KTCP

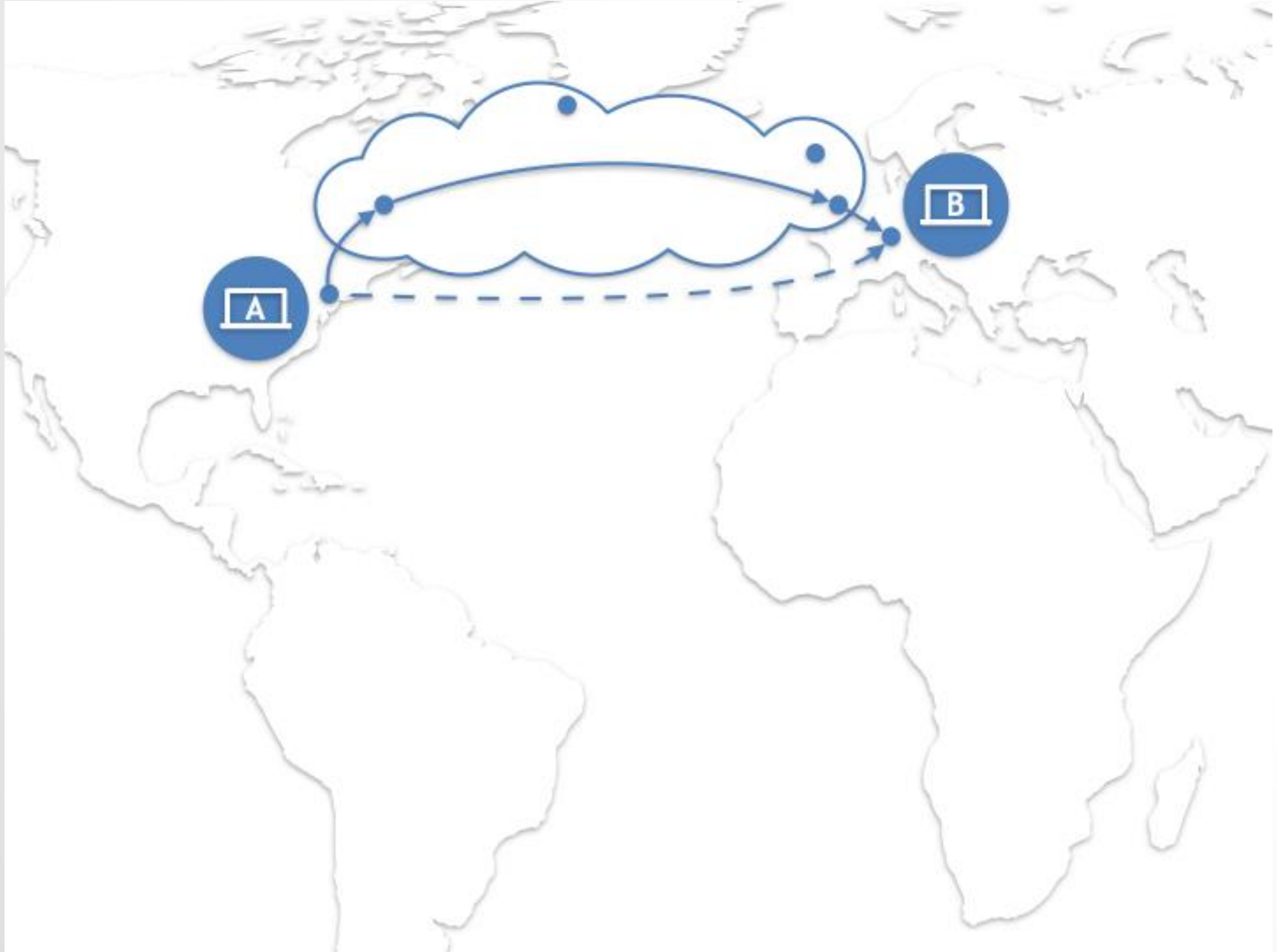
Kernel Split-TCP Proxy

Alex Markuze, VMware

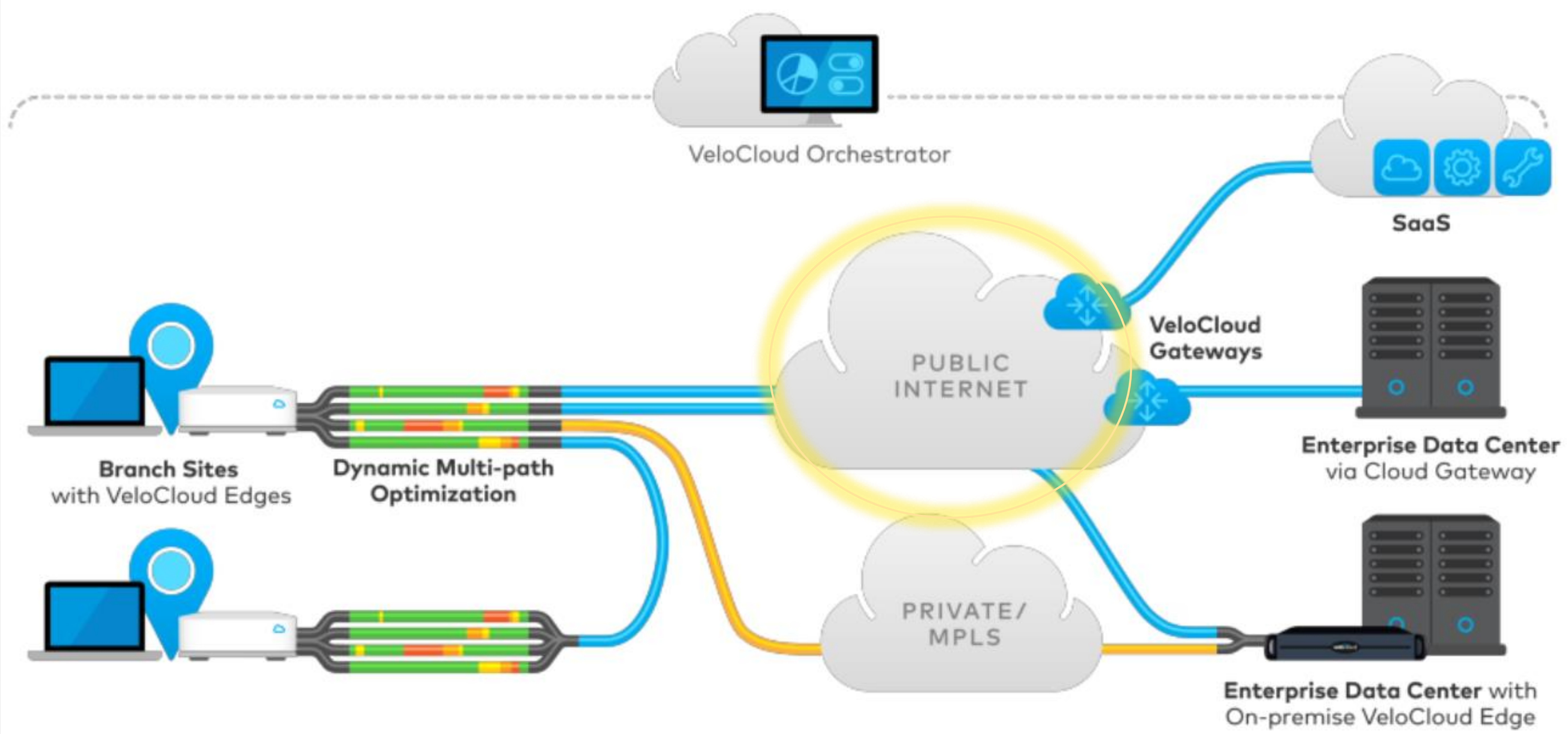
TALK LAYOUT

- Background
  - Project Pathway
  - What is TCP (Kernel) Splitting is good for?
- KTCP Features.
  - Show me the benefits.
- KTCP Implementation.
  - The nuts n’ bolts.

