Waverunner: An Elegant Approach to Hardware Acceleration of State Machine Replication

Reza Alimadadi, Hieu Mai, Shenghsun Cho, Michael Ferdman, Peter Milder, Shuai Mu

Hardware Support for Distributed Systems

- Distributed protocols are at the core of many critical systems
  - Cloud infrastructure coordination services (e.g., Chubby)
  - Large-scale distributed databases (e.g., Spanner)
- In these systems, protocol handling incurs massive perf. overheads
  - Beyond network latency, messages must cross PCIe and reach software
  - Beyond software logic, CPU must parse, process, and generate packets
- Hardware accelerators hold the key to high performance
  - Reach high throughputs at minimal latencies ✓
  - Pose major programmability challenges ✗
    - Every change requires hardware development

Our goal: Hardware accelerator performance, Software flexibility

Waverunner: Elegant Distributed Protocol Acceleration

- Two existing approaches to resolve performance challenge
  - Leave implementation in software, minimize overheads
  - Move everything to hardware, pay programmability cost
- We present a new hybrid software/hardware approach
  - Leave all the complexity in software
  - Design way to move only simplest common operations to hardware
- Our Waverunner prototype accelerates off-the-shelf software
  - Take off-the-shelf complex Raft protocol & application (e.g., Memcached)
  - Move only the most common function to hardware (~200 lines of code)
  - Achieve 100Gbps line-rate at 1.8µs tail latency

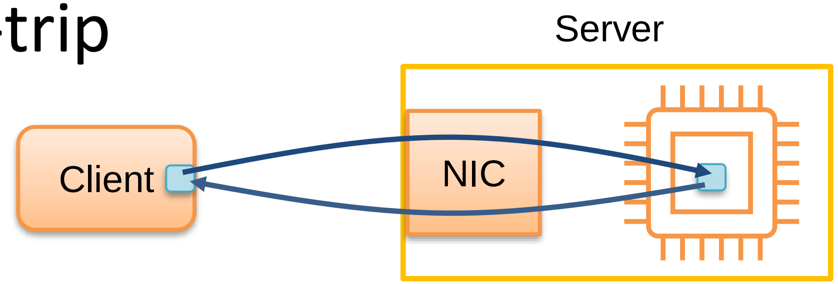
Hardware speeds with mostly unmodified Raft software

Outline

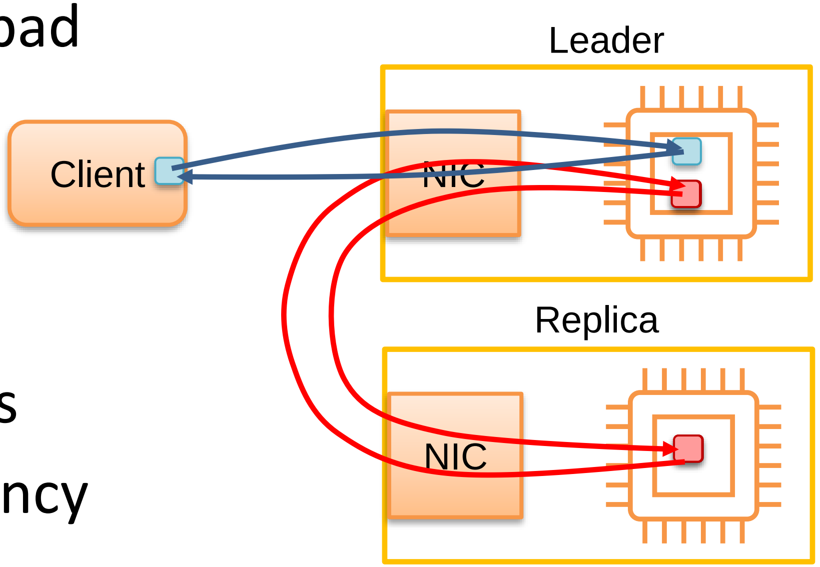
- Introduction
- Distributed Protocol Performance
- Waverunner Design
- Result Highlights
- Conclusions

