NFD: Using Behavior Models to Develop Cross-Platform Network Functions

Hongyi Huang, Wenfei Wu, Yongchao He, Bangwen Deng, Ying Zhang, Yongqiang Xiong, Guo Chen, Yong Cui, Peng Cheng

Tsinghua University, Facebook, Microsoft Research, Hunan University
Content

01 Background
02 Goal&Challenge
03 Design
04 Evaluation
05 Conclusion
Network Function (NF) / Middlebox

Various network functions are widely deployed in network & between hosts in addition to switches & routers

-- hard-coded
-- wired

Wan optimizer
Proxy
IDS
CDN
Firewall
NAT
BRAS
NFV replaces specialized middleboxes with software Virtual Network Functions (VNFs) consolidated on Commodity Off-The-Shelf (COTS) hardware.

-- Flexible deployment
-- Quick evolution
Deployment Way

Virtual Machine: NetVM, ClickOS

Container: OpenNetVM

Process/Function: Click, NetBricks
Accelerating Technique

Hardware-assisted: Floem, ClickNP, ResQ, SGX(Sgx-box), DPDK(many), GPU(apunet)

Software: OpenNF, S6
Integration: Cross-Product

NF Developer

IDS (Intrusion Detection System)
NAT (Network Address Translator)
Load Balancer
Rate Limiter
Cache
Monitor
...
...

m

Platform Provider/Developer

NetVM
OpenNetVM
Click
NetBricks
OpenNF
...
...

n

Network Operator

Explosion

m*n
IF \( m+n \) Possible?

\[ m \text{ NFs} + n \text{ Platforms(Compiler)} \Rightarrow m \times n \text{ (integrations)} \]

\[ m \text{ NF models} + n \text{ Platform(Abstraction + Compiler)} \Rightarrow m \times n \text{ (models)} + n \text{ compilers} \]

auto integration

\[ m \text{ NF models} + 1 \text{ Framework(Abstraction + Compiler)} + n \text{ Platform Plugins} \Rightarrow m \text{ (models)} + n \text{ (plugins)} + 1 \text{ Framework} \]

auto integration
Content

01 Background
02 Goal&Challenge
03 Design
04 Evaluation
05 Conclusion
Example 1

Kernel bypassing I/O

Change packet I/O with DPDK-enabled NICs

```c
//=== Snort.c with libpcap ===
int main(int argc, char* argv[]){
    ...
    // initialization
    pcap_loop(phandle, -1, pcap_handle, NULL);
}
```

```c
//=== Snort.c with DPDK ===
int main(int argc, char* argv[]){
    // initialization
    for (;;) {
        struct rte_mbuf *bufs[SIZE];
        rte_eth_rx_burst(port, 0, bufs, SIZE);
        ...
    }
}
```
Example2

State migration and management

Modifying PRADS and Snort to integrate with OpenNF takes more than 100 man-hours [OpenNF, StateAlyzr]

```c
//=== Snort.c without OpenNF ===
int main(int argc, char* argv[])
{
    ... // initialization
    pcap_loop(phandle,-1,pcap_handle,NULL);
}

//=== Snort.c with OpenNF ===
int main(int argc, char* argv[])
{
    ... // initialization
    locals.put_allflows = &put_allflows;
    sdmbn_init(&locals); // start agent
    pcap_loop(phandle,-1,pcap_handle,NULL);
}
```
2.5k extra lines of code in the modification when porting an IDS to Intel SGX [SGX-box]

```c
// === PRADS.c ===
void check_vlan (packetinfo *pi) {
    config.pr_s.vlan_recv++; // a state
    ...
}

void prepare_ip4 (packetinfo *pi)
{
    config.pr_s.ip4_recv++; // a state
}
```

```c
// === SGX Config ===
enclave {
    ...
    trusted { public void check_vlan (...);
                 public void prepare_ip4 (...);
    };
}
```
Intuitions to Design New Framework

a). Most integration targets a specific piece of logic (e.g., IO in DPDK, states in OpenNF)

b). The framework should provide interfaces for logic identification in NFs.

c). Integration operations are tedious but regular.

