Evaluating Viability of Network Functions on Lambda Architecture

By Arjun Singhvi, Anshul Purohit and Shruthi Racha

Network Functions (NFs)

- Examine and modify packets and flows in sophisticated ways
- Ensure security, improve performance, and providing other novel network functionality
- Examples of Network Functions : Firewall, Network Address Translators, Intrusion Detection Systems

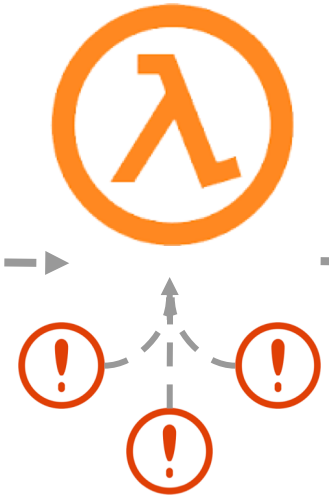
Network Functions (NFs)

- Lie in the Critical Path between source and destination

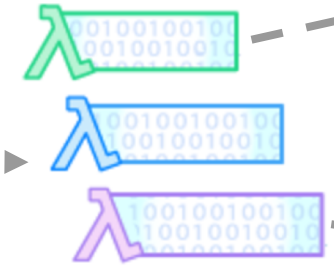
- Should be capable of
  - Handling packet bursts
  - Failures

Lambda Architecture - Working

Upload your code to Lambda



Setup your code to trigger from other cloud services, HTTP endpoints or in-app activity



Lambda runs your code only when triggered, using only the compute resources needed

Pay just for the compute time used

Lambda Frameworks

- Lambda Frameworks are popular

- Public cloud lambda offerings
  - AWS Lambda
  - Azure Functions
  - Google Cloud Functions

Lambda Frameworks - Advantages

- Elimination of server management
- Continuous Scaling on demand
- High-availability
- Pay-as-you-go model
- Developer just writes event-handling logic

Problem Statement

Does it make sense to implement network functions on lambda architectures?

Our Focus

- Investigate the performance of standalone NFs on Lambda architectures
- Implement and evaluate a locality-aware, event-based NF chaining system - LENS

Key Takeaways

- Naively implementing NFs on Lambda architecture leads to scalability at the cost of
  - High end-to-end latency
  - High overhead
- Porting standalone NFs onto Lambda architecture is not a viable option
- Lambda architectures are too restrictive - users cannot control the placement of lambda functions

Outline

- Standalone NFs Implementation
- Standalone NFs Evaluation Results
- LENS Design
- LENS Implementation Choices
- LENS Evaluation Results
- Summary
- Conclusion

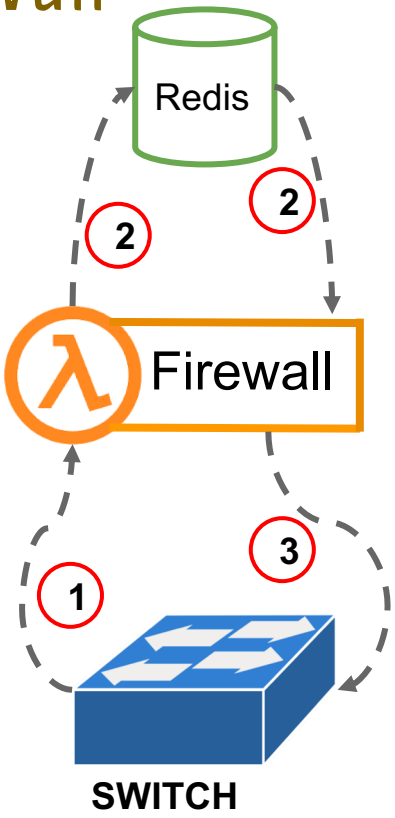
Standalone Network Functions

- Firewall
- NAT (Network address translation)
- PRADS (Passive Real-time Asset Detection System)

Standalone Network Functions - Firewall

- Monitors and controls the incoming and outgoing network traffic based on predetermined security rules

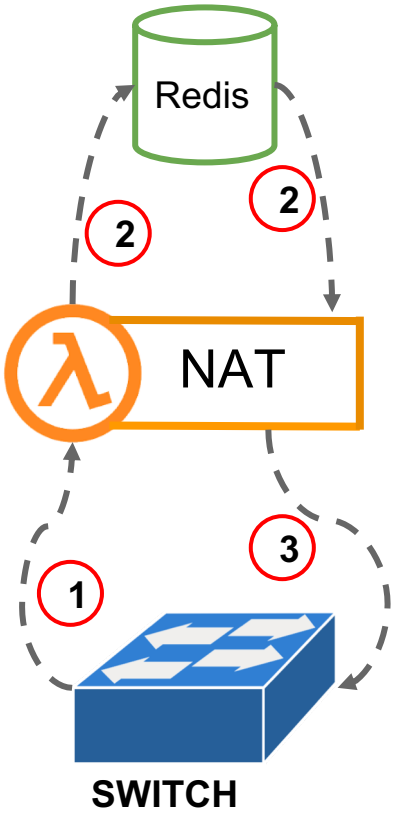
- Control Flow -
  i. Switch triggers Firewall
  ii. Fetch security rules
  iii. Block malicious packets



Standalone Network Functions - NAT

- Remaps IP addresses across private and public IP address space

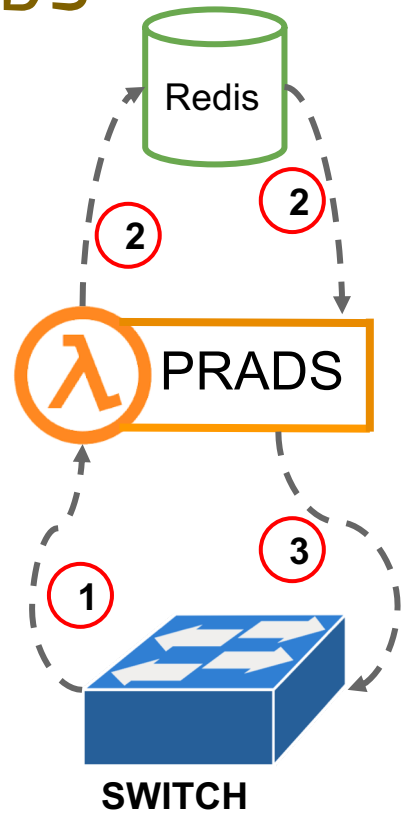
- Control Flow -
  i. Switch triggers NAT
  ii. Extract IP address from packet
  iii. Lookup for IP from external store
  iv. Modify the IP address in packet



Standalone Network Functions - PRADS

- Gathers information on hosts/services

- Control Flow -
  i. Switch triggers PRADS
  ii. Extract relevant packet fields
  iii. Store to external store



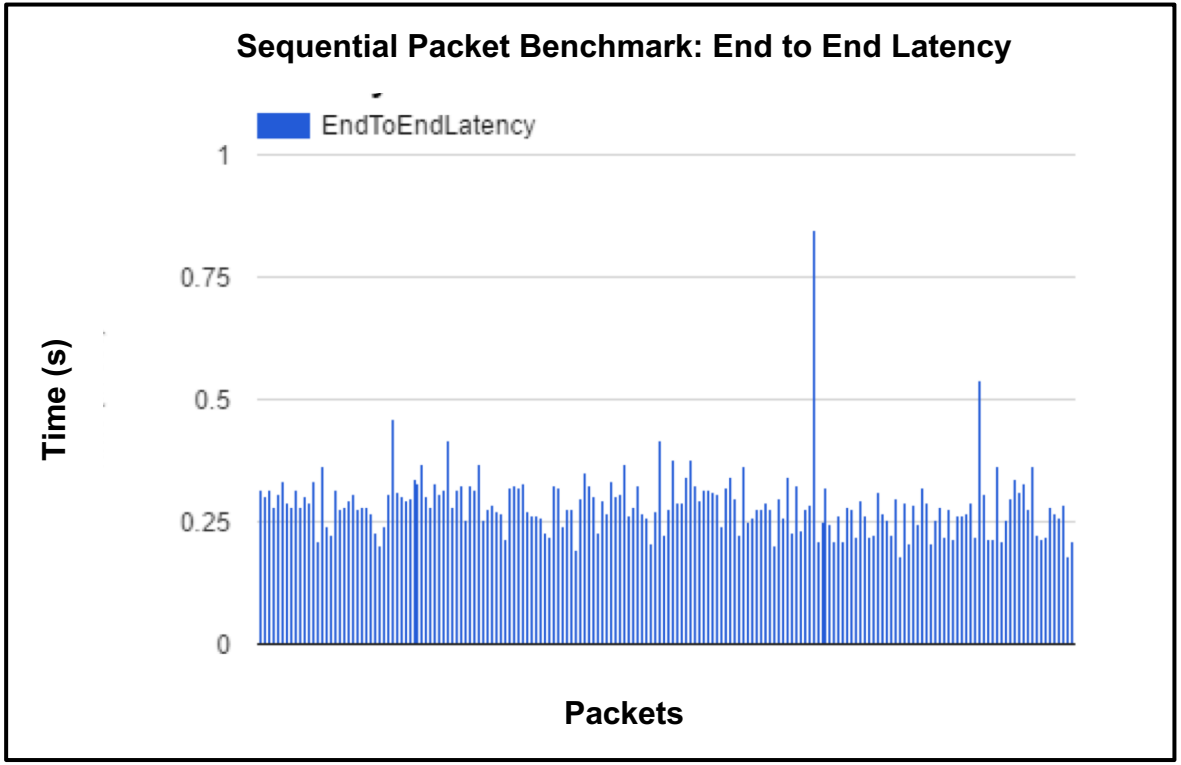
Experimental Setup

- Experiments run on Cloudlab

- Synthetic Benchmarks -
  - Sequential Packet Benchmark
    - Analyse latency breakdown
  - Concurrent Packet Benchmark
    - Analyze latency with scale

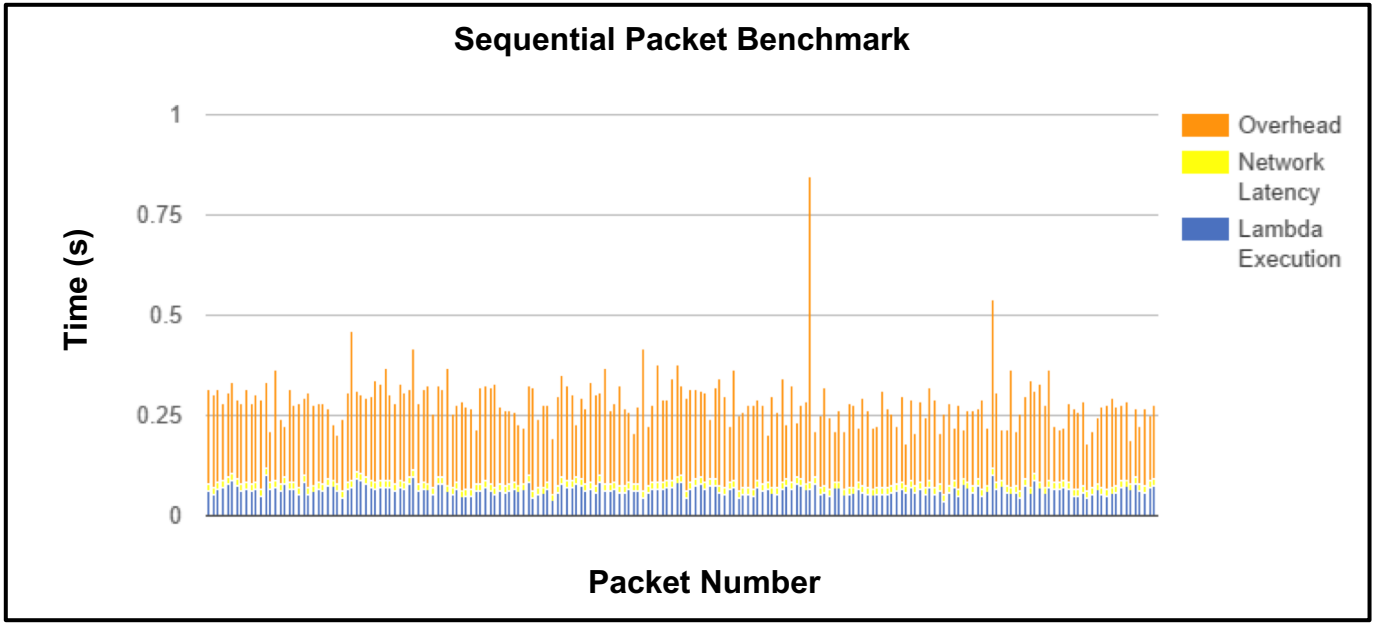
- Lambda Region
  - AWS: us-east-1 region