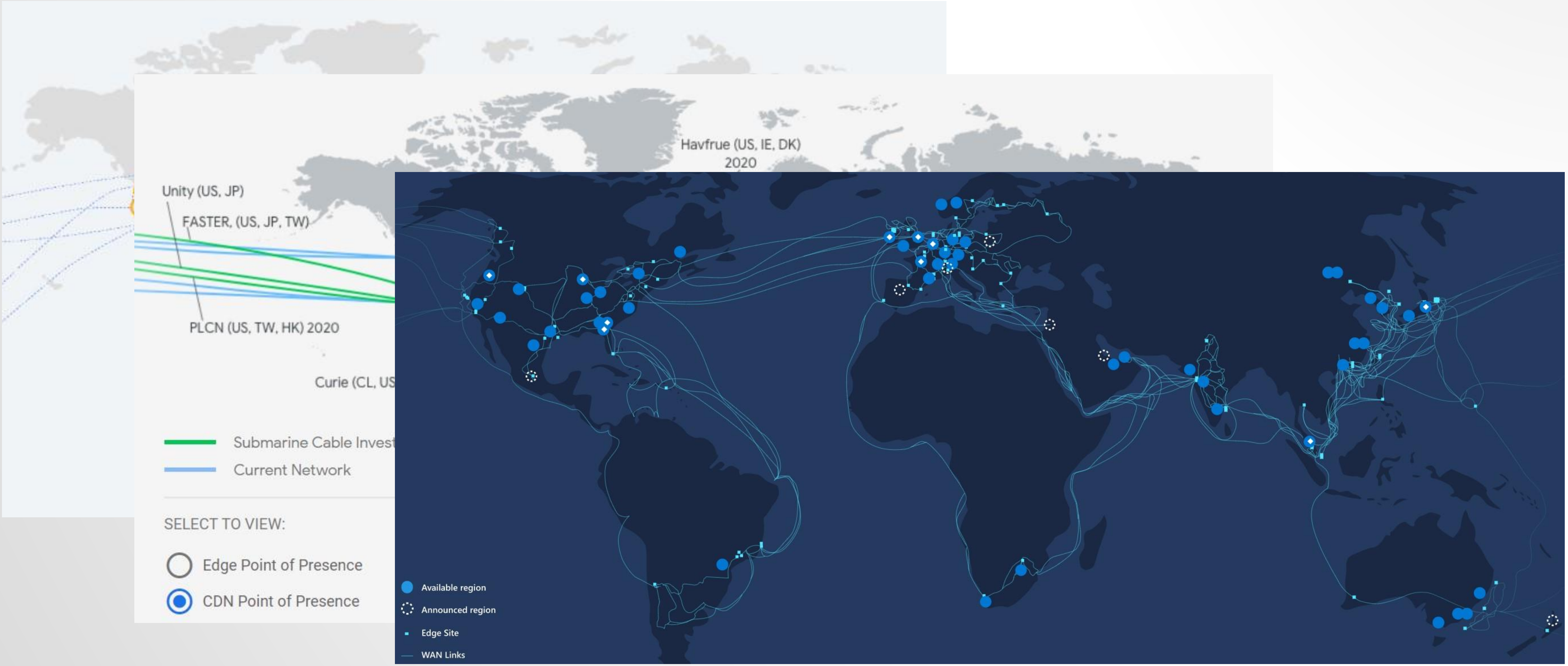
PROJECT PATHWAY



SD-WAN


VeloCloud Orchestrator
SaaS
PUBLIC INTERNET
VeloCloud Gateways
Branch Sites
with VeloCloud Edges
Dynamic Multi-path Optimization
Enterprise Data Center
via Cloud Gateway
PRIVATE/ MPLS
Enterprise Data Center with
On-premise VeloCloud Edge

NETWORK INFRASTRUCTURE



PROJECT PATHWAY https://arxiv.org/abs/1812.05582

- Link Health monitoring
- Management
- TCP-Split Proxy

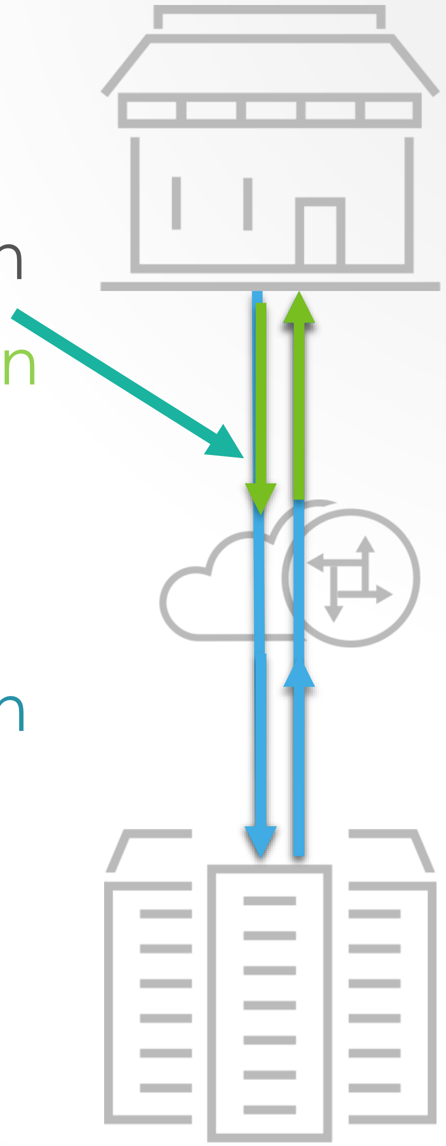
Regions: alicloud: 12 aws: 16 azure: 33 gcp: 24

WHAT IS SPLIT-TCP?

Single TCP connection

Leg A. TCP Connection

Leg B. TCP Connection

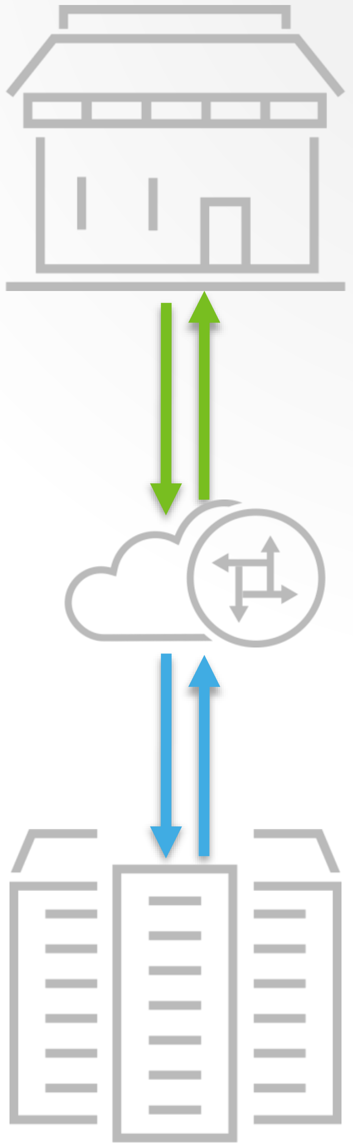


WHY SPLIT?

TCP Congestion Control contains a feedback loop (ACKs)

Smaller Round-Trip-Time (RTT)

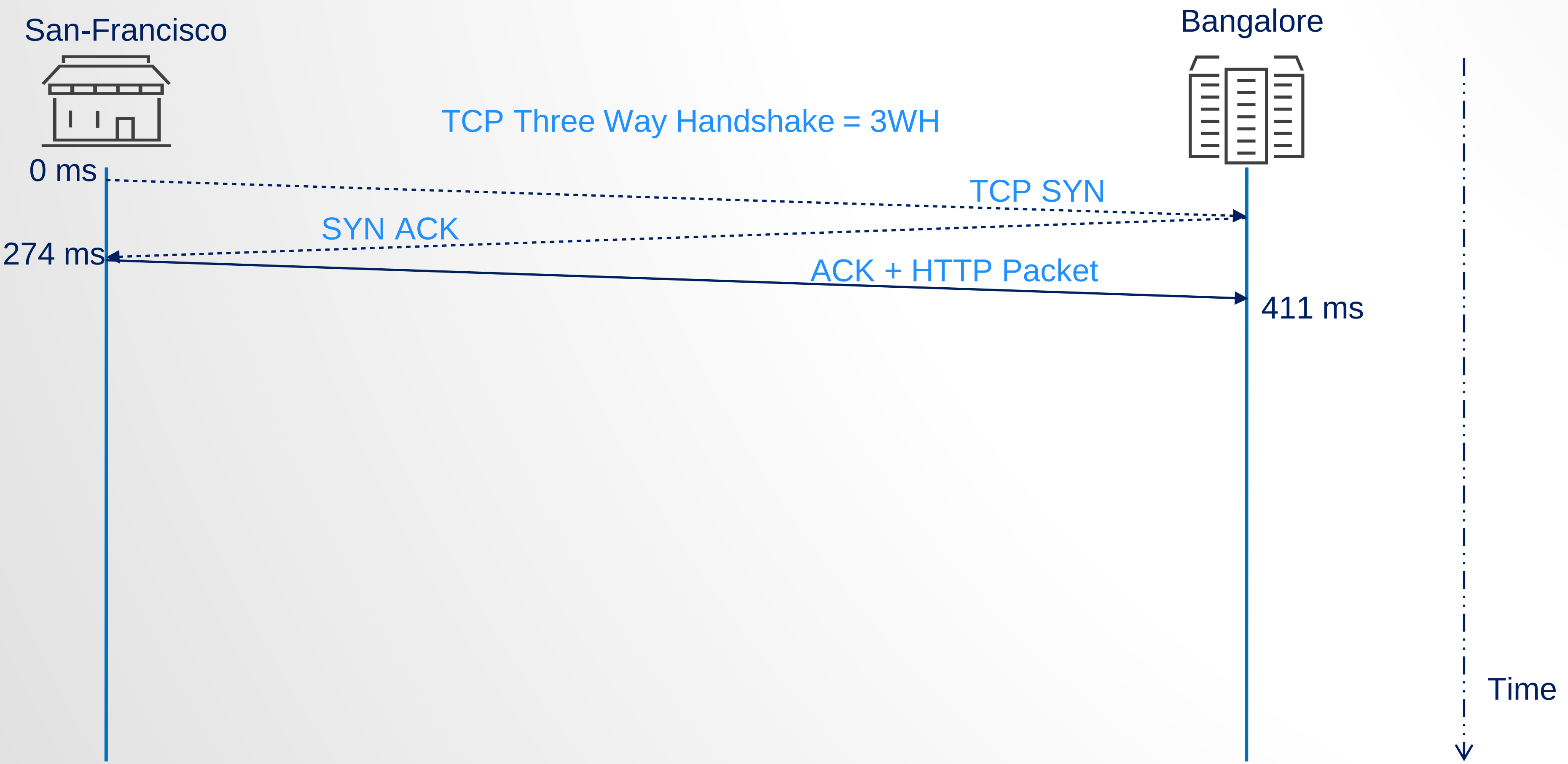
- Faster recovery from drops
- Faster throughput ramp-up
- Can support higher throughput with a smaller buffer (rwin)
- Reduces unfairness due to competition with small RTT



