libVNF: building VNFs made easy

Priyanka Naik, Akash Kanase, Trishal Patel, Mythili Vutukuru

Dept. of Computer Science and Engineering
Indian Institute of Technology, Bombay
NFV ecosystem

NFV: Network Function Virtualization
VNF: Virtual Network Function
NFV ecosystem

- Will they give good performance?
- Is it easy to build them?

NFV: Network Function Virtualization
VNF: Virtual Network Function
How to build VNF?

VNF code developed by VNF developer
How to build VNF?

VNF code developed by VNF developer

38% EPC code $\rightarrow$ read/write packets

CORD Intel EPC: https://gerrit.opencord.org/ngic
How to build VNF?

VNF code developed by VNF developer

38% EPC code → read/write packets

VNF Processing logic

VNF Framework

CORD Intel EPC: https://gerrit.opencord.org/ngic
How to build VNF?
How to build VNF?

What is missing in these frameworks?
What is required from VNF frameworks?

- Requirement 1: Support for both L3 and Transport VNF
- Requirement 2: Flexibility of network stack
- Requirement 3: Support for distributed state management
What is required from VNF frameworks?

- Requirement 1: Support for both L3 and Transport VNF
- Requirement 2: Flexibility of network stack
- Requirement 3: Support for distributed state management
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer
Support for L3 and transport VNFs

Layer 3 VNFs

Network address translator
Layer 3 Load balancer

Header manipulations

N/W layer
Data link layer
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer

Header manipulations

N/W layer

Data link layer

Frameworks: netbricks, YANFF
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer

Header manipulations

N/W layer

Data link layer

Frameworks: netbricks, YANFF

Transport Layer VNFs

vEPC

internet
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer

Header manipulations

N/W layer

Data link layer

Frameworks: netbricks, YANFF

Transport Layer VNFs

- vEPC
- internet

Request processing

Connection termination

Transport Layer

N/W layer

Data link layer

Connection initiation
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer

Header manipulations

N/W layer

Data link layer

Frameworks: netbricks, YANFF

Transport Layer VNFs

- vEPC
- internet

- Request processing

- Connection termination

- Transport Layer

- N/W layer

- Data link layer

Frameworks: mTCP, TLDK

Netbricks: Taking the v out of nfv. In Proc. of OSDI’16
YANFF: https://www.openhub.net/p/yanff
mTCP: A highly scalable user-level tcp stack for multicore systems. In Proc. of NSDI’14
TLDK: https://wiki.fd.io/view/TLDK
Support for L3 and transport VNFs

Layer 3 VNFs

- Network address translator
- Layer 3 Load balancer

Header manipulations

N/W layer

Data link layer

Frameworks: netbricks, YANFF

Transport Layer VNFs

- vEPC
- Internet

Request processing

Connection termination

Transport Layer

N/W layer

Data link layer

Frameworks: mTCP, TLDK

Are these frameworks enough?

Netbricks: Taking the v out of nfv. In Proc. of OSDI’16
YANFF: https://www.openhub.net/p/yanff
mTCP: A highly scalable user-level tcp stack for multicore systems. In Proc. of NSDI’14
TLDK: https://wiki.fd.io/view/TLDK
Event driven I/O

Existing transport-layer frameworks are event-driven
Event driven I/O

Pros:

Existing transport-layer frameworks are event-driven
**Event driven I/O**

Pros:

- Efficient for multi-core scalability

Existing transport-layer frameworks are event-driven
Event driven I/O

Pros:

- Efficient for multi-core scalability

Existing transport-layer frameworks are event-driven
Event driven I/O

Pros:

- Efficient for multi-core scalability

Cons:

Existing transport-layer frameworks are event-driven
Event driven I/O

Pros:

- Efficient for multi-core scalability

Cons:

- Needs explicit request state storage

Existing transport-layer frameworks are event-driven
Need to maintain request state
Need to maintain request state

State at B to process A’s request

- A’s request
- C’s reply
- Connection identifiers
Need to maintain request state

State at B to process A’s request

A’s request
C’s reply
Connection identifiers

VNF processing layer
(abstraction ?)

network stack (mTCP)
(connection)

DPDK and netmap layer
(packet)
Need to maintain request state

Existing frameworks do not provide this support
What is required from VNF frameworks?

