System Architecture
StatelessNF
StatelessNF Architecture

- **Controller**: Visualize/Modify, Monitor/Manage
- **SDN Switch**: Traffic to network functions
- **Data Store**: Timeout Manager
- **Network Function**
- **Network Function Host**
- **Network Function Host**
- **Visualize/Modify**
- **OF Rules**

**Network Function Host**

**SDN Switch**

**Controller**
Data Store

- Low latency, etc.
- Also needs (or could use) support for timers, atomic updates, queues
Network Function Instances
High-Performance Network I/O

e.g., DPDK, netmap

To remote data store

Input

Thread 1

NIC 1 ➔ RX

TX ➔ NIC 1

Output
Deployable Packet Processing Container

e.g., Docker

To remote data store

Pipe 1
NIC 1   Pull   Queue 1   Parse, Lookup, and Process   NIC 1

Pipe 2
NIC 2   Pull   Queue 2   Parse, Lookup, and Process   NIC 2

Pipe N
NIC N   Pull   Queue N   Parse, Lookup, and Process   NIC N

Input  Output
Optimized Data Store Client Interface

e.g., Batching, Buffer Alloc

To remote data store

Data Store Client Interface

Buffer Pool

Request Batching

Pipe 1

Thread 1

NIC 1

Pull

Queue 1

Thread 2

NIC 1

Parse, Lookup, and Process

Pipe 2

Thread 3

NIC 2

Pull

Queue 2

Thread 4

NIC 2

Parse, Lookup, and Process

Pipe N

Thread Nx2-1

NIC N

Pull

Queue N

Thread Nx2

NIC N

Parse, Lookup, and Process
Orchestration

- Failure handling – speculative failure detection (much faster reactivity)
- Scaling in and out – no need to worry about state when balancing traffic
Implementation

Network Functions (NAT, Firewall, Load balancer)
• DPDK
• SR-IOV
• Docker
• Infiniband to Data store (DPDK since paper)

Data store
• RAMCloud (Redis since paper)
• Extending

Controller
• Extended FloodLight, basic policies for handling scaling and failure. (complete re-write since paper)
StatelessNF System Evaluation
Goal: in this extreme case architecture, can we get similar throughput and latency as other software solutions, but with better handling of resilience and failure?
Tests:
- Raw throughput, latency
- Handling failure
- Handling scaling in-out

Network Functions:
- Baseline Network Functions (state and processing are coupled)
- Stateless Network Functions (state and processing are decoupled)
Throughput

Raw packets per second – lower until about 256 byte packets

Enterprise Trace – Stateless
Roughly matches Baseline

Note: similar to systems which have added support for scaling or failure
Latency

NAT (Firewall and Load balancer has slight less latencies)
Scaling In and Out

![Graph showing goodput (Gbps) over time with lines for Ideal, Baseline FW, and Stateless FW.]
Commercialization Effort
About

Murad Kablan  Eric Keller

+ 5 Engineers, 1 BizDev/Marketing, 1 intern
Target Customers

• Initial: Managed Service Providers, Next: Cable / Telco

Key Business Drivers:

New revenue streams
Gain more customers
Streamline operations
Reduce risk
Improve customer satisfaction

Managed hosting
21% CAGR
"Building and running a network service is difficult and expensive."

<table>
<thead>
<tr>
<th></th>
<th><strong>Hardware Infrastructures</strong></th>
<th><strong>Virtualized Infrastructures</strong></th>
<th><strong>Network as a Service</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DESIGN</strong></td>
<td>Slicing is hard</td>
<td>Lots of dev. effort</td>
<td>Ready to use</td>
</tr>
<tr>
<td><strong>REQUEST</strong></td>
<td>Support tickets handled by network operator</td>
<td>Support tickets handled by network operator</td>
<td>Automated via platform</td>
</tr>
<tr>
<td><strong>PURCHASING</strong></td>
<td>Specialized hardware with long delivery &amp; deployment times</td>
<td>Commodity hardware with restrictive license pools</td>
<td>Plug &amp; play commodity Hardware, pay per use</td>
</tr>
<tr>
<td><strong>CONFIG</strong></td>
<td>Extensive and time-consuming</td>
<td>Quick, but complex to scale</td>
<td>Automated and scalable via platform</td>
</tr>
<tr>
<td><strong>FUNCTIONS</strong></td>
<td>Complex to update and scale</td>
<td>Easy updates and scaling, but with disruption</td>
<td>Seamless updates without disruption</td>
</tr>
<tr>
<td><strong>UPGRADE</strong></td>
<td>Once every three to five years</td>
<td>Once a year for new license pools</td>
<td>Anytime, on-demand</td>
</tr>
</tbody>
</table>
Prove Technology outside of Lab

**PoC**

**Mechanism:**
- Deploy in sandbox
- Setup for fake tenants
- Simulate traffic / events (failure)

**Goal:**
- Demonstrate ease of use
- Product functionality feedback

**Exit Criteria:**
- Pass initial tests of stability, performance, and resilience
- Positive customer experience

**Mechanism:**
- Step1: Tapped real traffic.
- Step2: low-profile tenants.
- Simulate events (failure)

**Goal:**
- Quantify perf. and resilience
- Quantify value (cost savings)

**Exit Criteria:**
- Metrics meet expectations

**Mechanism:**
- Offer out as service to tenants.

**Support:**
- Support to initial customers 24/7.
- Frequent product updates

1 started, 1 committed, 2 in discussion, willing to bring on 2 more over next 12-18 months
From the Academic Paper to Product
Network Function Design

Data Store Client

- Reduce interaction
- Hide optimizations
  - Easy to write NFs
  - (code is agnostic to opt.)
- Processing as graph of fine or coarse functions

Near term: clean API, leverage ubiquity of DPDK
Standard Distributed System Issues

Consistency

- Data store scaling
- add
- re-shard

Transactions

- Failure

Configuration

- Control
- Config
ONAP, OpenStack, ...  
- Doesn’t (shouldn’t) matter to us

Public Cloud?  
- Current impl. hindered by lack of control in virtualization layer, and network layer  
  - (e.g., lack of DPDK support, limitation on tunneling, unpredictable network)
Conclusions and Future Work

• Networks need agile network functions
  • Seamless scalability, failure resiliency, without sacrificing performance

• StatelessNF is a design from the ground up
  • Zero loss scaling, zero loss fail-over

• Main potential drawback... performance, but in this extreme point:
  • Throughput similar to other solutions
  • 100-300us added latency (similar to other solutions)

• Future work: Evolve data store design for network functions
Thanks!

eric@bestateless.com