Distributed Protocol Performance Overheads

- Traditional software has one NIC-CPU round-trip



- Distributed protocol incurs many “extra” round-trips
  - Modern networks are fast: network hops not so bad
  - Multiple (4 here) extra PCIe traversals
  - Large amount of extra CPU work
- Overheads reduce performance
  - CPU is busy processing packets, throughput drops
  - PCIe crossings and software interaction adds latency



Existing Hardware Acceleration Techniques

- Kernel bypass to avoid OS overheads
  - Passes data directly between NIC and software (e.g., DPDK, RDMA)
  - Improves throughput and latency
  - Leaves CPU as the bottleneck for processing all protocol packets

- FPGA hardware for bespoke accelerators
  - Re-implements entire protocol and application in hardware (e.g., ZABFPGA)
  - Achieves line rate throughput and minimal latency
  - Eliminates ability to evolve or bugfix system
    - Any change takes several hours to build

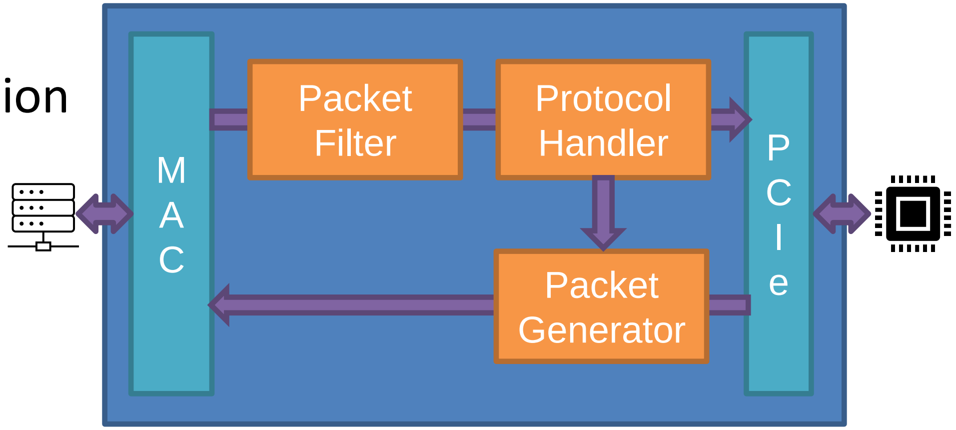
Waverunner Insight

- Possible to partition distributed protocol into two parts
  - Complex software logic (90% of Raft code)
    - Leader election, failure recovery, etc.
    - Too rare to meaningfully impact performance
  - Simple common routines (10% of Raft code)
    - Broadcast to replicas, commit on quorum, etc.
- We build Waverunner accelerator based on (these) observations
  - Repetitive common case is handled in hardware
    - Maximum throughput, minimum latency, no CPU involvement for protocol handling
    - Application handles only the client requests (like traditional software)
  - Any deviation from the common case switches to software
    - Waverunner passes all unknown/unhandled messages to software
    - Waverunner detects error conditions, but punts to software to handle them

Make the common case fast, and the uncommon case correct

Waverunner Hardware Design

- Based on a conventional NIC
- Adds three new Raft-specific hardware components
  - Packet Filter
    - Identifies common Raft packets and instructs Protocol Handler to process them
  - Protocol Handler
    - Raft logic for handling client requests on leader and leader requests on followers
  - Packet Generator
    - Used by Protocol Handler to initiate packet transmission
- Protocol Handler: ~200 lines of C++
  - Simple code, HLS-translated to hardware



Waverunner Software Design

- Waverunner operates in full-software and accelerated modes
  - In full-software mode, CPU executes all Raft functionality
    - Leader election, failure recovery, etc.
  - In accelerated mode, CPU runs “commit” message handler
    - Switches to full-software mode if failure is detected
- Waverunner integrated with real-world applications
  - Memcached, Redis, and simple K-V hash (std::map<>)
  - Requests forwarded to application without OS involvement (similar to DPDK)

