NetClone: Fast, Scalable, and Dynamic Request Cloning for Microsecond-Scale RPCs

Gyuyeong Kim

성신여자대학교
SUNGSHIN WOMEN'S UNIVERSITY

Microsecond-scale RPCs

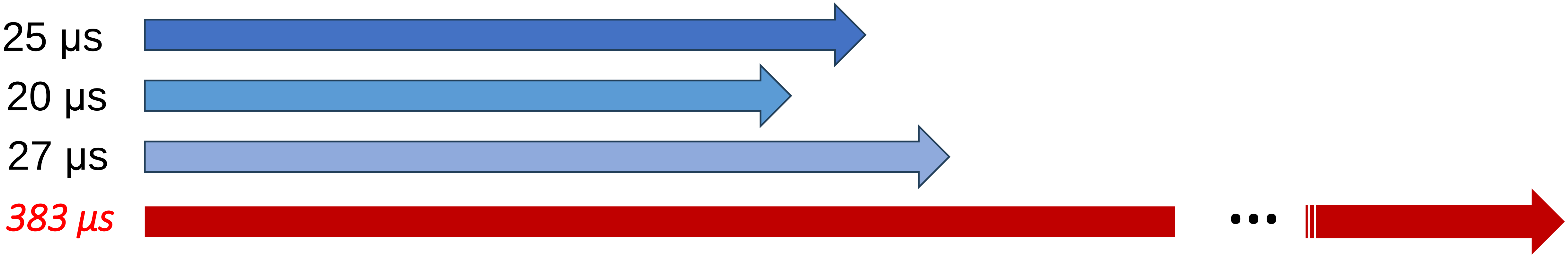
- Microservice components interact via RPCs
- The RPC is getting smaller and shorter
  - 75% of requests are < 512B, 90% of responses < 64B*
  - e.g., ~ 20 μs to access key-value stores

- We need ***microsecond-scale*** tail latency for better user experience

*Y. Gan et al., "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud and Edge Systems," in *Proc. of ACM ASPLOS,* 2019.

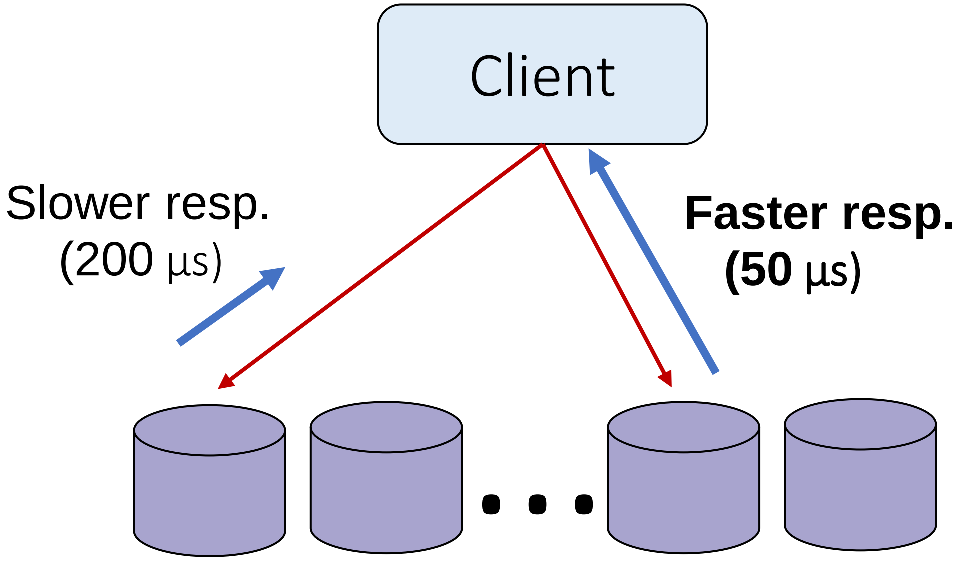
Service-time Variability

- RPC requests may experience unexpected latency variability
- Hard to eliminate because sources are diverse
  - Load fluctuation, background tasks, OS scheduling, garbage collection, …



Request Cloning to Mask Variability

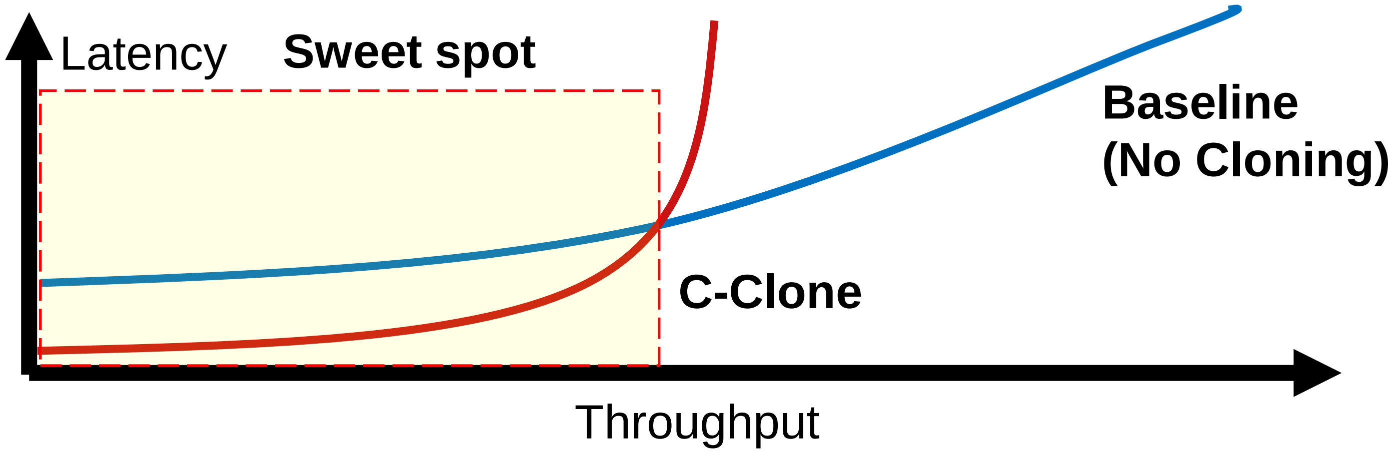
- Client sends duplicate requests and takes the faster response
  - Client-side Cloning (C-Clone) [CoNEXT'13]*



*Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker, "Low Latency via Redundancy," in *Proc. of ACM CoNEXT*, 2013.