Goal: Build a cross-platform development framework for NFs
Challenge

1. Expressiveness.
   Expressive to describe the packet processing logic in various NFs

2. EasyDev.
   How to save the development workload for platform provider

   The outcome NFs should be logically correct and have comparable performance compared with existing legacy NFs.
Content

01 Background
02 Goal&Challenge
03 Design
04 Evaluation
05 Conclusion
Overview

NF Developer

NF Models

NF Abstractions

Platform Developer

Platform-specific Plugin

NF Compiler

Network Operator

NF Executables

Runtime Environment
Overview

Expressiveness

NF Developer
- NF Models
- NF Abstractions

Platform Developer
- Platform-specific Plugin
- NF Compiler

Network Operator
- NF Executables
- Runtime Environment
Expressive Language

Existing programming frameworks:
NetCore, SNAP, SDN, Click, …
# Semantics

<table>
<thead>
<tr>
<th>symbols</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f[h] )</td>
<td>( h ) is a header field (Figure 4 does not list all fields), and ( f[h] ) is the field ( h ) in packet ( f ).</td>
</tr>
<tr>
<td>( f[TAG] )</td>
<td>We append tags to each packet for flexible processing[34], which can be viewed fields of a packet.</td>
</tr>
<tr>
<td>( f[output] )</td>
<td>Record the output ports of a packet. ( f[output] := {p_1, p_2} ) means sending packet ( f ) to port ( p_1 ) and ( p_2 ). ( f[output] := \epsilon ) means dropping the packet.</td>
</tr>
<tr>
<td>( r )</td>
<td>Abbreviation for A rule: ( h_1 = v_1 \land h_2 = v_2 \land \ldots )</td>
</tr>
<tr>
<td>( f \in r )</td>
<td>Abbr. for a flow-rule match: ( f[h_1] = v_1 \land f[h_2] = v_2 \land \ldots )</td>
</tr>
<tr>
<td>( R )</td>
<td>Abbreviation for a rule set: ( {r_1, r_2, \ldots} )</td>
</tr>
<tr>
<td>( f \in R )</td>
<td>Abbreviation for a flow-ruleset match ( (f ) match one of rules in ( R )): ( f \in r_1 \lor f \in r_2 \lor \ldots )</td>
</tr>
</tbody>
</table>
Programming Abstractions

Packet processing abstraction: parse, deparse, transform
Bytestream processing abstraction: TCP flow
User-defined abstraction: custom abstractions
State abstraction: manage state in granularity

```c++
    string type="int";
    Value value = 0;
    int granularity = sip&dip&sport&dport&proto;
    map<unsigned, Value> instances;
    State_Counter& operator ++()
    {    
        key = hash(pkt&MaskOf(granularity));
        if (instances.find(key) == instances.end())
            instances[key] = value;
        ++instances[key];
    }
```

Time-driven logic abstraction: timer
Uniform Structure

Stateful Match Action Table

| entry | entry | ::= | \( (x_f \land x_s) \) then \((p_f ; p_s)\) else \( \bot \) |
| SMAT | smat | ::= | entry|entry; model |

Rationality:

1. Existing practices in Microsoft Azure [VFP]
2. Stateless -> compatible with switch policy [SDN, P4]

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>State</td>
</tr>
<tr>
<td>Stateful Firewall</td>
<td>Configuration: OK={r1, r2, ...}</td>
</tr>
<tr>
<td>f∈OK</td>
<td>-</td>
</tr>
<tr>
<td>f f∈seen</td>
<td></td>
</tr>
<tr>
<td>f f∉seen</td>
<td></td>
</tr>
</tbody>
</table>
Overview

Expressiveness

NF Models

NF Abstractions

EasyDev

Platform-specific Plugin

NF Compiler

NF Executables

Runtime Environment

NF Developer

Platform Developer

Network Operator
Syntax Tree

Intermediate Representation (IR)

leaf nodes

non-leaf nodes: derived to leaf nodes
Interface 1: Override (DPDK, GPU)

-- replace a piece of logic by a platform enhanced implementation
(replace pcap_loop)

```c
//=== Snort.c with libpcap ===
int main(int argc, char* argv[])
{
    ... // initialization
    pcap_loop(phandle, -1, pcap_handle, NULL);
}

//=== Snort.c with DPDK ===
int main(int argc, char* argv[])
{
    ... // initialization
    for (;;) {
        struct rte_mbuf *bufs[SIZE];
        rte_eth_rx_burst(port, 0, bufs, SIZE);
        ... 
    }
}
```
Interface2: Modification (OpenNF)

