CESSNA: Resilient Edge Computing

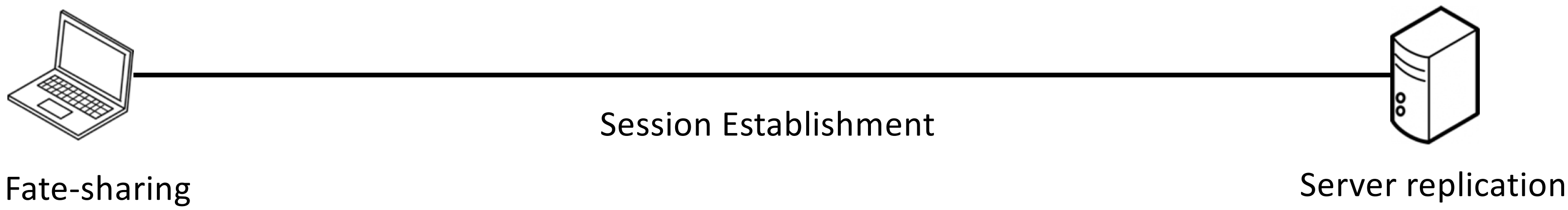
Yotam Harchol

UC Berkeley

Joint work with: Aisha Mushtaq, Murphy McCauley, Aurojit Panda, Scott Shenker

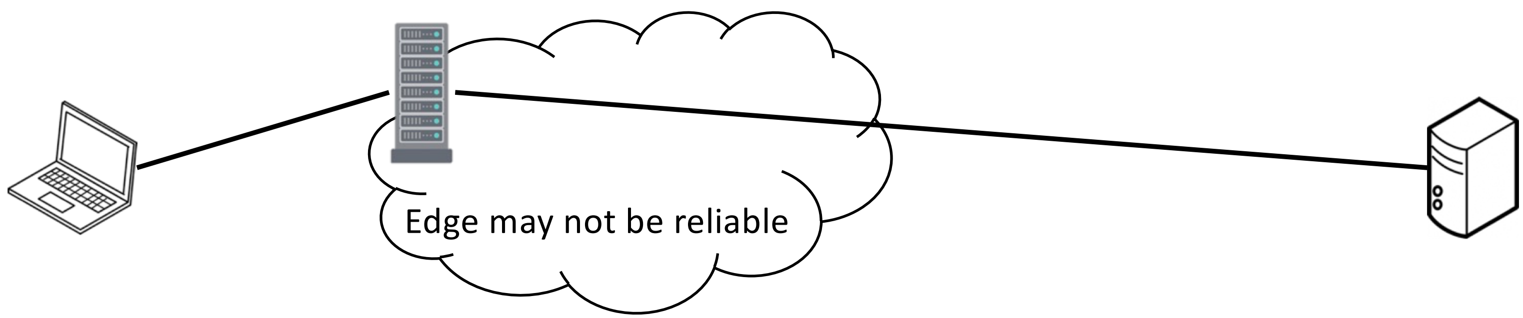
SIGCOMM MECOMM Workshop, Budapest, Hungary, August 2018

Client-Server Computing



Client-Edge-Server Computing

Session goes through the edge



Edge application can be stateful
State depends on packets from both sides ***and*** their interleave ordering

Problem: How to maintain ***correctness*** of the state at the edge, under failover / mobility

Examples for Stateful Edge Applications

Compression
at the edge

Video
conferencing*

Online
gaming

Data aggregation
(e.g., for IoT)

* Control channel is stateful, video channel may not be

Goals

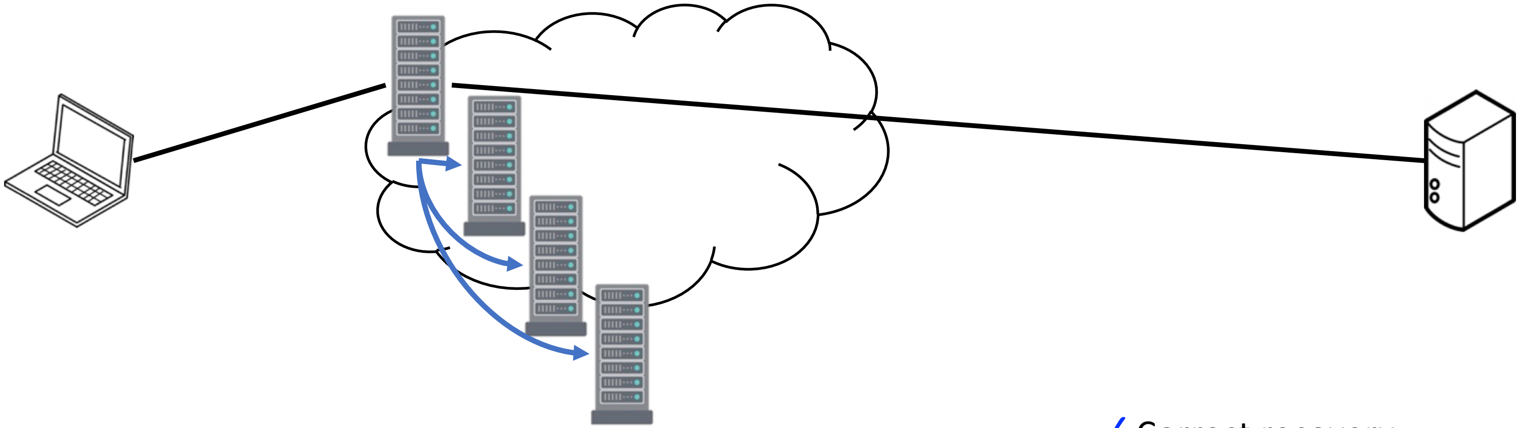
Correct Recovery
- New edge “sees” the same sequence of messages
- Transient “stall”