Request Cloning to Mask Variability

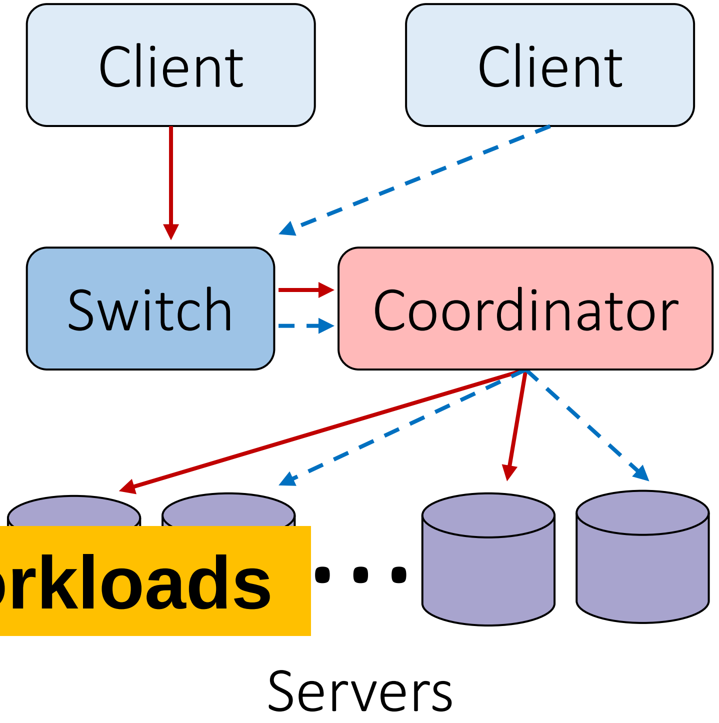
- Client sends duplicate requests and takes the faster response
  - Client-side Cloning (C-Clone) [CoNEXT'13]*
- Static cloning: only beneficial within a sweet spot



*Ashish Vulimiri, Philip Brighten Godfrey, Radhika Mittal, Justine Sherry, Sylvia Ratnasamy, and Scott Shenker, "Low Latency via Redundancy," in *Proc. of ACM CoNEXT*, 2013.

Coordinator-based Cloning

- A coordinator performs cloning decisions
  - LÆDGE [NSDI'21]*
- Dynamic cloning with load-awareness
  - Clones requests only if at least two servers are idle
  - No sweet spot

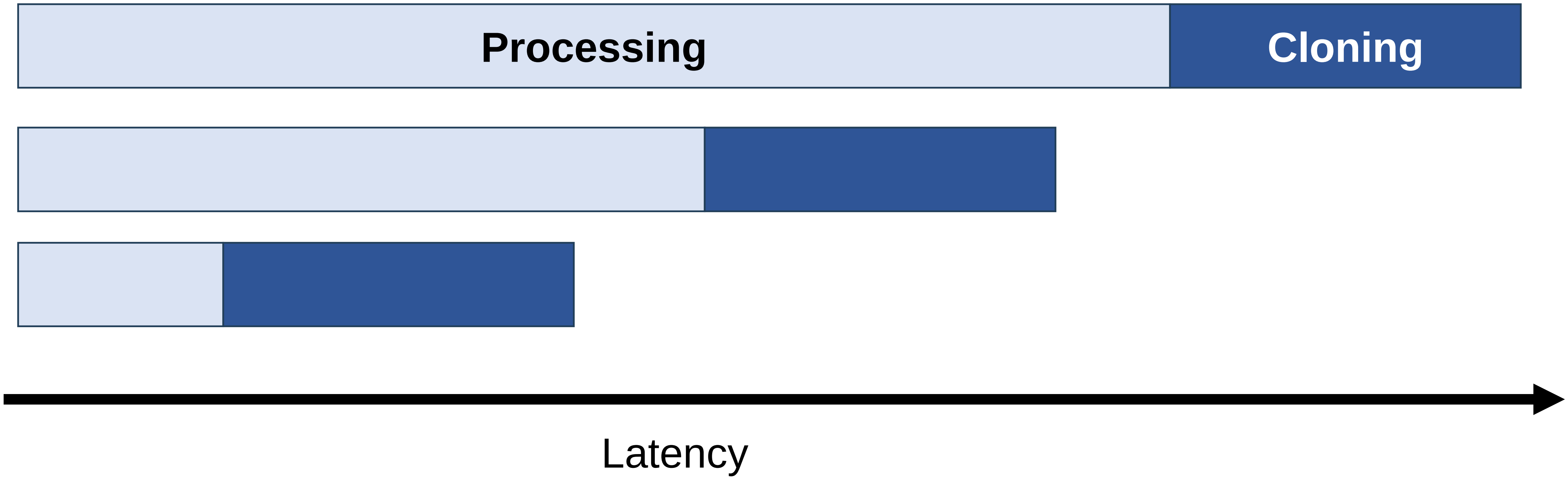


Not enough to serve *microsecond-scale* workloads

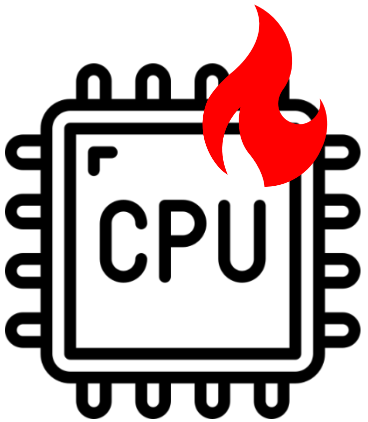
*Mia Primorac, Katerina Argyraki, and Edouard Bugnion, "When to Hedge in Interactive Services," in *Proc. of USENIX NSDI*, 2021.

