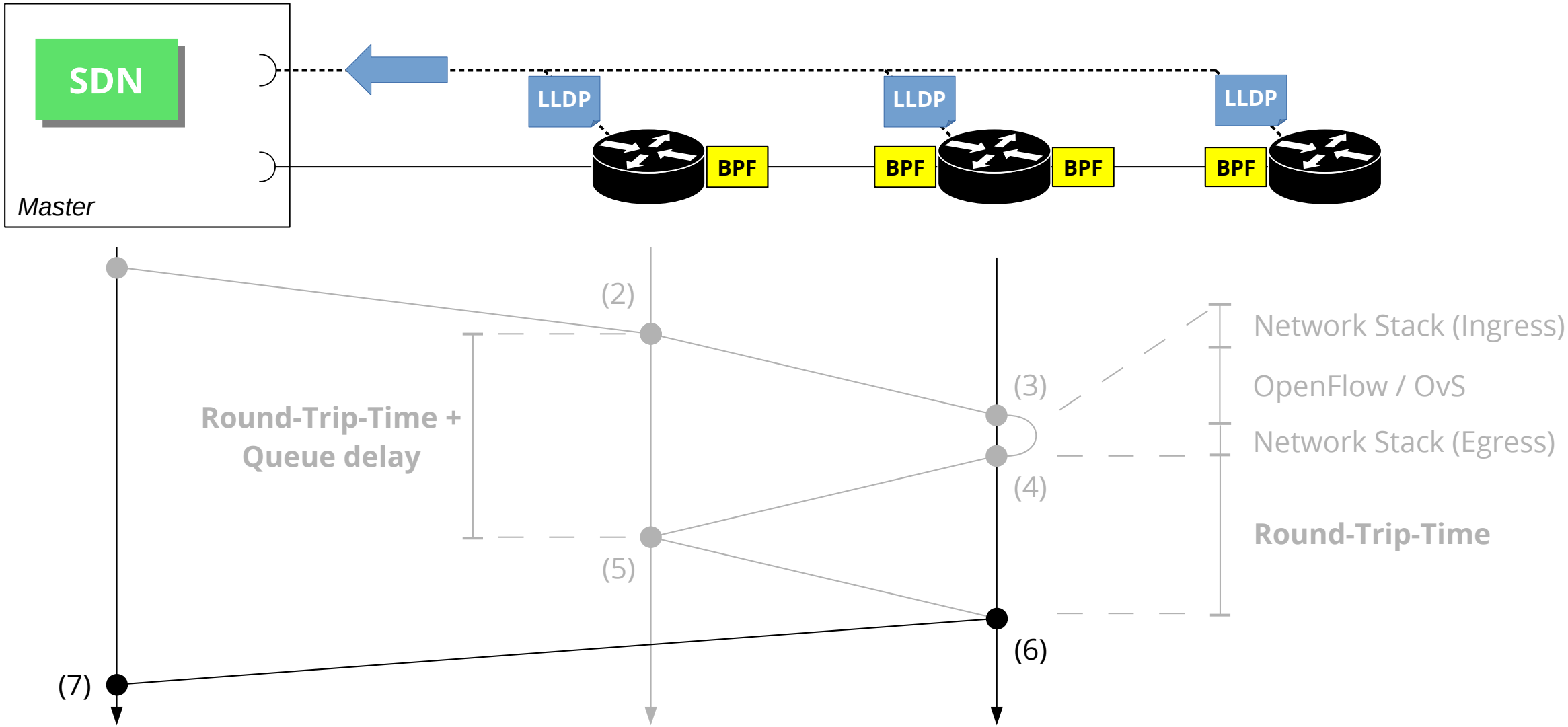
1. Controller sends probing packets to registered switches



2. Switches attempt to do LLDP Looping with all adjacent nodes

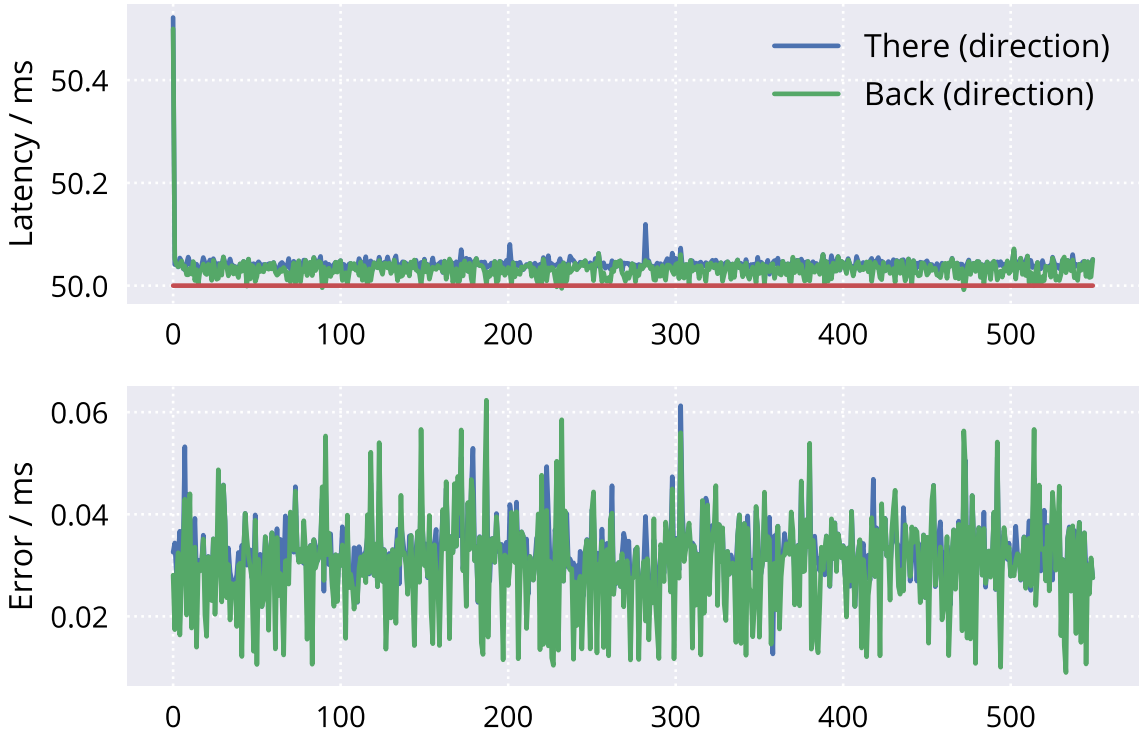


3. Adjacent nodes send probing results back to Controller



Review: LLDP Monitoring

- XDP allows to return packet **without incurring queueing delays**
  - this enables **seperation of individual delays**
- **No clock synchronization** needed
- Transmission delay is assumed to be symmetric