Survivability
- Arbitrary # of lost edges
- Edge failure never kills session

Client Mobility
Recovery may be needed at a remote edge

High Throughput
Edge should provide high throughput

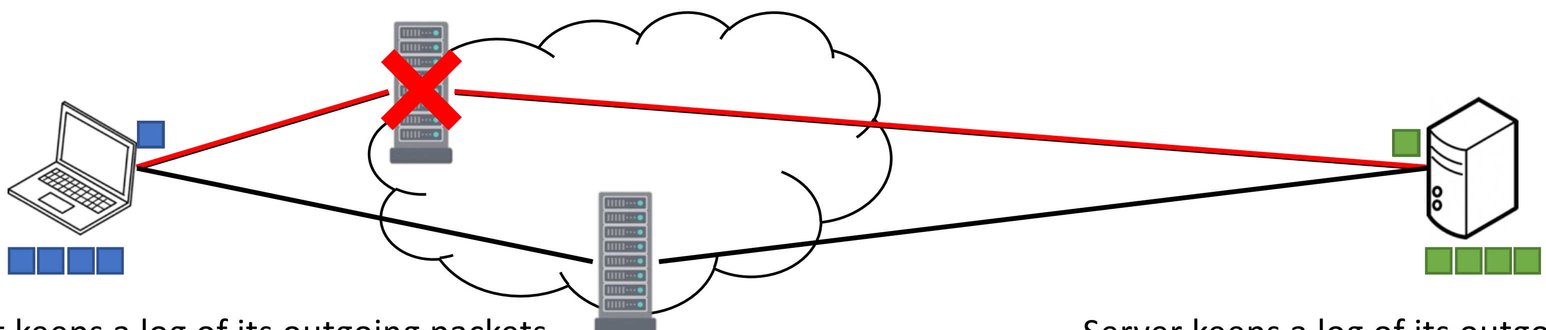
Strawman Solution #1: Replication



Edge is replicated
➔ Must have multiple hot backups, actively running and consistently updated
➔ Not applicable for client mobility

✓ Correct recovery
✗ Survivability
✗ Client mobility
✗ High throughput

Strawman Solution #2: Message Replay



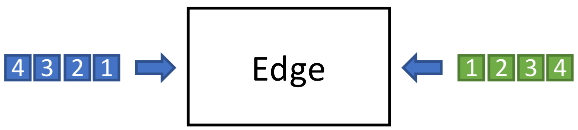
Client keeps a log of its outgoing packets

Server keeps a log of its outgoing packets

Problem 1: Packet logs may become very long ➔ can use periodic snapshots
Problem 2: Need to know the replay order between client and server packets ➔ ??

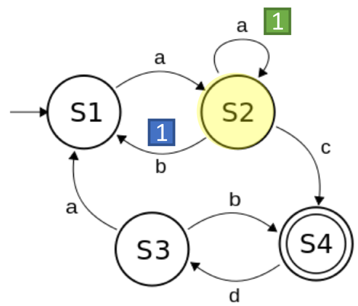
✗ Correct recovery
✓ Survivability
✓ Client mobility
✓ High throughput

The Challenge of Interleave Ordering



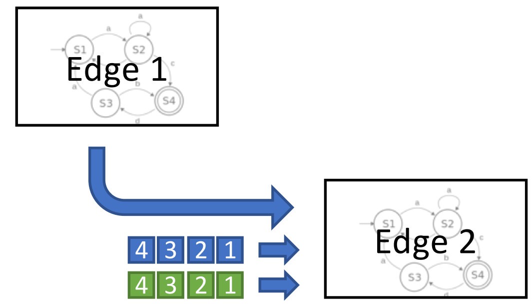
Messages arrive at the edge at two different sockets, simultaneously

Multiple possible ordering sequences of messages



The edge is a state-machine - Each packet changes the state (state transition)

Multiple ***correct*** states we could be at after receiving more than one message



