SNF: Serverless Network Functions

Arjun Singhvi, Junaid Khalid, Aditya Akella, Sujata Banerjee

* This work does not have any affiliation with Google
Network Functions (NFs) 101

NFs are typically **stateful** and maintain **per-flow internal state**

- IDS: flow to automaton state mapping
- LB: flow to backend server mapping
- ....
How can one offer NFs as a Service (NFaaS)?

- Intuitive programming model
- Delivers low-latency processing requirements
- Automatically scale up/down to meet the demand
- Usage-based billing
Serverless Computing for NFaaS?

Seems to have the right building blocks

- Event-driven programming model
- Usage-based billing
- Automatic compute elasticity

Upload your function to the serverless computing platform and register for event-based triggers

Execution is triggered when an event occurs

Platform runs your function only when triggered

Pay just for the computation time
Issue 1: Stateless Function Abstraction

Incoming Events (e.g., packets) → Controller → Remote Storage Service
Issue 1: Stateless Function Abstraction

Functions are stateless by design – no (guaranteed) local state across events

State transfer via remote storage

- High per-packet processing latencies due to physical decoupling of compute and state
Issue 1: Stateless Function Abstraction

Functions are stateless by design – no (guaranteed) local state across events

State transfer via remote storage

- High per-packet processing latencies due to physical decoupling of compute and state
Why not opt for a “Server-full” NFaaS?

Compute and state are **physically coupled**

Leads to coarse-grained flow-level work allocation
- Commitment between a flow and compute unit

Trade-off between efficiency and performance
- Overload impacts performance
- Under utilization impacts efficiency

State migration during traffic redistribution has **high overheads**
Issue 2: Work Allocation at Billing Granularity

Coupling between units of controller work allocation and billing - events

- Leads to trade-off between resource efficiency, performance and usage-based billing.
  - Packet
    - Efficiency and ideal billing at the cost of performance
  - Flow
    - Performance at the cost of efficiency and ideal billing
Issue 2: Work Allocation at Billing Granularity

Coupling between units of controller work allocation and billing - events leads to trade-off between resource efficiency, performance, and usage-based billing.

Packet efficiency and ideal billing at the cost of performance. Performance at the cost of efficiency and ideal billing.

**SNF** is a serverless platform that provides support to stateful NFs and allows offering NFaaS.
SNF Design Overview: Key Ideas

**Key Idea #1**
Decouple work allocation and billing granularity

**Key Idea #2**
Middleground for compute-state (de)-coupling

**Key Idea #3**
Leverage the existence of flowlets

Flowlets are **burst of packets** that is separated in time from other bursts by a sufficient gap — called the flowlet **timeout**
SNF Design Overview: Key Abstractions

Incoming Events
(e.g., packets)

Controller

NAT₁

NATₙ
SNF Design Overview: Key Abstractions

Incoming Events (e.g., packets) → Controller → $NAT_1$ → $NAT_n$.
SNF Design Overview: Key Abstractions

**Ephemeral Stateful Functions**
- All events (packets) within a flowlet are sent to the same function and state is maintained locally

**Peer-Peer Distributed Storage**
- NFs share state with each other in a peer-peer fashion
- Leverage the flowlet timeout to *proactively* replicate state
SNF Design: State Management - Why Proactive?

A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$. 
A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$. 
A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$
SNF Design: State Management - Why Proactive?

A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$
SNF Design: State Management - Why Proactive?

A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$
SNF Design: State Management - Why Proactive?

A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$.
SNF Design: State Management - Why Proactive?

A flowlet $f_{i+1}$ of flow F can be assigned to a different compute unit as compared to flowlet $f_i$.

Such reactive approaches degrade performance!
SNF Design: State Management

Proactively replicate state before the next flowlet $f_{i+1}$ arrives

When to proactively replicate?
- Balance between making unnecessary transfers and wait times
  - Replicate at half the flowlet inactivity timeout

Where to proactively replicate?
- Future flowlet to compute unit assignment not known
- **Top K** compute units in a weighted randomized manner
  - weights correspond to how controller prioritizes allocation to a particular compute unit
SNF Design: Compute Management

Weighted greedy bin-packing algorithm
- Maximally pack flowlets into few compute units
- Prefers units to which have been proactively replicated

Score = Utilization + $\alpha \times \text{StateExists}$

$\alpha$ balances utilization against proactive benefits
SNF Design: Compute Management

Weighted greedy bin-packing algorithm
- Maximally pack flowlets into few compute units
- Prefers units to which have been proactively replicated

\[ \text{Score} = \text{Utilization} + \alpha \times \text{StateExists} \]

\( \alpha \) balances utilization against proactive benefits
SNF Design: Compute Management

Weighted greedy bin-packing algorithm
- Maximally pack flowlets into few compute units
- Prefers units to which have been proactively replicated

Score = Utilization + $\alpha \times \text{StateExists}$

$\alpha$ balances utilization against proactive benefits
SNF Design: Compute Management

Weighted greedy bin-packing algorithm
- Maximally pack flowlets into few compute units
- Prefers units to which have been proactively replicated

Score = Utilization + $\alpha \times \text{StateExists}$

$\alpha$ balances utilization against proactive benefits
SNF Evaluation: Implementation and Testbed

Prototype – built from scratch

Workload – replay previously captured packet traces (3.8M packets with 1.7K connections)

NFs – NAT, LB, IDS, UDP Whitelister, QoS Traffic Policer

Compute units configured to handle 1Gbps incoming workload
SNF Evaluation: Compute Management

Can SNF provision compute as per the incoming traffic demand at fine time scales?