Negligible

$$Latency = T_{Transmission} + T_{Propagation} + T_{Queueing} = RTTQ - RTT/2$$

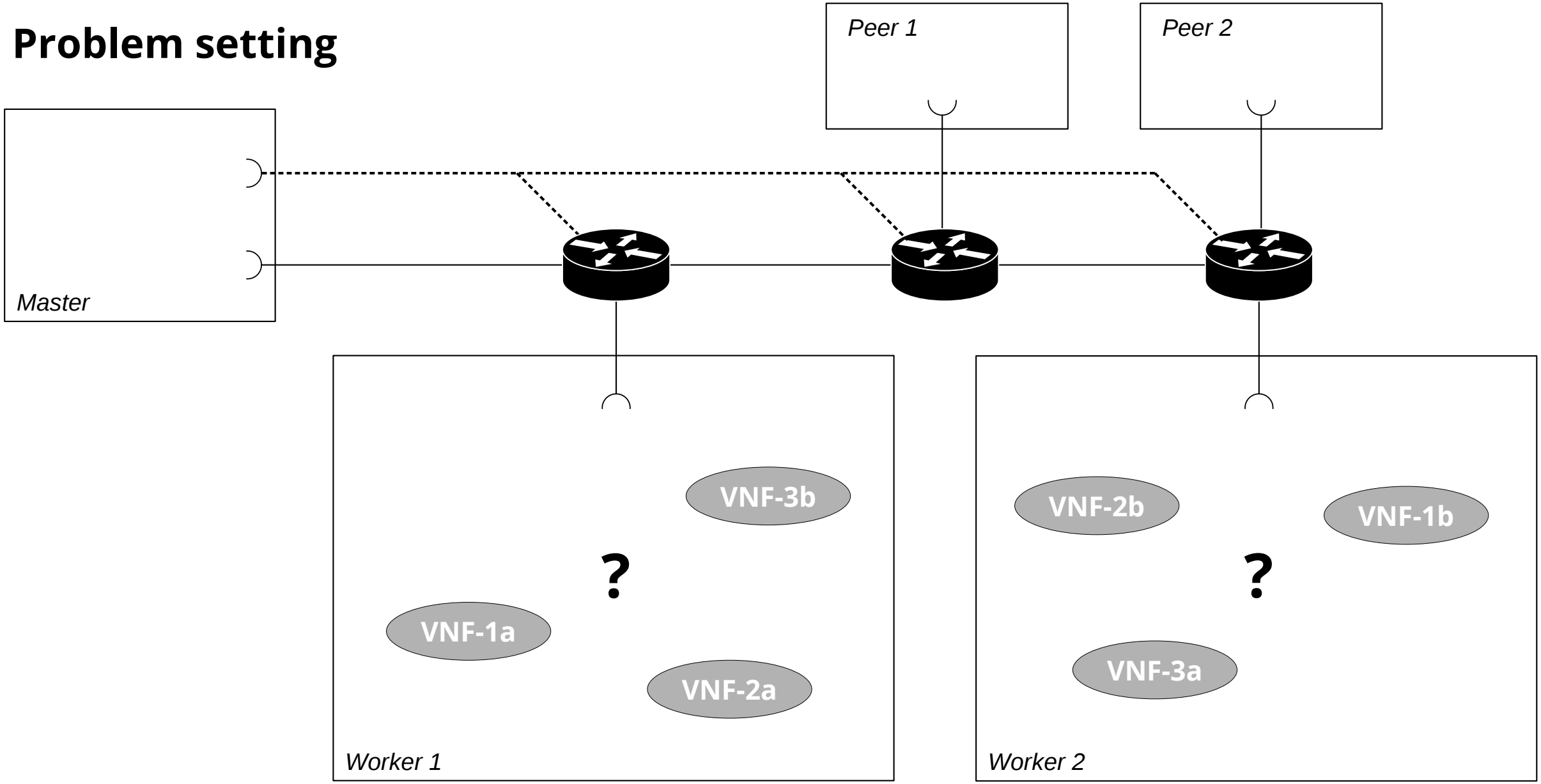
Symmetric

Precision on empty link

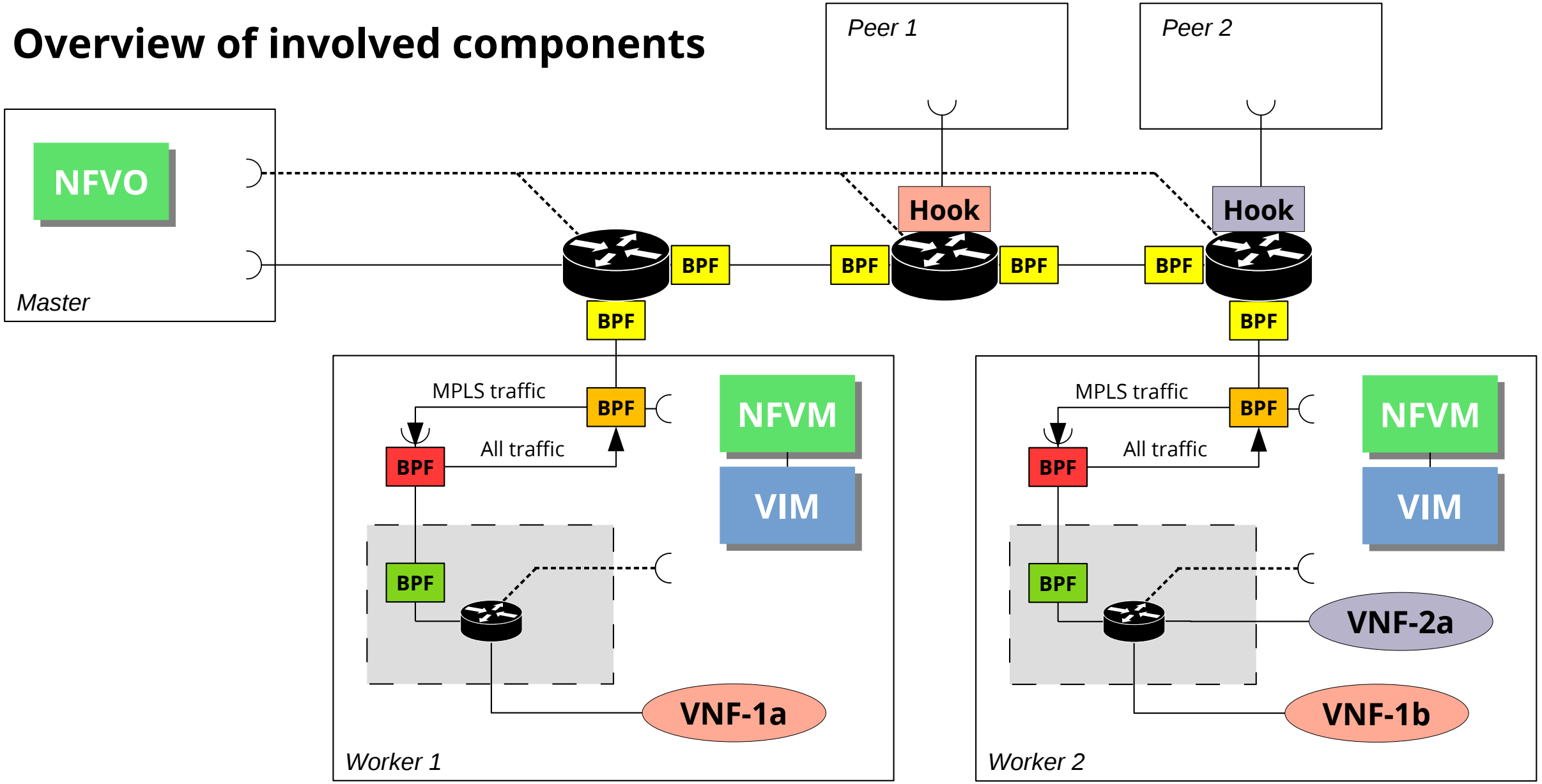
Reference	50 ms
Mean value	50.030141 ms
Std. dev.	0.009187 ms

Management and Orchestration

Problem setting



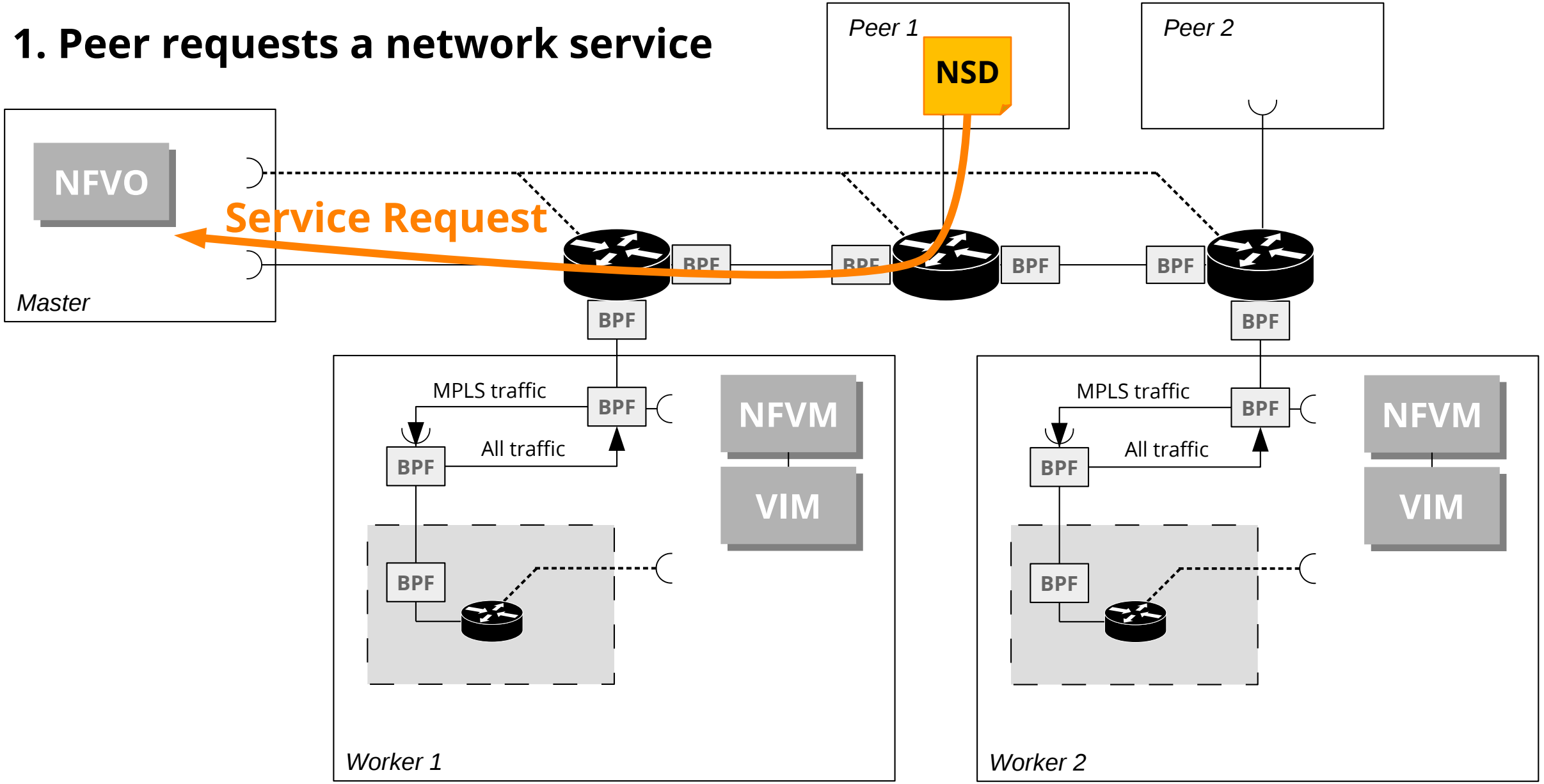
Overview of involved components



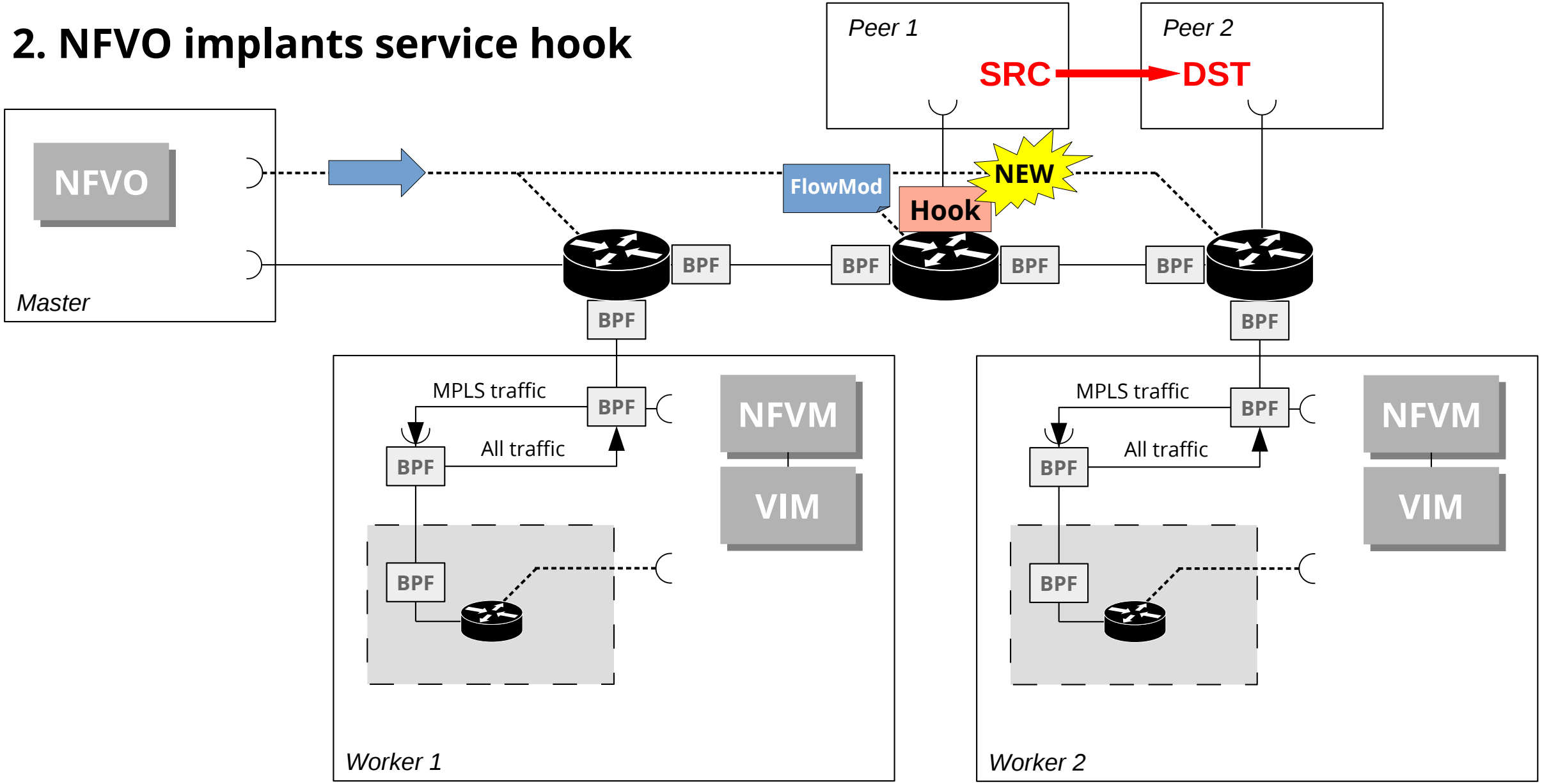
Background: The Serverless Paradigm

- Core idea
  - Transfer the **responsibility for** infrastructure **provision and deployment** from the administrator **to the platform**
- Deployment units are *functions*
  - typically **stateless**
  - running inside containers or virtual machines
  - functions are **handling full HTTP requests** in conventional platforms
- Provision **as the need arises**
  - based on incoming requests
  - with **automatic scaling** under consideration of resource usage (CPU)
- Concept most effective when **request rate shows high volatility**