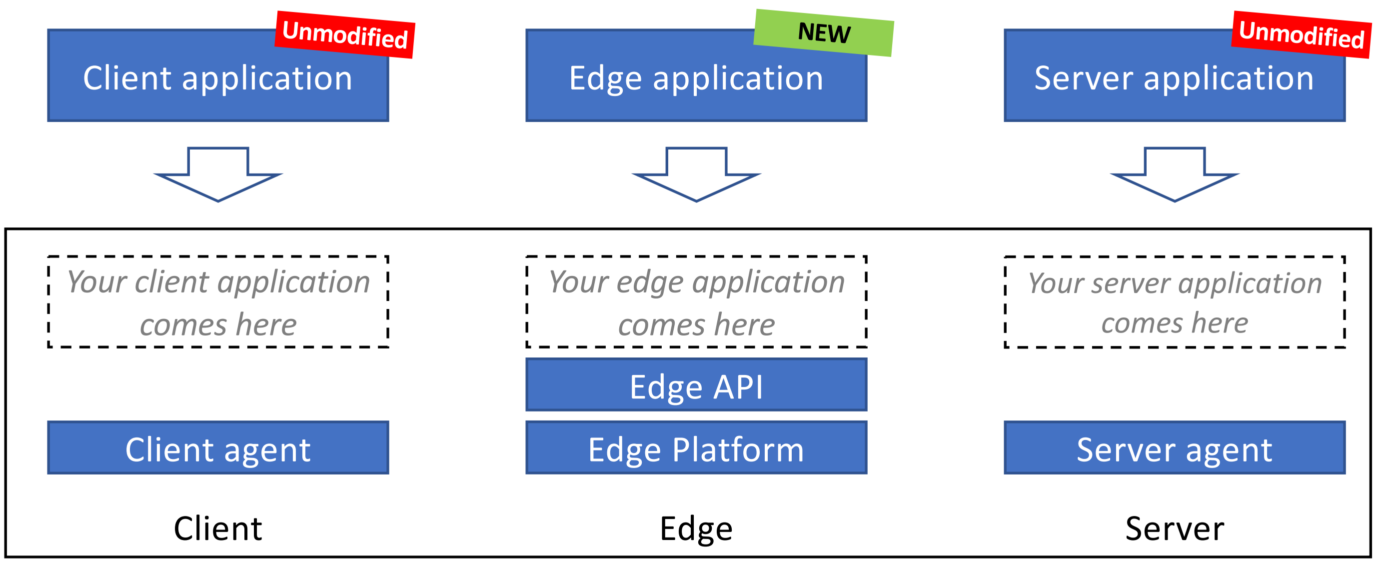
Faithful Replay: We want to replay messages in the exact same order

Exactly the same state traversal order

Exactly the same correct state

CESSNA –
Client-Edge-Server for Stateful Network Applications

A software framework for running resilient edge applications



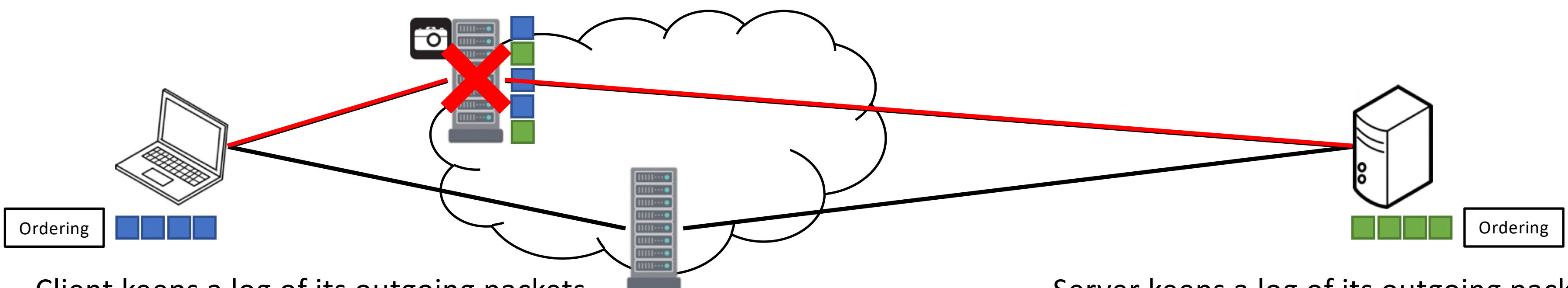
CESSNA Framework

Assumptions:

1. Edge application instance per client-server session
2. Deterministic edge application: no real randomness, no multithreading within an instance

CESSNA

Edge tracks ordering as it handles packets
Attaches ordering information to outgoing packets