Sequential Packet Benchmark Results - NAT



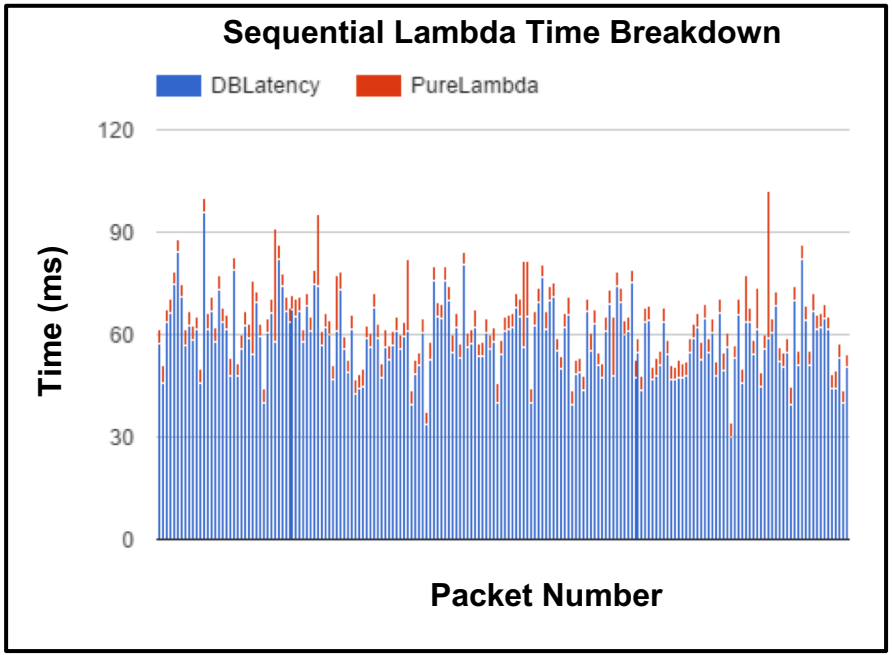
Sequential Packet Benchmark Results - NAT

Total Latency = Lambda Execution Time + Network Latency + AWS Overhead



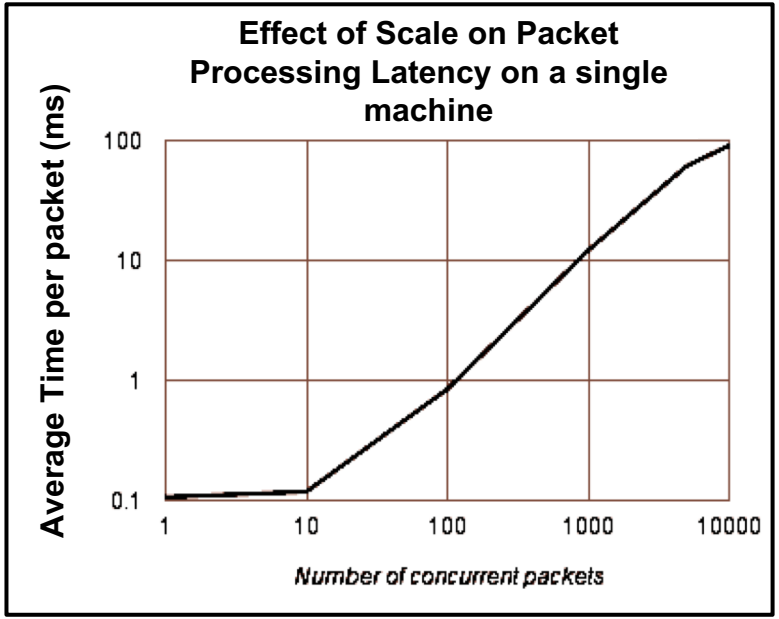
Sequential Packet Benchmark Results - NAT

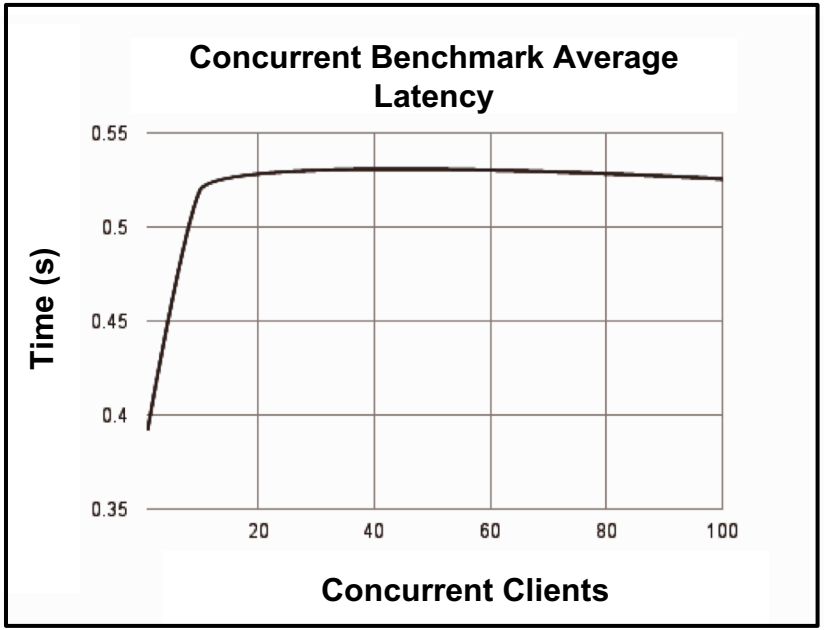
Lambda Execution Time = External Store Access Time + Pure Lambda Execution Time



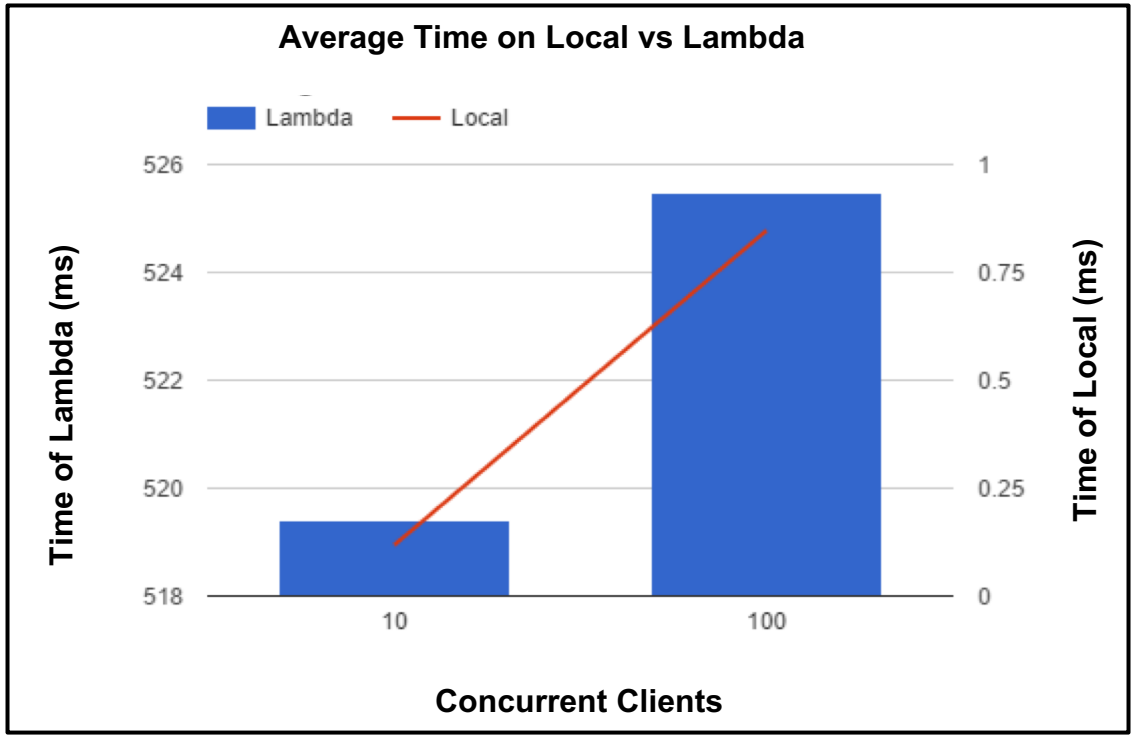
Concurrent Packet Benchmark Results - NAT

Network Functions Scale on Lambda Frameworks



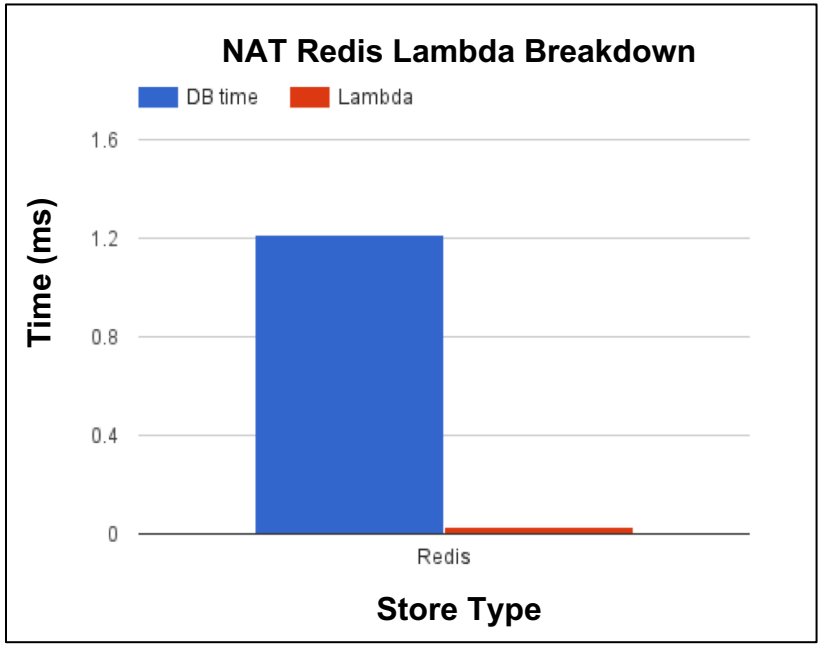


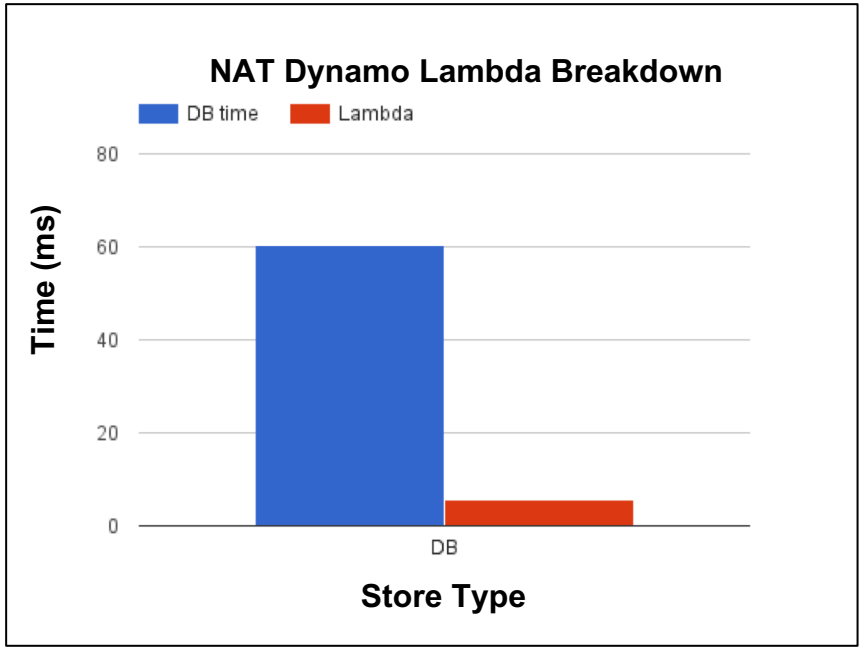
Concurrent Packet Benchmark Results - NAT