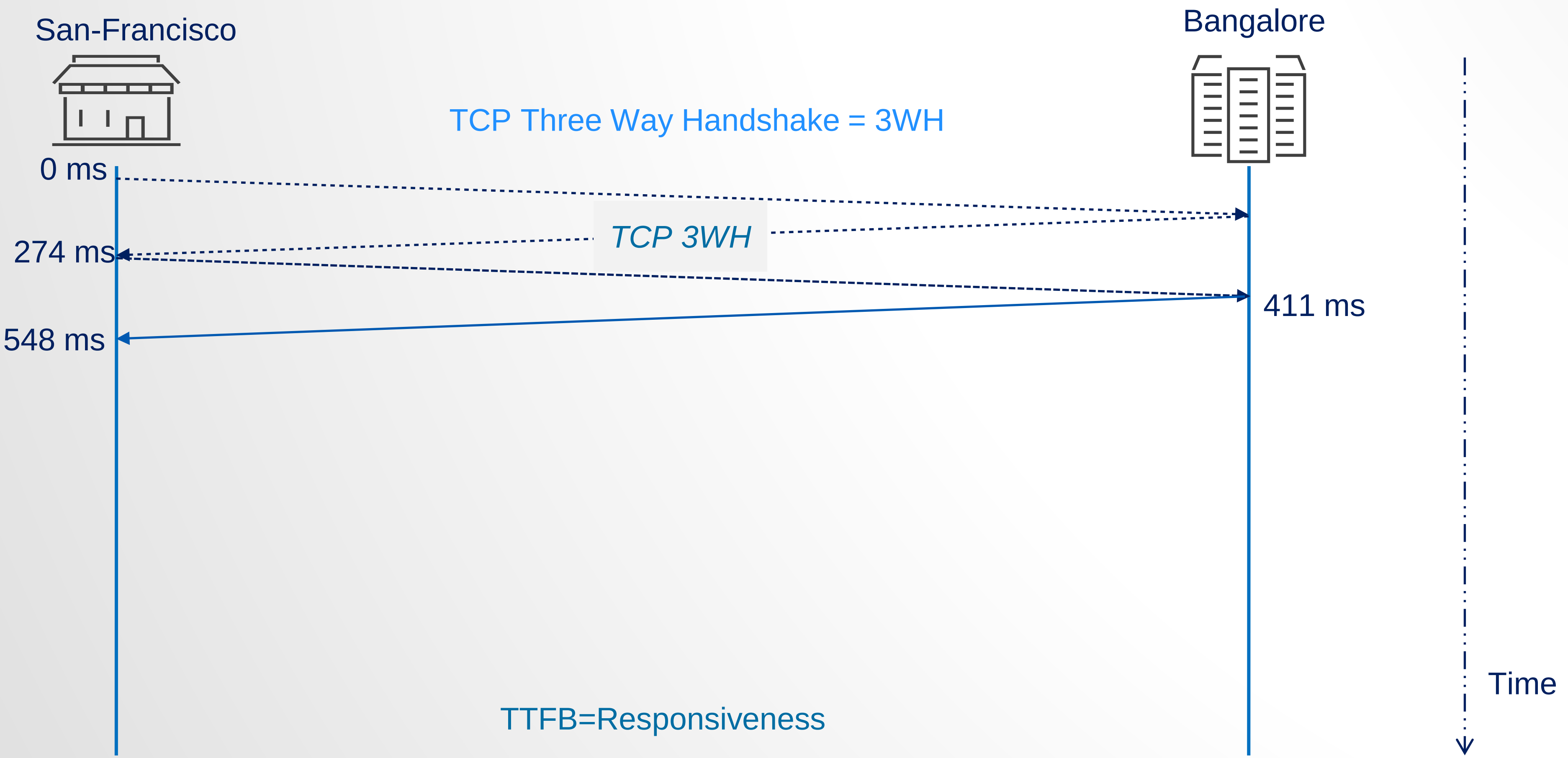
TALK LAYOUT

- Background
  - Project Pathway
  - What is TCP (Kernel) Splitting is good for?
- KTCP Features.
  - Show me the benefits.
- KTCP Implementation.
  - The implementation drill-down.

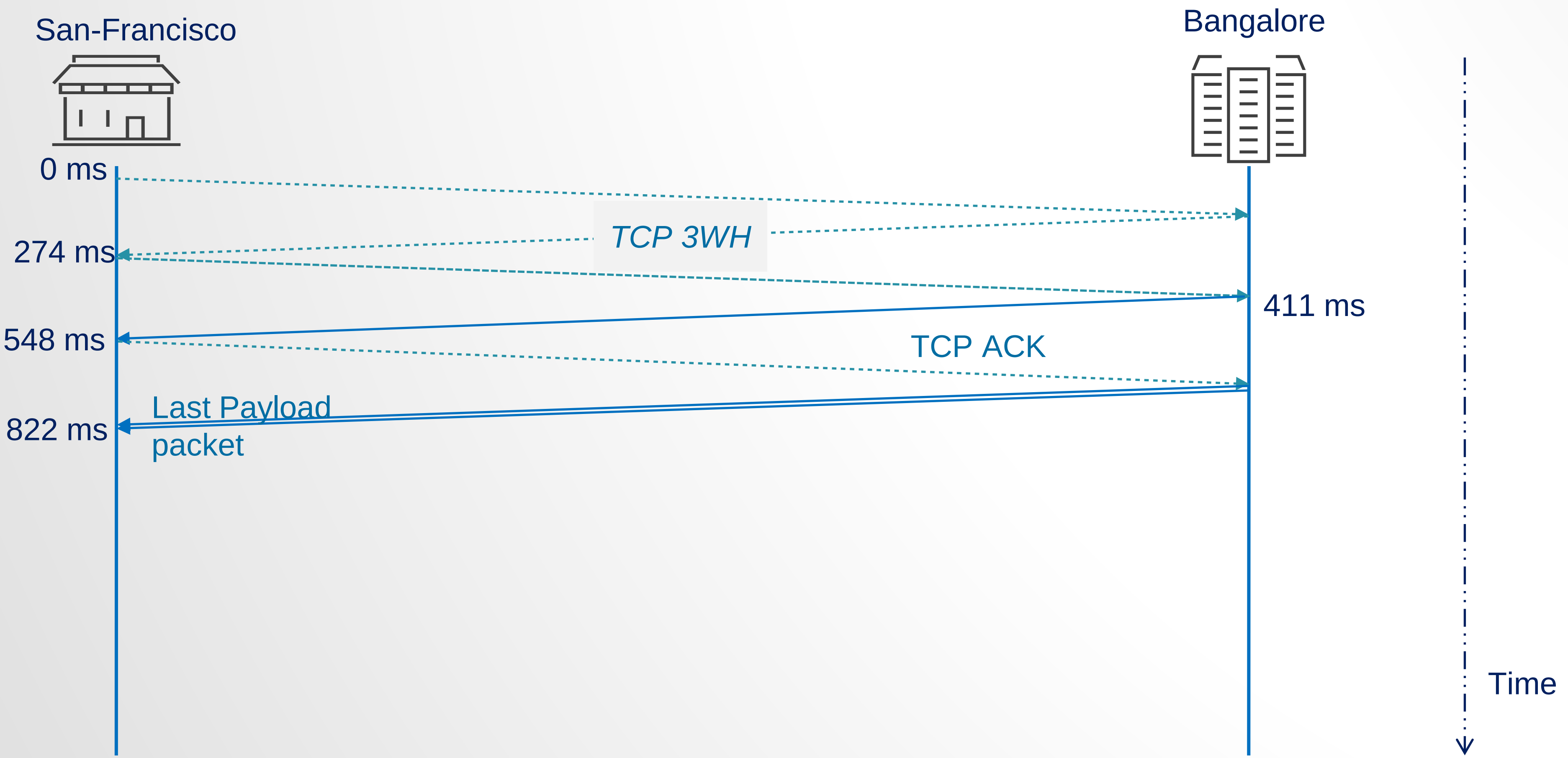
TCP 3WH


San-Francisco
Bangalore
TCP Three Way Handshake = 3WH
0 ms
TCP SYN
SYN ACK
274 ms
ACK + HTTP Packet
411 ms
Time