Client keeps a log of its outgoing packets

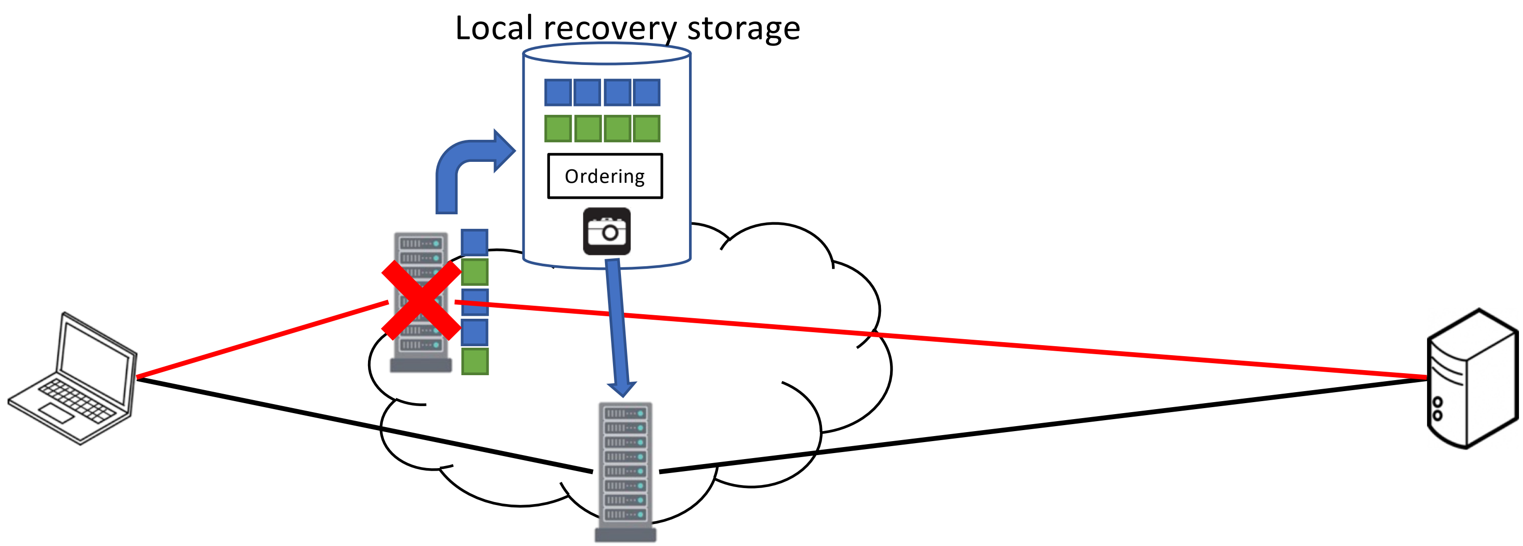
Server keeps a log of its outgoing packets

One recovery option: remote (cold) recovery

Edge takes periodic snapshots and sends to client, or to another edge
→ Packet logs and ordering info are safely pruned

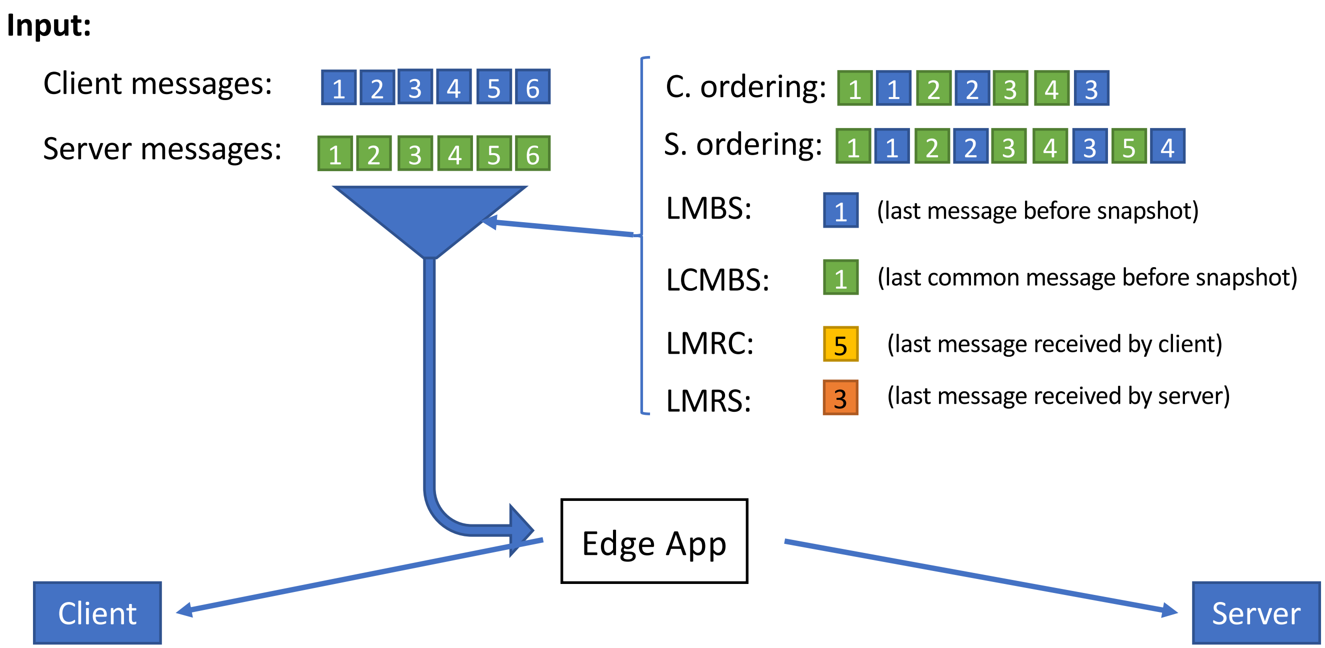
Recovery algorithm: enables ***faithful replay***