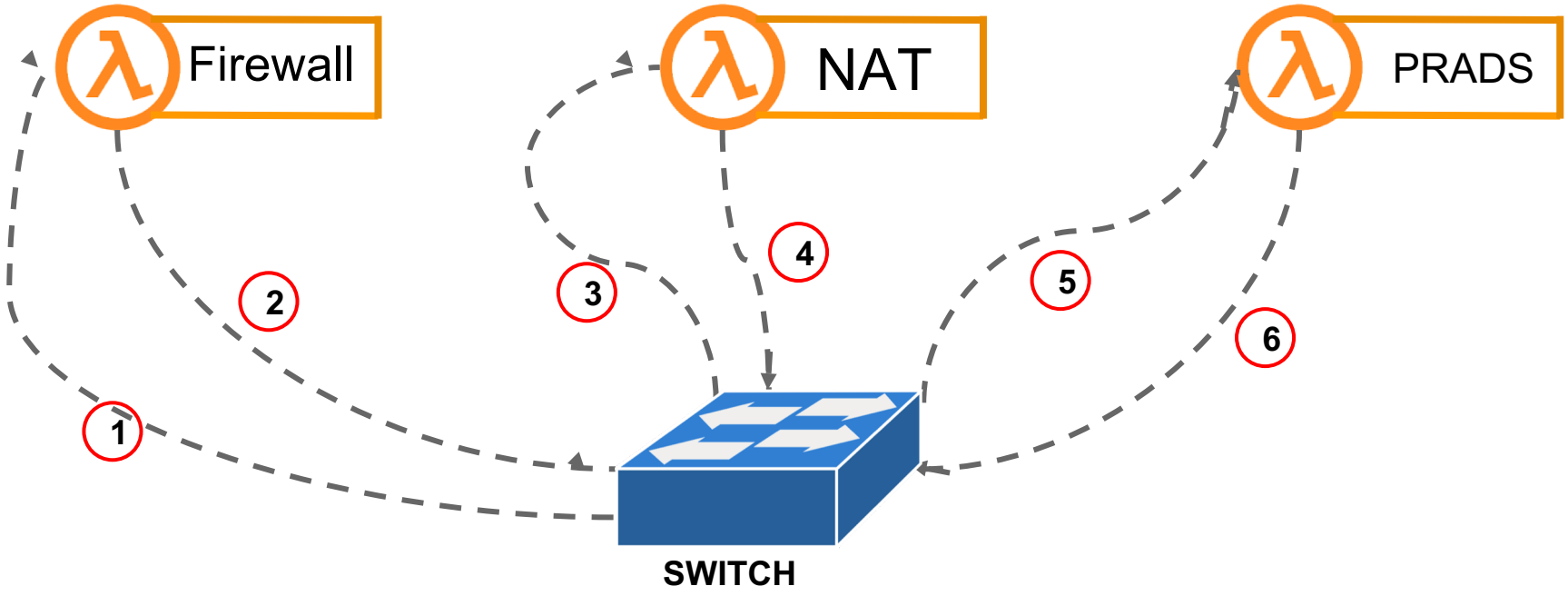
DynamoDB vs Redis

Use of in-memory redis state operations provides much lower latencies





Middlebox Chaining Solution (Naive Approach)

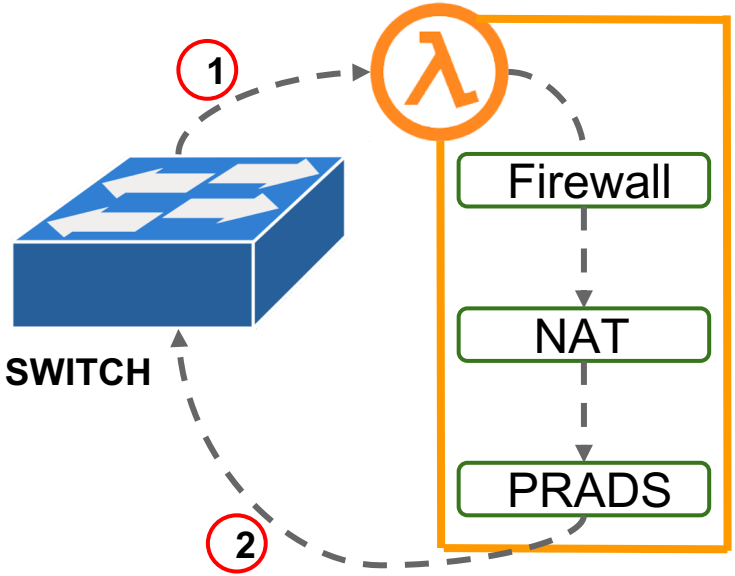


LENS

Locality-aware, Event-based NF Chaining System

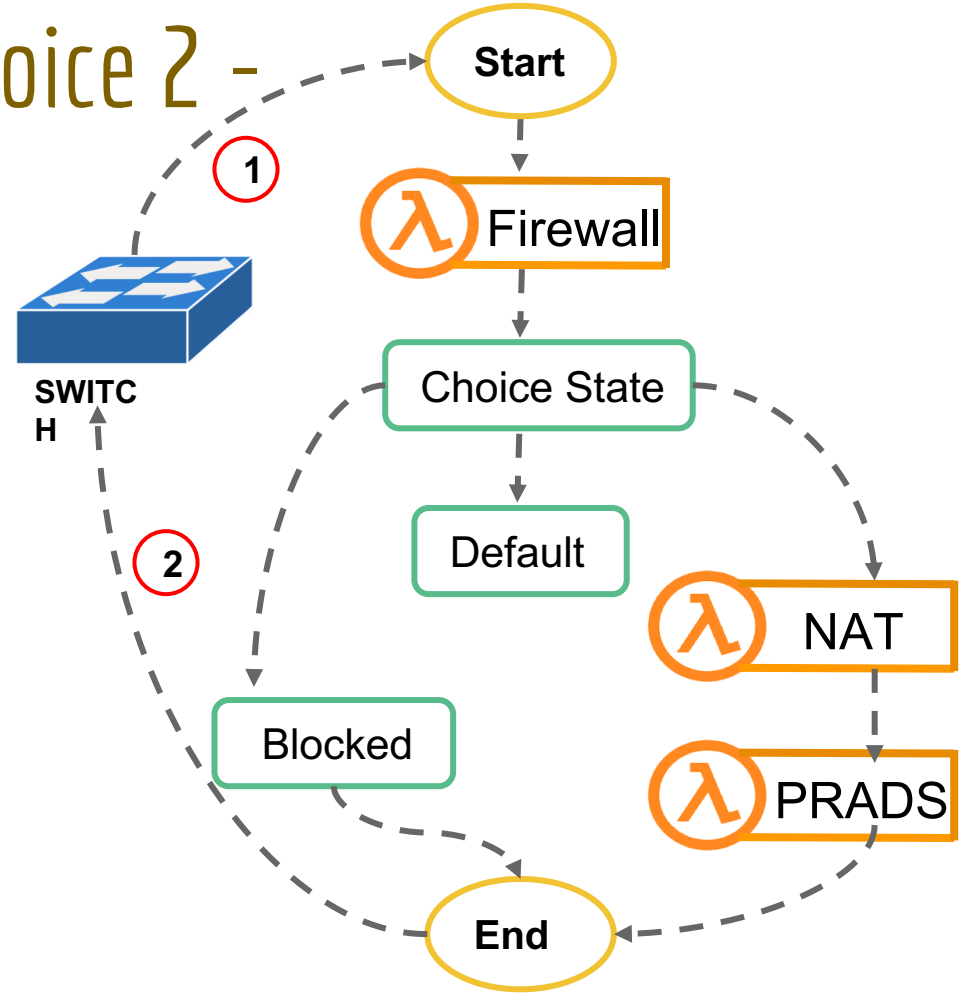
LENS Implementation Choice 1 - **All In One**

- Functionality of 3 middleboxes in single function
- Pros
  - Locality Aware
- Cons
  - One hot middlebox leads to unnecessary relaunch of all 3 middleboxes.
  - One middlebox corruption renders other middleboxes on same lambda instance unusable



LENS Implementation Choice 2 - Step Functions

- Interpose each middlebox lambda onto a node in step function
- Pros
  - Easy to model complex workflows
- Cons
  - Overhead in Lambda States and Transitions
  - Can not enforce locality



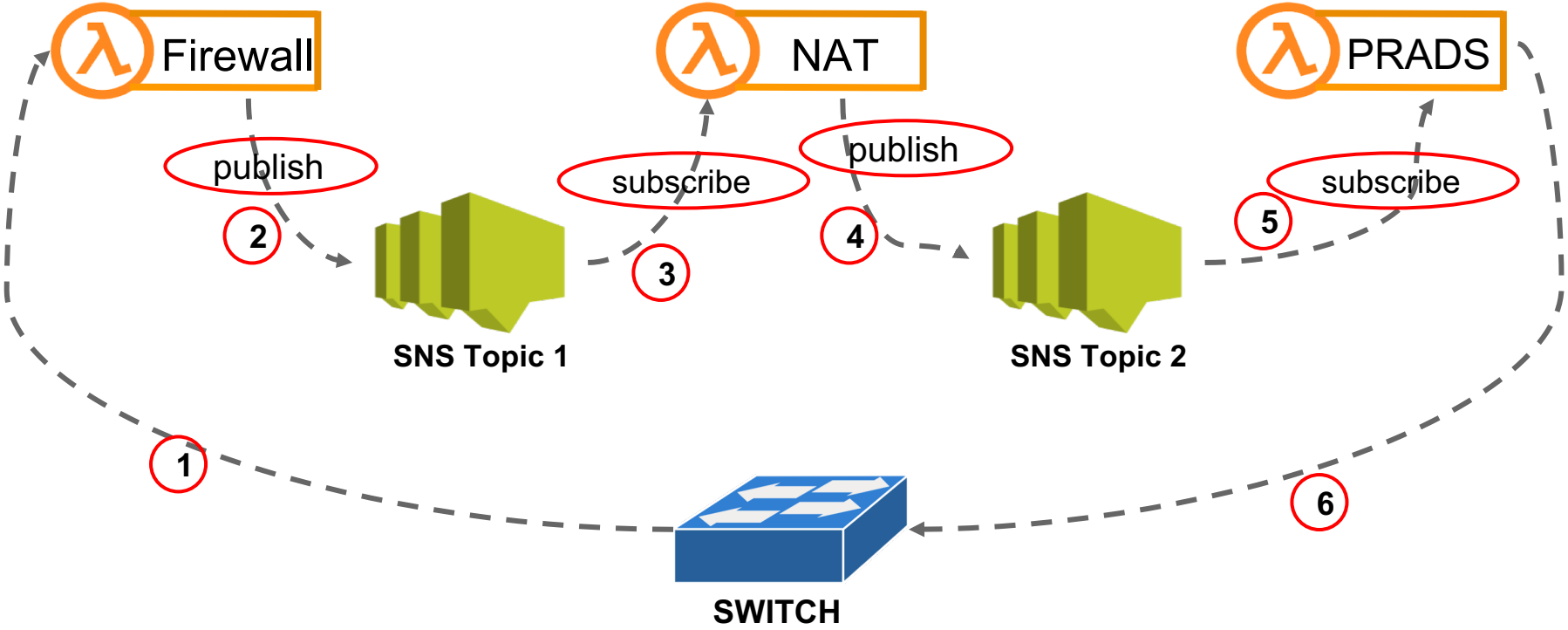
Implementation Choice 3 - **Simple Notification Service**

- Simple Notification Service (SNS)
  - Fast
  - Flexible
  - Push Notification Service
  - Send individual messages
  - Fan - out messages
  - Publisher - Subscriber Model

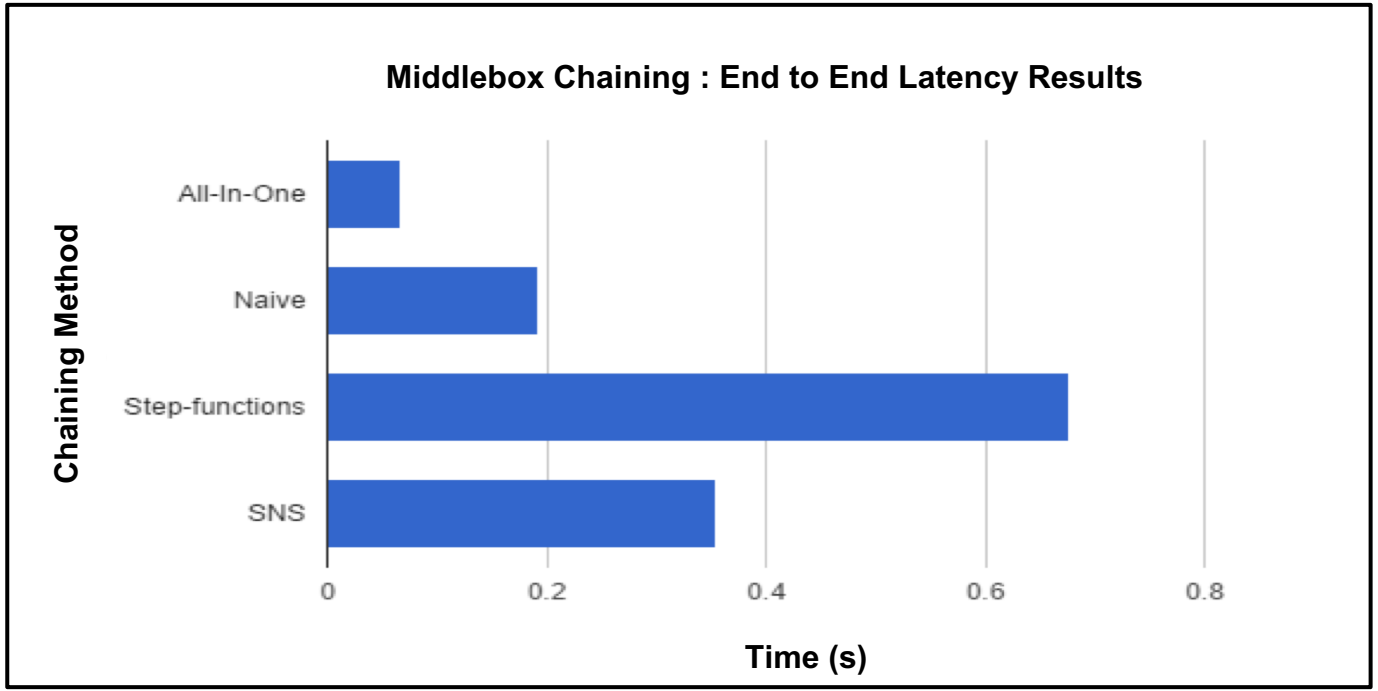
- Pros
  - Simplifies Event based handling