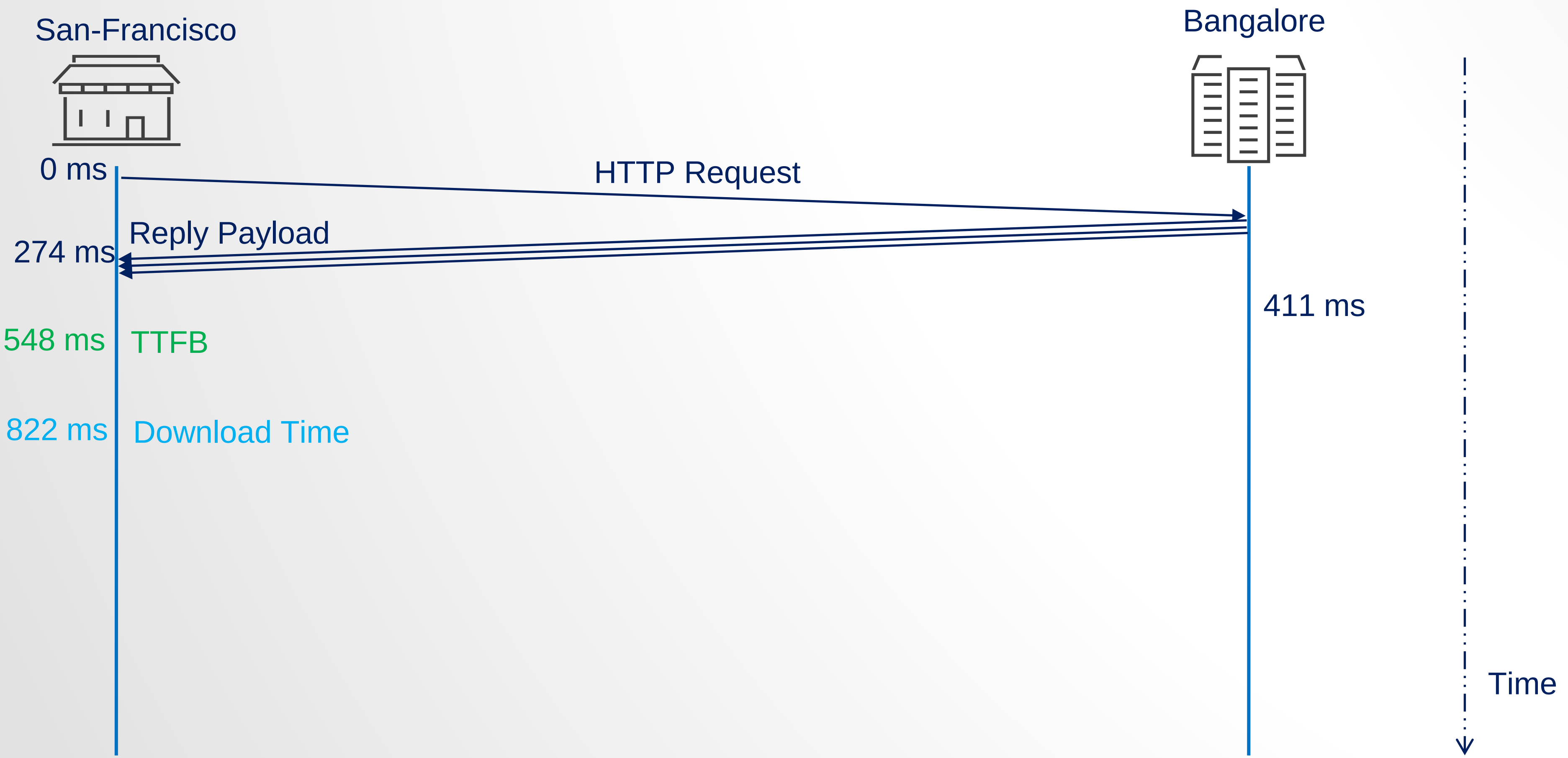
TTFB


San-Francisco
Bangalore
TCP Three Way Handshake = 3WH
0 ms
TCP 3WH
274 ms
411 ms
548 ms
Time
TTFB=Responsiveness