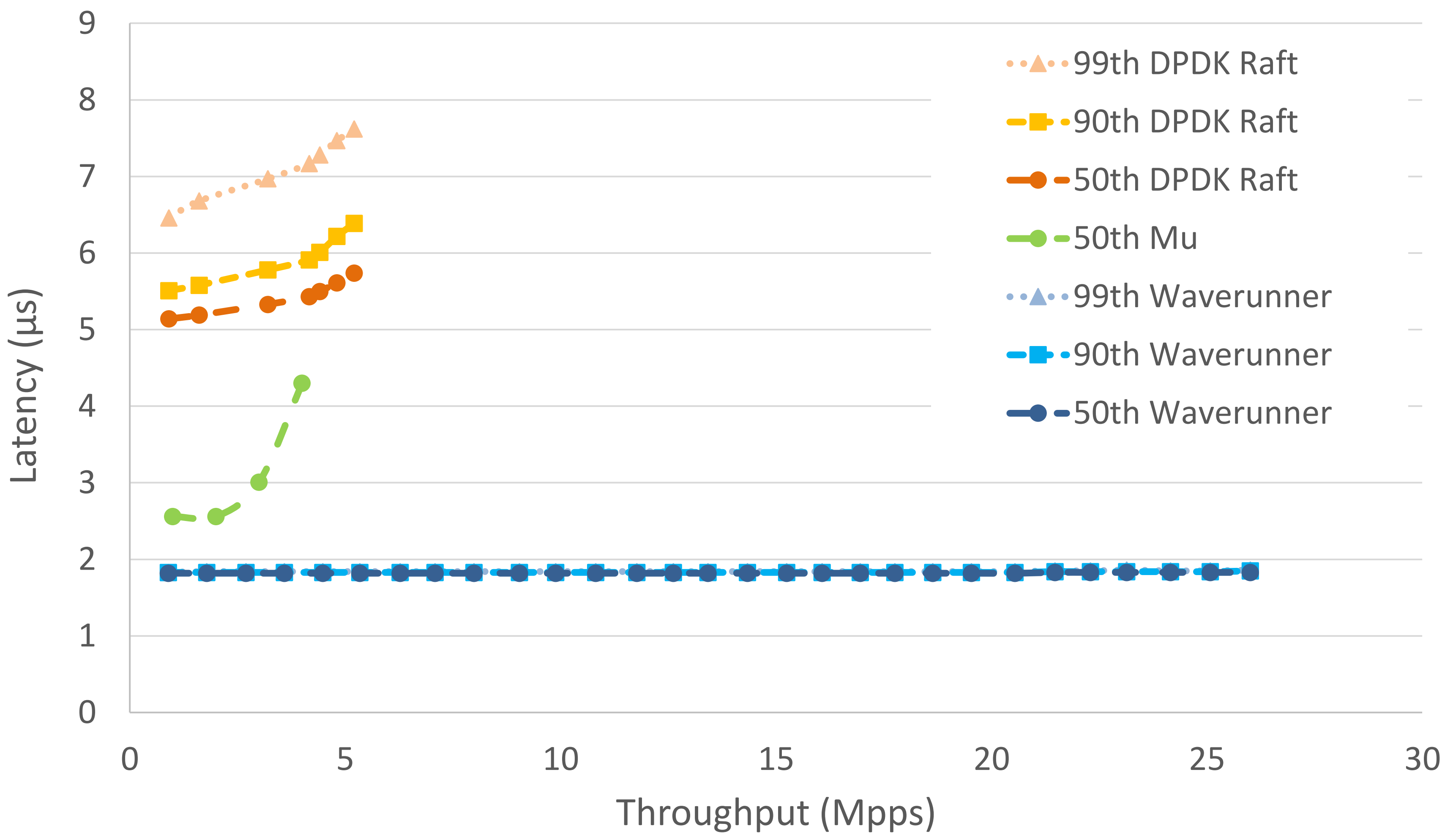
Outline

- Introduction
- Distributed Protocol Performance
- Waverunner Design
- Result Highlights
- Conclusions