#1 Latency overhead for cloning decisions

- As the runtime decreases, the portion of overhead increases
- Even a small overhead can increase latency excessively

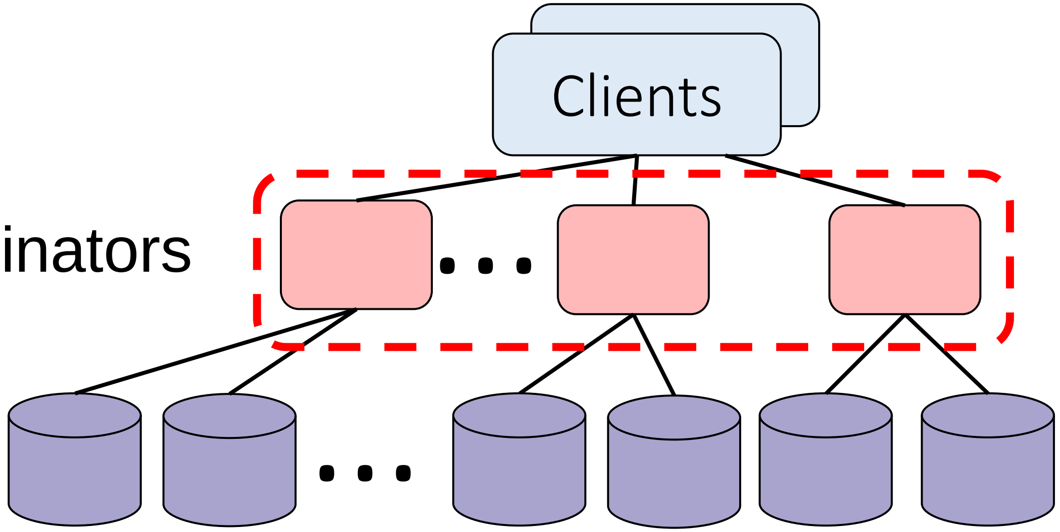


#2 Limited Scalability of CPUs



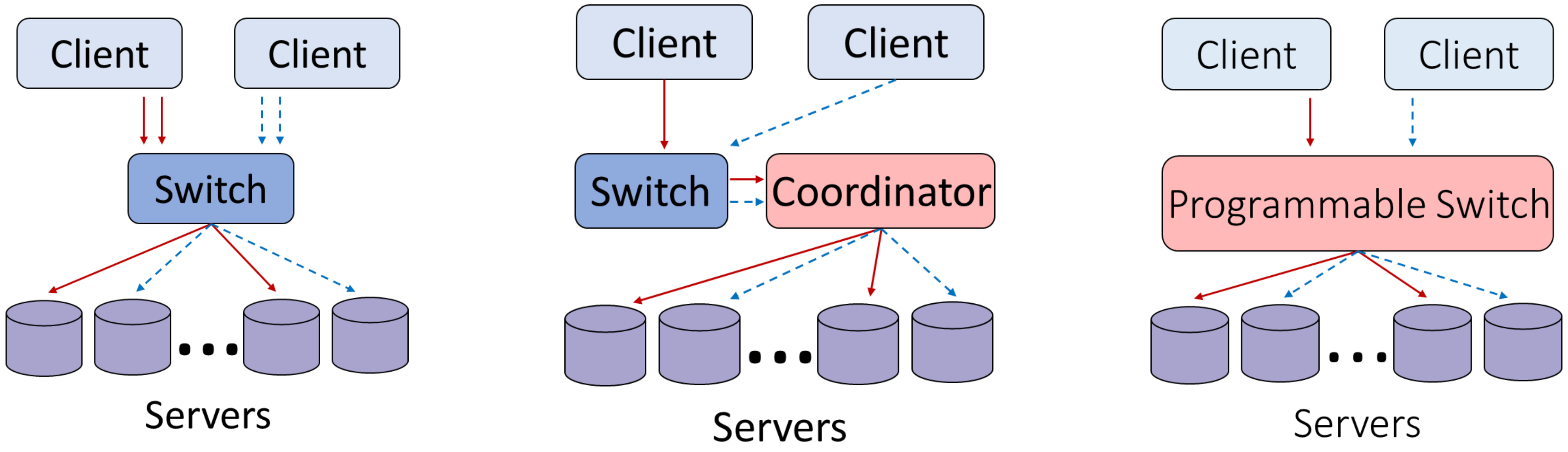
- The coordinator uses the CPU for request handling
- Limited packet processing performance

- Multiple coordinators to scale out
  - Costs to build and maintain a tier of coordinators



The Case for In-Network Cloning

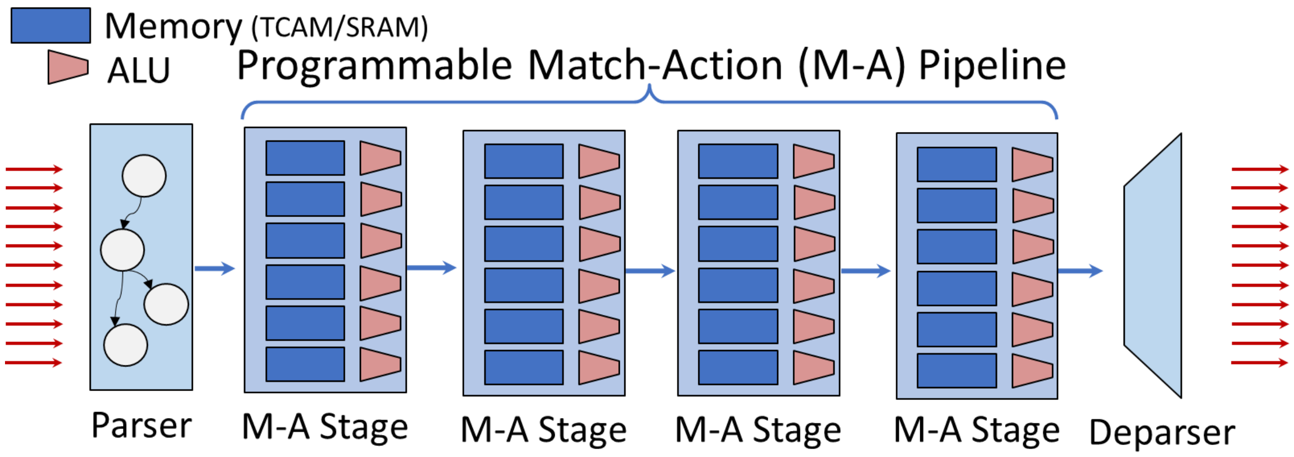
- Q: How can we perform dynamic request cloning quickly at scale?
- A: NetClone: switch-based dynamic request cloning



Client-based (C-Clone) Coordinator-based (LÆDGE) **Switch-based (NetClone)**

Why In-Network Cloning?