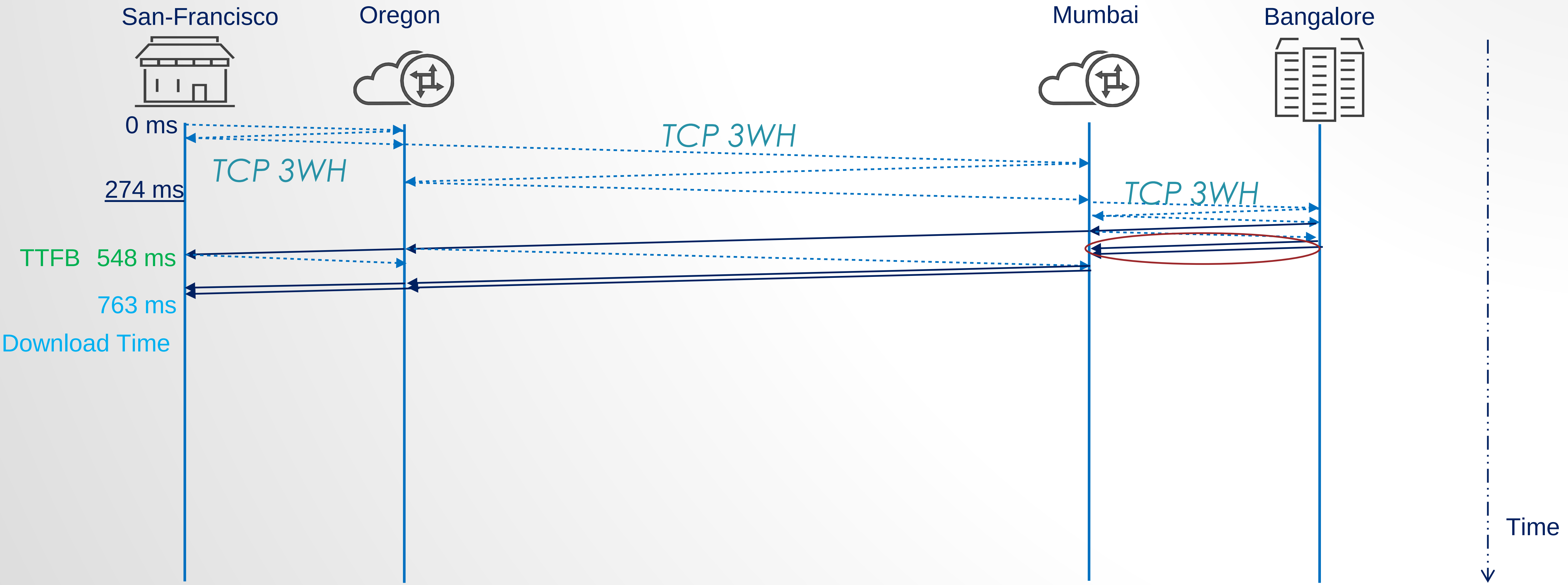
TCP SLOW START



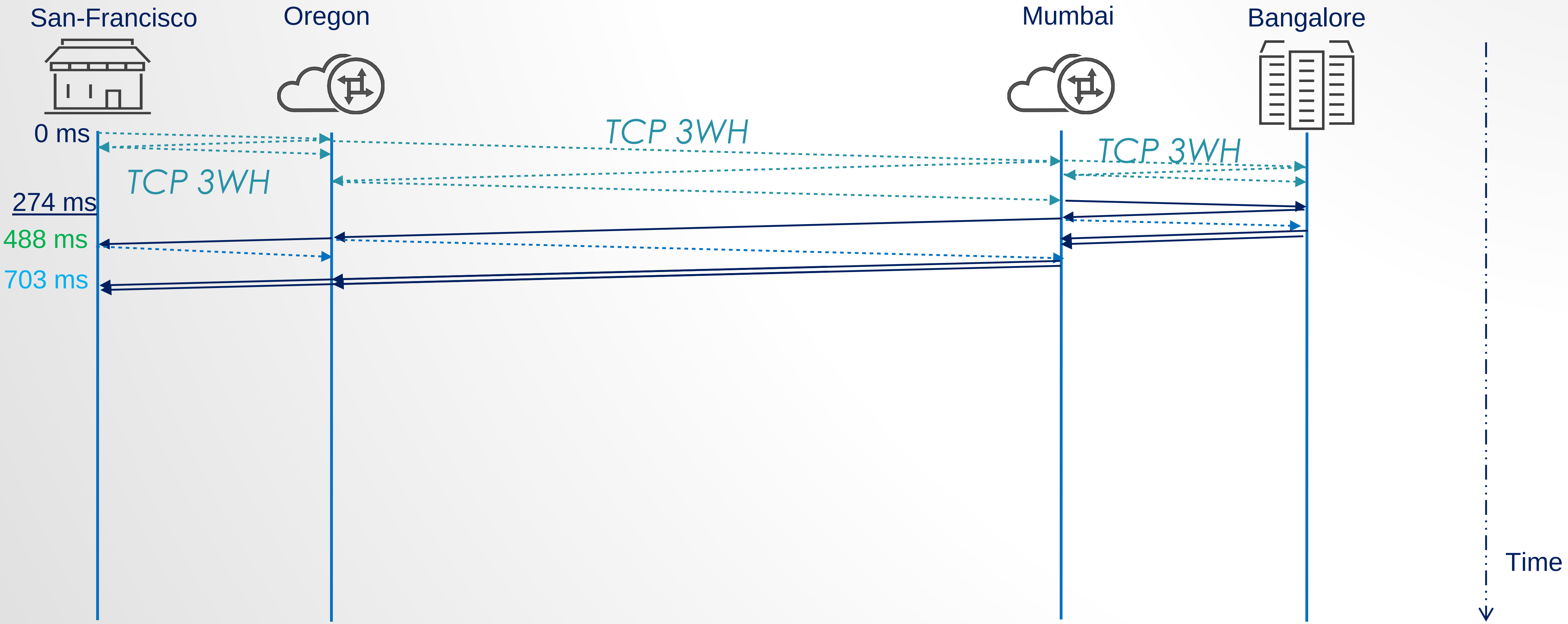
IDEAL TRANSMISSION



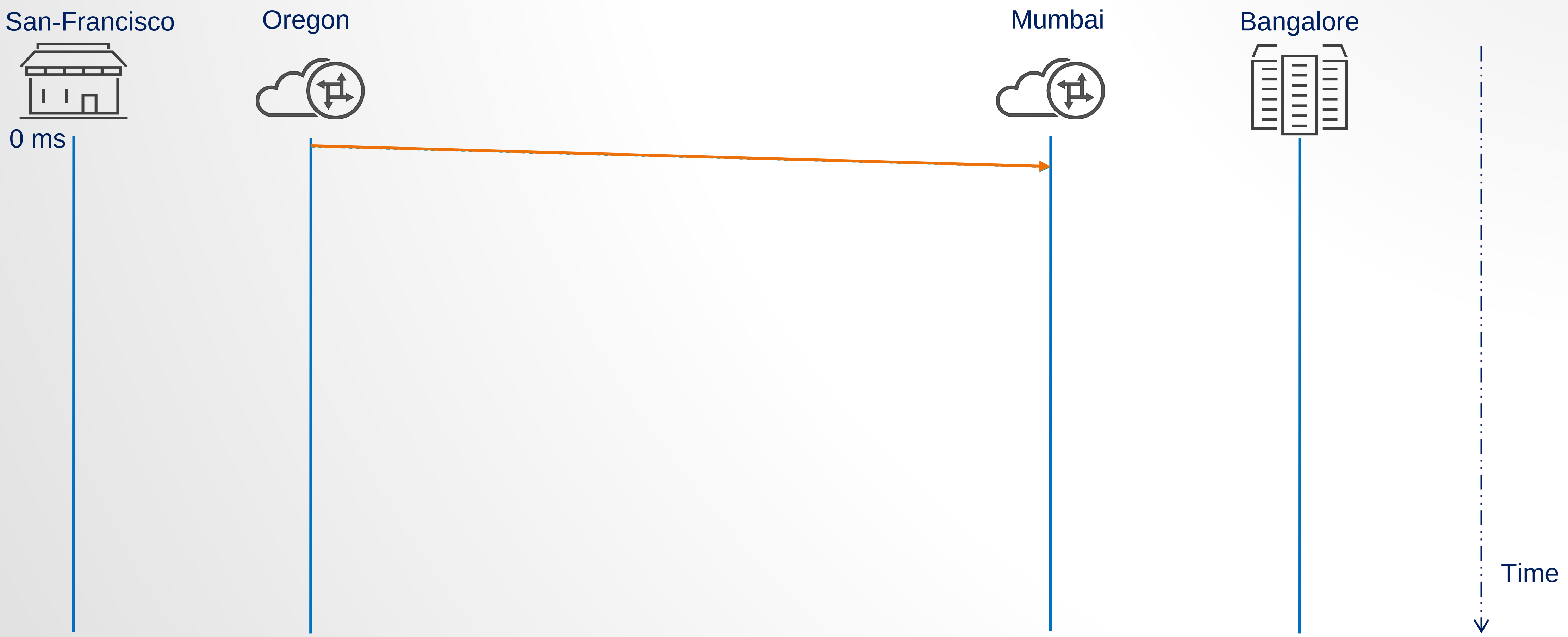
SPLIT-TCP


San-Francisco
Oregon
Mumbai
Bangalore
0 ms
TCP 3WH
TCP 3WH
274 ms
TCP 3WH
TTFB 548 ms
763 ms
Download Time
Time

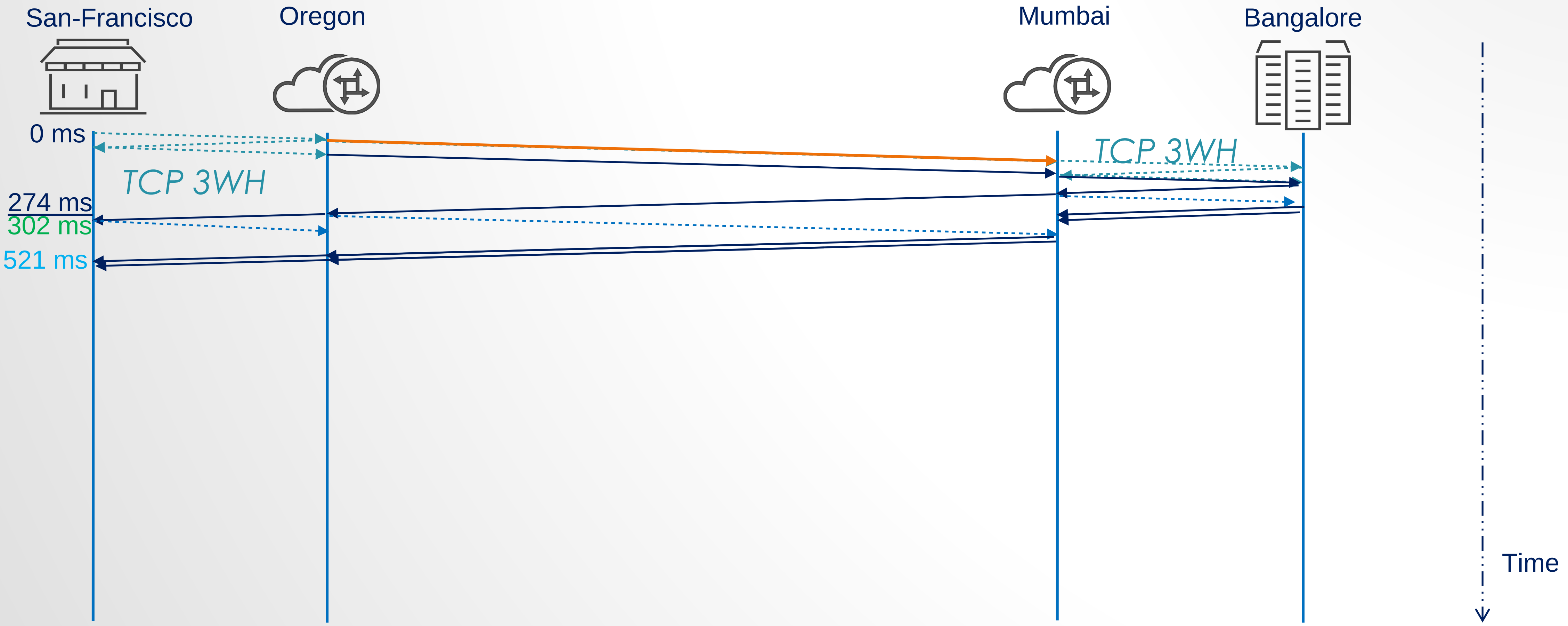
EARLY SYN


San-Francisco
Oregon
Mumbai
Bangalore
0 ms
TCP 3WH
TCP 3WH
TCP 3WH
274 ms
488 ms
703 ms
Time