Experimental Setup

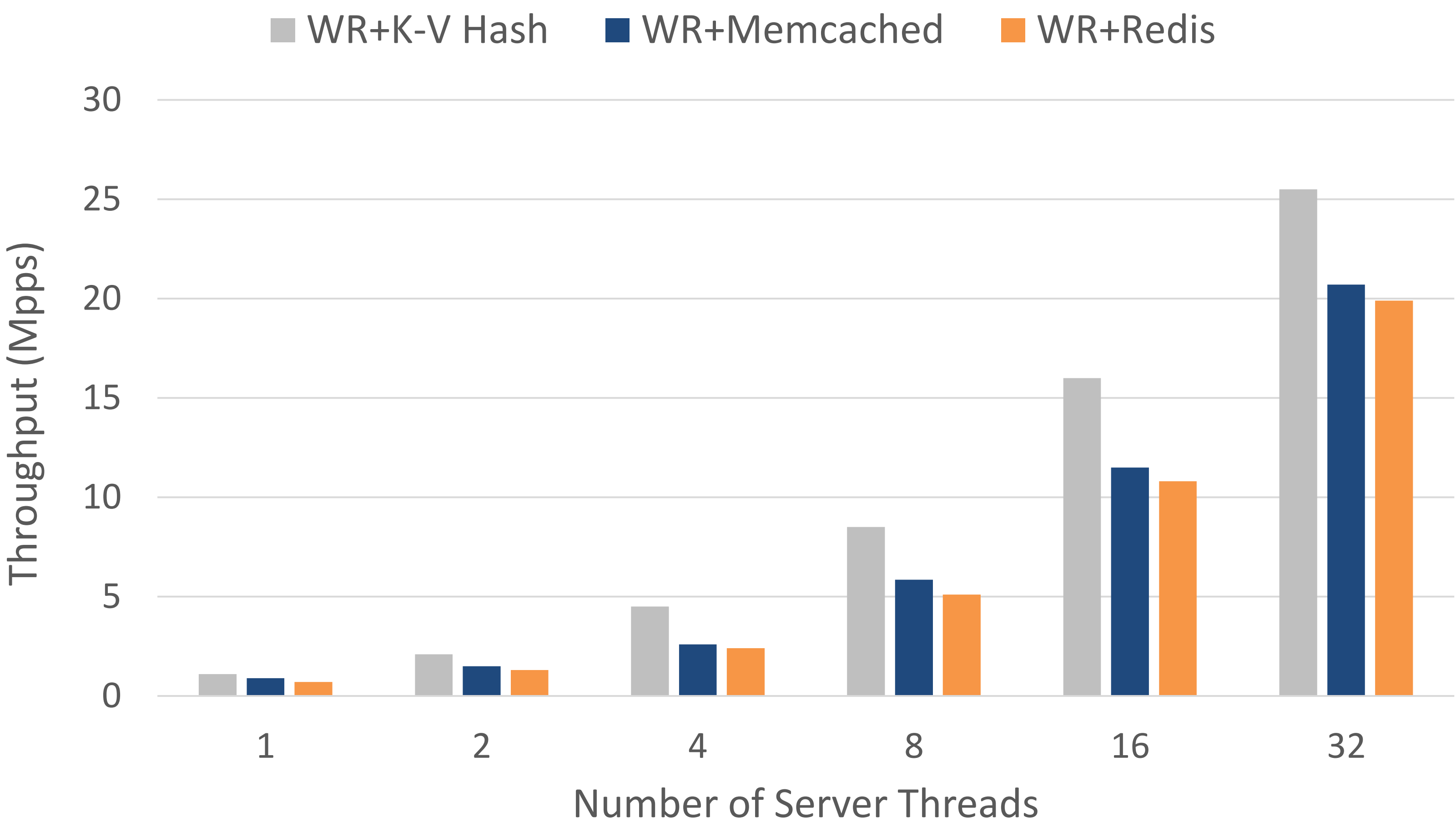
- Three Linux servers
  - Xilinx U280 FPGA with two 100G ports
  - Mellanox ConnectX-4 (100G NICs)
- Eight client hosts
  - Each with ConnectX-4 Lx (2x 25G NICs)
- Open-loop clients bombard leader with min-size requests
  - 32 Bytes payload (133 Bytes packet size)
  - Perform no batching at client