- Requirement 1: Support for both Layer 3 and Transport VNF
- Requirement 2: Flexibility of network stack
- Requirement 3: Support for distributed state management
Flexibility of network stack

Kernel Stack

- Application VNF
- Kernel network stack
- vNIC
### Flexibility of network stack

<table>
<thead>
<tr>
<th>Kernel Stack</th>
<th>Kernel Bypass Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application VNF</td>
<td>Application VNF + userspace stack</td>
</tr>
<tr>
<td>Kernel network stack</td>
<td>DPDK/netmap</td>
</tr>
<tr>
<td>vNIC</td>
<td>vNIC</td>
</tr>
</tbody>
</table>
Flexibility of network stack

Kernel Stack

| Application VNF | Kernel network stack | vNIC |

Kernel Bypass Stack

| Application VNF + userspace stack | DPDK/netmap | vNIC |

Easy switch between stacks
What is required from VNF frameworks?

- Requirement 1: Support for both L3 and Transport VNF
- Requirement 2: Flexibility of network stack
- Requirement 3: Support for distributed state management
Support for distributed state management
Support for distributed state management

VNF 2  VNF 2

State Synchronization
Support for distributed state management

State Synchronization

State Migration
openNF, split/merge
Support for distributed state management

![Diagram showing state synchronization and migration](image)

**Stateless network functions**: Breaking the tight coupling of state and processing. In *Proc. of NSDI’17*

**Split/merge**: System support for elastic execution in virtual middleboxes. In *Proc. of NSDI’13*

**Opennf**: Enabling innovation in network function control. In *Proc. of SIGCOMM’14*
Support for distributed state management

Stateless network functions: Breaking the tight coupling of state and processing. In Proc. of NSDI’17
Opennf: Enabling innovation in network function control. In Proc. of SIGCOMM’14
## Summary of VNF Frameworks

<table>
<thead>
<tr>
<th>Requirement/Framework</th>
<th>netbricks</th>
<th>Flick</th>
<th>StatelessNF</th>
<th>Split-Merge/OpenNF</th>
<th>libVNF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 3 + App-layer support</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Flexibility of network stack</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Distributed State Management</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Netbricks:** Taking the v out of nfv. In *Proc. of OSDI’16*

**Flick:** Developing and running application-specific network services. In *Proc. of USENIX ATC’16*

**Stateless network functions:** Breaking the tight coupling of state and processing. In *Proc. of NSDI’17*

**Split/merge:** System support for elastic execution in virtual middleboxes. In *Proc. of NSDI’13*

**Opennf:** Enabling innovation in network function control. In *Proc. of SIGCOMM’14*
libVNF Design Goals

- Flexibility of network stack
- Support for network and transport layer VNF
- Distributed State Management

R: Requirement

R1: Flexibility of network stack
Handled by libVNF

R2: Support for network and transport layer VNF
Handled by libVNF

R3: Distributed State Management
Handled by VNF developer
libVNF overview
libVNF overview

- mTCP+ netmap/DPDK initialization
- Kernel stack initialization
- Stack initialization

VNF code

API Calls

libVNF API

Stack initialization
libVNF overview

- libVNF API
  - Stack initialization
  - Per-core threads

VNF code

API Calls

- mTCP+ netmap/DPDK initialization
- Kernel stack initialization

- Per-core threads
libVNF overview

- mTCP+ netmap/DPDK initialization
- Kernel stack initialization

- Lock-free
- Cache optimized

Stack initialization
Per-core threads
Per-core data structures
libVNF API

Communication

Request state

libVNF API

State Management
Communication API

VNF code

libVNF API

Pre-allocated memory pools (Per-core packet pools)

Per-core data structures

Per-core packet pool
**Communication API**

- VNF code
- `registerCallback(socket, fn)`
- Store mapping
- Pre-allocated memory pools (Per-core packet pools)
- Per-core data structures
- Per-core packet pool

Diagram:
- VNF code
- `registerCallback(socket, fn)`
- Store mapping
- Pre-allocated memory pools (Per-core packet pools)
Communication API

Packet arrives on socket

registerCallback(socket, fn)

Pre-allocated memory pools (Per-core packet pools)

VNF code

Store mapping

fn(packet)

Per-core packet pool

Per-core data structures

libVNF API
Communication API

VNF code

libVNF API

getPktBuf

Per-core data structures

Per-core packet pool

Pre-allocated memory pools (Per-core packet pools)
Communication API

Pre-allocated memory pools (Per-core packet pools)
VNF Design Requirements

Communication

State Management
libVNF API

VNF Design Requirements

Communication

Request state

State Management
Need for request state

VNF processing layer (abstraction ?)