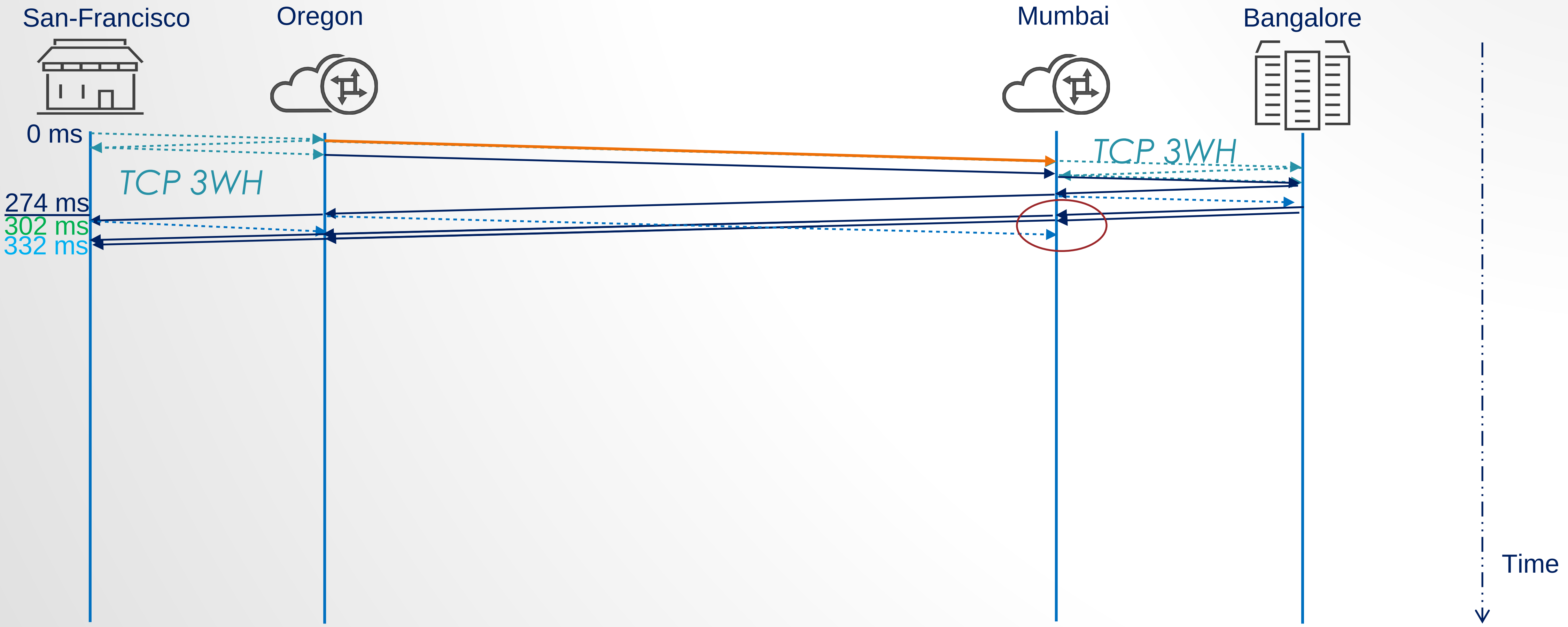
PRE-CONNECTION



PRE-CONNECTION

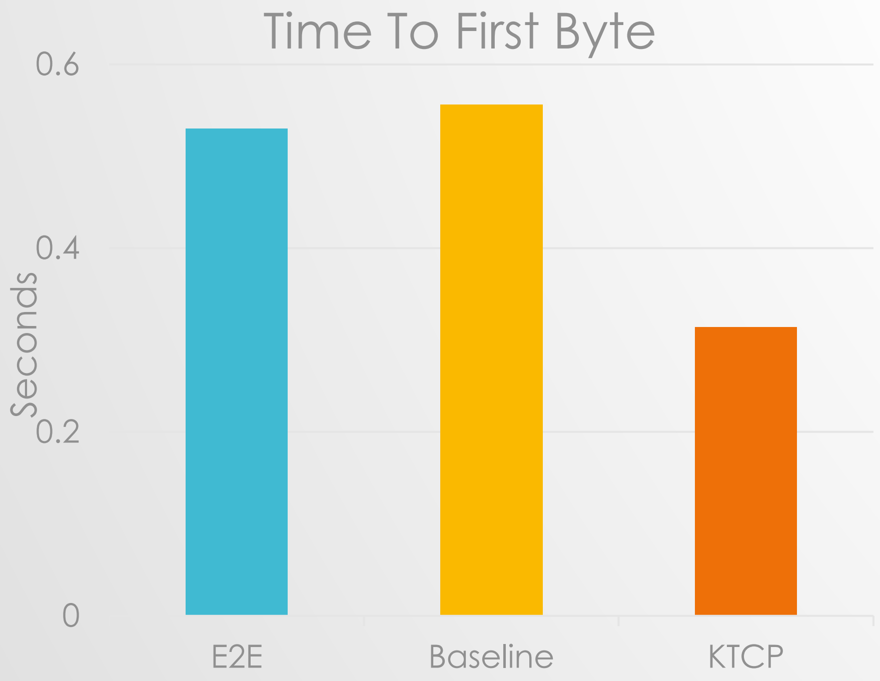


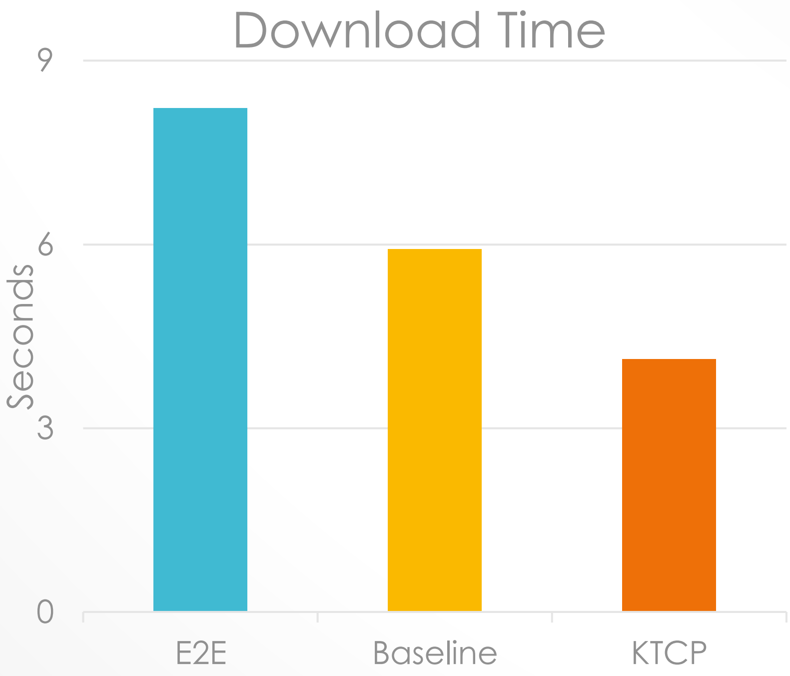
TURBO START


San-Francisco
Oregon
Mumbai
Bangalore
0 ms
TCP 3WH
TCP 3WH
274 ms
302 ms
332 ms
Time

EXPERIMENTAL RESULTS

50MB DOWNLOAD RESULTS





TALK LAYOUT

- Background
  - Project Pathway
  - What is TCP (Kernel) Splitting is good for?
- KTCP Features.
  - Show me the benefits.
- KTCP Implementation.
  - The implementation drill-down.

WHY THE KERNEL?

User space

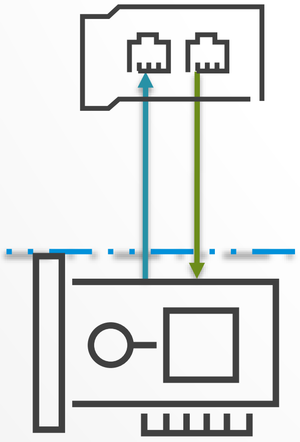
POSIX Threads
Copying
System Calls

Slow & Limited

Kernel space

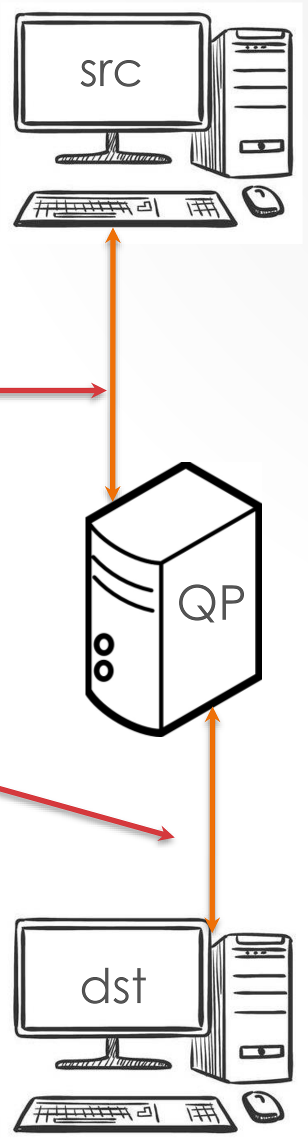
TCP Stack
Routing
Tunneling
Netfilter
NAT
Connection tracking
eBPF
Kernel Threads

TCP socket



CONNECTION QP

- Split connection meta-data
  - 12B Key : src: Port | dst: Port | src: IP | dst: IP
  - struct socket *rx
  - struct socket *tx
- AUX fields
  - atomic_t refcount;
  - struct rb_node. node;
  - wait_queue_head_t wait;



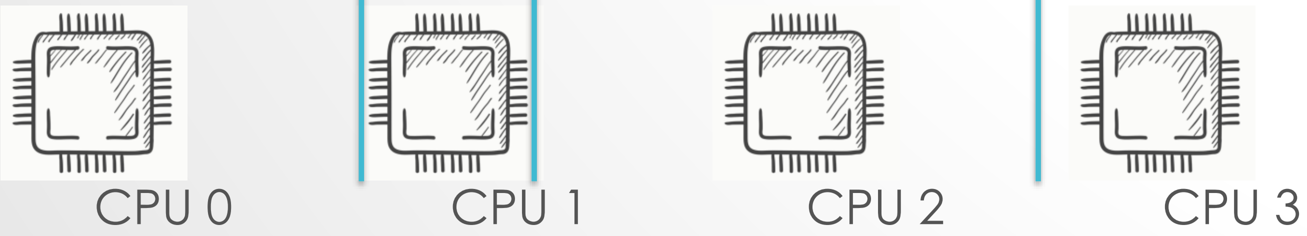
QP CREATION FLOW

1. SYN : Alloc TX Sock Thread on Core X
2. Connection established: Alloc RX Sock Thread on Core X
3. Lookup in QP Tree[X].
4. Send/Receive: full CPU mask.