- High performance
  - Can process **a few billion** packets per second
  - Can process a packet in **hundreds of nanoseconds**
- High flexibility
  - We can **customize the switch data plane** thanks to the programmable switch ASIC like Intel Tofino



Programmable switch

Requirements and Challenges

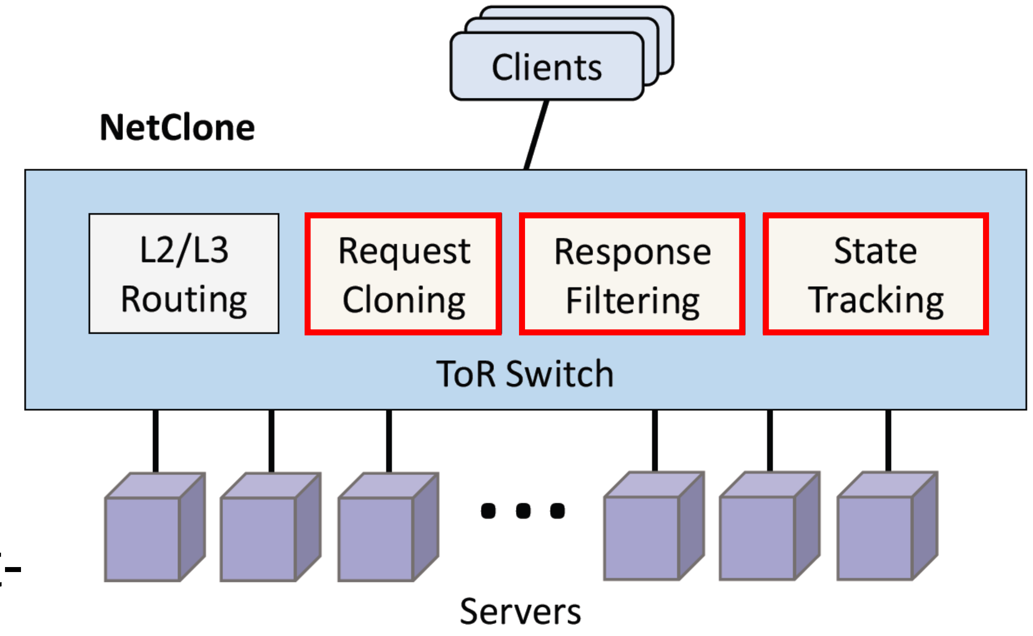
- Requirements achieved by using the switch
  - **Scalability**: Tbps-scale packet processing throughput
  - **Low latency**: cloning decisions in a nanosecond-scale
  - **No sweet spot**: dynamic request cloning in the switch

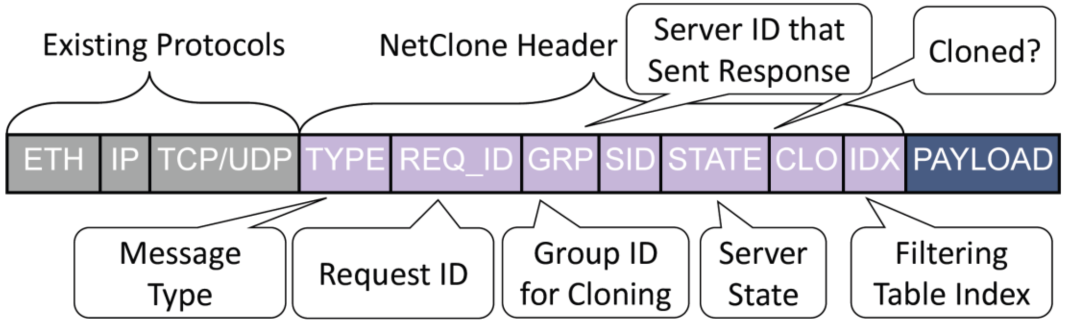
- Strict hardware recourse constraints
  - Limited memory space
  - Limited computational capability

} Design the custom switch data plane by addressing technical challenges

NetClone Architecture

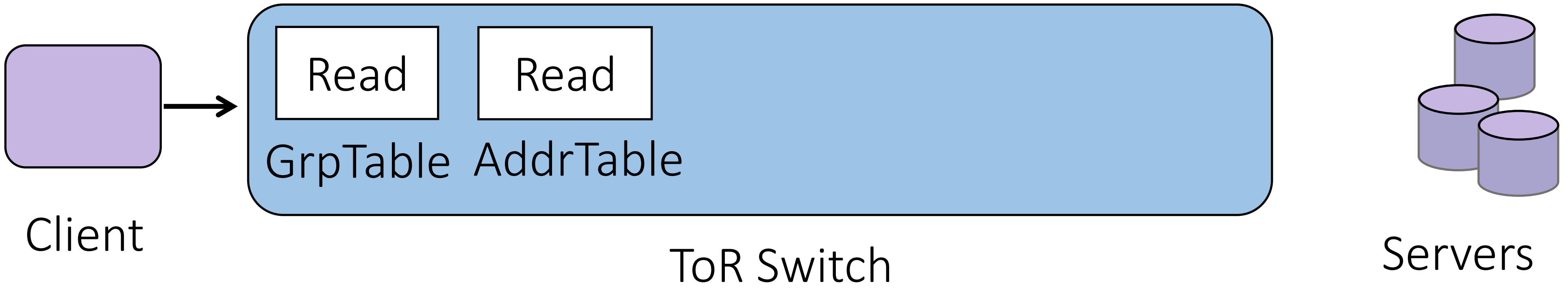
- **Request cloning module**
  - Clones requests only if the two selected servers are idle
- **Response filtering module**
  - Drops the slower response to reduce client-side overhead
- **State tracking module**
  - Keep track server states
- **Custom header**
  - Support NetClone functionality