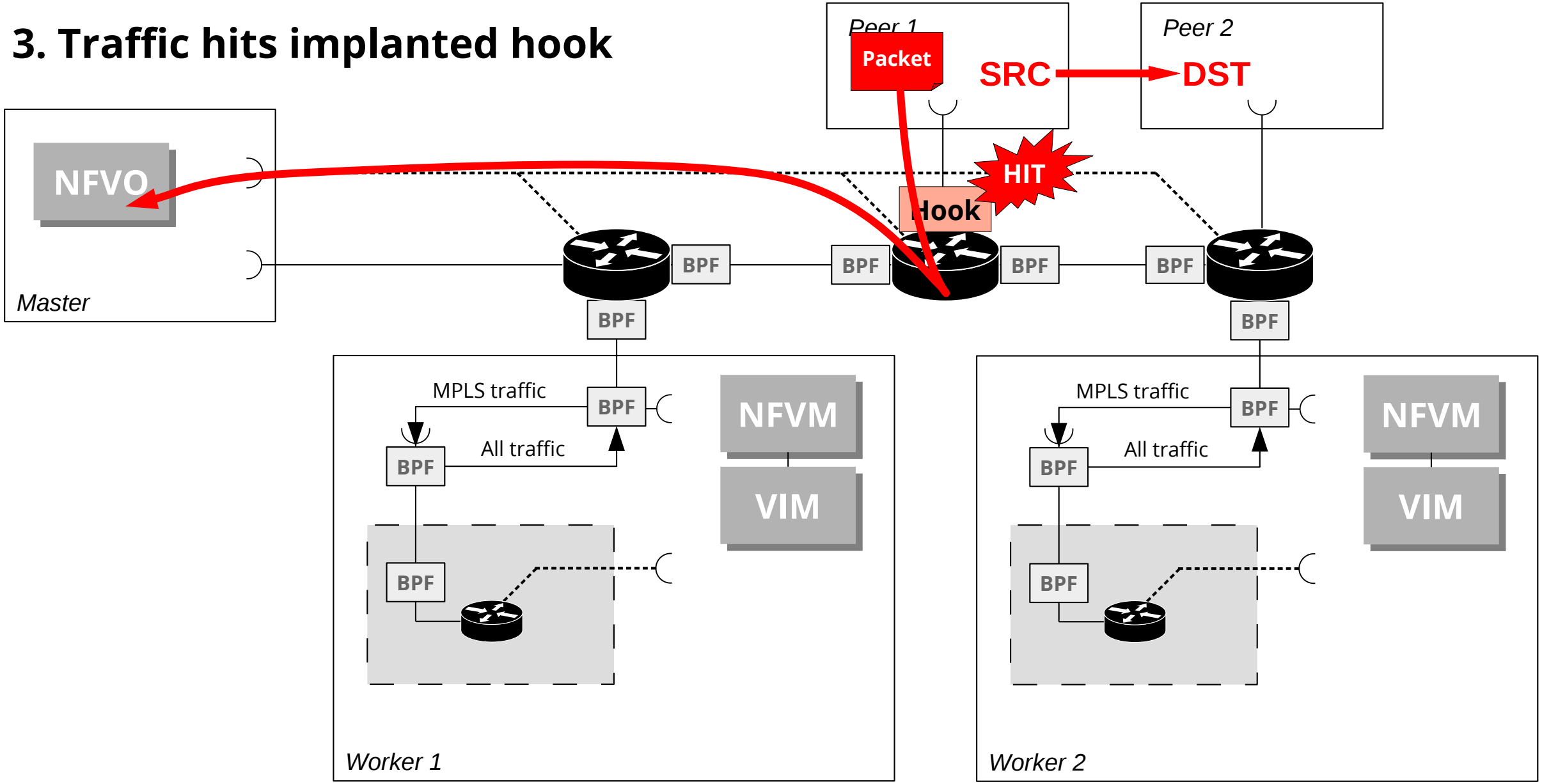
1. Peer requests a network service



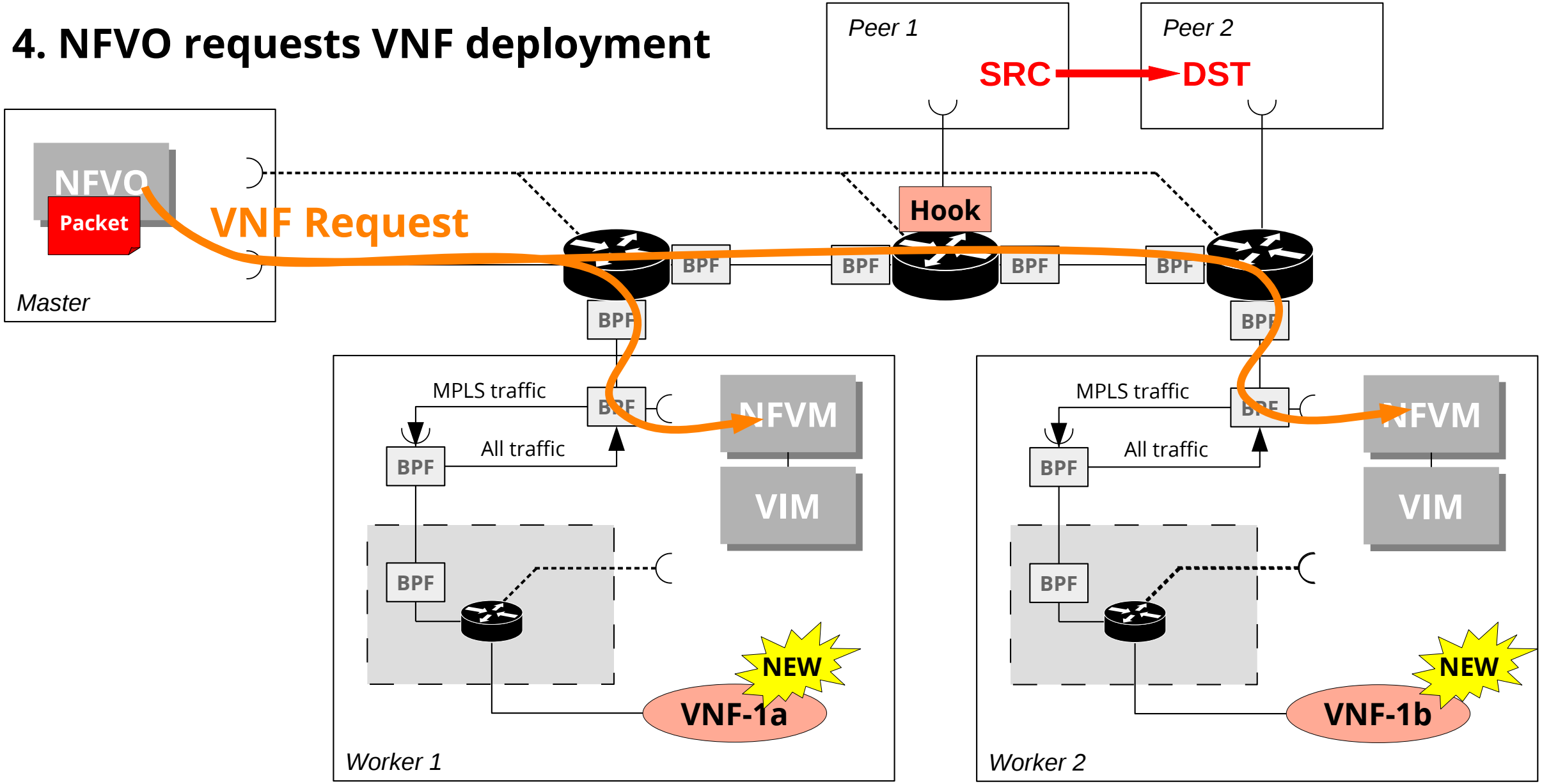
2. NFVO implants service hook



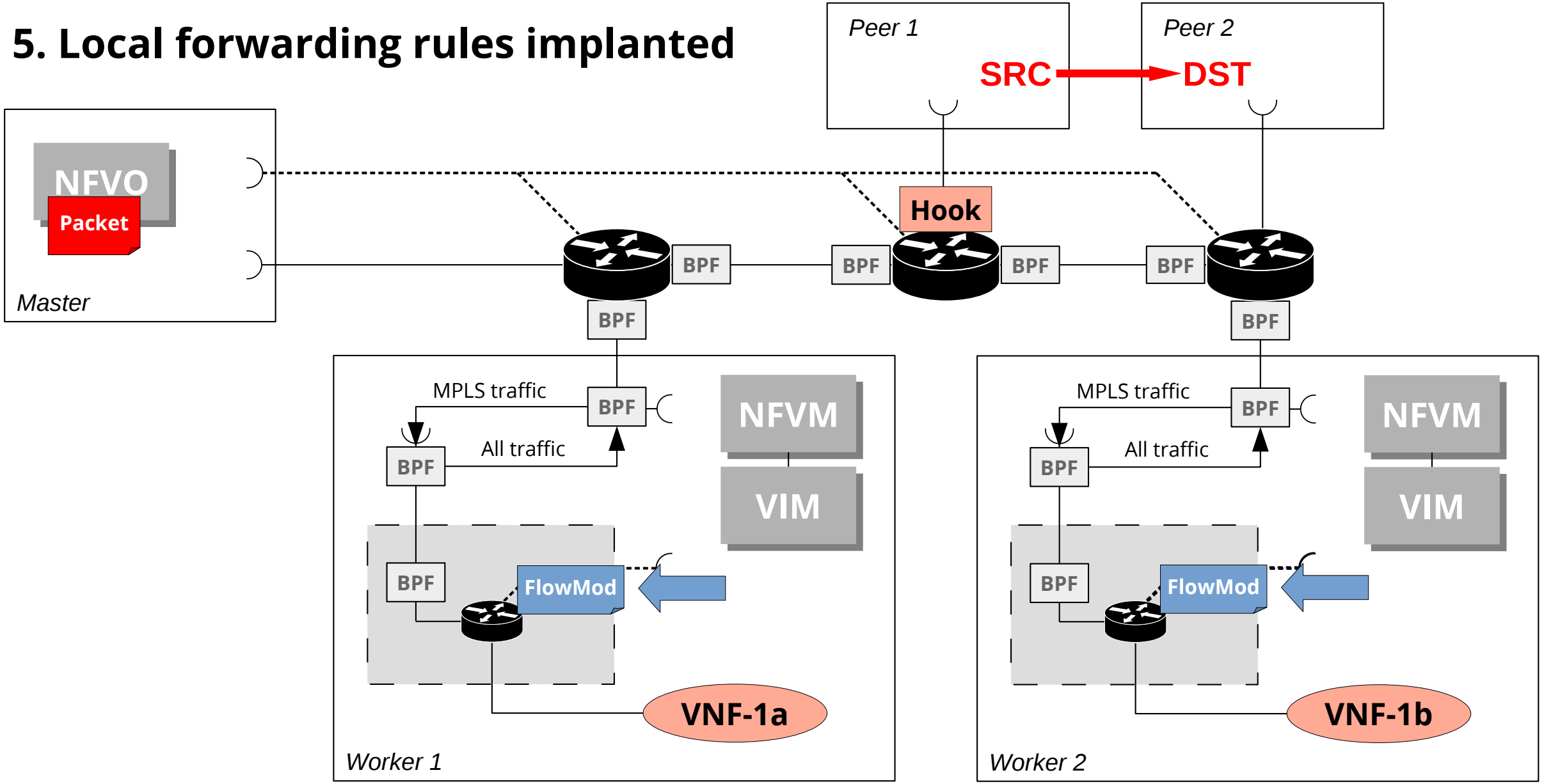
3. Traffic hits implanted hook



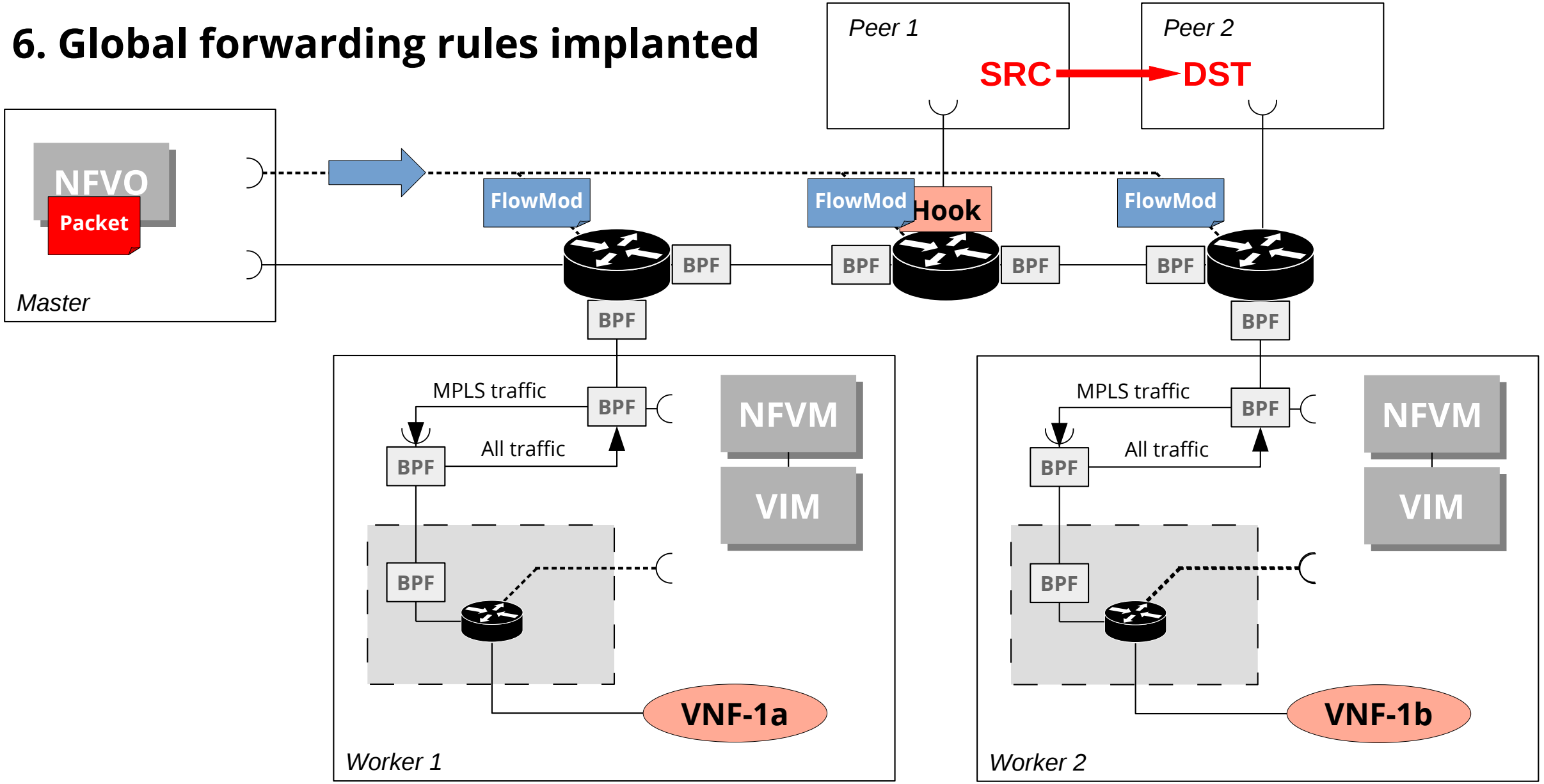
4. NFVO requests VNF deployment



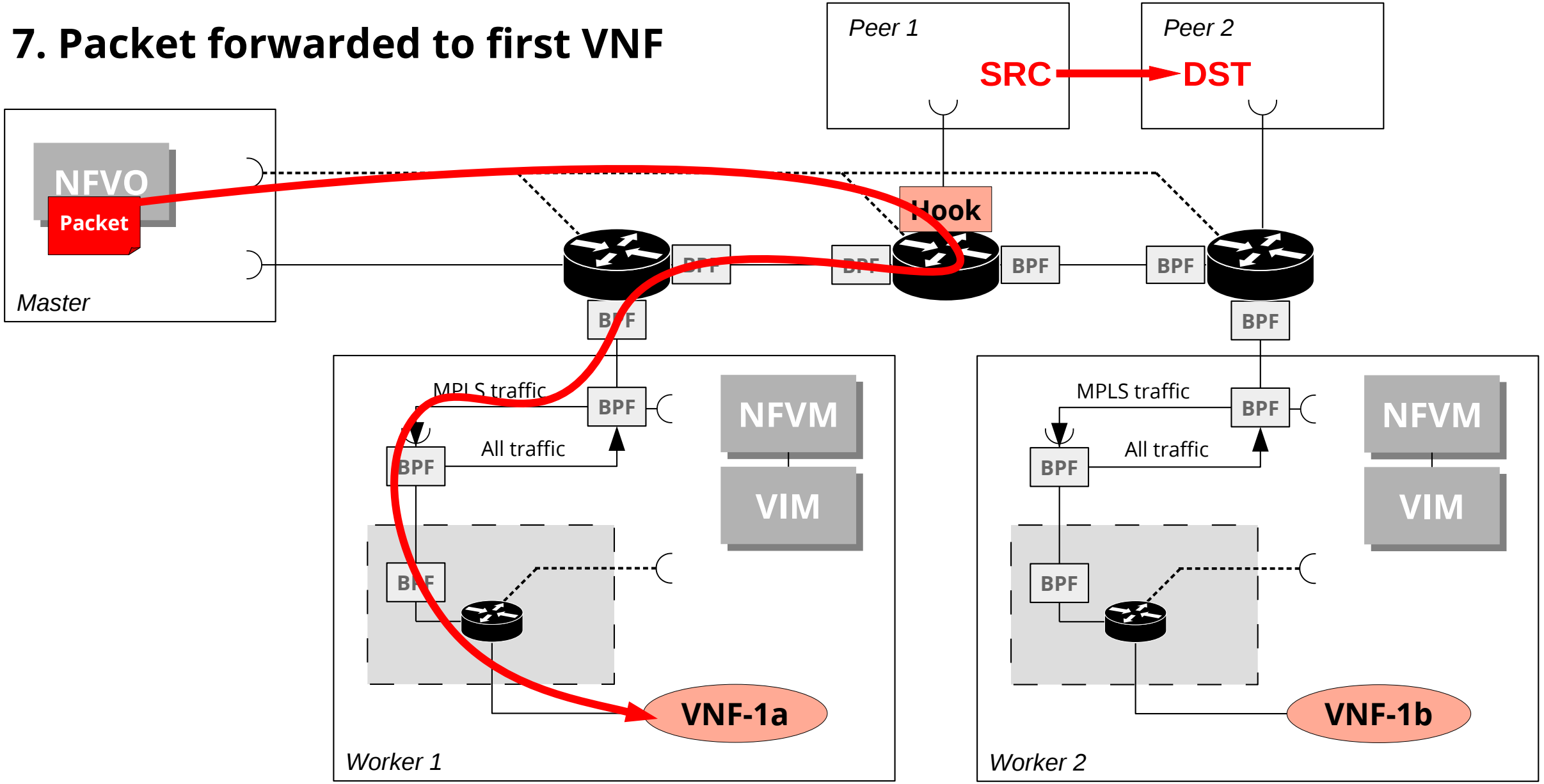
5. Local forwarding rules implanted



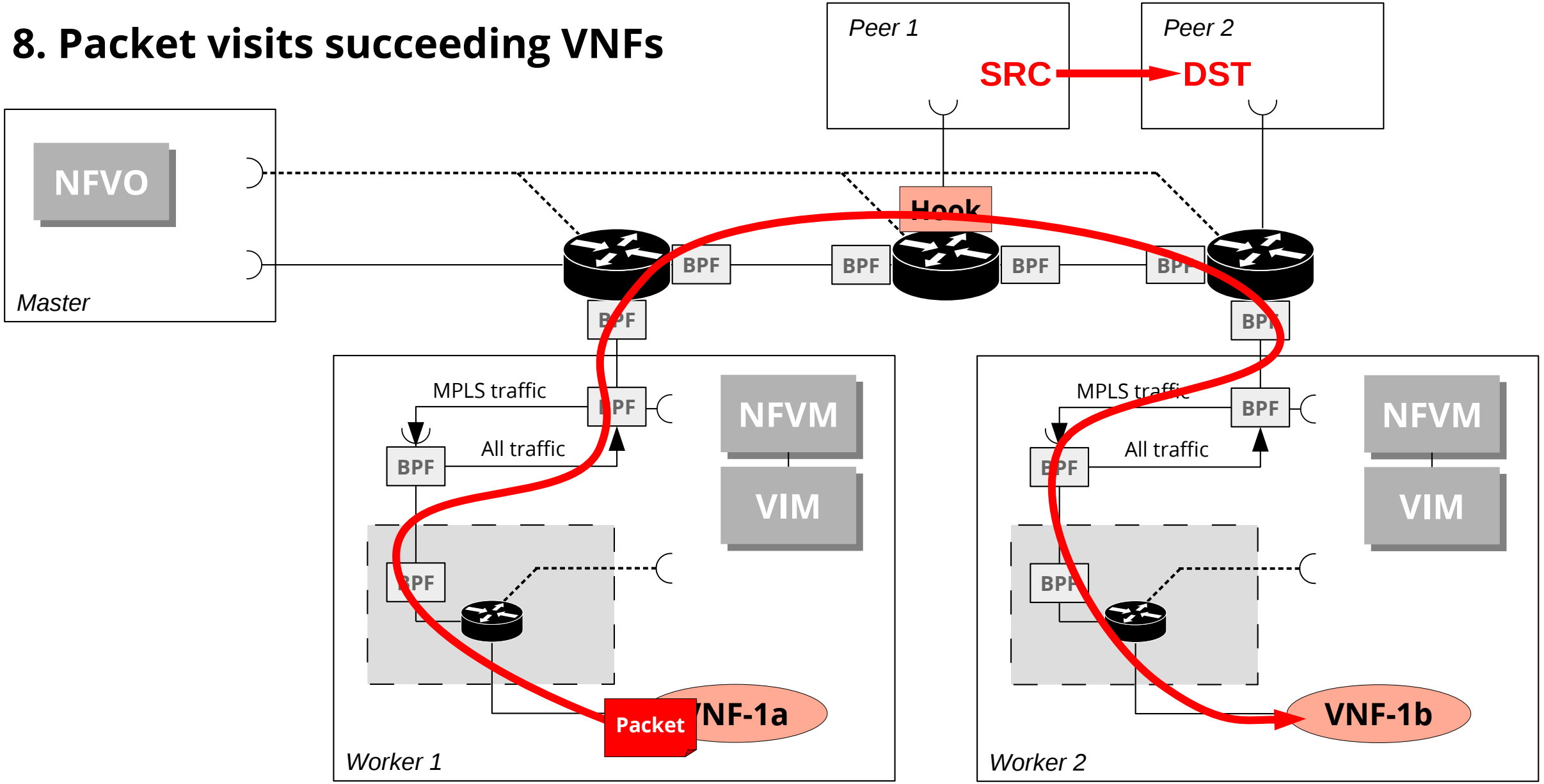
6. Global forwarding rules implanted



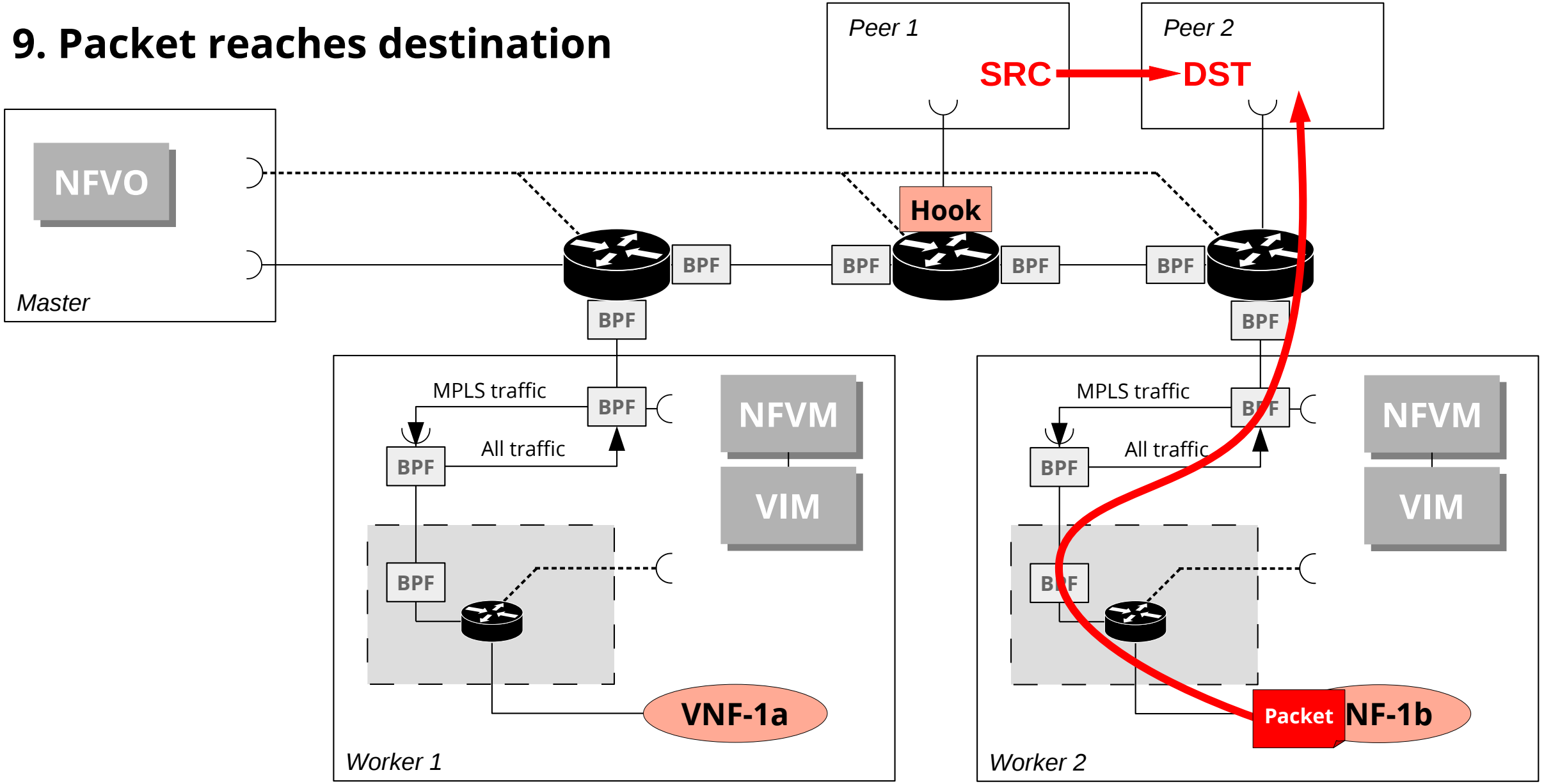
7. Packet forwarded to first VNF



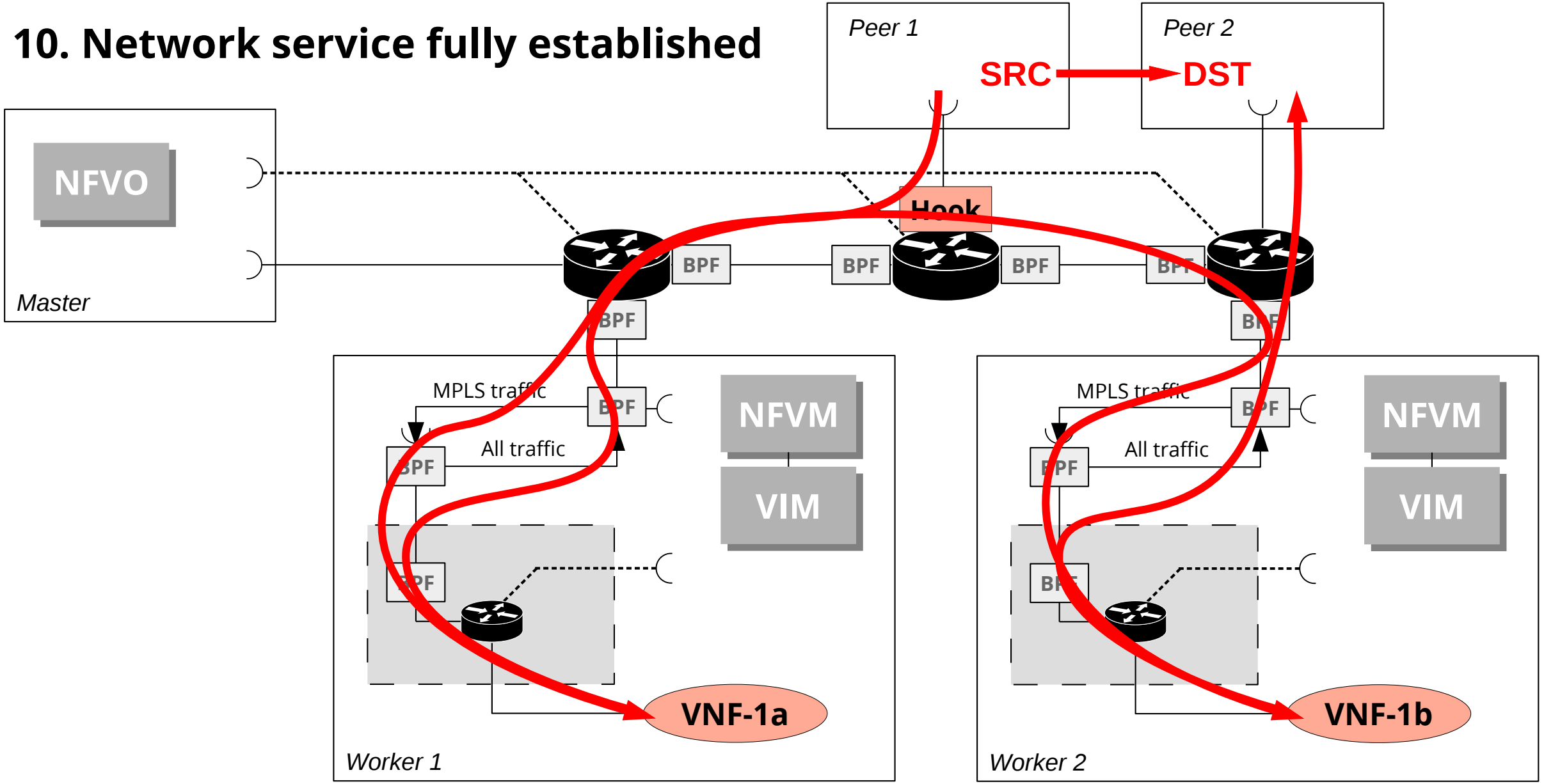
8. Packet visits succeeding VNFs



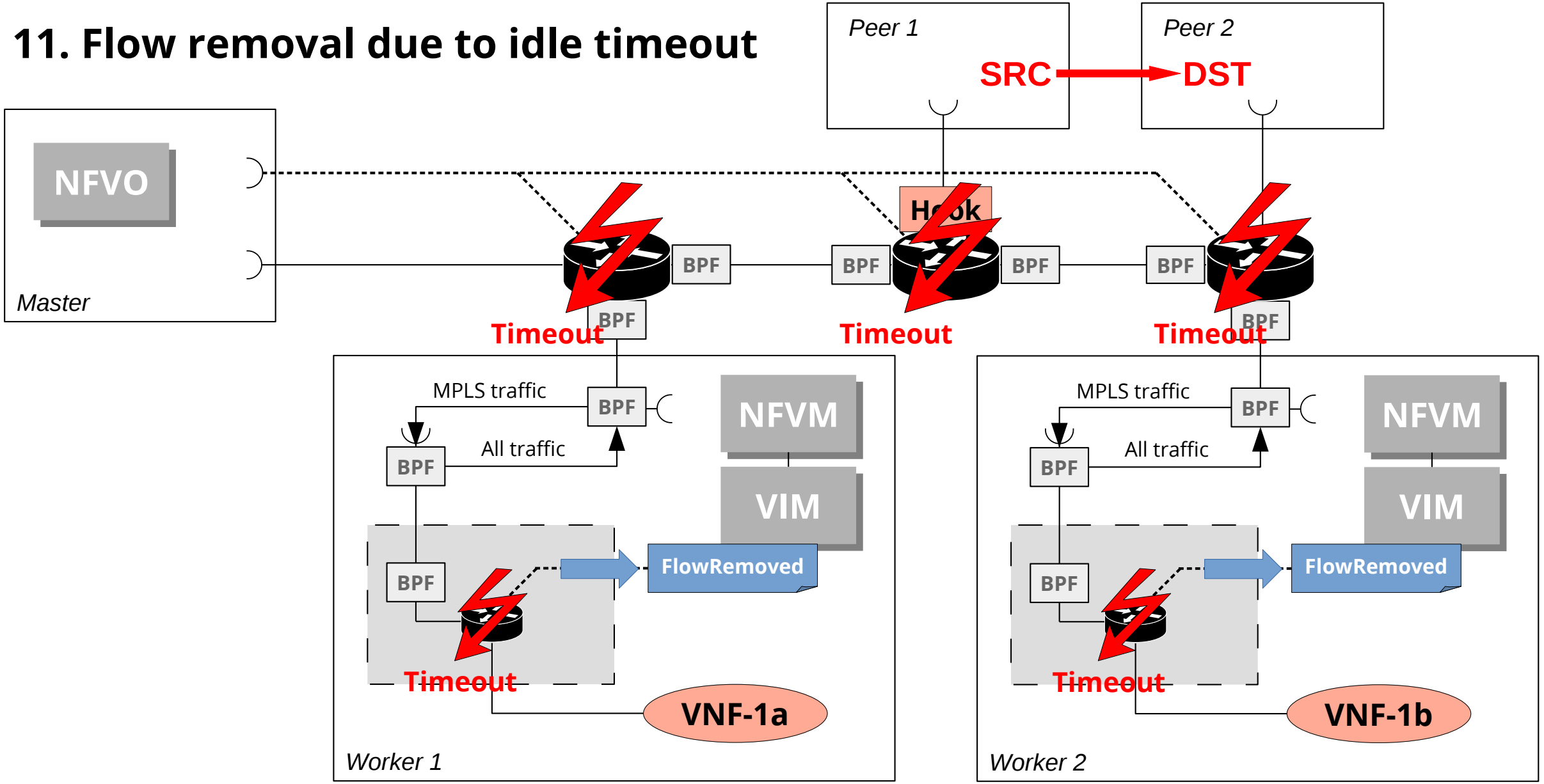
9. Packet reaches destination



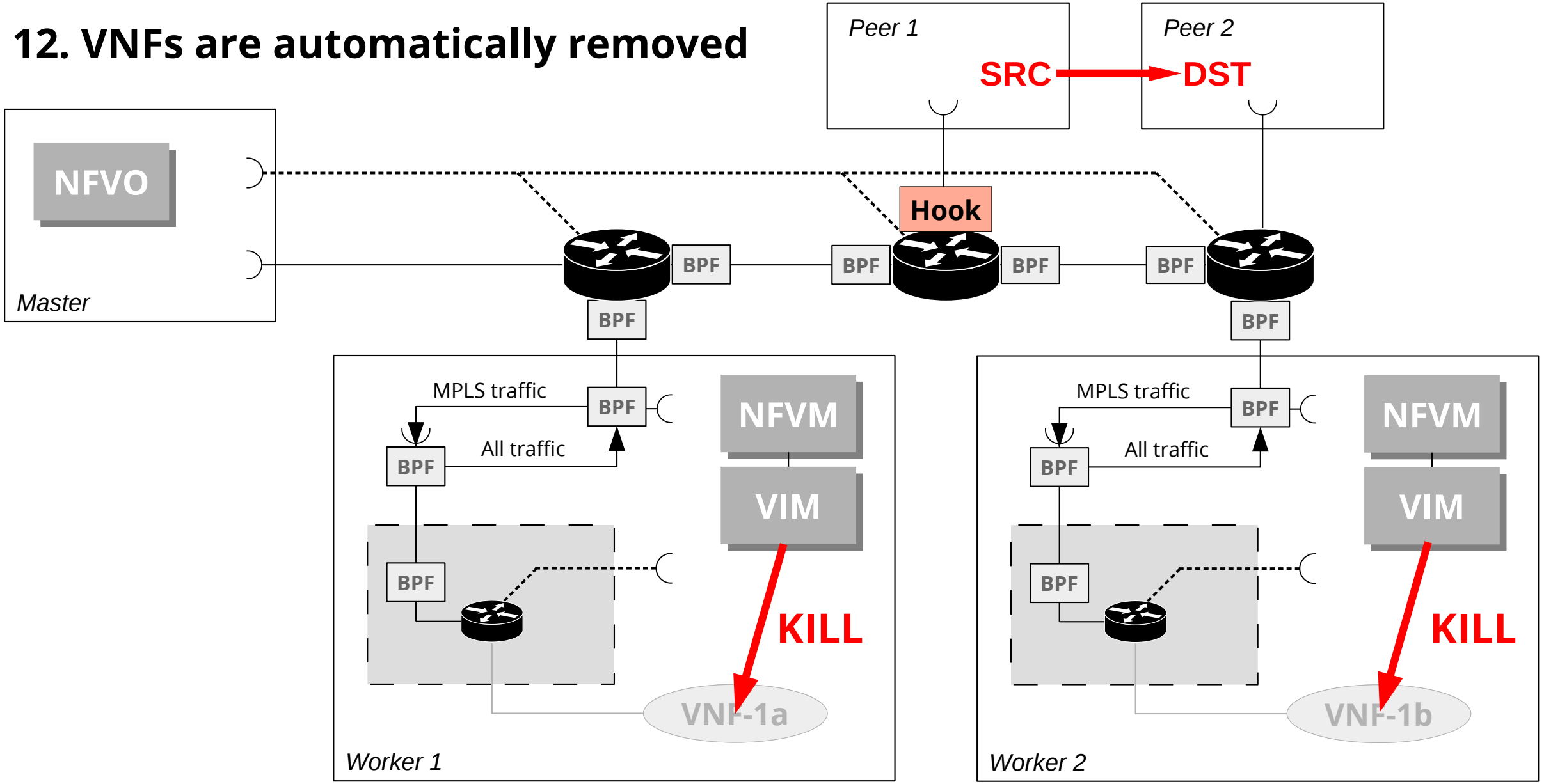
10. Network service fully established



11. Flow removal due to idle timeout