Local Recovery

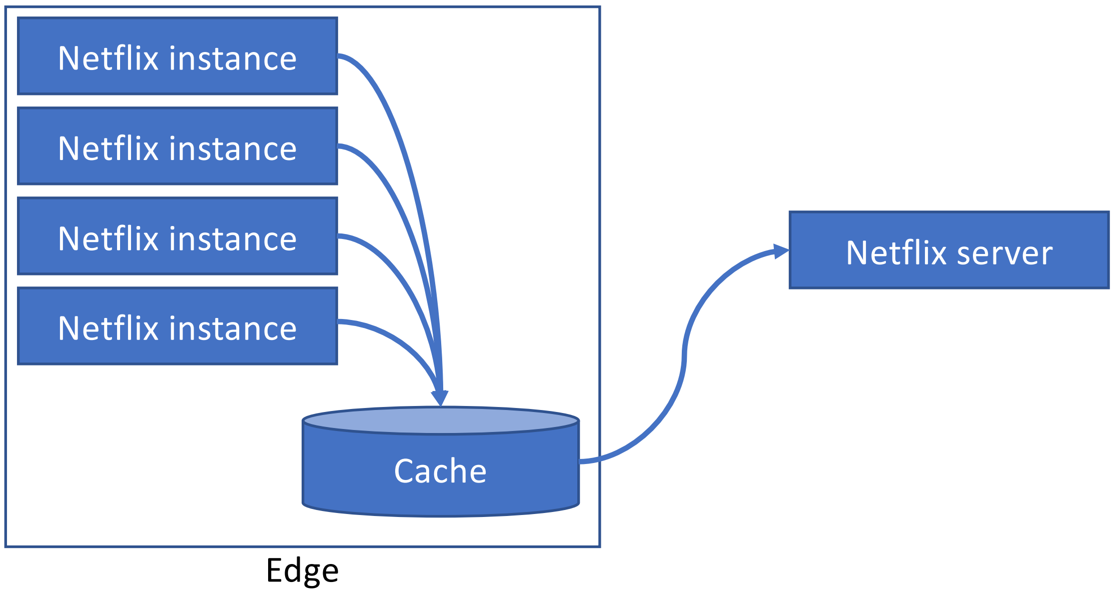


<u>Two operational modes:</u>
Cold standby: Upon failure, instantiate alternate edge
Hot standby: Alternate edge always running with latest snapshot

Recovery Algorithm



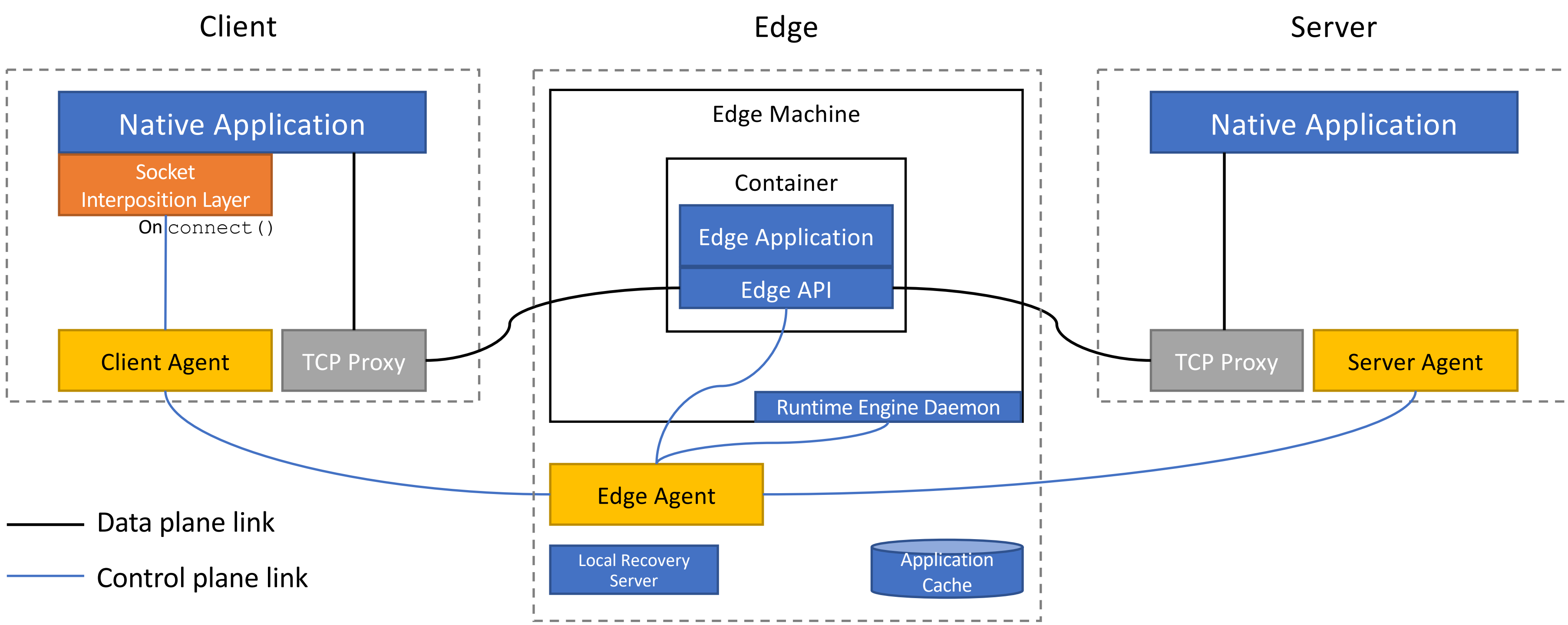
Local Cache



Netflix instance
Netflix instance
Netflix instance
Netflix instance
Cache
Edge
Netflix server

CESSNA Design

(somewhat different than in the paper)



Edge App API

Must implement:

- recv_client_msg(data)
- recv_server_msg(data)

Optional:

- init()
- accept_client_connection()
- shutdown()