- Cons
  - Locality unaware

LENS Implementation Choice 3 - Simple Notification Service (SNS)



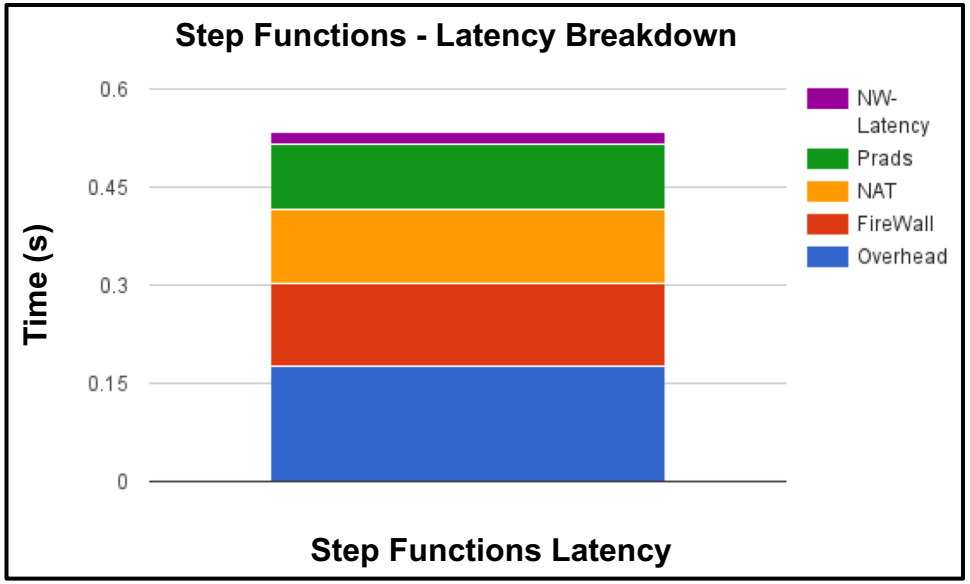
LENS Evaluation Results



LENS Evaluation Results - Analysing Step Functions

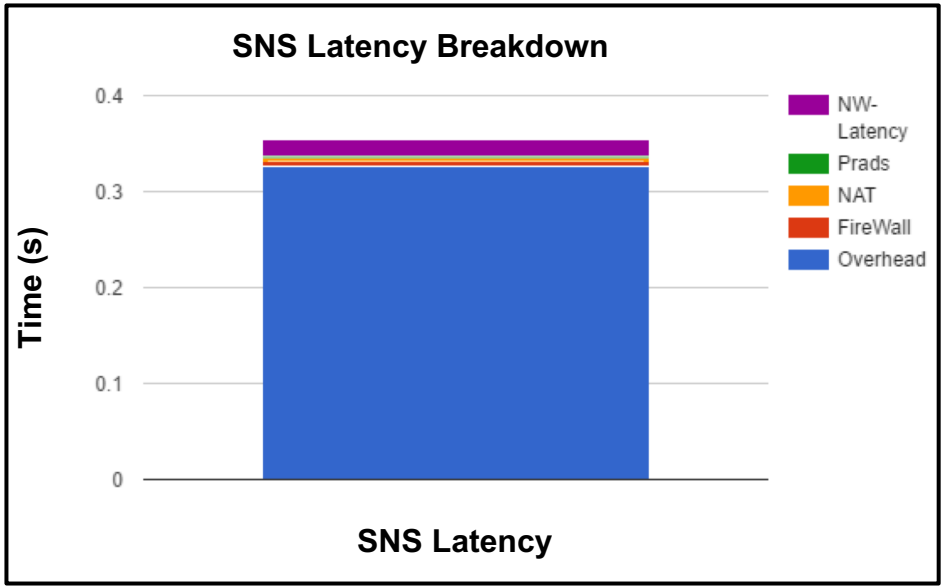
Total Latency = Network Latency + Lambda Execution Time + AWS Step Function Overhead

- ~100ms to execute
- ~3ms for Lambda Execution
- High setup cost
- AWS Step Function Overhead represents
  - State Transitions
  - Non-Task State time



LENS Evaluation Results - Analysing SNS Execution

- SNS
  - 92% overhead
- Overhead includes
  - Pub-Sub delay
  - Lambda Setup costs



Summary

- Implementing standalone NFs/middleboxes on Lambda is not a viable option
  - High latency and overhead

- Chaining middleboxes hides the high latency

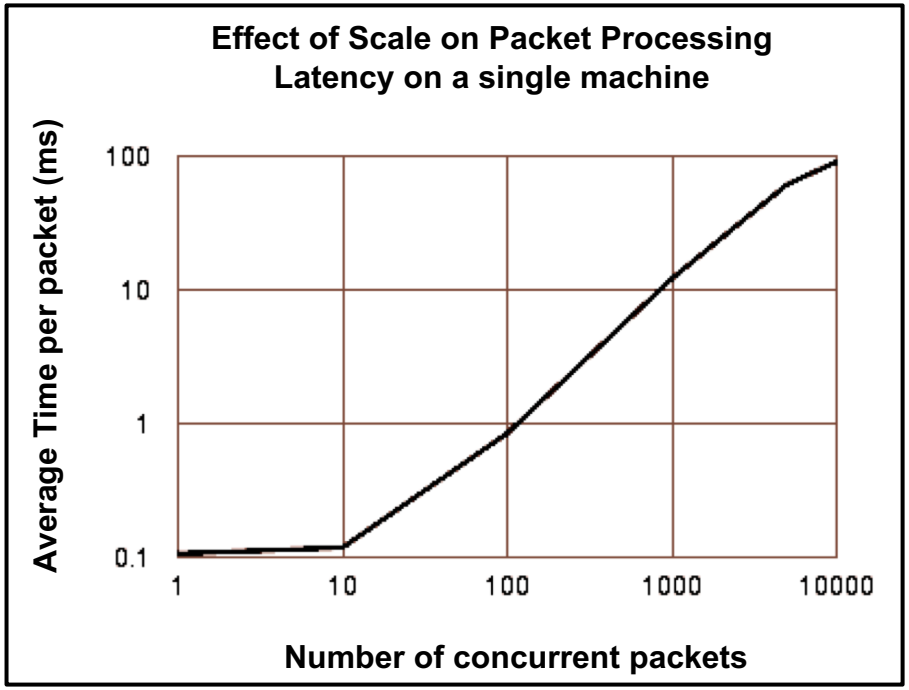
- After exploring various chaining methods
  - Services provided by AWS lambda are
    - Very restrictive
    - Have high overhead
  - Chaining is most beneficial in the All-In-One case
    - Provides locality
    - High memory footprint
    - Only suitable when all NFs scale equally

Questions?

Graph Slides

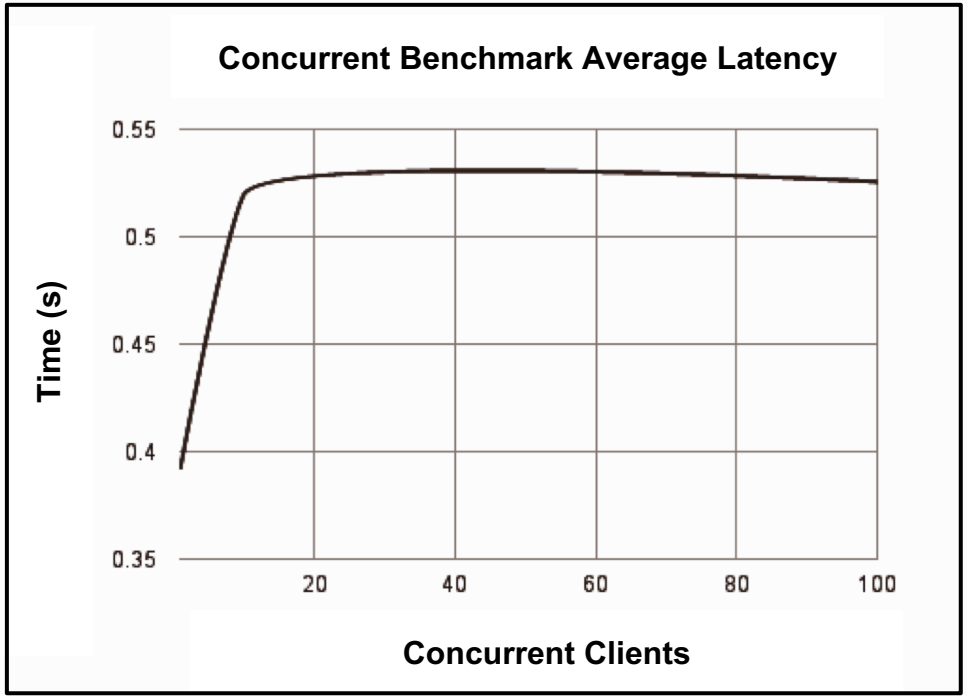
Graph 1

- Plot illustrating average NAT response time with concurrent clients
- Highlights the problem of scaling on a single machine
- Motivation for investigating a an implementation in a distributed setting



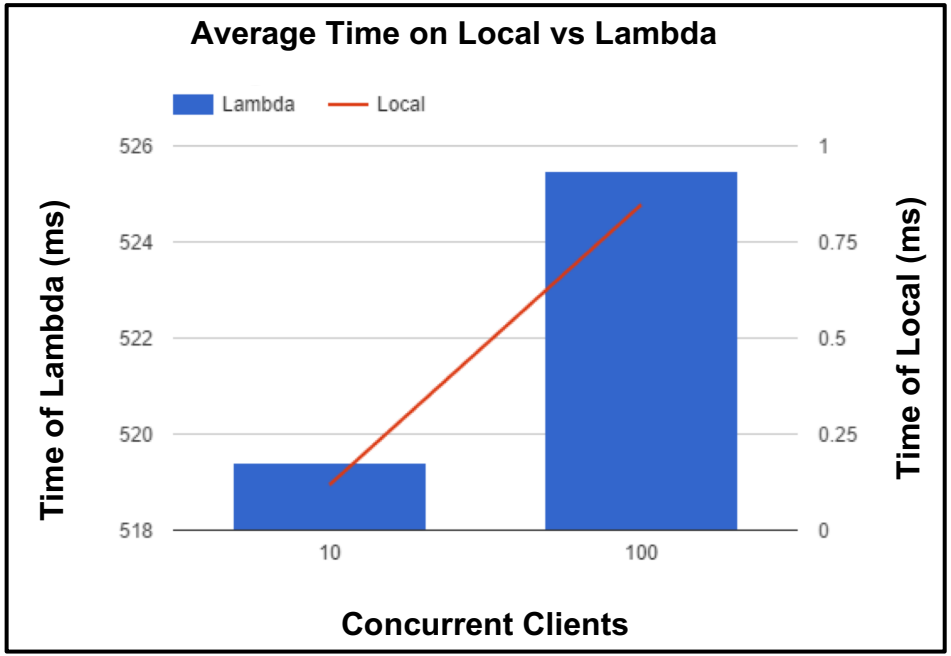
Graph 2

- NAT implementation on AWS lambda scales well
- AWS lambda: maximum parallel executions set to 100
- Latency is mostly unaffected
- High end to end latencies