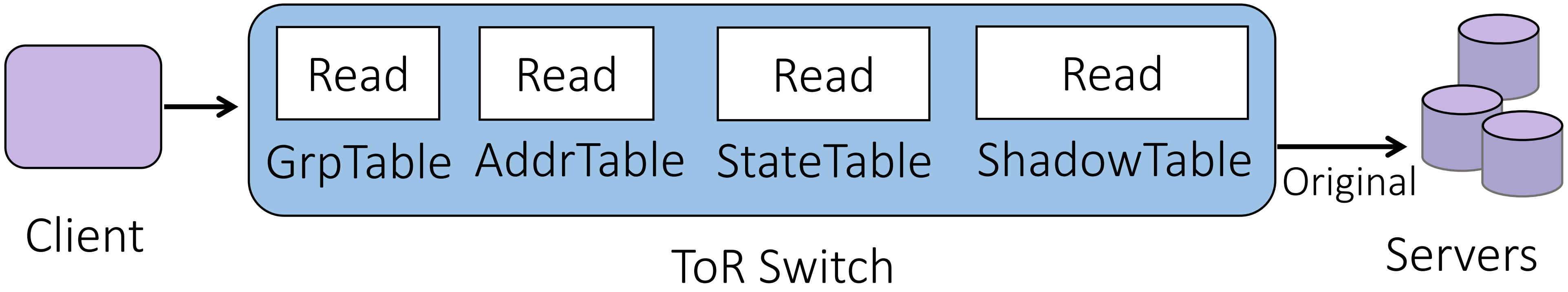
Dynamic Request Cloning

- Step 1: gets the IDs of two candidate servers
- Step 2: sets the dest. IP address to server 1



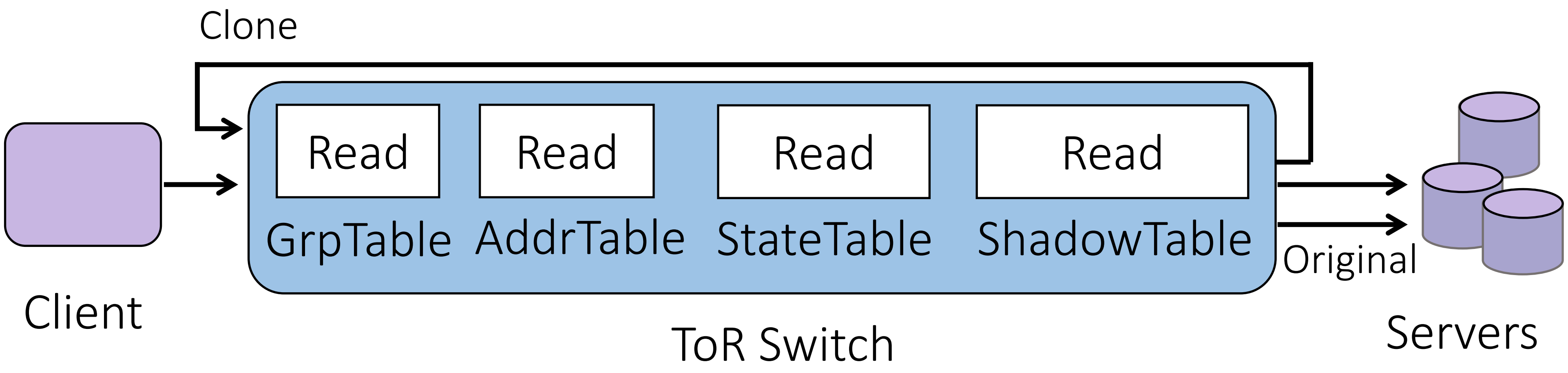
Dynamic Request Cloning

- Step 3: read the state of server 1
- Step 4: read the state of server 2
- Step 5 (If any server is busy): no cloning



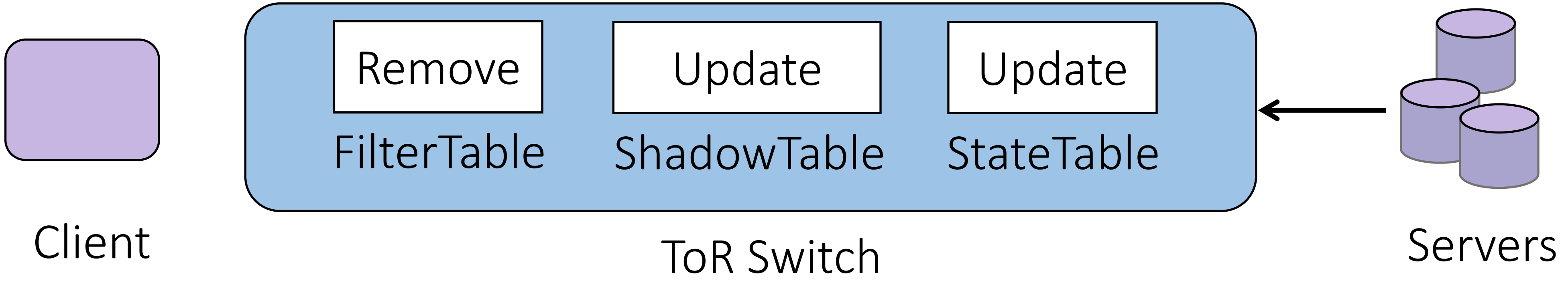
Dynamic Request Cloning

- Step 5 (If both servers are idle) – Clone the request
  - Forward the original request to server 1
  - *Recirculate* the cloned request
- Step 6: update the dest. IP address of the clone to server 2