Replication Latency



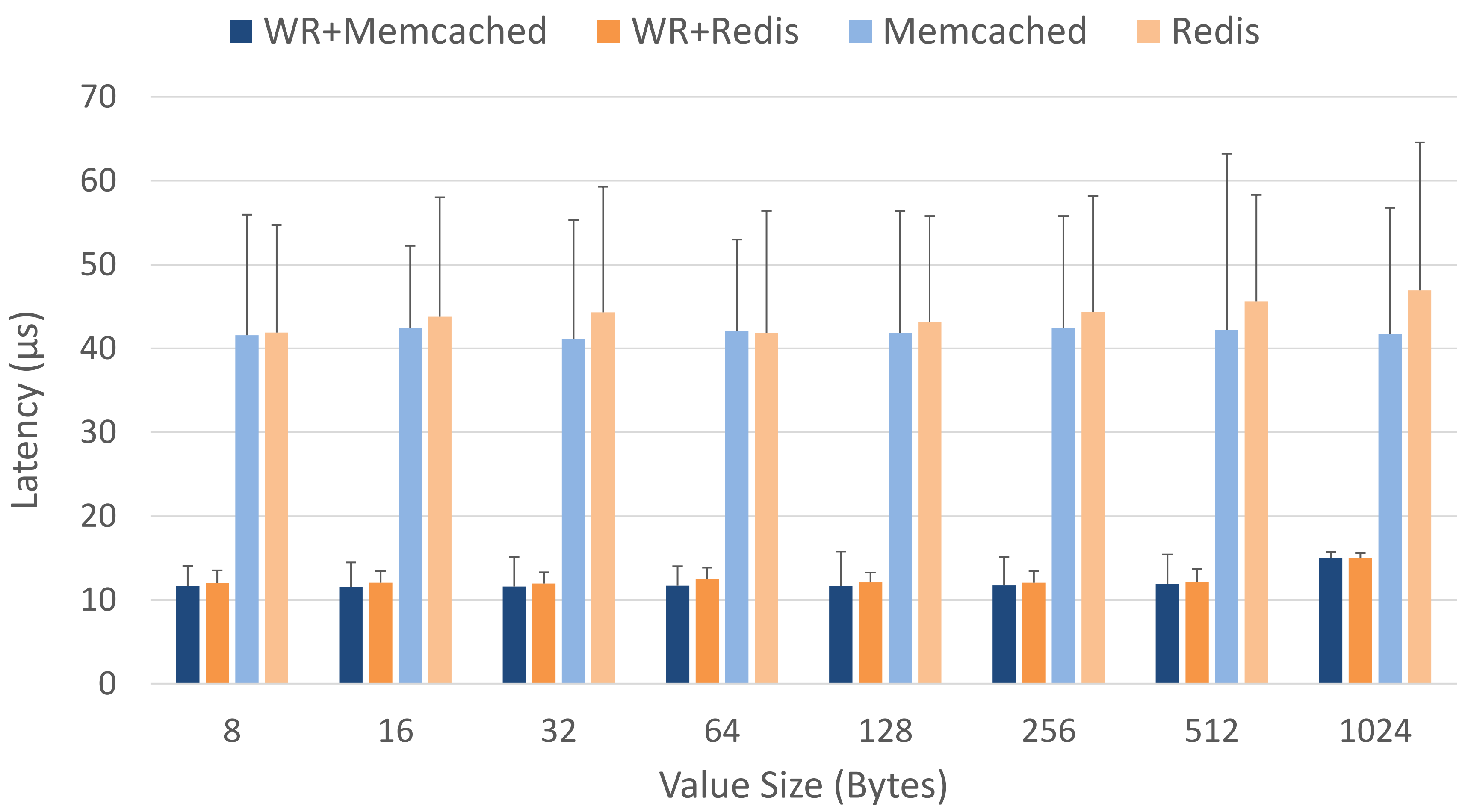
Waverunner reaches 100G line rate (85.5G goodput) with min latency