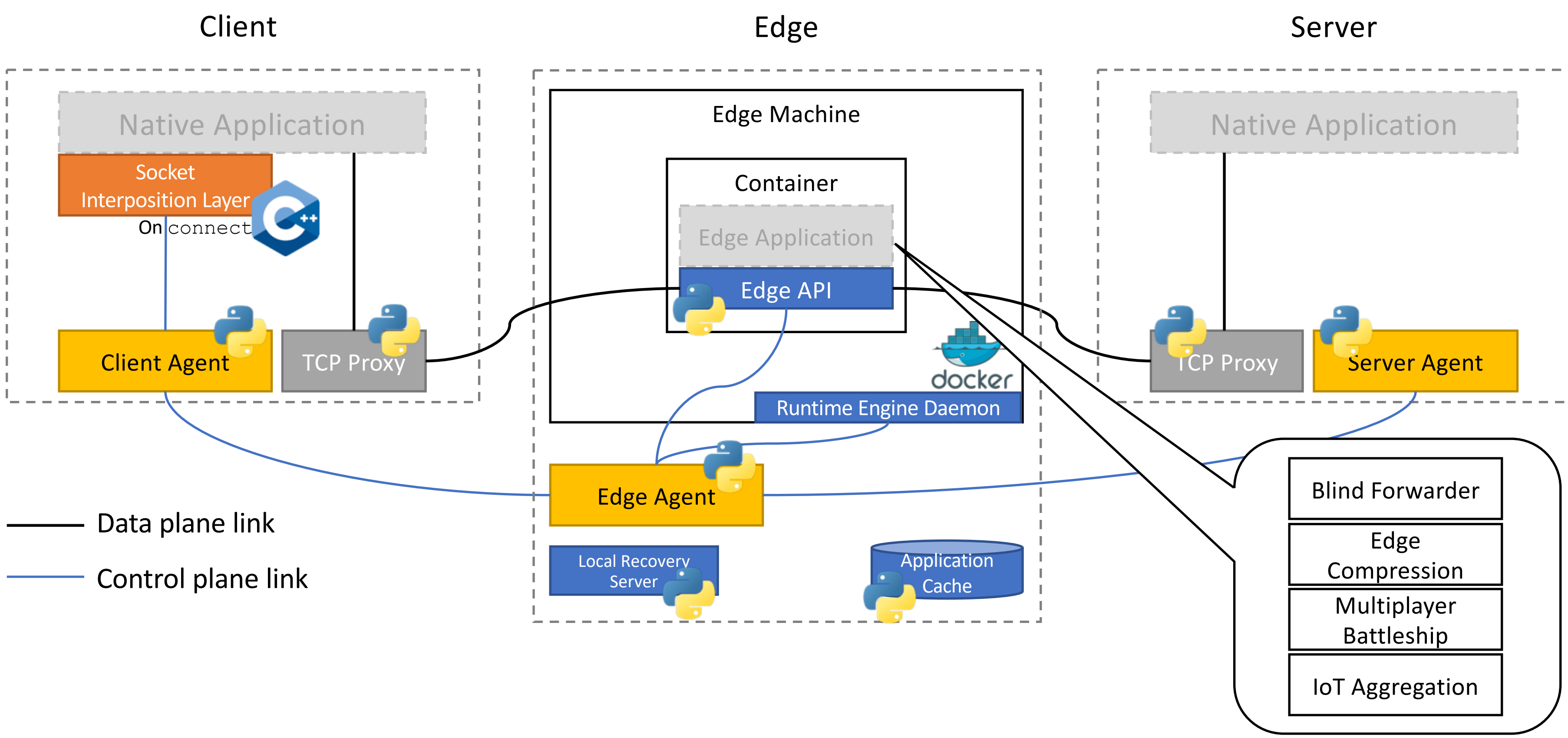
Provided:

- send_msg_to_client(data)
- send_msg_to_server(data)
- cache_read(obj_name)
- set_timeout(func, time)