Response Processing

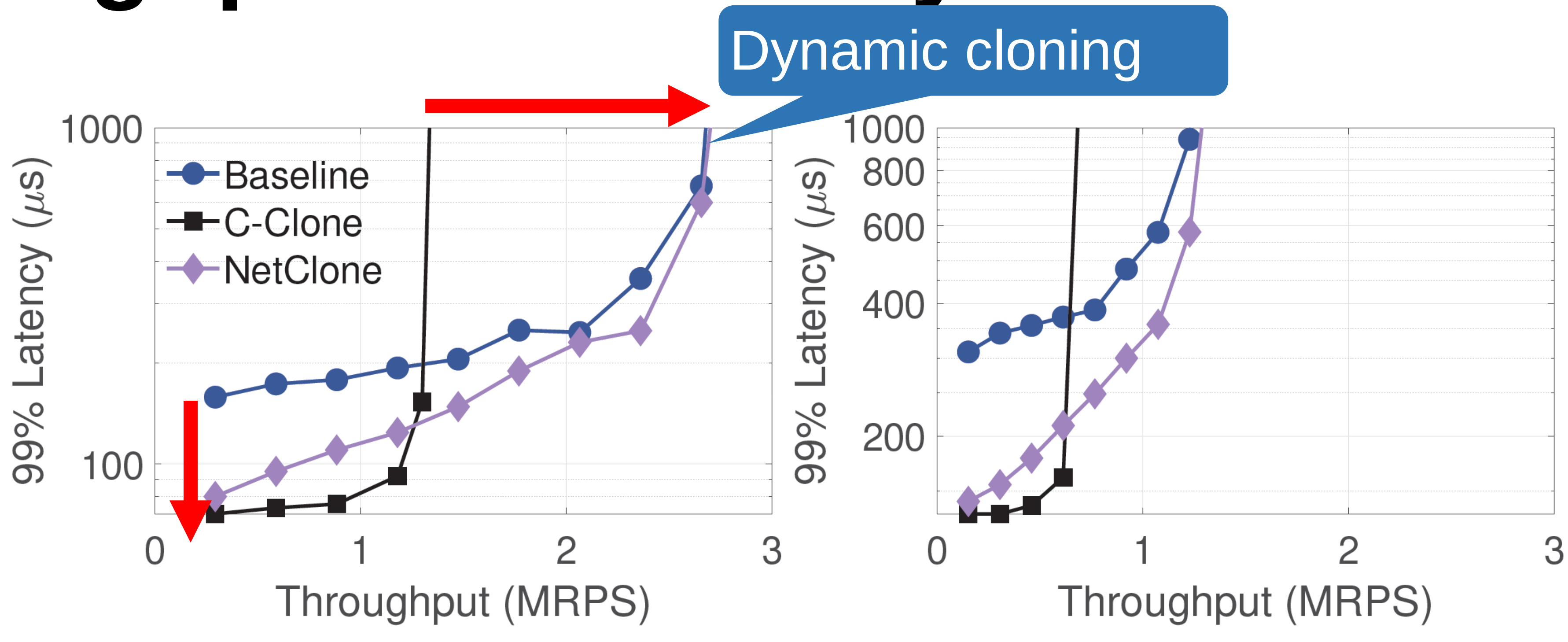
- Step 1: update server states (responses carry server states)
- Step 2: Checks the filter table
  - No matched ID exists: put request ID into the filter table (Faster response)
  - Matched ID exists: drop the response (Slower response)



Evaluation

- Implementation
  - 6.5Tbps Intel Tofino switch ASICs
  - Open-loop multi-threaded applications
- Testbed
  - 6.5Tbps Intel Tofino switch
  - 8 servers with Nvidia ConnectX-5 100G NIC
- Workloads
  - Synthetic workload: exponential and bimodal distributions with dummy RPCs
  - Key-value stores with zipf-0.99

Throughput vs. Latency

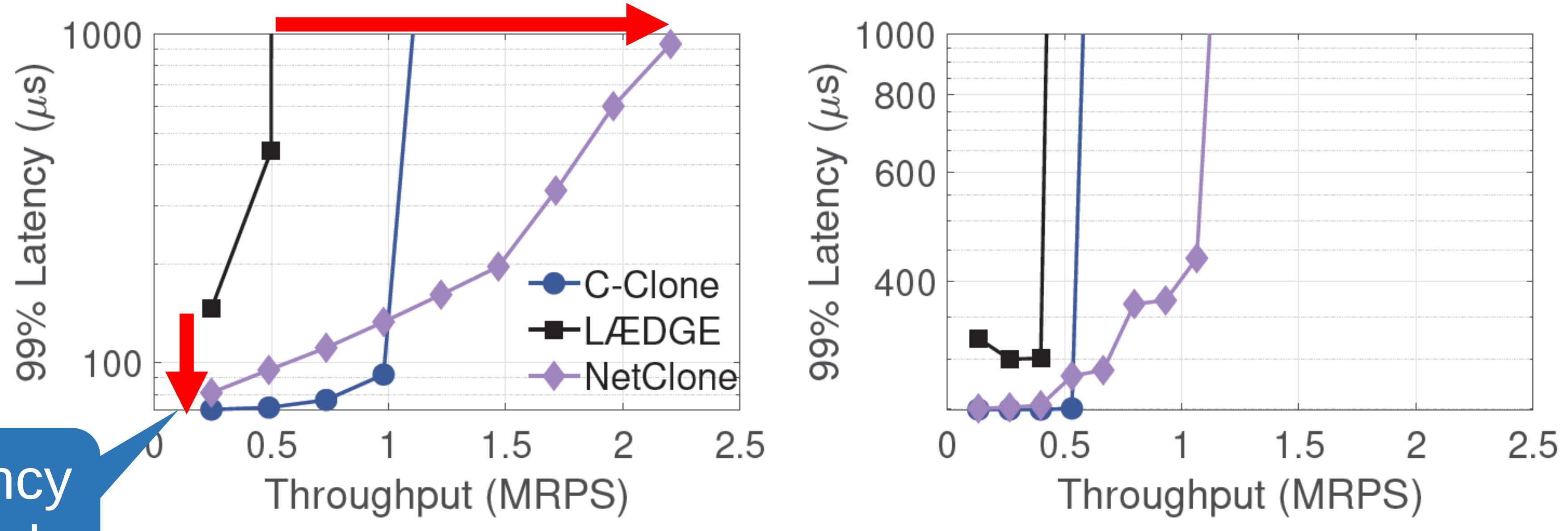


(a) Exp(25)

(b) Bimodal(90%-25,10%-250)