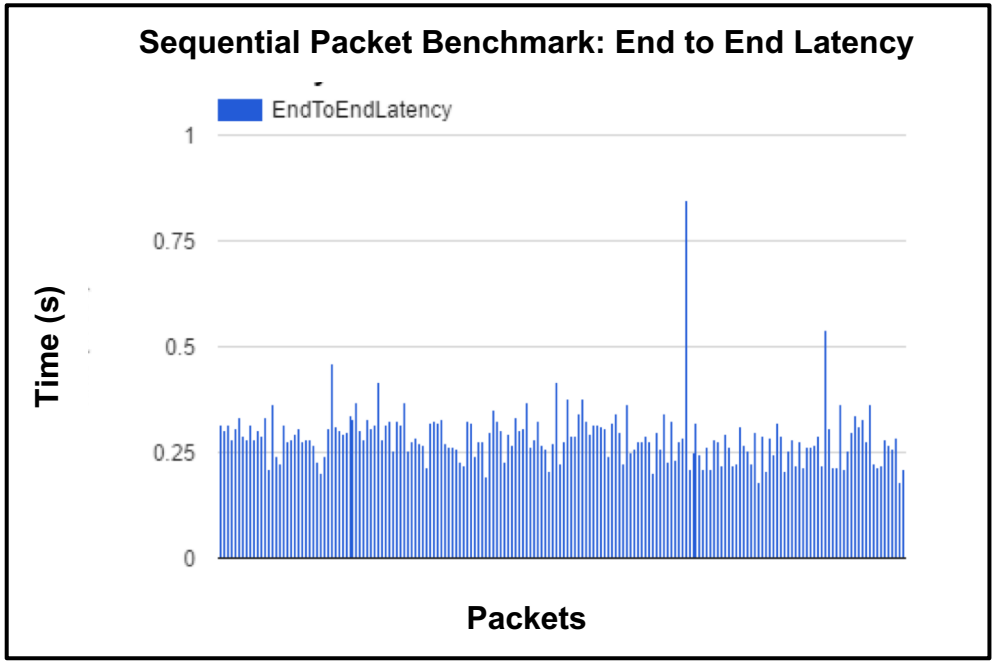
Graph 3

- Comparison between lambda and local NAT
- Very higher rate of change of local latency
- Lambda is unaffected
- Lambda addresses the scaling problem
  - At the cost of very high end-to-end latency
  - Further analysis



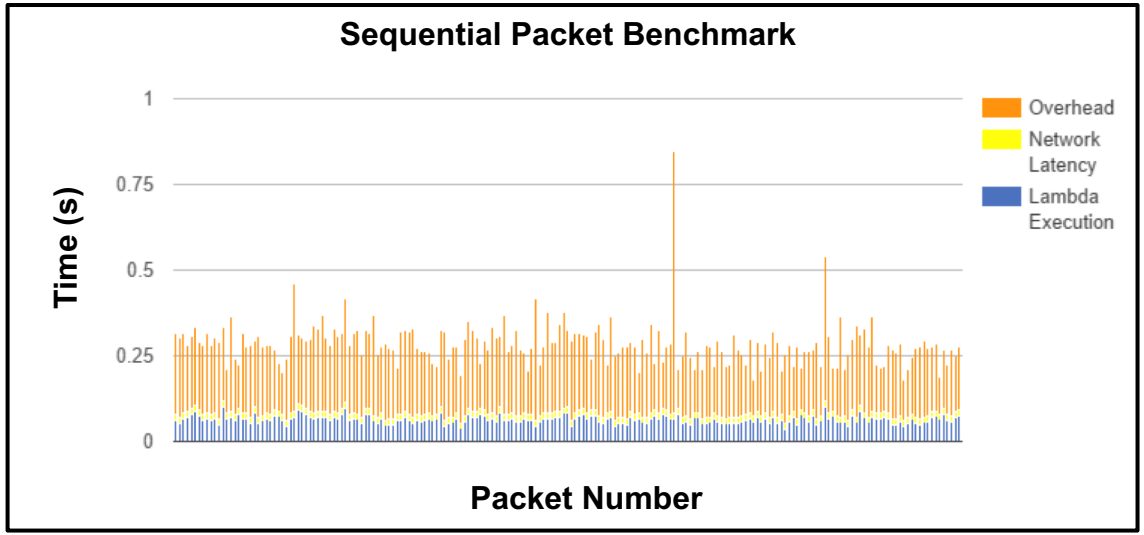
Graph 4

- Distribution of NAT latencies for 100 sequential packets.
- Need to breakdown the latency into known components
  - Network Latency
  - Lambda Execution
  - AWS overhead



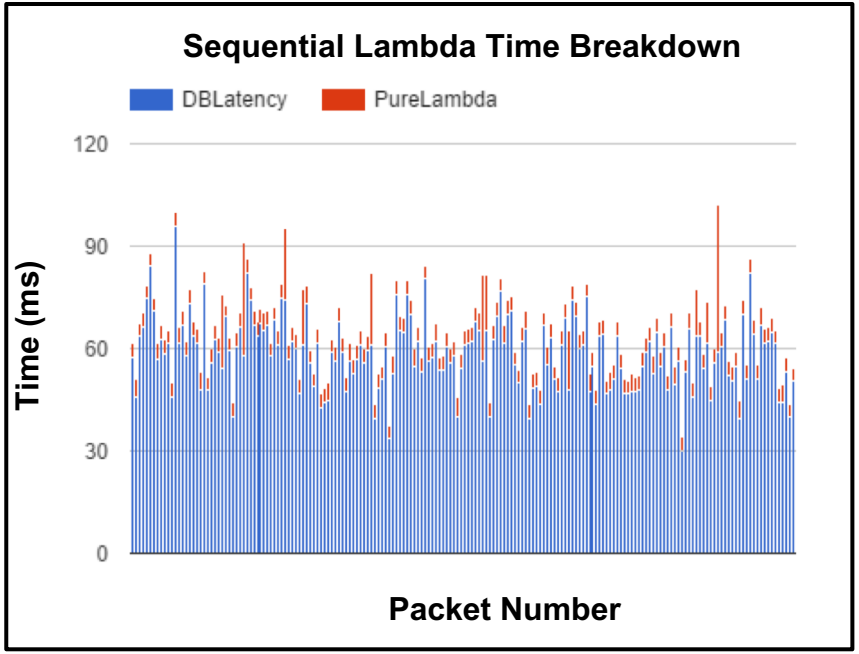
Graph 5

- Distribution with the Lambda, Network and AWS overhead components
- High cost for launching lambda instances



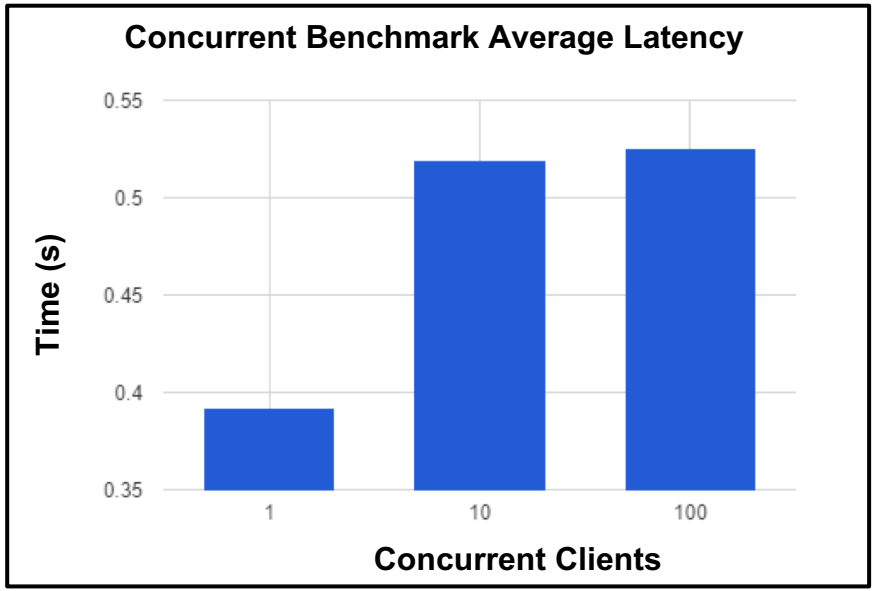
Graph 6

- Breakdown of Lambda Execution Time
- State operations take higher fraction of time
- DynamoDB Update operations are costly
  - Provides High Consistency



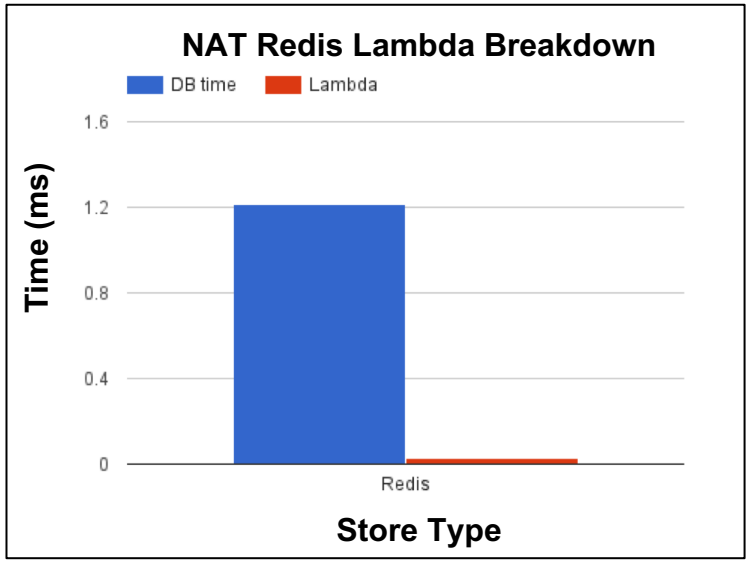
Graph 7

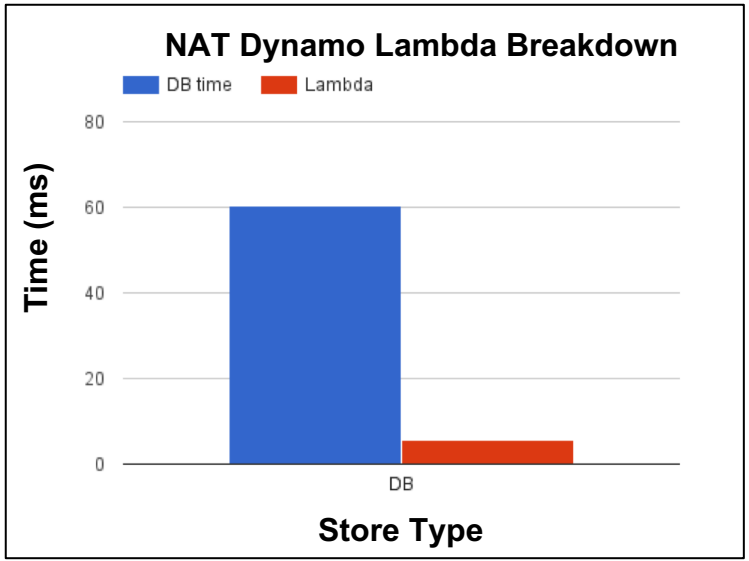
- Illustrating the scaling property provided by the lambda architecture
- Similar trend observed for Firewall and PRADS middleboxes
- Average latency remains mostly unaffected



Graph 8

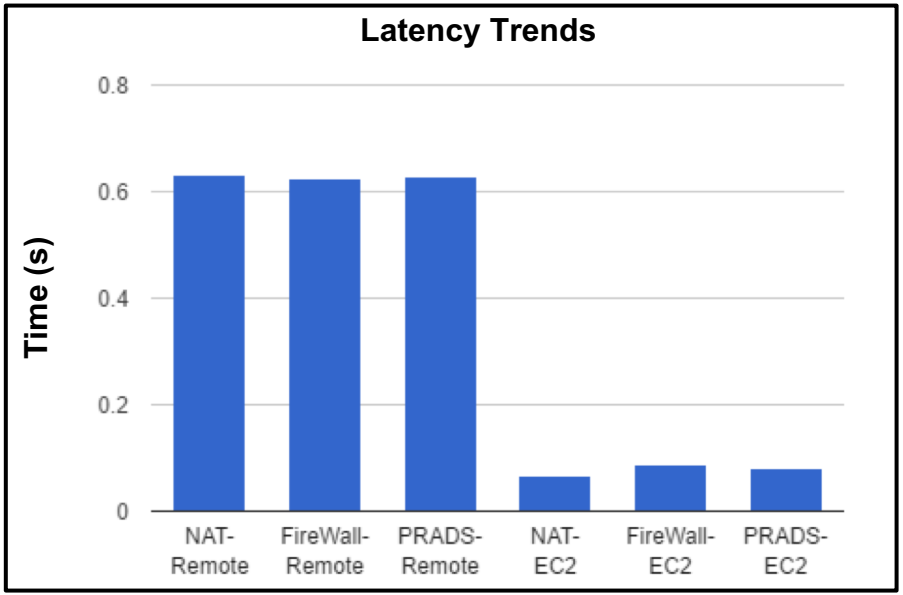
- Use of in-memory redis state operations provides much lower latencies
- The state mapping will not be persistent
  - Backup state in the DynamoDB
  - Replication in Redis