Example: Edge Compression Service

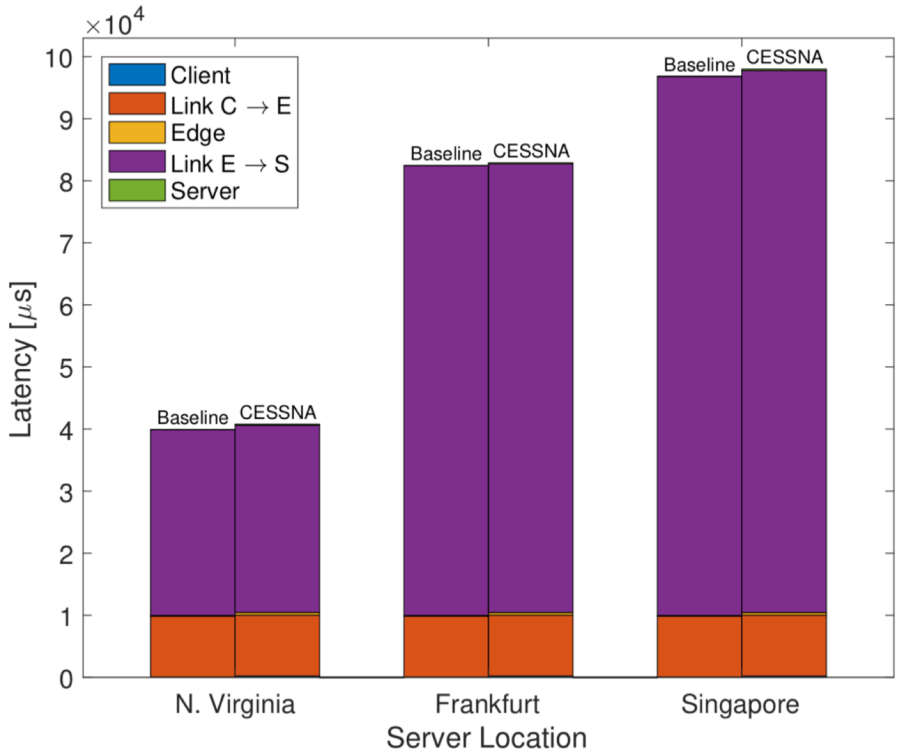
```
class CompressionApp(cessna_app.Application):
    def __init__(self):
        cessna_app.Application.__init__(self)
        self.compressor = zlib.compressobj()
        self.decompressor = zlib.decompressobj()

    def recv_server_msg(self, data):
        decomp = self.decompressor.decompress(data)
        decomp += self.decompressor.flush()
        self.send_msg_to_client(decomp)

    def recv_client_msg(self, data):
        comp = self.compressor.compress(data)
        comp += self.compressor.flush(zlib.Z_FULL_FLUSH)
        self.send_msg_to_server(comp)
```

Initial Implementation

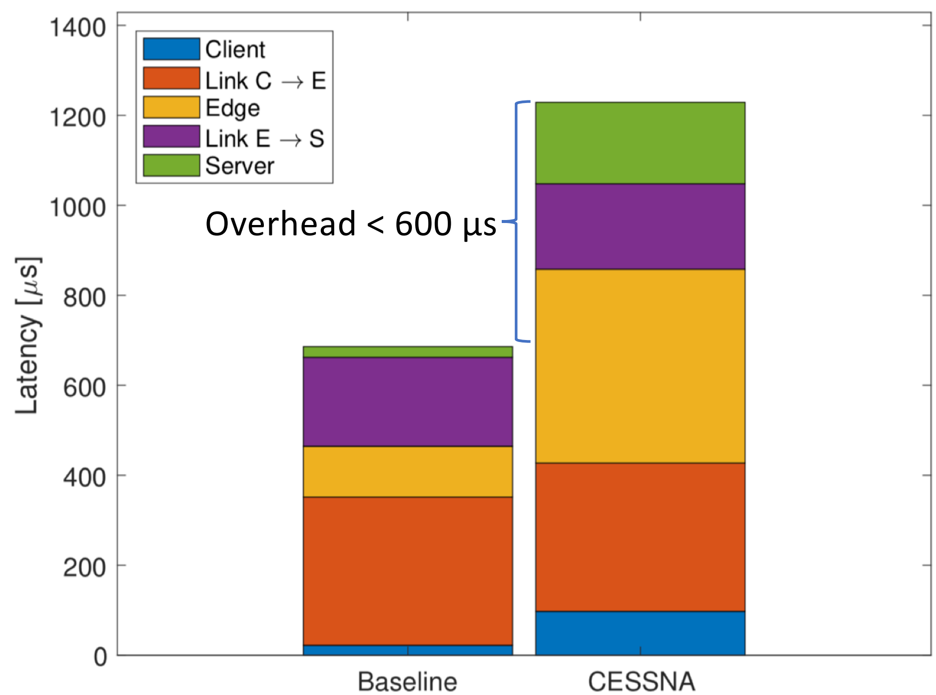


Initial Evaluation

(Not part of the workshop paper)