QP CREATION FLOW

Netfilter (IRQ)

- Alloc Thread: SYN handler
- Schedule on 2; 2 = hash(5 tuple)



QP CREATION FLOW

SYN handler

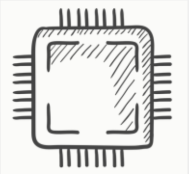
- Add QP
- Wait for RX socket (90 sec)

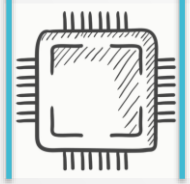
Connection handler

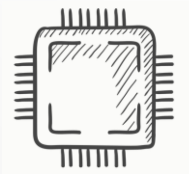
- Lookup QP
- Remove from tree
- Wakeup waiting
- Start forwarding RX to TX

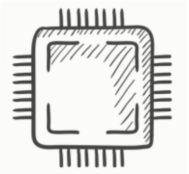
Proxy server

- Accept connection from source (RX socket)
- Alloc QP
- Alloc Thread: Connection handler
- Schedule on 2;


CPU 0


CPU 1


CPU 2


CPU 3

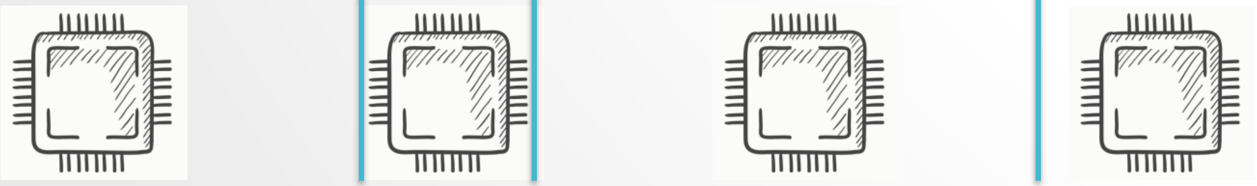
QP CREATION FLOW

SYN handler

- Start forwarding TX sock to RX

Connection handler

- Start forwarding RX to TX



CPU 0 CPU 1 CPU 2 CPU 3

THREAD POOL

- Alloc:

- Slow: may take several milliseconds
- Limited: Cant be called from NAPI/IRQ

```
struct task_struct *kthread_create_on_node(int (*threadfn)(void *data),
                                           void *data,
                                           int node,
                                           const char namefmt[], ...);

/**
 * kthread_create - create a kthread on the current node
 * @threadfn: the function to run in the thread
 * @data: data pointer for @threadfn()
 * @namefmt: printf-style format string for the thread name
 * @arg...: arguments for @namefmt.
 *
 * This macro will create a kthread on the current node, leaving it in
 * the stopped state.  This is just a helper for kthread_create_on_node();
 * see the documentation there for more details.
 */
#define kthread_create(threadfn, data, namefmt, arg...) \
        kthread_create_on_node(threadfn, data, NUMA_NO_NODE, namefmt, ##arg)
```

BUILDING BLOCK: POOL_ELEM

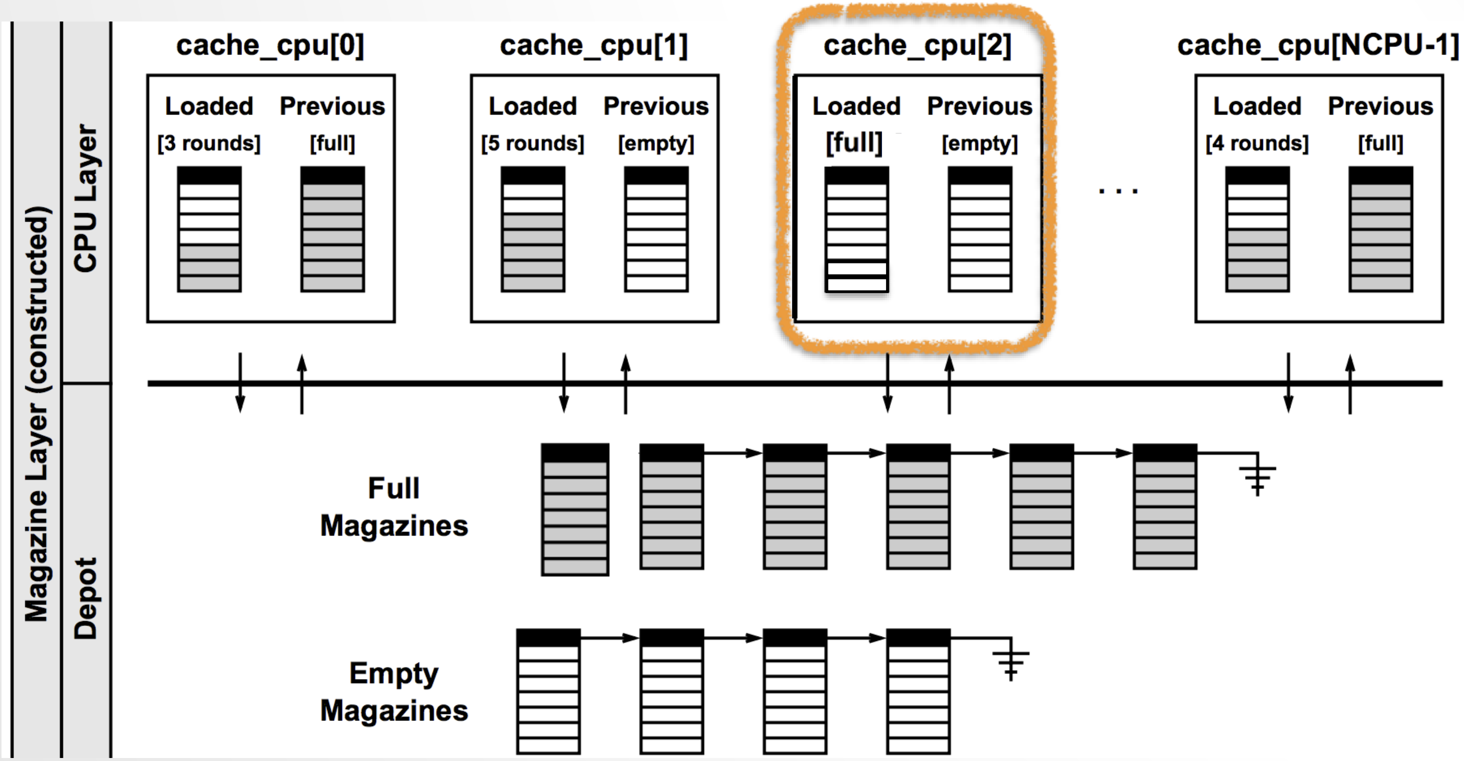
```
struct task_struct *kthread_create_on_node(int (*threadfn)(void *data),
                                           void *data,
```

- POOL_ELEM:
  - task_struct * task;
  - Int (*pool_task)(void *data) pool_task;
  - void * data;

- threadfn:
  - While (!kthread_should_stop())
    - this->pool_task(this->data);
    - set_current_state(TASK_INTERRUPTIBLE);
    - kthread_pool_reuse(elem);
    - schedule();

- Fast alloc:
  - set; pool_task and data
  - Wake task
- Can start anywhere

MAGAZINE ALLOCATOR (ATC'01)



ZERO-COPY

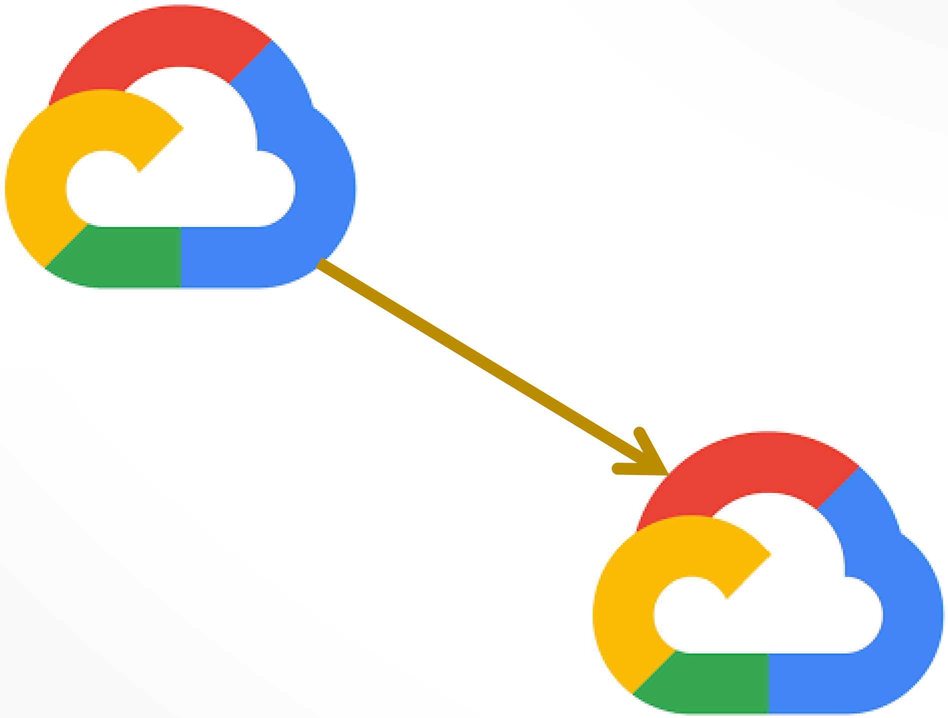
```c
/*
 * This routine provides an alternative to tcp_recvmsg() for routines
 * that would like to handle copying from skbuffs directly in 'sendfile'
 * fashion.
 * Note:
 *      - It is assumed that the socket was locked by the caller.
 *      - The routine does not block.
 *      - At present, there is no support for reading OOB data
 *        or for 'peeking' the socket using this routine
 *        (although both would be easy to implement).
 */
int tcp_read_sock(struct sock *sk, read_descriptor_t *desc,
                  sk_read_actor_t recv_actor)
```

- RX:
  - NEW: tcp_read_sock_zcopy:
    - based on tcp_read_sock (tcp_splice_read)
- TX:
  - do_tcp_sendpages: used in splice.
  - Modify tcp_sendmsg_locked

```c
ssize_t do_tcp_sendpages(struct sock *sk, struct page *page, int offset,
                         size_t size, int flags)
```