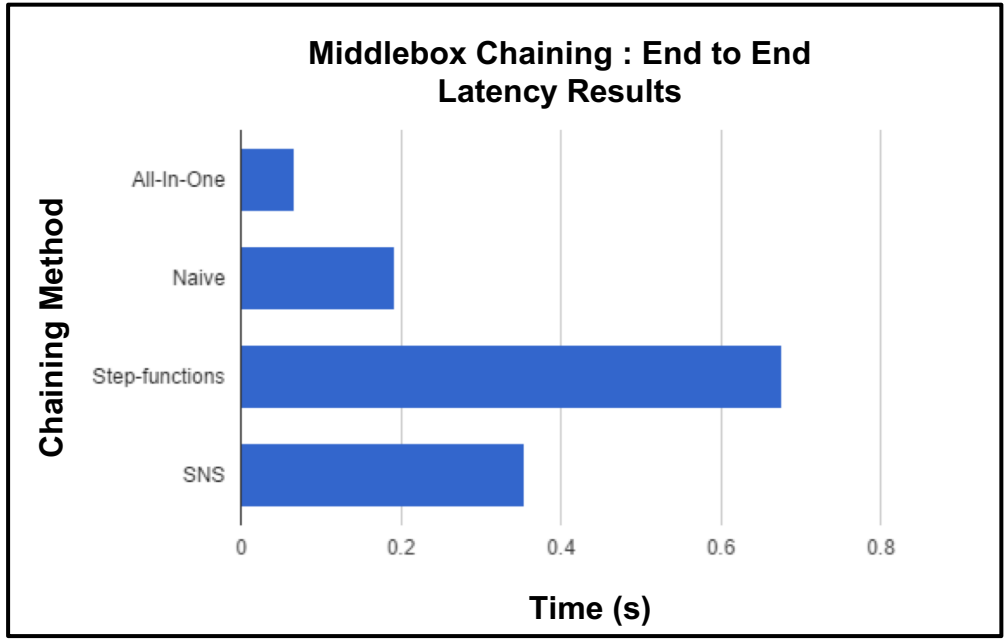
Graph 9

- Running the benchmarks from an EC2 instance
- Avoids the Wide Area Network Latency by calling an internal API and Lambda trigger
  - Lower Network Latency
  - Lower AWS Overhead
- Latency characteristics are comparable among the middleboxes



Graph 10

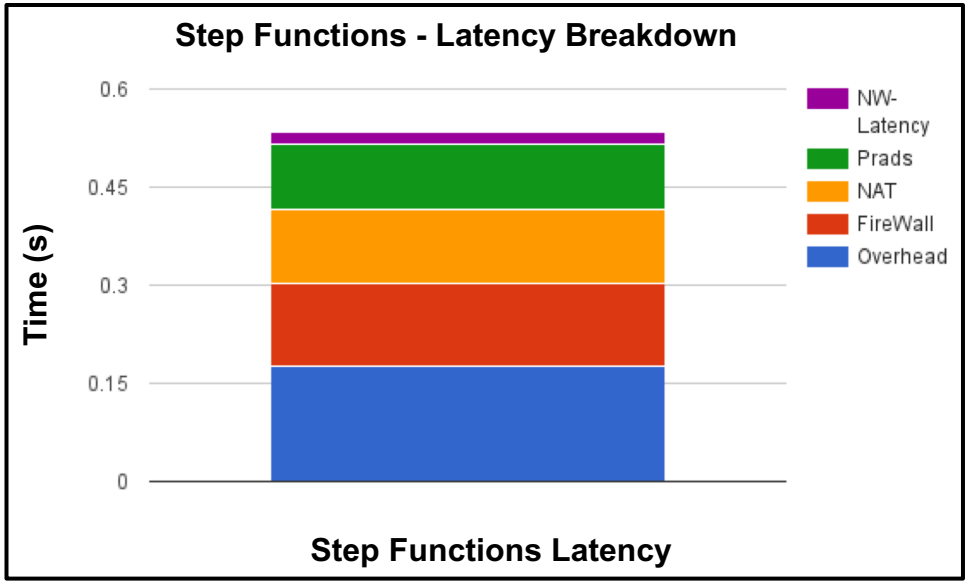
- AWS chaining constructs have very high latency
- All-In-One illustrates low overhead
  - 1 Lambda instance
- Naive launches 3 lambdas



Graph 11

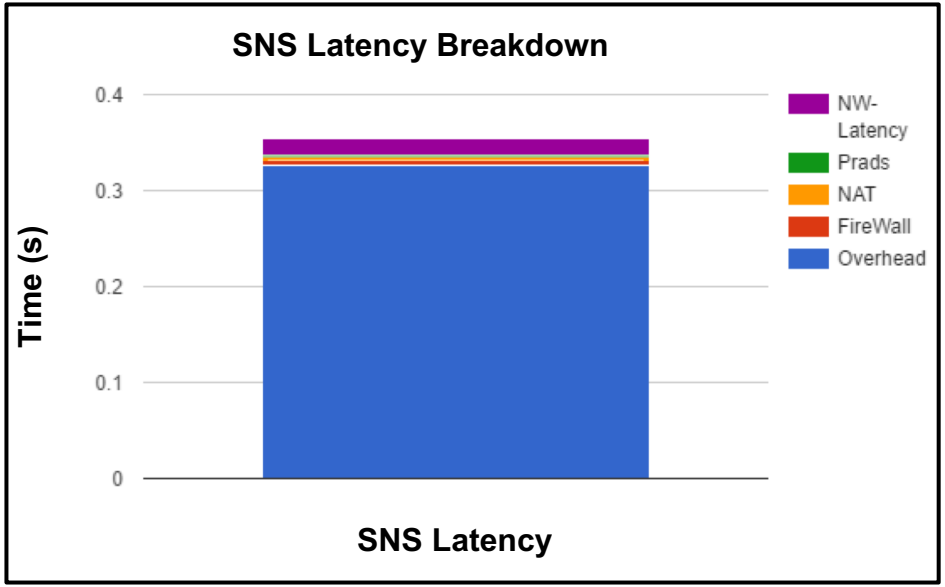
- States executing lambdas
  - ~100ms to execute
  - ~3ms for Lambda execution
  - High setup cost

- Overhead represents
  - State Transitions
  - Non-Task State time



Graph 12

- SNS
  - 92% overhead
- Overhead includes
  - Pub-Sub delay
  - Lambda Setup costs



Backup Slides

Middleboxes

- The term middlebox was coined in “1999” also now known as network functions.
- Systems that examine and modify packets and flows
- Ensure security & improve performance in enterprise and service provider networks.
- Recent Trend: **Network Functions Virtualization (NFV)**
  - Replace dedicated hardware appliances with software-based network functions running on generic compute resources.

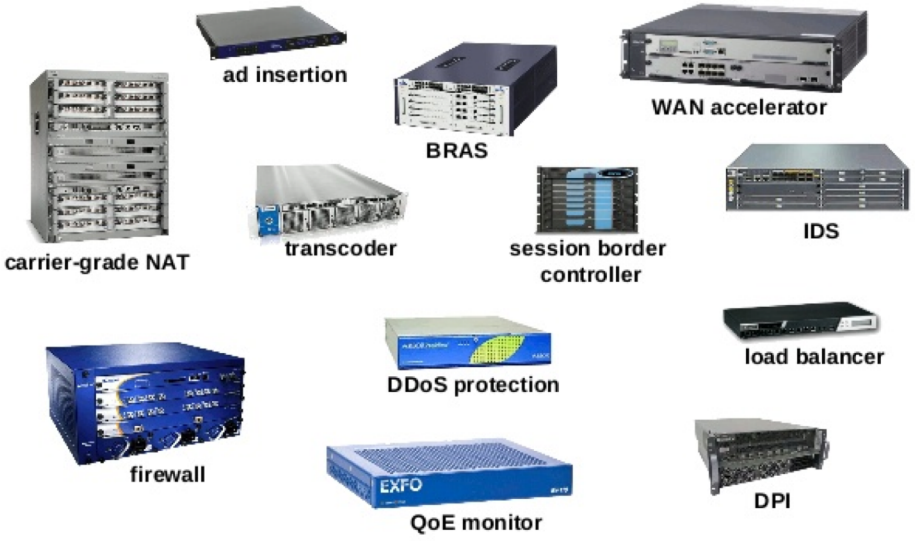


Figure 1: Various Middleboxes

Lambda Architecture?

- Architecture for real-time & serverless computing
  - Execution of stateless functions

- Public cloud lambda offerings
  - AWS Lambda
  - Azure Functions
  - Google Cloud Functions

- Generic Advantages
  - Elimination of server management
  - Continuous Scaling on demand
  - High-availability
  - Event-based triggering mechanism
  - Developer just writes event-handling logic

AWS Lambda

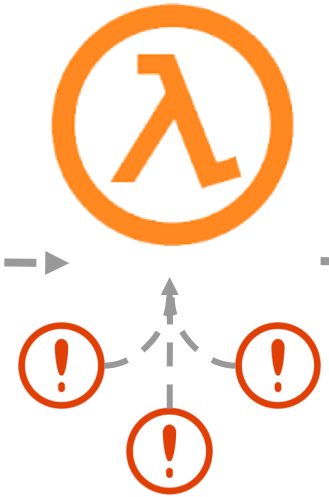
Microsoft Azure

Google Cloud Platform

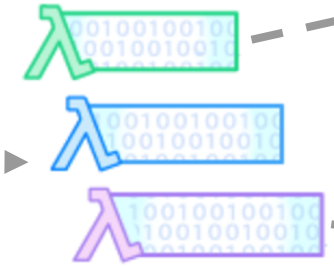
Lambda Architecture

- How it works ?

Upload your code to Lambda



Setup your code to trigger from other cloud services, HTTP endpoints or in-app activity



Lambda runs your code only when triggered, using only the compute resources needed

Pay just for the compute time used

Problem Statement and Motivation

- Evaluate the viability of implementing Network Middleboxes on the Lambda architecture based on
  - Scale
  - Performance
  - Consistency
  - State Maintenance

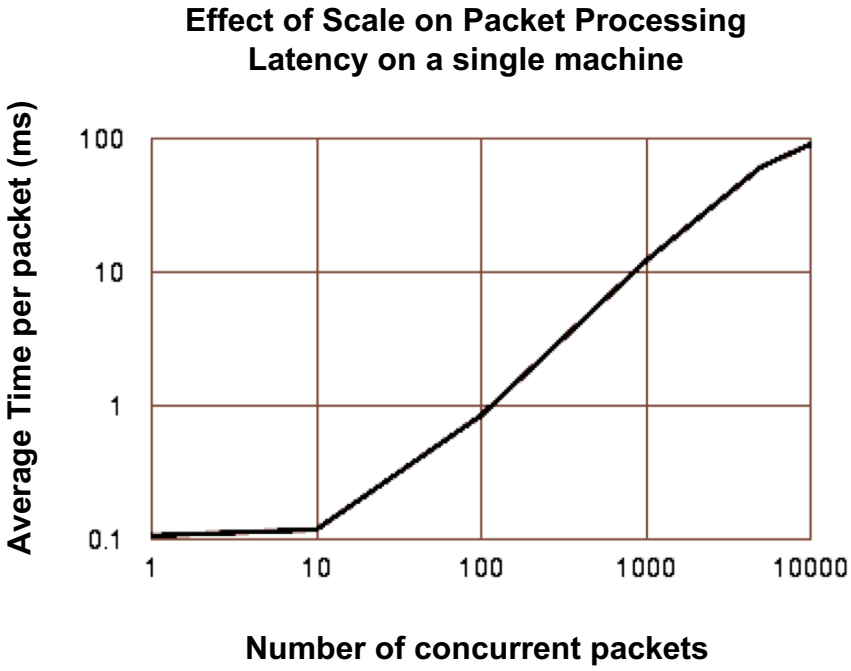
- Implementing Middleboxes on a single machine does not scale well.