12. VNFs are automatically removed



Review: Serverless MANO

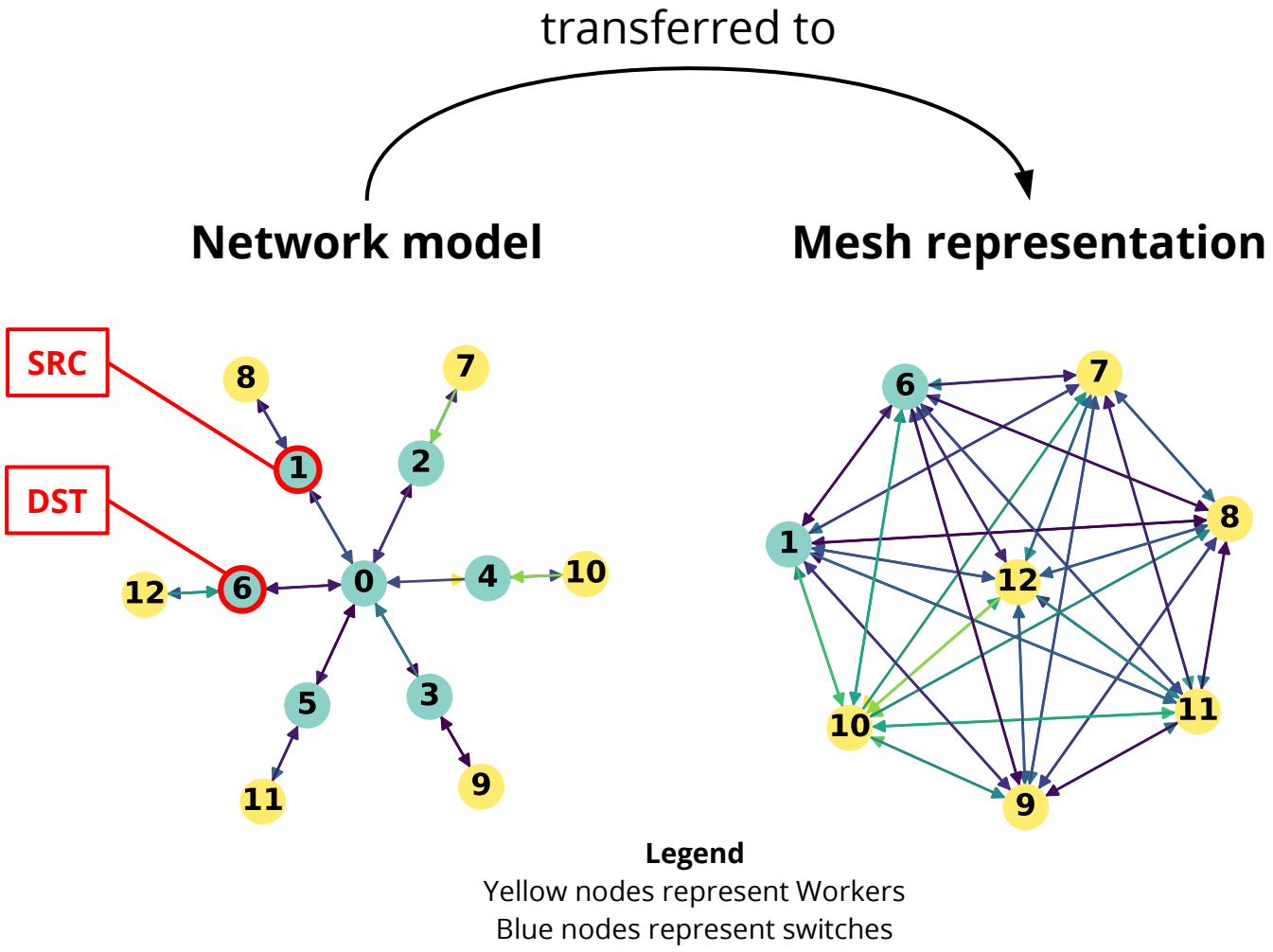
- The developed **serverless network service lifecycle**
  - Is based on OpenFlow and label based switching
  - Reduces the amount of **manual administration to zero**
  - Keeps **resource usage as low as possible**
- Startup delay of network services is typically > 100 ms
  - Drawback that has to be considered when designing services

⟹ Short demo at the end of the presentation

Placement

Hasty Traveller Algorithm: Concept

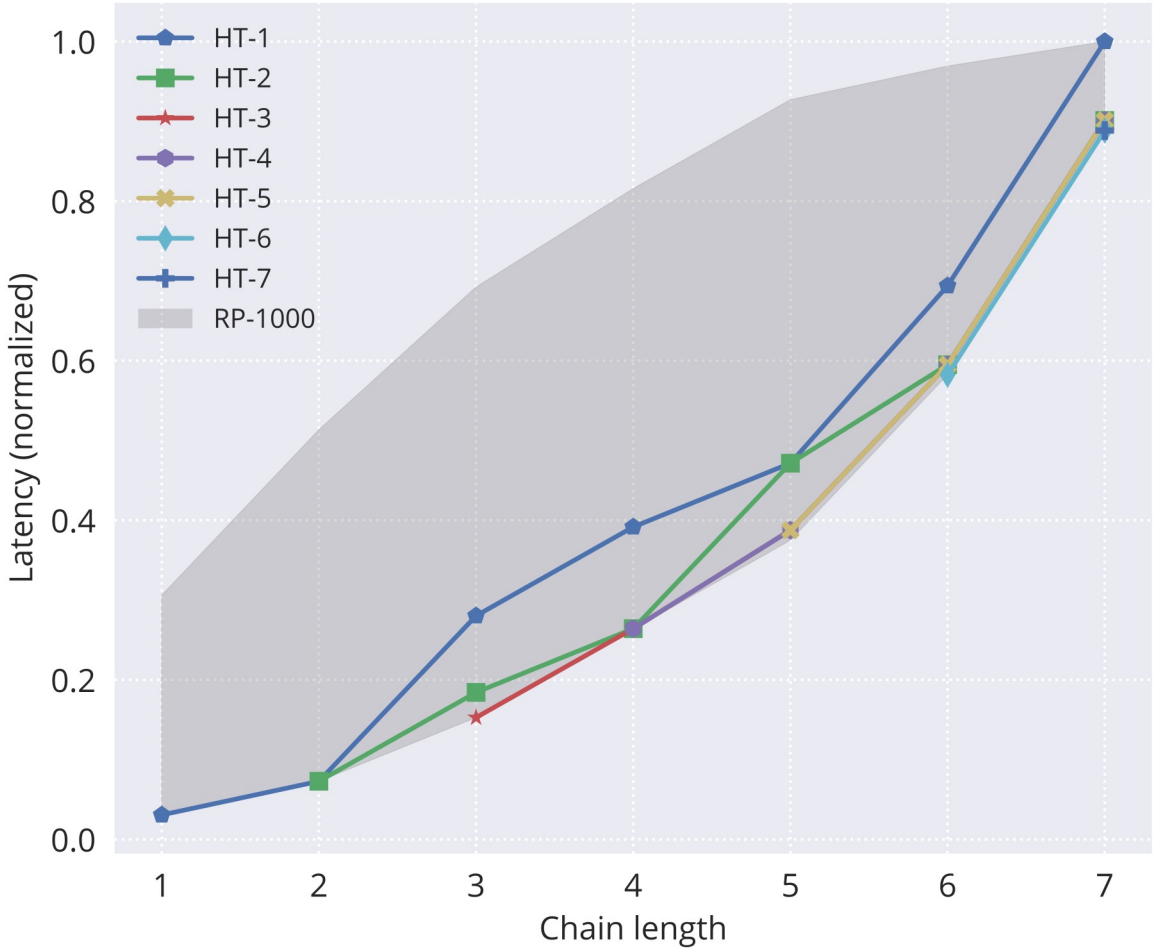
- Design Goal
  - find shortest path **visiting a defined number of workers** along the way (chain length)
  - Travelling Salesman problem
- **Recursive search** in mesh representation