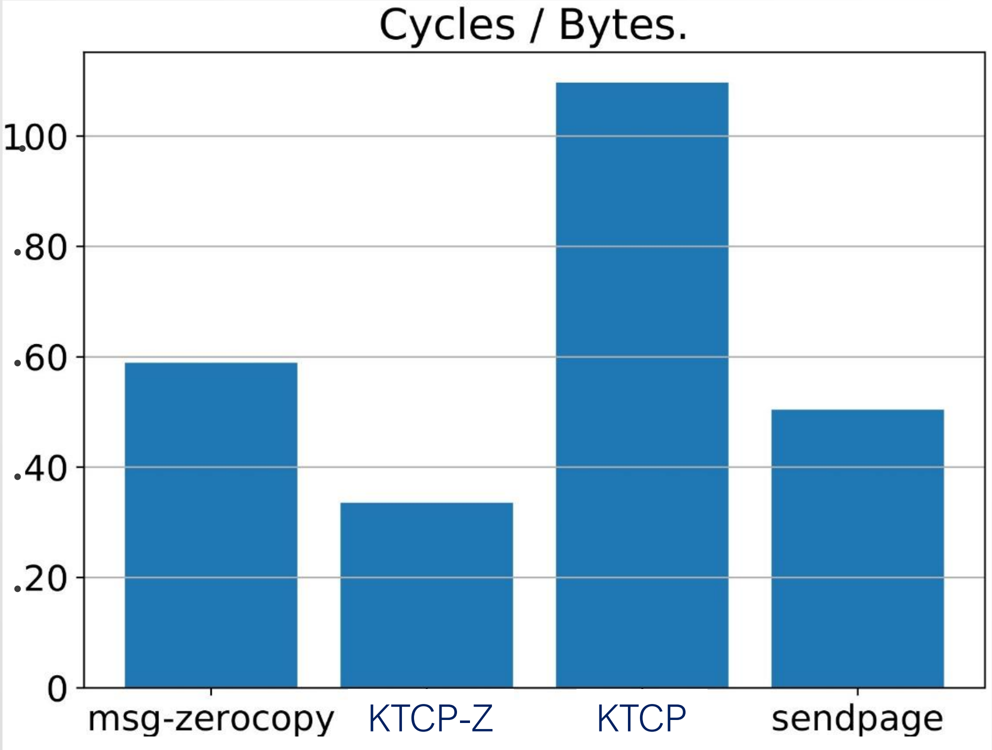
ZERO-COPY EVALUATION

- 16 core Intel Cascade Lake
- 64GB RAM
- 32Gb/s Egress



ZERO-COPY SEND EVALUATION

- 16 core Intel Cascade Lake
- 64GB RAM
- 32Gb/s Egress

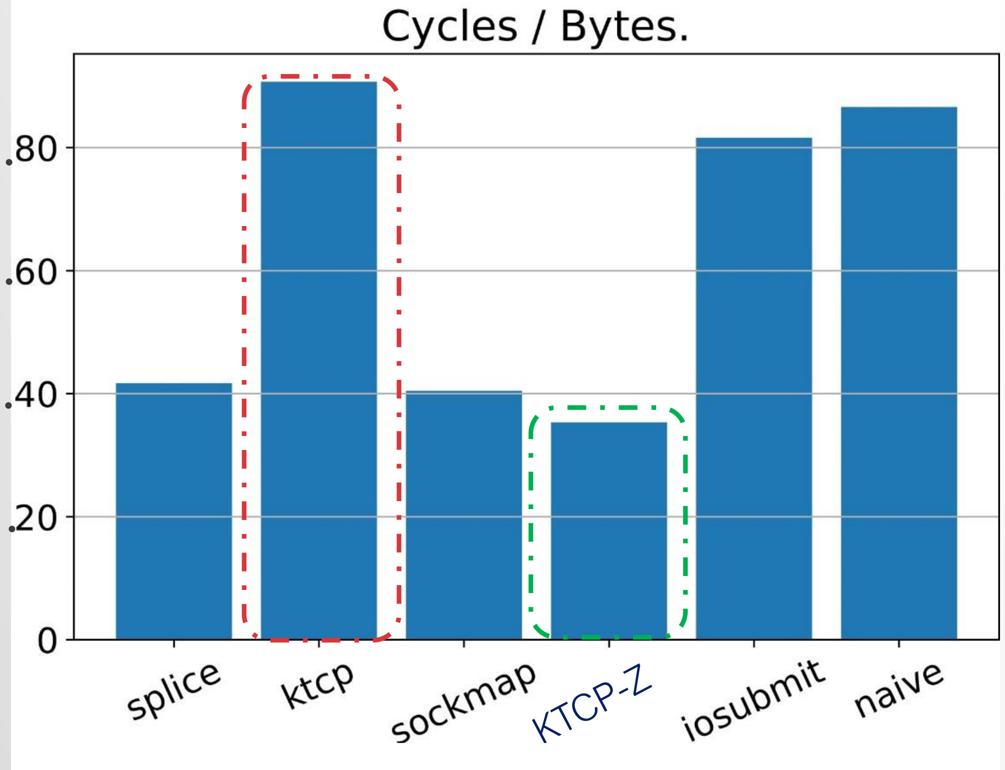


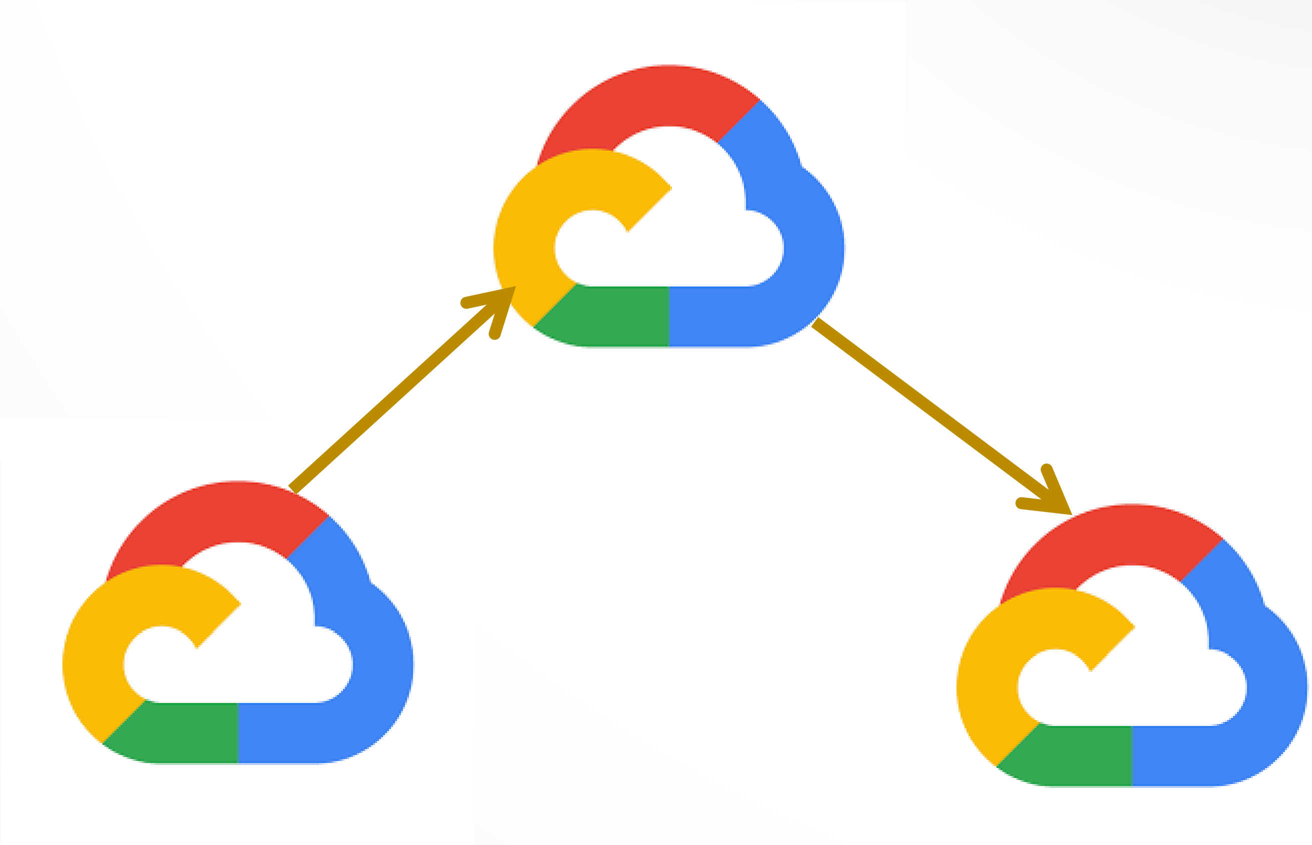
https://blog.cloudflare.com/sockmap-tcp-splicing-of-the-future/

https://blog.cloudflare.com/io_submit-the-epoll-alternative-youve-never-heard-about/

ZERO-COPY SPLICE EVALUATION

- 16 core Intel Cascade Lake
- 64GB RAM
- 32Gb/s Egress





CONTACT

- E-mail
  - amarkuze@vmware.com

- Code Availability
  - https://github.com/Markuze/ktcp.git