Baselines:

<table>
<thead>
<tr>
<th>Vanilla Flow Allocation</th>
<th>Smart Flow Allocation (Xms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate when new flow</td>
<td>Allocate when new flow</td>
</tr>
<tr>
<td>arrives</td>
<td>arrives and every X ms</td>
</tr>
<tr>
<td>(associated with compute</td>
<td></td>
</tr>
<tr>
<td>unit for entire lifetime)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Adopted by generic</td>
</tr>
<tr>
<td></td>
<td>server-full alternatives</td>
</tr>
<tr>
<td></td>
<td>Adopted by NF specific</td>
</tr>
<tr>
<td></td>
<td>solutions</td>
</tr>
</tbody>
</table>
SNF Evaluation: Compute Management

Vanilla Flow

Smart Flow (100ms)

Smart Flow (50ms)

Flowlet
Vanilla Flow Allocation does not adapt well to incoming workload. 

Smart Flow (Xms) Allocation leads to most overprovisioning of compute units. 

Flowlet Allocation closely matches the incoming load.
Vanilla Flow Allocation does not adapt well to incoming workload. Smart Flow (Xms) Allocation leads to most overprovisioning of compute units. Flowlet Allocation closely matches the incoming load.
SNF Evaluation: Compute Management

Vanilla Flow Allocation does not adapt well to incoming workload.
Smart Flow (Xms) Allocation leads to most overprovisioning of compute units.
Flowlet Allocation closely matches the incoming load.
Vanilla Flow Allocation does not adapt well to incoming workload.
Smart Flow (Xms) Allocation leads to most overprovisioning of compute units.
Flowlet Allocation closely matches the incoming load.

Flowlet Allocation gets $3.36x$ more opportunities to assign work.
SNF Evaluation: Compute Management
SNF Evaluation: Compute Management

Vanilla Mode
High Latencies
SNF Evaluation: Compute Management

**Vanilla Mode**

- High Latencies
- Pinning of flows to compute
SNF Evaluation: Compute Management

Vanilla Mode
High Latencies

Smart Flow Mode
Moderately High Latencies

Pinning of flows to compute
SNF Evaluation: Compute Management

**Vanilla Mode**
- High Latencies
- Pinning of flows to compute

**Smart Flow Mode**
- Moderately High Latencies
- Unable to handle low-time scale overloads
SNF Evaluation: Compute Management

**Vanilla Mode**
- High Latencies
- Pinning of flows to compute

**Smart Flow Mode**
- Moderately High Latencies
- Unable to handle low-time scale overloads
- Reactive State Transfers
SNF Evaluation: Compute Management

- **Vanilla Mode**
  - High Latencies
  - Pinning of flows to compute

- **Smart Flow Mode**
  - Moderately High Latencies
  - Unable to handle low-time scale overloads

- **Flowlet Mode**
  - Low Latencies
  - Reactive State Transfers
SNF Evaluation: Compute Management

Vanilla Mode
- High Latencies
- Pinning of flows to compute

Smart Flow Mode
- Moderately High Latencies
- Unable to handle low-time scale overloads

Flowlet Mode
- Low Latencies
- Reactive State Transfers
- Proactive State Transfers
SNF Evaluation: Compute Management

- **Vanilla Mode**
  - High Latencies
  - Pinning of flows to compute

- **Smart Flow Mode**
  - Moderately High Latencies
  - Unable to handle low-time scale overloads
  - Reactive State Transfers

- **Flowlet Mode**
  - Low Latencies
  - Proactive State Transfers
  - Frequent Work Allocation

Graph: Cumulative Probability vs. Packet Processing Latency (us) with lines indicating different latencies for Vanilla, Flowlet, and Smart Flow modes.
SNF Evaluation: Compute Management

Can SNF provision compute as per the incoming traffic demand at fine time scales?

Does SNF provide its performance while not sacrificing utilization?
Flowlet allocation mode ensures that there is rare overload and lesser underutilization.
SNF Evaluation: State Management

Does proactive state replication help curtail tail latencies?

Baselines:

<table>
<thead>
<tr>
<th>Reactive</th>
<th>Optimized State External</th>
</tr>
</thead>
<tbody>
<tr>
<td>On arrival of new flowlet, state pulled from compute unit where previous flowlet was processed</td>
<td>State proactively pushed to external store and pulled on arrival of new flowlet</td>
</tr>
</tbody>
</table>

NF State Management Solutions

State Management in Serverless Platforms Today
SNF Evaluation: State Management
SNF Evaluation: State Management

**Baselines**
Tail latencies dominated by RTT
SNF Evaluation: State Management

**Baselines**
Tail latencies dominated by RTT

**Proactive (SNF)**
Low tail latencies due to proactive state transfers
SNF Evaluation: State Management

Baselines
Tail latencies dominated by RTT

Proactive (SNF)
Low tail latencies due to proactive state transfers

12-15x improvement in tail latencies
SNF Summary

Performant and efficient serverless platform that offers NFaaS

Decouple work allocation granularity (flowlet) from billing (packet) granularity

Realize the notion of ephemeral stateful functions
SNF Summary

Performant and efficient serverless platform that offers NFaaS

Decouple work allocation granularity (flowlet) from billing (packet) granularity

Realize the notion of ephemeral stateful functions

More details in paper

- How to ensure state fault tolerance?
- How to deal with adversarial flowlets?
Thank you!

asinghvi@cs.wisc.edu

SNF: Serverless Network Functions

Arjun Singhvi, Junaid Khalid, Aditya Akella, Sujata Banerjee