End-to-End Application Throughput



Memcached and Redis achieve 20Mpps, 14x higher than non-accelerated

End-to-End Application Latency



Waverunner latency under 12.5µs, 3.5x better than non-HA application

Conclusions

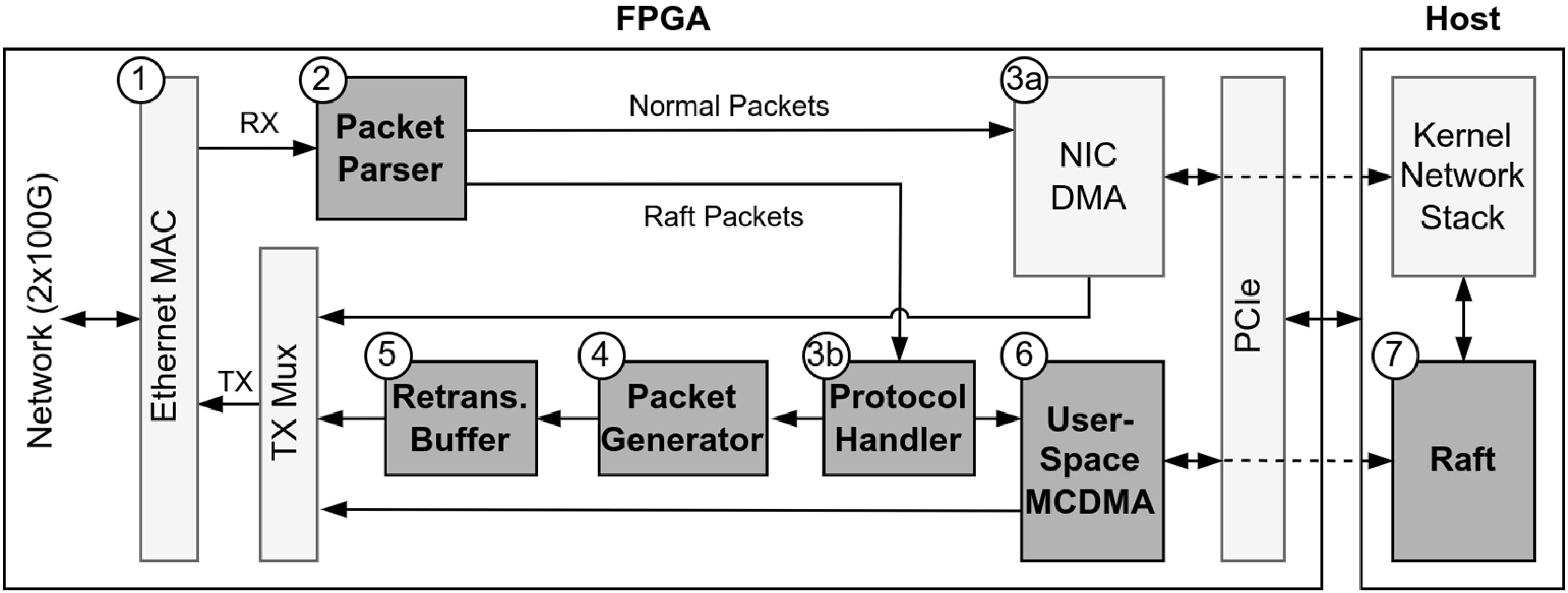
- Distributed protocol handling incurs massive perf. overheads
  - Software implementations too slow, hardware implementations too inflexible
- We developed Waverunner: elegant hybrid software/hardware approach
  - Handles simple common operations in hardware
  - Leaves everything else (infrequent and complex operations) in software
- Waverunner runs Raft protocol w/min-size packets
  - 100Gbps and constant tail latency, using only ~200 lines of C++ hardware code

Backup Slides

CPU Cycle Analysis

	Usage Ratio (%)		
	Control Plane	Data Plane	Application
Our Raft	~0 (1e-8)	88	12
NuRaft	~0 (1e-4)	92	8
etcd	~0 (1e-4)	72	28

Waverunner Hardware Detailed



Replication Latency