  - at each node, visit adjacent nodes and sum up incurred latency
  - to **limit the complexity**, only visit some adjacent nodes with lowest link latency
- From all visited paths, **the one with lowest latency is chosen**



Legend
Yellow nodes represent Workers
Blue nodes represent switches

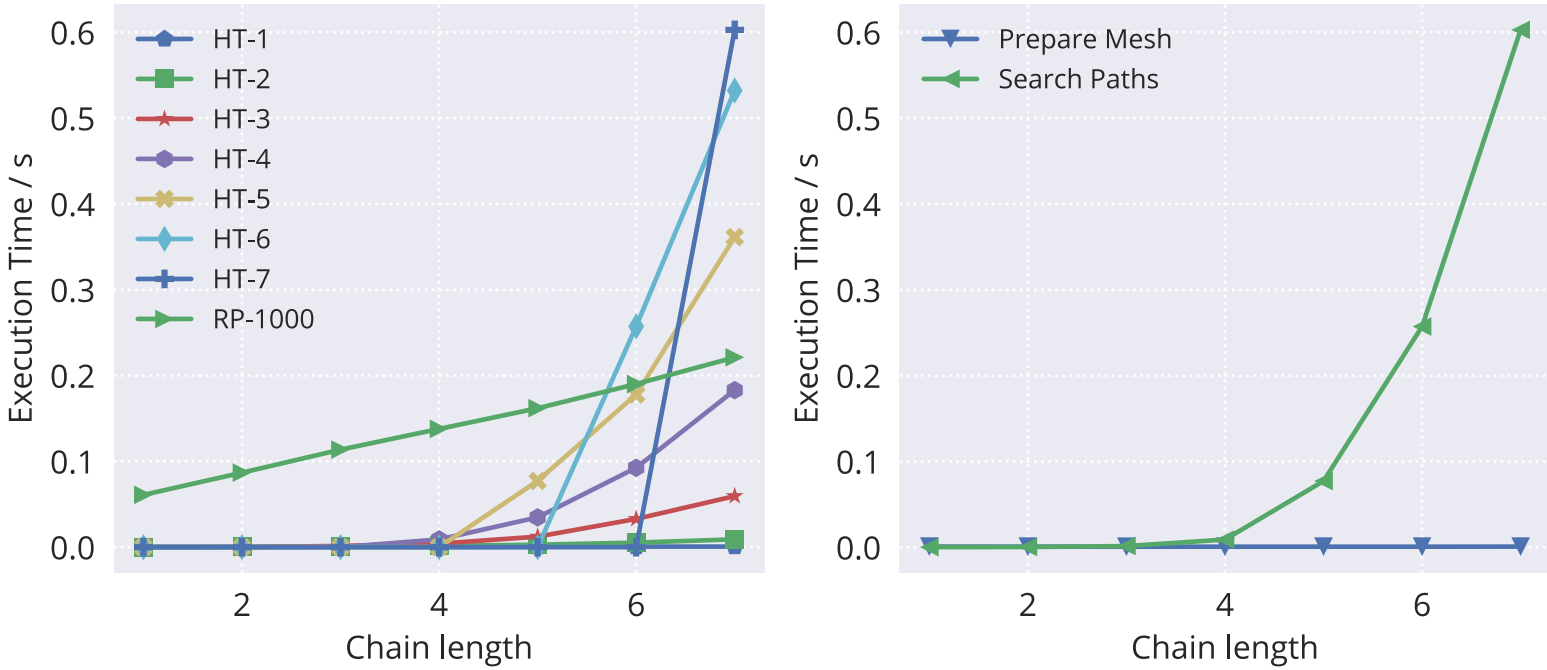
Hasty Traveller Algorithm: Expected Latency

- Expected Latency Plot
  - Hasty Traveller with various limit values (**HT-*limit***)
  - compared to 1000 random placements (**RP-1000**)
- Lower boundary of grey area represents best placement
- Upper boundary of grey area represents worst placement
- Results naturally depend on the underlay topology



Hasty Traveller Algorithm: Timing

- Plots show time it took to reach a solution
- With increasing limit, Hasty Traveller becomes very expensive
- At some point, choosing the best of 1000 random placements is faster
- A limit of 3 shows **reasonable complexity and good results** for typical chain lengths



Conclusion

Contributions

- State of the art NFV platforms
  - are missing awareness of the underlay network
  - can therefore not provide latency optimized deployments
- This work introduced new concepts to address these issues
  - **eBPF based integrated LLDP Monitoring** → increases network awareness
  - **Serverless network service lifecycle** → reduces administration complexity
  - **Graph theoretic placement strategy** → supports latency optimized deployments
- These ideas enabled the **design of a lightweight NFV platform**
  - Proof of concept implementation can be accessed via https://git.comnets.net/patrickziegler/nfv-mano