- Use the benefits of scaling and high availability provided by Lambda Architecture

- Obtain better performance characteristics for Middleboxes at scale

Motivation

- Performance degrades with the number of concurrent packets
- Natural way to handle scale is to launch more concurrent instances
- Network middleboxes are on the critical path
  - Handle low latency
  - Handle concurrent connections
- Each stage of processing in a middlebox chain must
  - Handle load
  - Handle hard/soft failures



Solution Approach

- Model middleboxes as event-based lambda functions
  - Stateful middleboxes
    - Eg. NAT, Firewall
    - Fetching and updating state on every packet
    - Use external stores for maintaining state
      - Trade-off between latency and consistency of the state
    - Stateless operations performed in the lambda handler
  - Stateless middleboxes
    - Fits the lambda framework naturally
    - All the middlebox box handled in a function

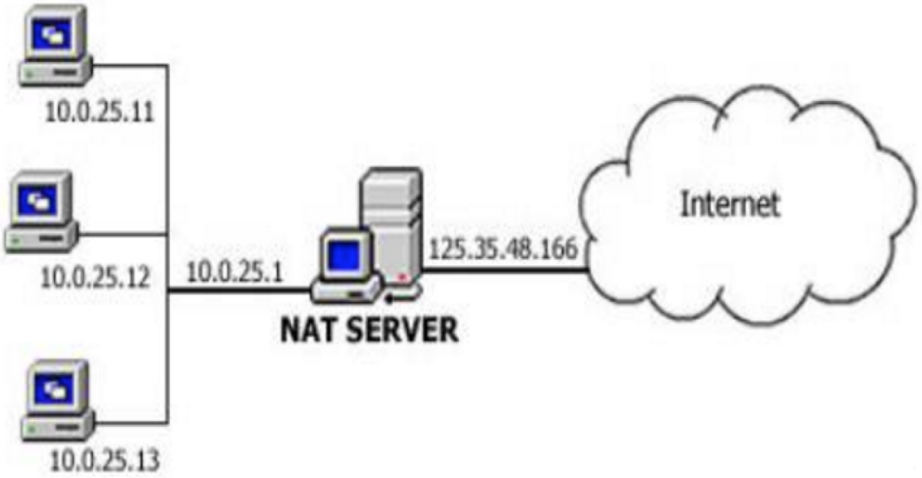
- Investigate the idea of middlebox chaining
  - Reduce communication time between middleboxes
  - Current implementations lead to multiple hops on the network

Concise Result

- We would be presenting some results based on our Lambda Middlebox applications
  - Preliminary investigation among public clouds
  - Choice of storing the stateful information
  - Baseline performance characteristics for the middleboxes
    - NAT
    - Firewall
    - PRADS
  - Breakdown and analysis of the total client observed end-to-end latency
  - Overheads and effect of network latency
  - Effect of chaining the middleboxes using various techniques
    - Naive
    - All in one lambda
    - Use of a graph-based step functions topology
    - Notification based triggering mechanism

Design (Middleboxes)

- Network Address Translation (NAT)
  - Remaps IP addresses across private and public IP address space
  - Design a 2 way mapping to perform lookups
  - Lambda functions would
    - Extract IP address from packet
    - Lookup for IP to external store
    - Modify the IP address in packet
  - Mapping is stored externally
    - Database
    - In-Memory Cache



Design (Middleboxes)

- Firewall
  - Inspects the IP address/port
  - Mapping contains rules for filtering
  - Lambda functions would
    - Extract IP/port fields
    - Lookup for filtering rules
    - Block malicious packets

- PRADS
  - Gathers information on hosts/services
  - Mapping stores relevant fields
  - Lambda function would
    - Extract IP/port fields
    - Store host fields

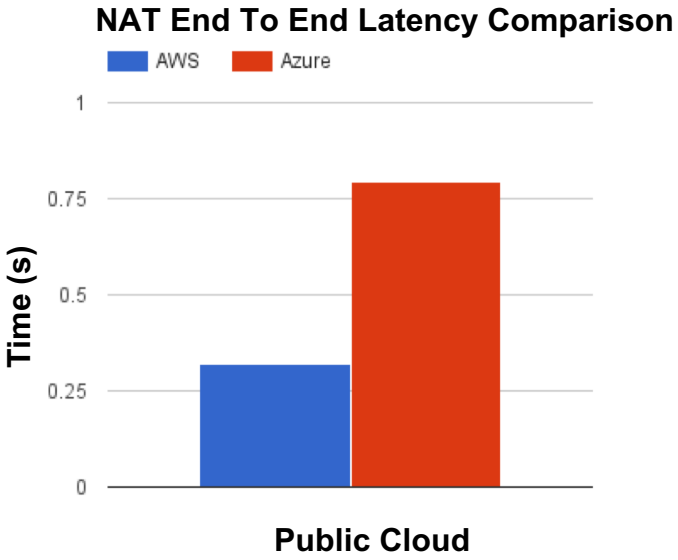
Generic workflow

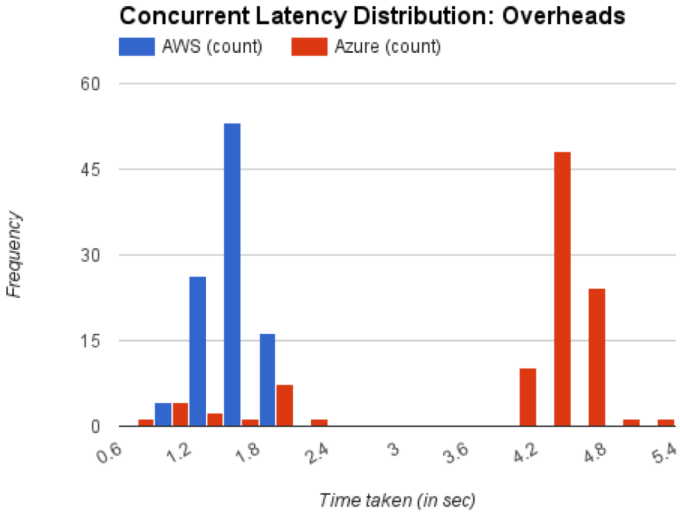
- Parse incoming packets
- Update/Retrieve state information
- Middlebox specific action

Baseline Latency Results

We pick AWS Lambda to implement the middleboxes discussed

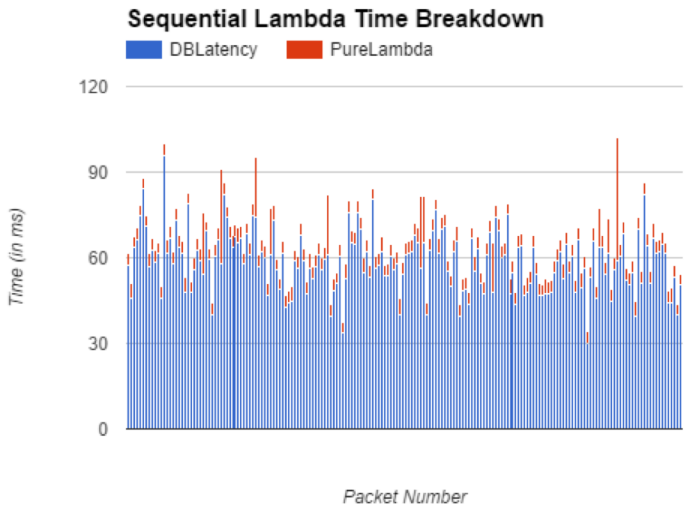
Trends for NAT implementation on AWS-Dynamo and Azure-SQL

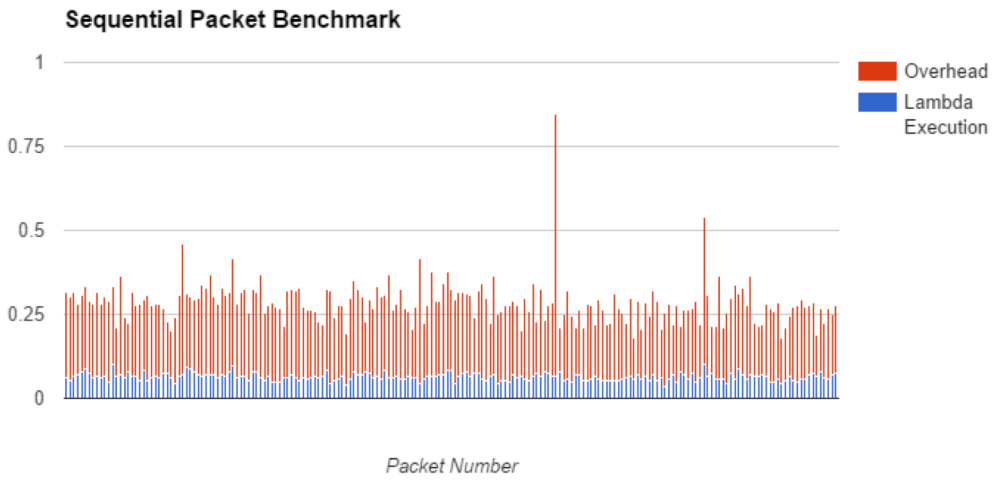




Breakdown of the Latency and Lambda Execution

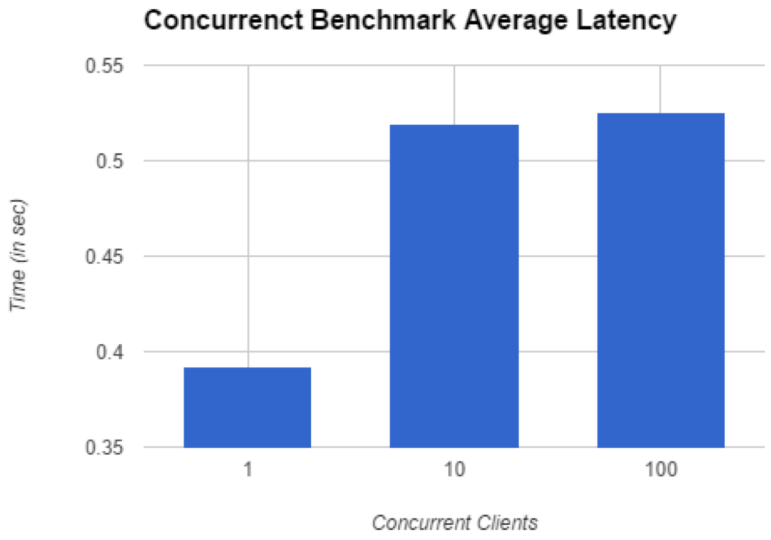
- Execution of lambda functions is a small fraction of the latency





Concurrent Benchmark Behaviour

- The workload launched concurrent clients
- AWS lambda has a limit of maximum 100 parallel executions
- Small variation in the average time taken for 10 and 100 clients
- Illustrates the scaling achieved by lambda.
    - Contrasts with the single machine trend

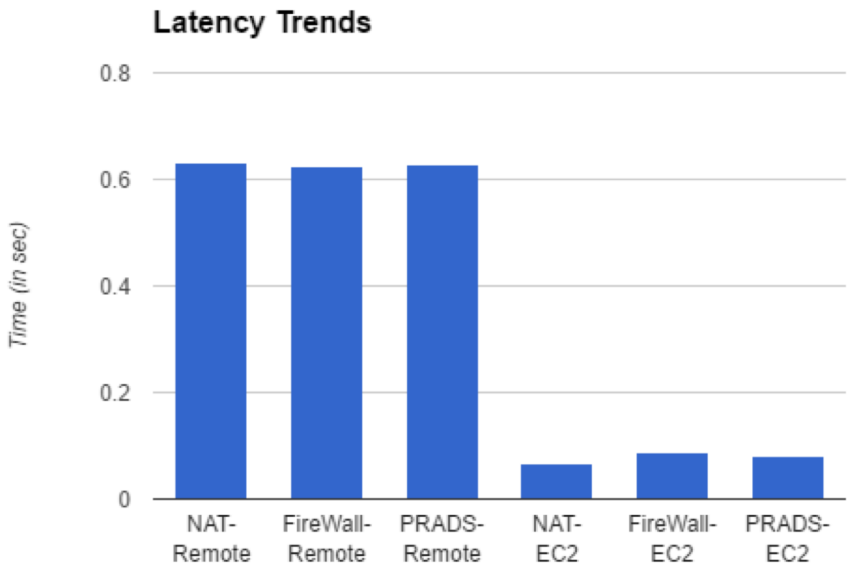


Effect of External Store and Network Latency

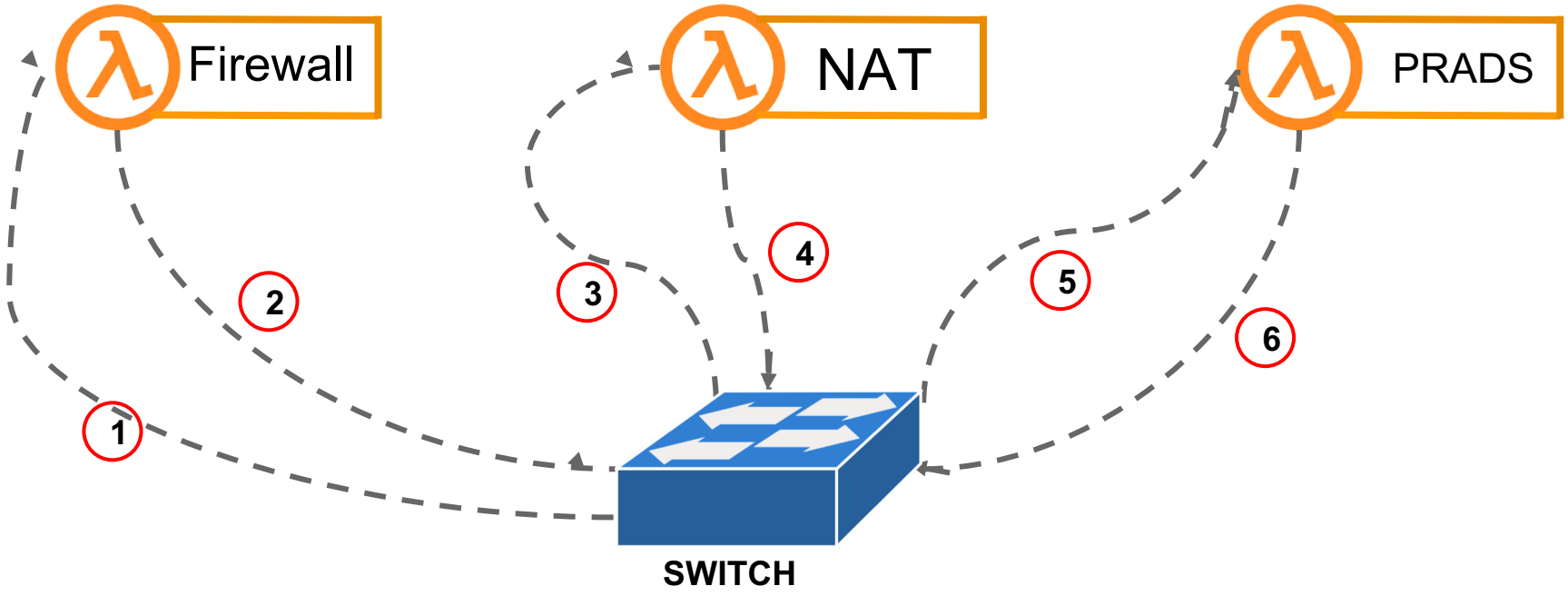
- Dynamo DB persists the state to leading to higher overhead
- Storing state in memory leads to faster lookups and updates
- This is not persistent and is lost on redis-server crash
- Majority of the lambda execution time is spent in state lookups