NetClone provides lower tail latency and maintains high throughput

Comparison with LÆDGE

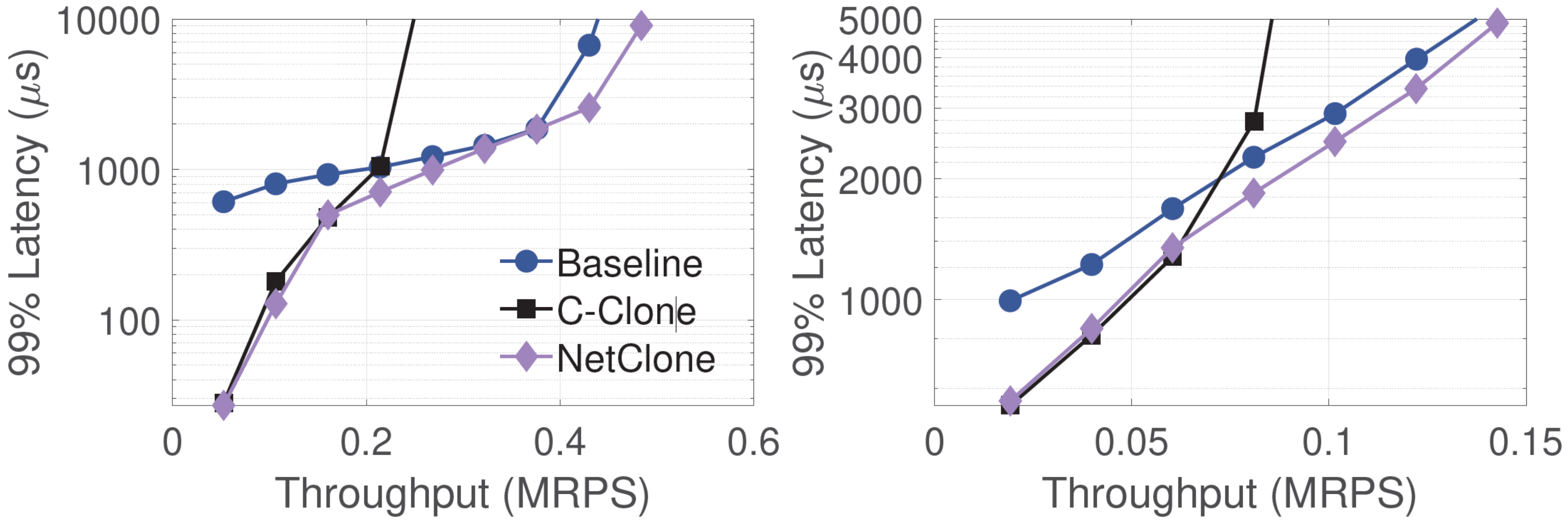


(a) Exp(25)

(b) Bimodal(90%-25,10%-250)

NetClone provies better performance than LÆDGE

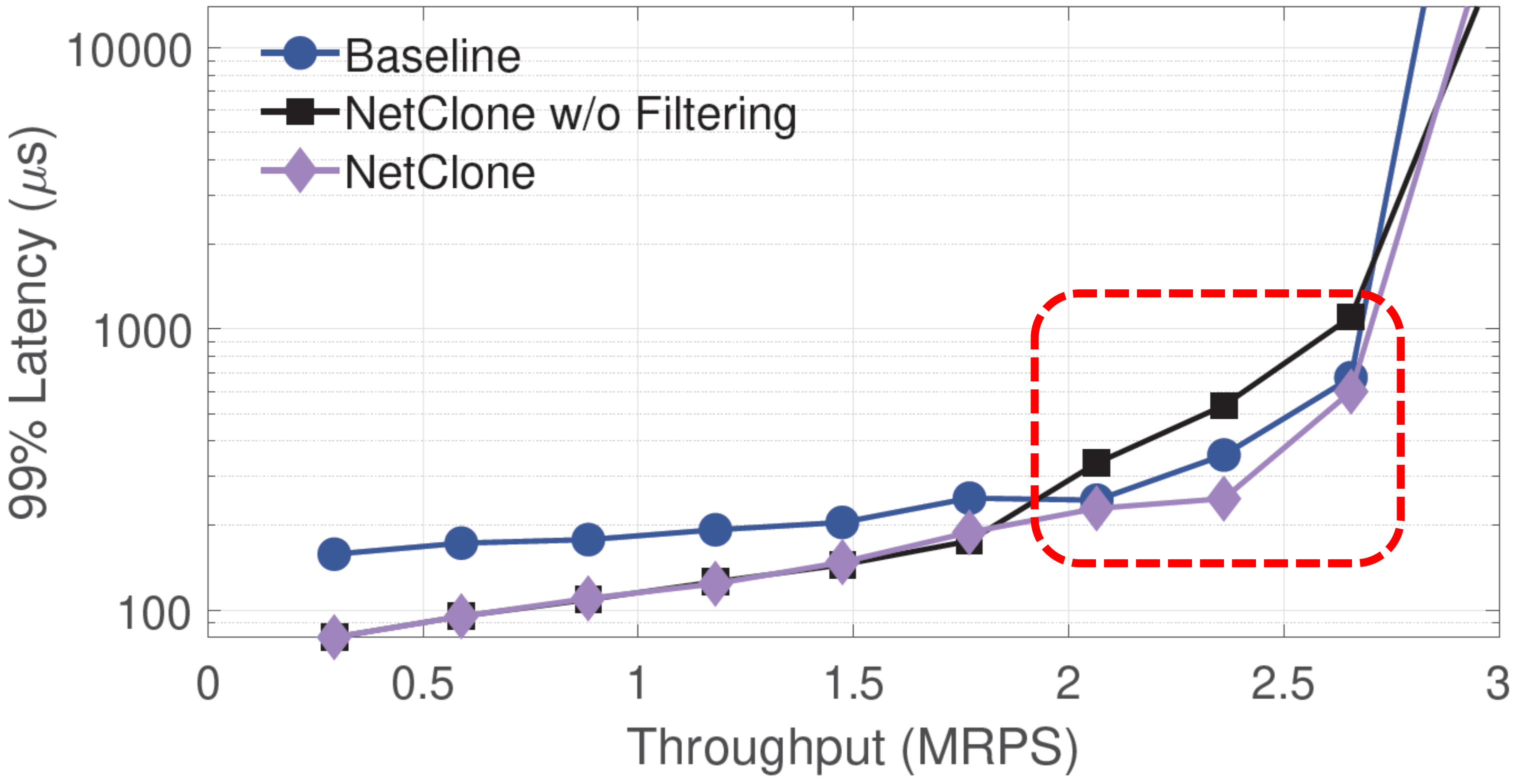
Application: Redis



(a) 99%-GET,-1%-SCAN **(b) 90%-GET,10%-SCAN**

NetClone improves the performance of real-world applications

Impact of Redundant Response Filtering



Response filtering reduces client-side overhead

Conclusion

- Microsecond-scale RPCs require **microsecond-scale tail latency**
- NetClone is a request cloning mechanism that performs **fast, scalable, and dynamic request cloning** by leveraging programmable switches
- Programmable switches can play **a critical role in the era of microseconds!**

Thank you!