Outlook

- The developed system lays the **groundwork for further research** on
  - **Resource allocation** and **VNF benchmarking** for predicting computing delays
  - Early adoption of recently proposed **Network Service Headers** (NSH)
  - Evaluating the operation of P4 programmable switches against the OpenFlow based approach used in this work
- The proof of concept implementation
  - can be further developed to serve as a platform for **teaching NFV**
  - serves as a **reproducible environment** for research on new network services

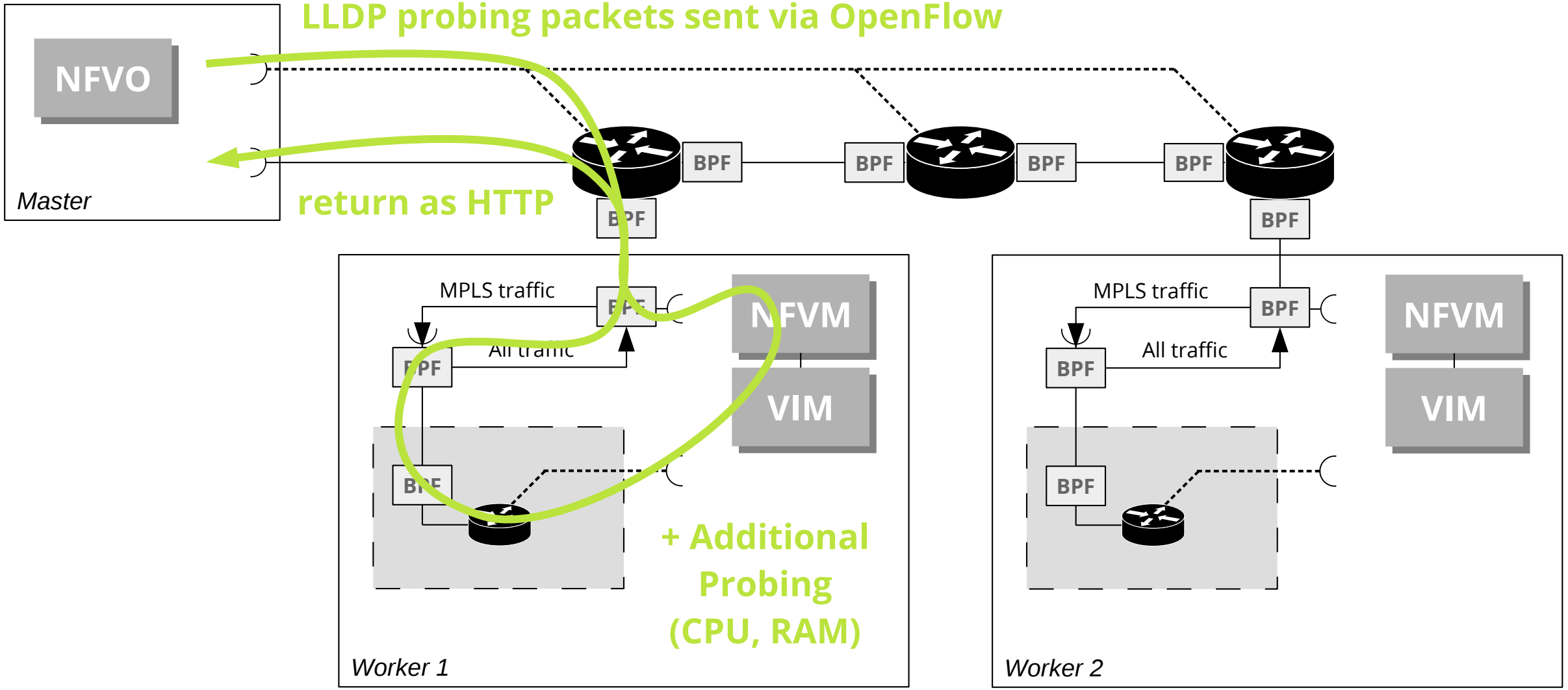
TECHNISCHE UNIVERSITÄT DRESDEN

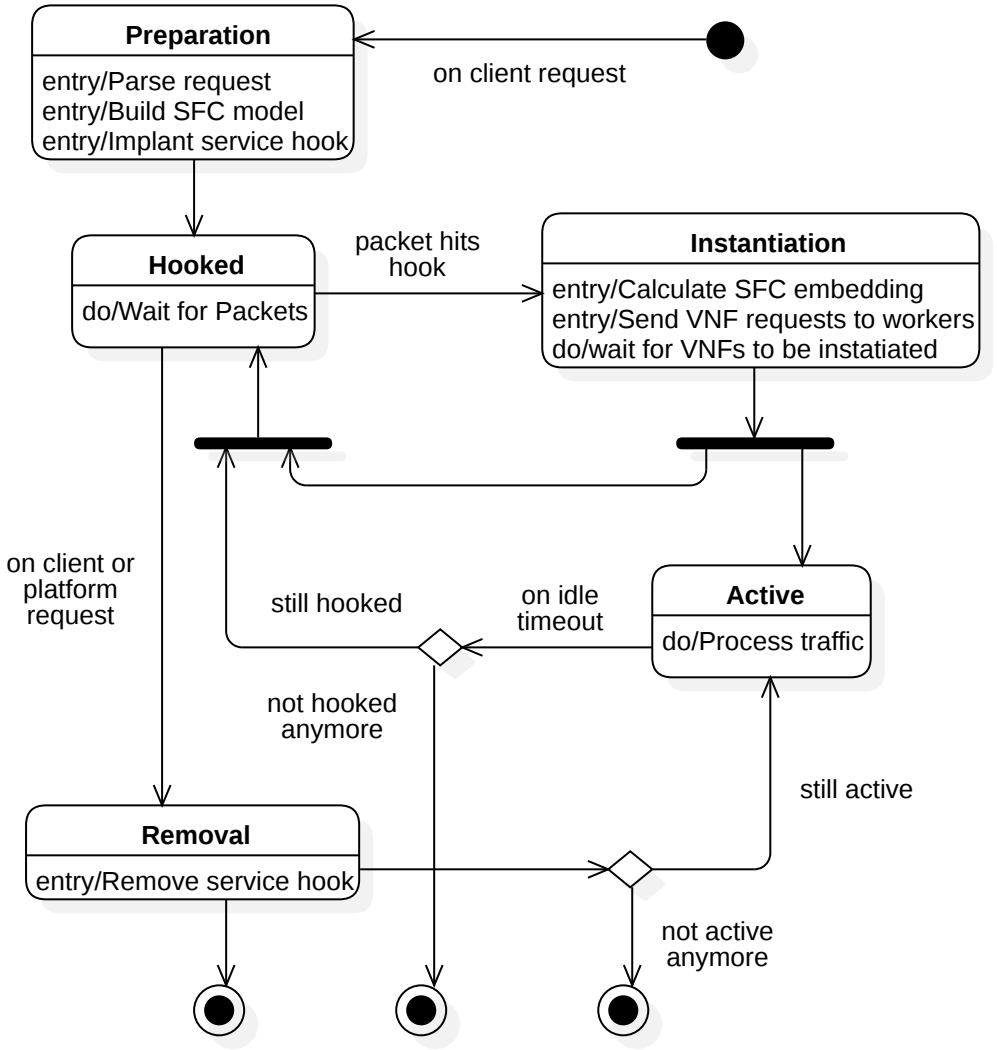
ComNets
Deutsche Telekom Chair of Communication Networks

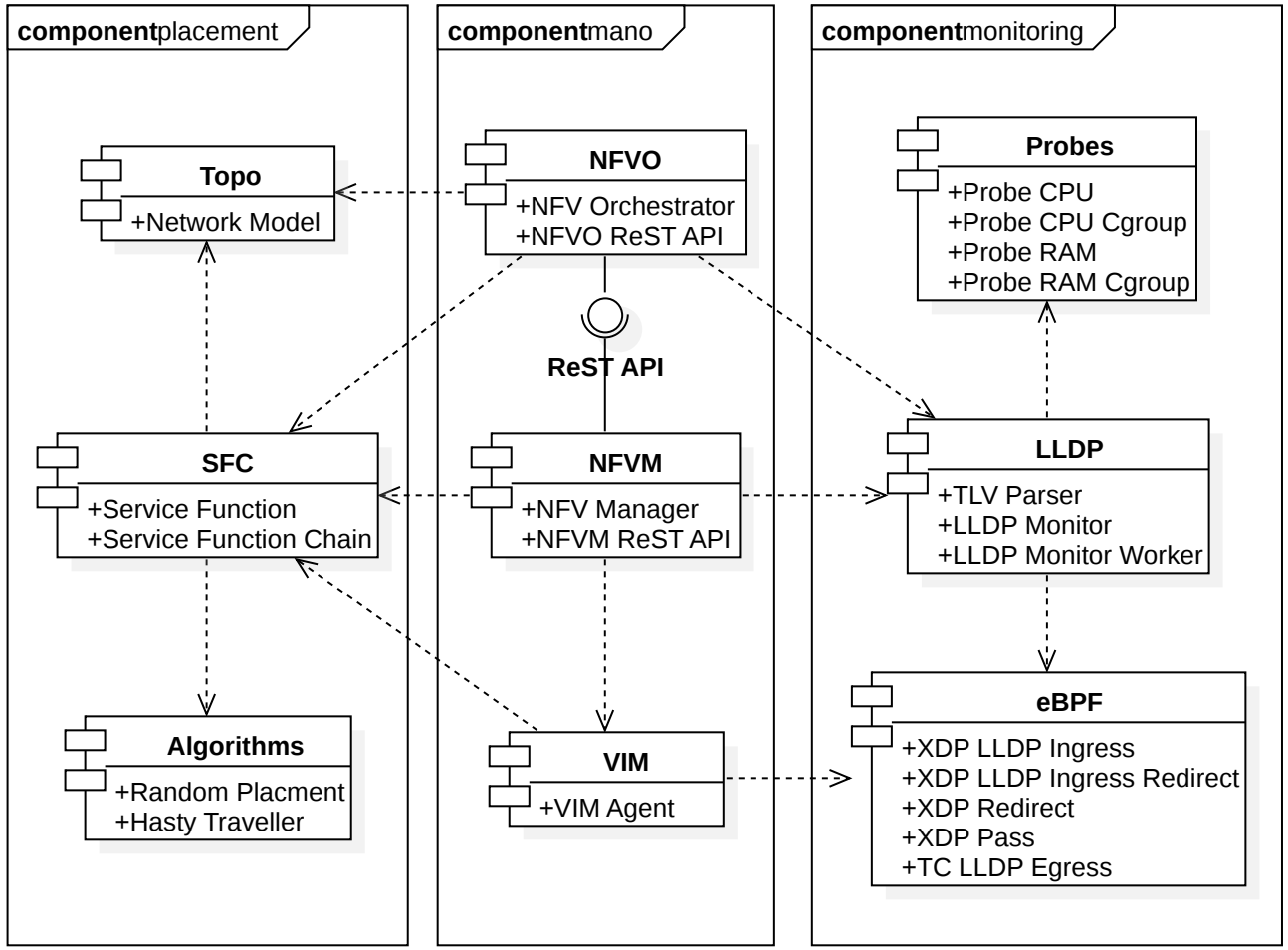
Live Demo

Patrick Ziegler
Diploma Thesis Defense, 24.10.2019
Deutsche Telekom Chair for Communication Networks

Monitoring system supports additional probing on Workers




Preparation
entry/Parse request
entry/Build SFC model
entry/Implant service hook
on client request
Hooked
do/Wait for Packets
packet hits hook
Instantiation
entry/Calculate SFC embedding
entry/Send VNF requests to workers
do/wait for VNFs to be instatiated
on client or platform request
still hooked
on idle timeout
Active
do/Process traffic
not hooked anymore
still active
Removal
entry/Remove service hook
not active anymore


componentplacement
componentmano
componentmonitoring
Topo
+Network Model
NFVO
+NFV Orchestrator
+NFVO ReST API
Probes
+Probe CPU
+Probe CPU Cgroup
+Probe RAM
+Probe RAM Cgroup
ReST API
SFC
+Service Function
+Service Function Chain
NFVM
+NFV Manager
+NFVM ReST API
LLDP
+TLV Parser
+LLDP Monitor
+LLDP Monitor Worker
Algorithms
+Random Placment
+Hasty Traveller
VIM
+VIM Agent
eBPF
+XDP LLDP Ingress
+XDP LLDP Ingress Redirect
+XDP Redirect
+XDP Pass
+TC LLDP Egress