Effect of Network Latency

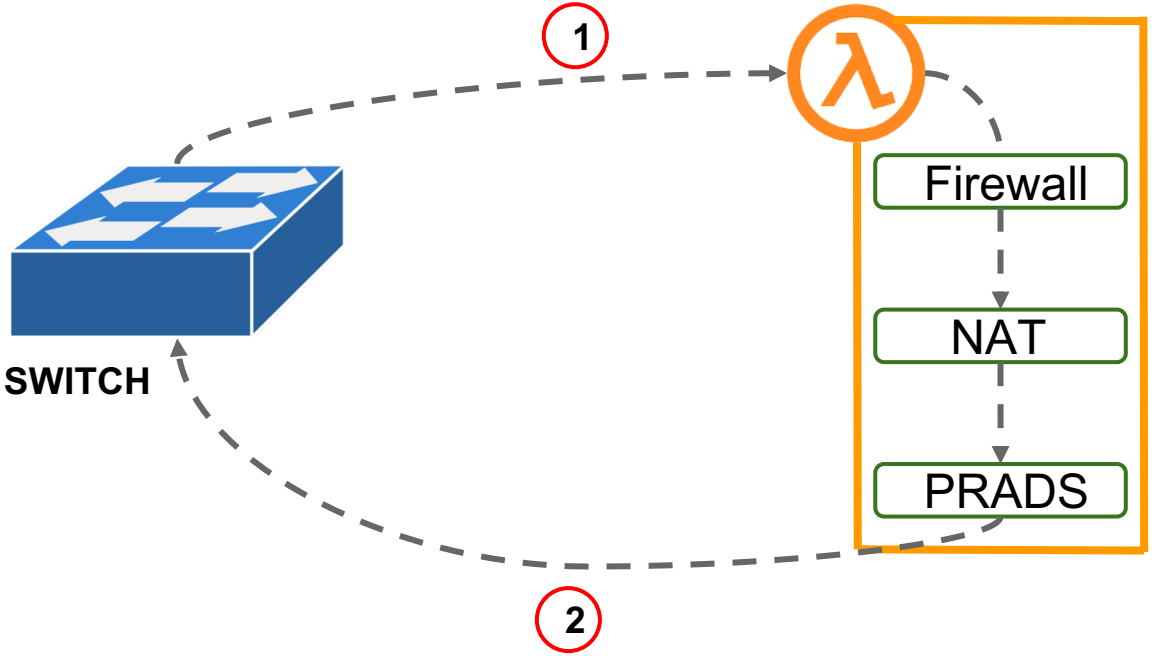
- Sending requests from the EC2 datacenter avoids the link over Wide Area Network.
- Network latency is a major component of the remote request latency



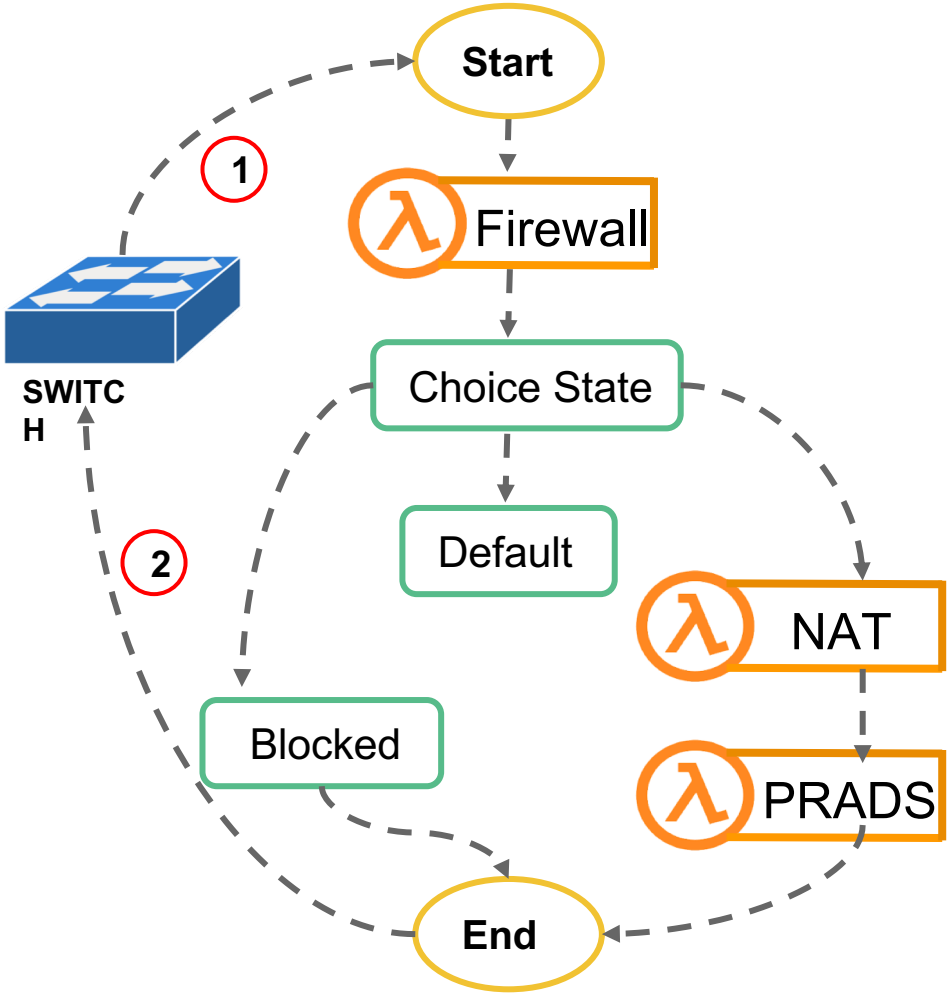
Design I (Lambda - Naive impln)



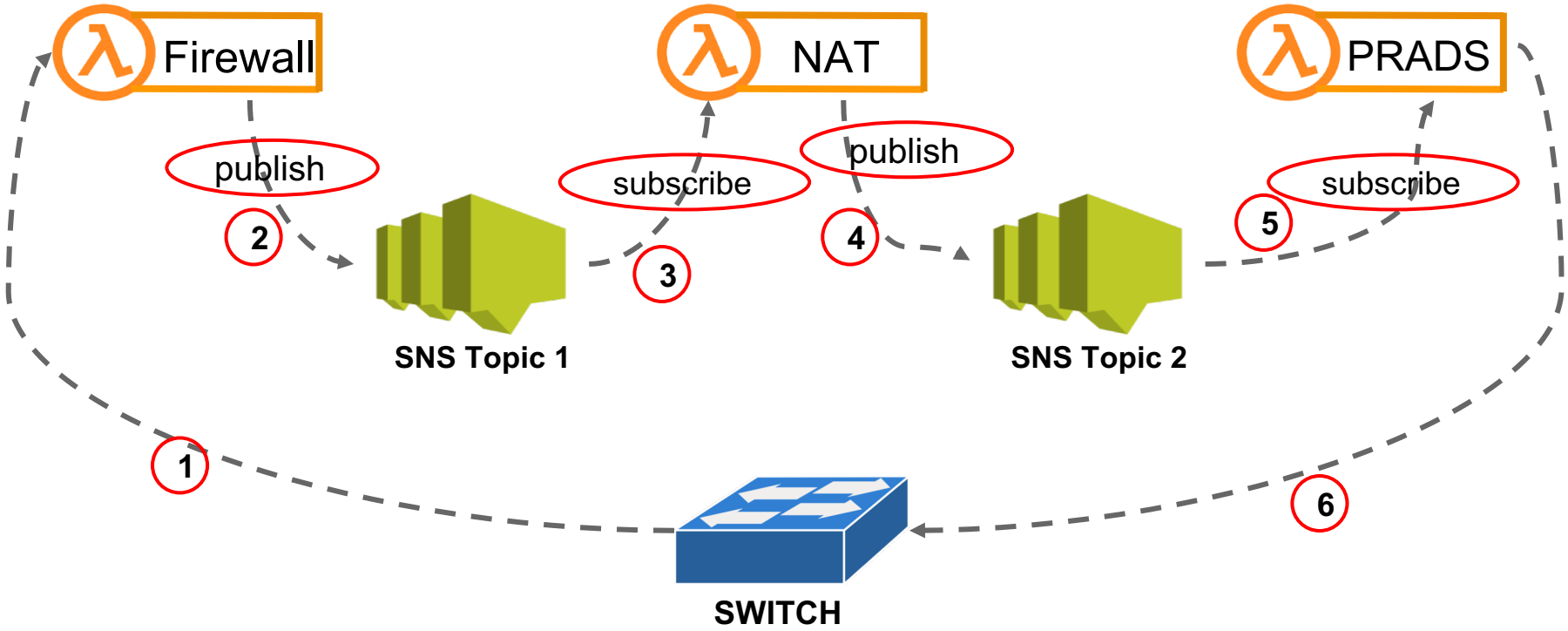
Design (Lambda - All In One)



Design (Step Fns)

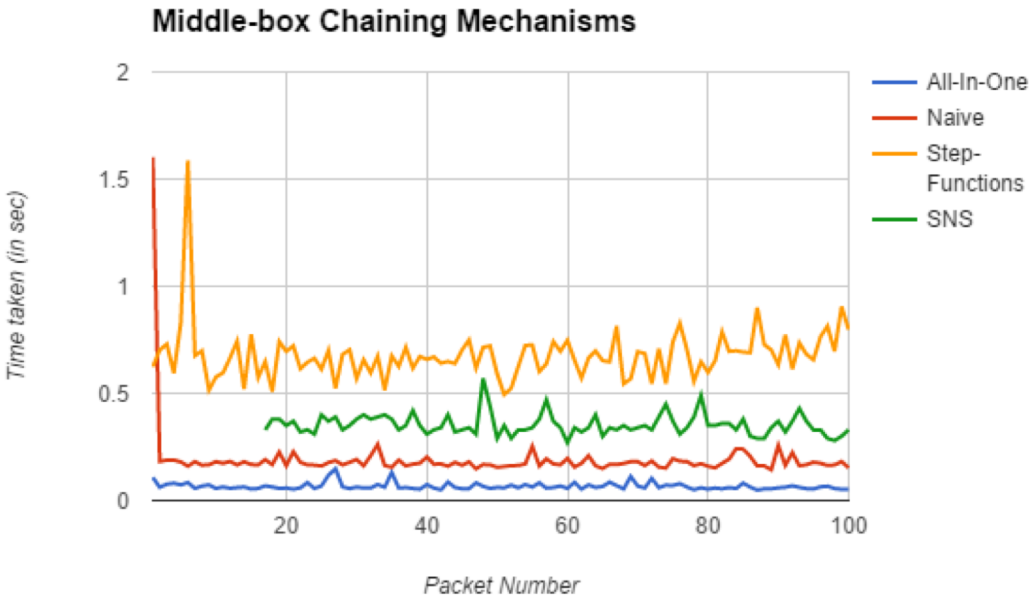


Design (SNS - Pub/Sub model)



Design (Lambda calling Lambda) - Future Work

Comparison between middlebox chaining



Summary

Conclusion