Questions?

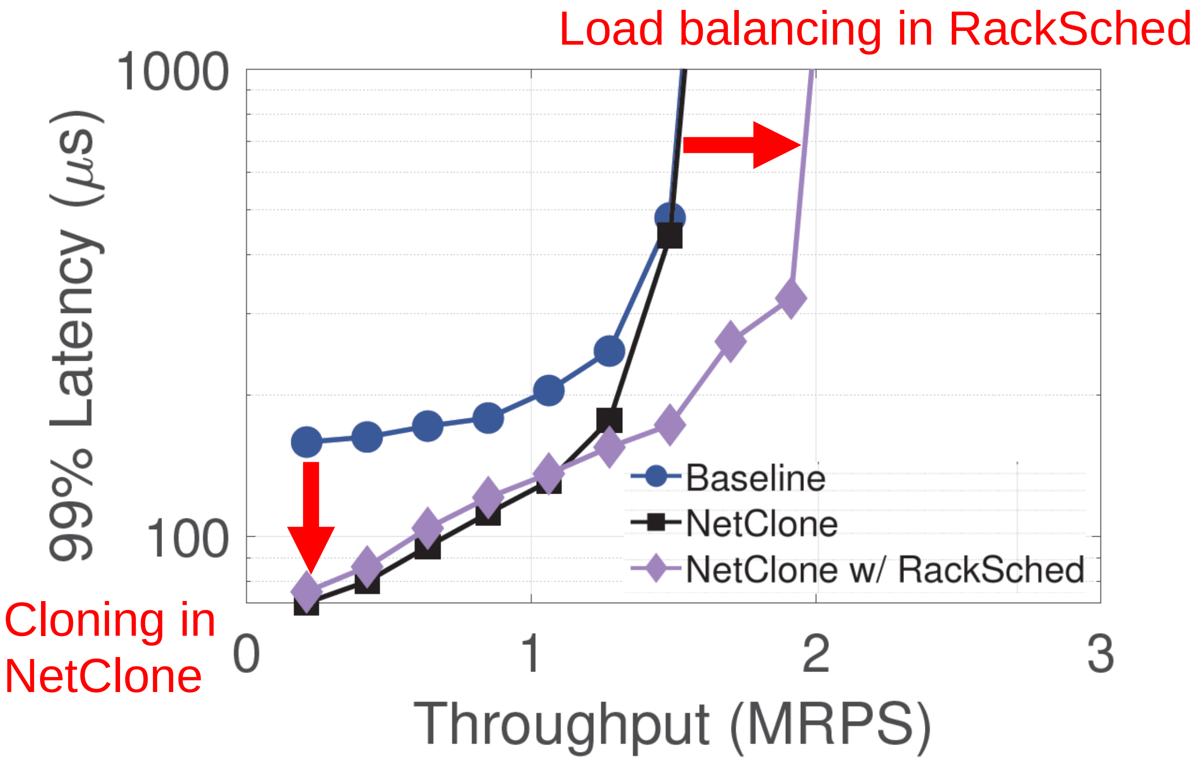
Contact: gykim@sungshin.ac.kr

NetClone prototype code is available at:

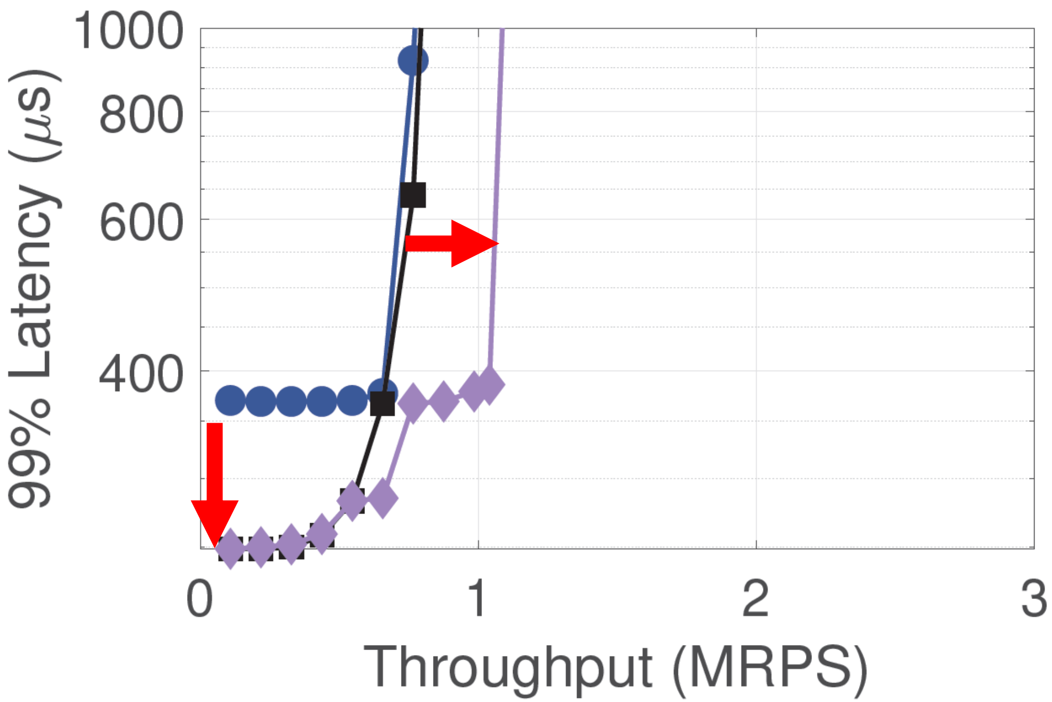
https://github.com/GyuyeongKim/NetClone-public

Appendix

Performance with RackSched [OSDI'20]



Exponential distribution



Bimodal distribution

NetClone can be integrated with an in-network request scheduler

Hang Zhu, Kostis Kaffes, Zixu Chen, Zhenming Liu, Christos Kozyrakis, Ion Stoica, and Xin Jin, "RackSched: A Microsecond-Scale Scheduler for Rack-Scale Computers," in *Proc. of USENIX ODSI*, 2020.