-- insert/delete/modify a node on the syntax tree using IR callback function
(add initialization)

```java
new OpenNFVisitor.visit(syntax_tree);
}

public class OpenNFVisitor implements NFDCompiler{
    @Override public T visitInit(...){
        AddAgentCode(...)
        InsertCode("List <State> allStates *")
        super.visitInit(...) // orig. compilation }
    @Override public T visitStateDeclaration(...){
        super.visitStateDeclaration(...)
        stateName = ... // get the state name
        InsertCode(String.format("allStates.add(%s),stateName"))
    }
}
Interface3: Retrieval (SGX)

-- collect extra information (sensitive state and function)
-- use the information for platform integration (SGX config)

```java
    new SGXVisitor.visit(syntax_tree);
)
    public class SGXVisitor implements NFDCompiler{
        List<String> sensitiveFunc;
        List<String> sensitiveData;
        @Override public T visitStateDeclaration (...){
            stateName =
            sensitiveData.add(stateName);
            @Override public T visitStateMatch (...){
                FuncName = ...
                sensitiveFunc.add(FuncName);
                @Override public T visitStateAction (...){
                    FuncName = ...
                    sensitiveFunc.add(FuncName);
                }
```
Example Workflow

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>match 1</td>
<td>action 1</td>
</tr>
<tr>
<td>match 2</td>
<td>action 2</td>
</tr>
</tbody>
</table>

Syntax Tree:
- Entry 1
  - SMAT
  - Entry 2

NFD:
- Match 1
  - Action 1
- Match 2
  - Action 2

void NF(packetinfo *pi, s) {
  if (match 1) action 1;
  else if (match 2) action 2;
}

trust function {
  action 1;
  action 2;
}

SGX Compiler

NF

Executable on SGX
Overview

NF Developer

- NF Models
- NF Abstractions

Expressiveness

Platform Developer

- Platform-specific Plugin
- NF Compiler

EasyDev

Network Operator

- NF Executables
- Runtime Environment

Performance
Prototype

Source: DSL
Compiler: ANTLR4
Target: C++

14 NFs + 6 Platforms

Comparing Workload (LOC)

NFD: models + framework + plugins = ~ 4k
manual: ~700k

<table>
<thead>
<tr>
<th>Component of NFD</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFD model grammar</td>
<td>234 (g4)</td>
</tr>
<tr>
<td>compiler frontend (automatically derived by ANtlr)</td>
<td>4.3k (Java)</td>
</tr>
<tr>
<td>compiler backend (generate C++ NF programs)</td>
<td>1137 (Java)</td>
</tr>
<tr>
<td>C++ template (program structure, operators) for NFs</td>
<td>752 (C++)</td>
</tr>
<tr>
<td>extension for OpenNF</td>
<td>489 (C++)</td>
</tr>
<tr>
<td>extension for GPU</td>
<td>668 (C++)</td>
</tr>
<tr>
<td>extension for DPDK</td>
<td>167 (C++)</td>
</tr>
<tr>
<td>extension for SGX</td>
<td>273 (C++)</td>
</tr>
</tbody>
</table>
Testbed

Server:

- Intel i9 CPU (10-core, 20-thread)
- 128GB memory
- 10Gbps NIC
- three NVIDIA GTX1080 Ti graphics cards
- 1TB SSD

Trace: [IMC10]

Opensource NFs: Snort, PRADS, Balance, HAProxy, Click NAT
Correctness

**Rate Limiter**: tuning rate
-- control the sending rate accurately as configuration
Performance

Firewall

Load Balancer

Optimization: reduce redundant logic
Integration

GPU
+ parallelism
- data transfer

(a) Encryption scaling up byte stream size
(b) Pattern matching scaling up number of patterns
(c) Pattern matching scaling up byte stream size
Integration

DPDK

SGX

405us v.s. 6952us
Content

01 Background
02 Goal&Challenge
03 Design
04 Evaluation
05 Conclusion
Conclusion

We built a cross-platform NF development framework NFD

- Platform-independent language
- Reconfigurable compiler
- Develop 14 NFs with 6 platforms
- Less workload, valid logic and performance, platform compatibility,
  and commodity-equivalent complex logic
Thank You