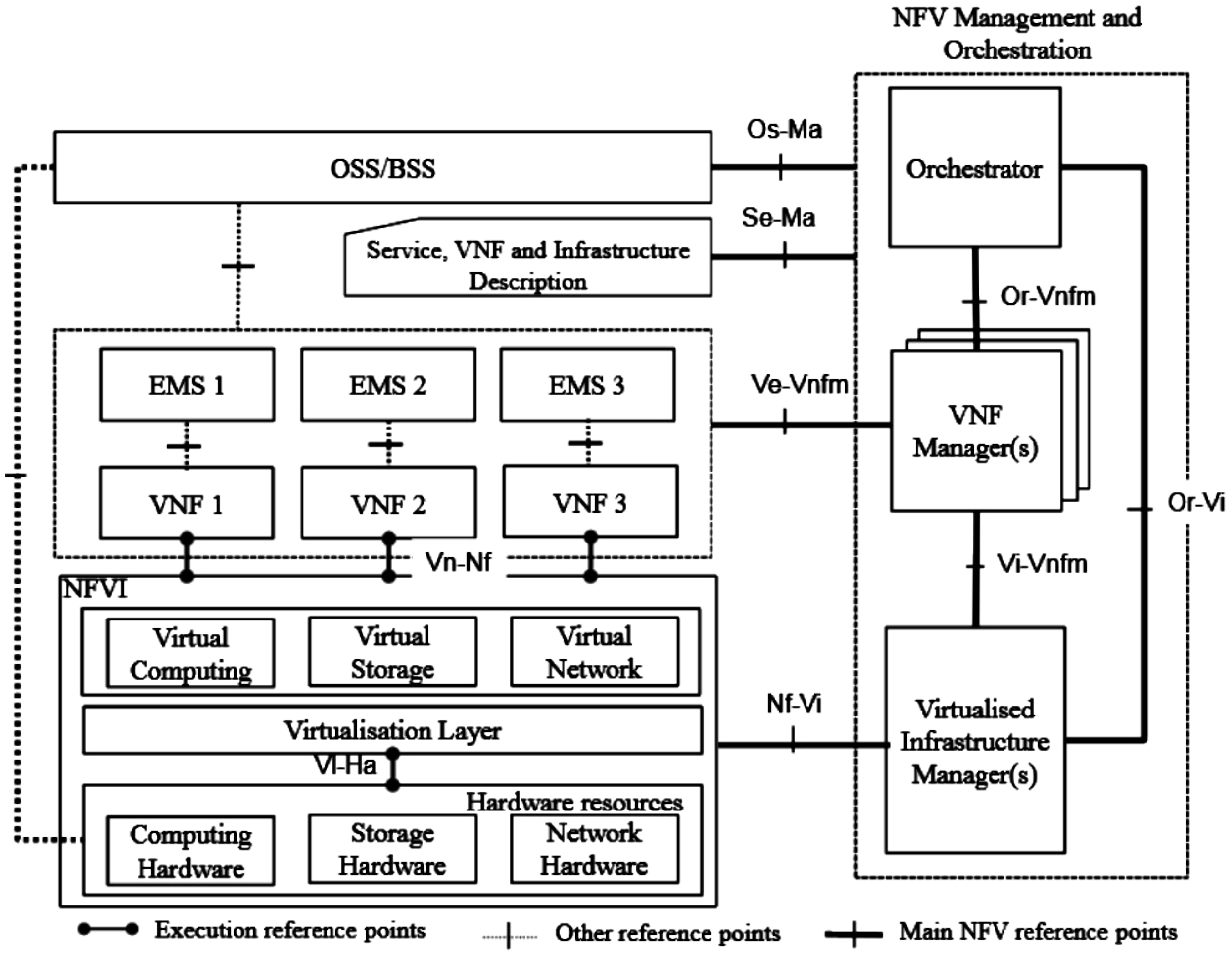
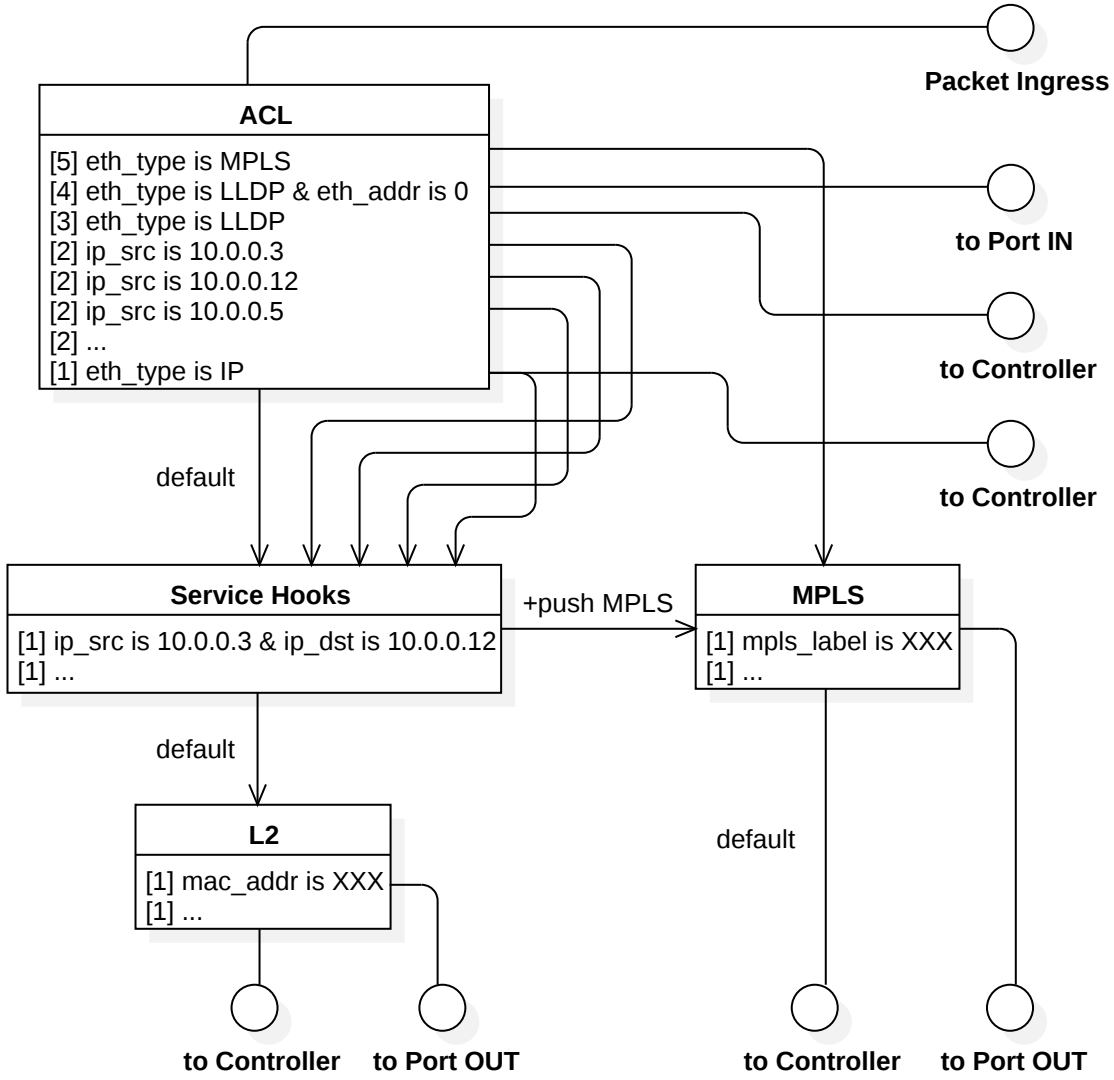
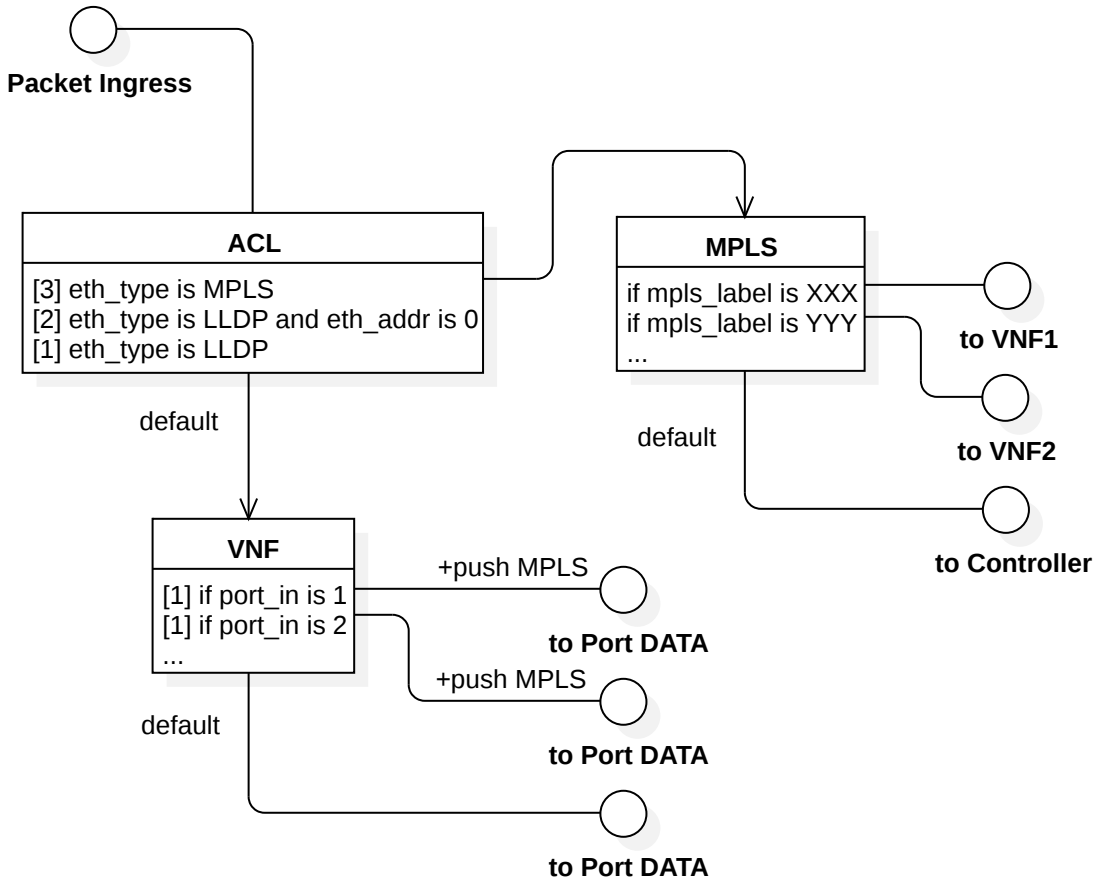


Image source:
S. K.Mohapatra, J. Bhuyan, and H. N.Narang, “Planning and managing virtual-ized next generation networks,” International journal of Computer Networks & Communications, vol. 7, pp. 01–16, 11 2015


Packet Ingress
ACL
[5] eth_type is MPLS
[4] eth_type is LLDP & eth_addr is 0
[3] eth_type is LLDP
[2] ip_src is 10.0.0.3
[2] ip_src is 10.0.0.12
[2] ip_src is 10.0.0.5
[2] ...
[1] eth_type is IP
to Port IN
to Controller
to Controller
default
Service Hooks
[1] ip_src is 10.0.0.3 & ip_dst is 10.0.0.12
[1] ...
+push MPLS
MPLS
[1] mpls_label is XXX
[1] ...
default
L2
[1] mac_addr is XXX
[1] ...
default
to Controller
to Port OUT
to Controller
to Port OUT


Packet Ingress
ACL
[3] eth_type is MPLS
[2] eth_type is LLDP and eth_addr is 0
[1] eth_type is LLDP
default
MPLS
if mpls_label is XXX
if mpls_label is YYY
...
default
to VNF1
to VNF2
to Controller
VNF
[1] if port_in is 1
[1] if port_in is 2
...
default
+push MPLS
to Port DATA
+push MPLS
to Port DATA
to Port DATA