network stack (mTCP) (connection)

DPDK and netmap layer (packet)

State at B to process A’s request

A’s request

C’s reply

Connection identifiers
Need for request state

VNF processing layer
REQUEST OBJECT

network stack (mTCP)
(connection)

DPDK and netmap layer
(packet)

State at B to process A’s request

Request object

A’s request
C’s reply
Connection identifiers

1 2
4 3

A’s request

C’s reply

Connection identifiers

A to B
B to C
C to A
Request Object API

libVNF API

Per-core request pool
Request Object API

A's request
C's reply
Connection identifiers

allocateReqObj(A connection_id)

Allocate request object block for A's request

Per-core request pool

libVNF API
Request Object API

libVNF API

allocatedReqObj(A connection_id)

Allocate request object block for A's request

Per-core request pool

Per-core packet pool
Request Object API

A's request
C's reply
Connection identifiers

1

2

3

4

linkReqObj(C connection_id)

libVNF API

Per-core request pool
Per-core packet pool
libVNF API

VNF Design Requirements

Communication

Request state

State Management
State Management API

VNF code

libVNF API
State Management API

VNF code

libVNF API

Local data store pool
State Management API

VNF code

setData(LOCAL)

Store in local datastore

libVNF API

Local data store pool
State Management API

VNF code

setData(LOCAL)

Store in local datastore

libVNF API

Local data store pool

libVNF data store wrapper

Redis KV store

Remote Data store
State Management API

VNF code

libVNF API

setData(REMOTE)

Cache locally

Store in remote data store

libVNF data store wrapper

Redis KV store

Local data store pool

Remote Data store
Evaluation

- Overhead of libVNF
- Scalability with cores
- Benefits of libVNF
Setup

Diagram:

- A
- B
- C

- Arrow 1 from A to B
- Arrow 2 from B to C
- Arrow 4 from C to A
- Arrow 3 from C to B
Setup

VNF A

VNF C

S/W switch (on kernel)
Setup

VNF A
S/W switch (on kernel)

VNF C
S/W switch (on kernel)

VNF B
S/W switch (like netmap-vale)

Physical NIC
NIC Queue
VNF A, C: 4 core, 4GB RAM
VNF B: 4 GB RAM, cores varied
Evaluation

- Overhead of libVNF
- Scalability with cores
- Benefits of libVNF
Overhead check
Overhead check

<5% overhead of libVNF
DPDK~ netmap performance
Evaluation

- Overhead of libVNF
- Scalability with cores
- Benefits of libVNF
Core scalability
Core scalability

scales linearly with cores
Evaluation

- Overhead of libVNF
- Scalability with cores
- Benefits of libVNF
### Building VNFs

<table>
<thead>
<tr>
<th>VNF</th>
<th>Performance Overhead of libVNF</th>
<th>LoC Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMS (IP Multimedia Subsystem)</td>
<td>3.4%</td>
<td>42%</td>
</tr>
<tr>
<td>EPC (LTE-Evolved Packet Core)</td>
<td>5.5%</td>
<td>38%</td>
</tr>
<tr>
<td>Layer 3 Load Balancer</td>
<td>14%</td>
<td>52%</td>
</tr>
<tr>
<td>VNF</td>
<td>Performance Overhead of libVNF</td>
<td>LoC Saved</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>IMS (IP Multimedia Subsystem)</td>
<td>3.4%</td>
<td>42%</td>
</tr>
<tr>
<td>EPC (LTE-Evolved Packet Core)</td>
<td>5.5%</td>
<td>38%</td>
</tr>
<tr>
<td>Layer 3 Load Balancer</td>
<td>14%</td>
<td>52%</td>
</tr>
</tbody>
</table>

Low overhead in app-layer VNF
Higher overhead in L3 VNF
Summary

- Library to ease building of VNFs
- Expressive to build L3 and App-layer VNF
- Supports multiple network stacks
- Low performance overhead

https://github.com/networkedsystemsIITB/libVNF
ppnaik@cse.iitb.ac.in
Thank You
Setup

VNF A

VNF C

S/W switch (on kernel)

LB VNF

VNF B

VNF B

Data store VM

S/W switch (vale on netmap)

Physical NIC

NIC Queue

A

1

B

2

C

4

3
VNF A, C: 4 core, 4GB RAM
VNF B: 4 GB RAM, cores varied
Data Store VM: 6 core, 16GB RAM
LB: 1 core, 4GB RAM