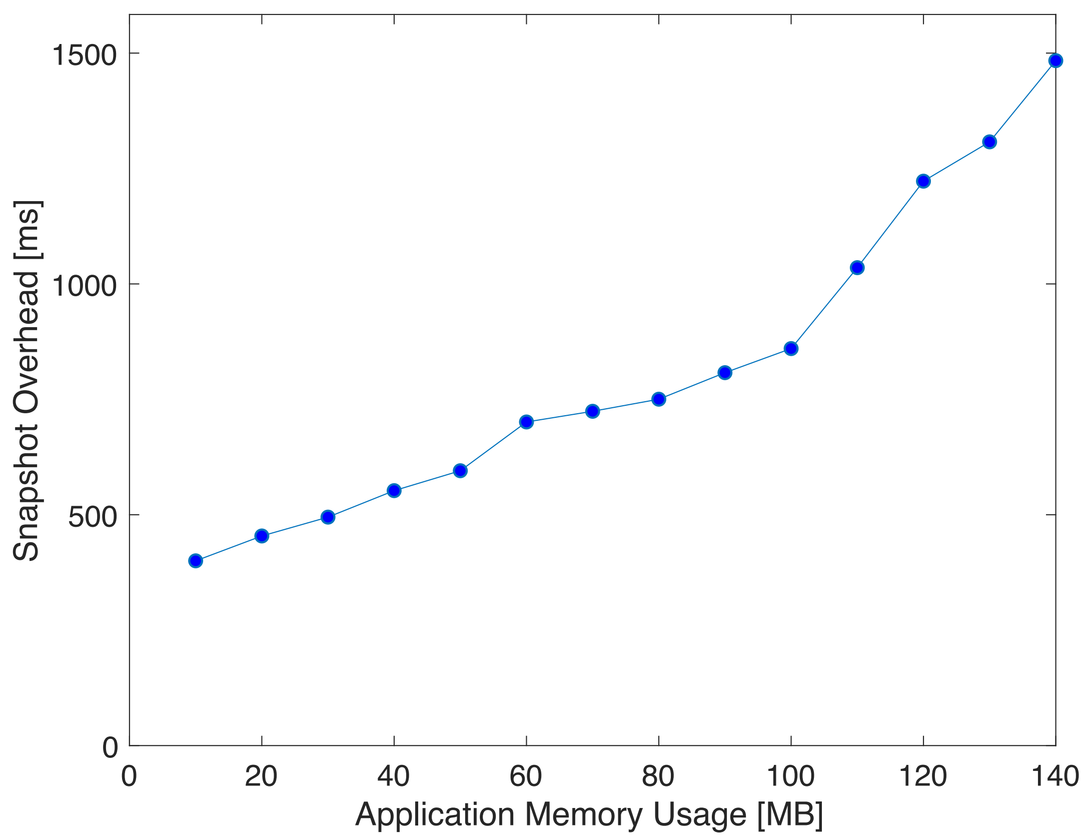


C,E – West US, S - varies

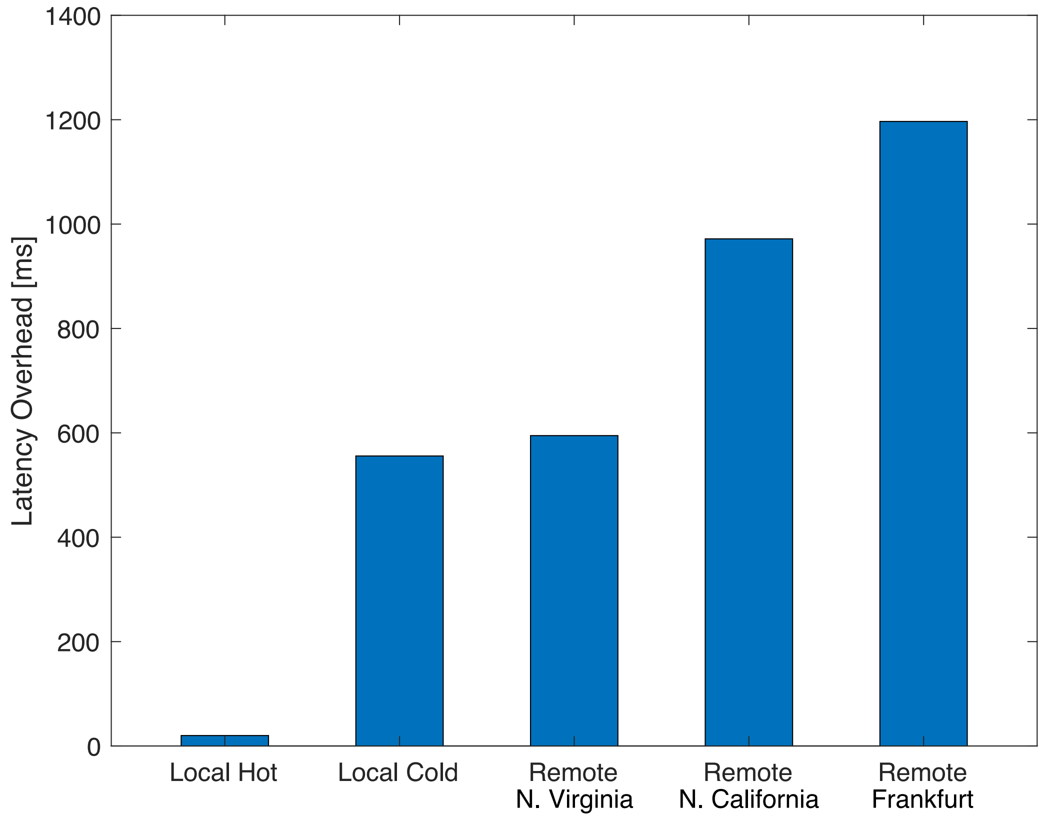


C,E,S co-located

Snapshot Latency Overhead



Recovery Latency Overhead



(Original edge in N. Virginia)

<u>For cold recovery:</u>
Docker restore: 87% (488 ms)
Snapshot loading: 10% (57 ms)
Recovery algorithm: 3% (20 ms)

Future Work

- Improve snapshot & recovery times
  - Use different edge runtimes
  - Use language-level snapshotting / serialization
- CESSNA over HTTP – work in progress
- Multiple clients per session – hard problem!

Conclusions

- Consistency of stateful edge applications is challenging
  - State is dependent on two parties
  - Edge platforms are considered less reliable
- CESSNA provides strong correctness guarantees
  - Also enables client mobility with edge
- Two recovery modes for efficient recovery
  - Local recovery – hot / cold standby
  - Remote recovery
- Per packet latency overhead < 700 µs

Questions?

Thank you