MegaPipe : A New Programming Interface for Scalable Network I / O

Sangjin Han+, Scott Marshall+, Byung-Gon Chun*, and Sylvia Ratnasamy+

+University of California, Berkeley; *Yahoo! Research

OSDI, 2012

Presented By Dong Yuan & Zhihui Deng
(2015210938 2015210926)

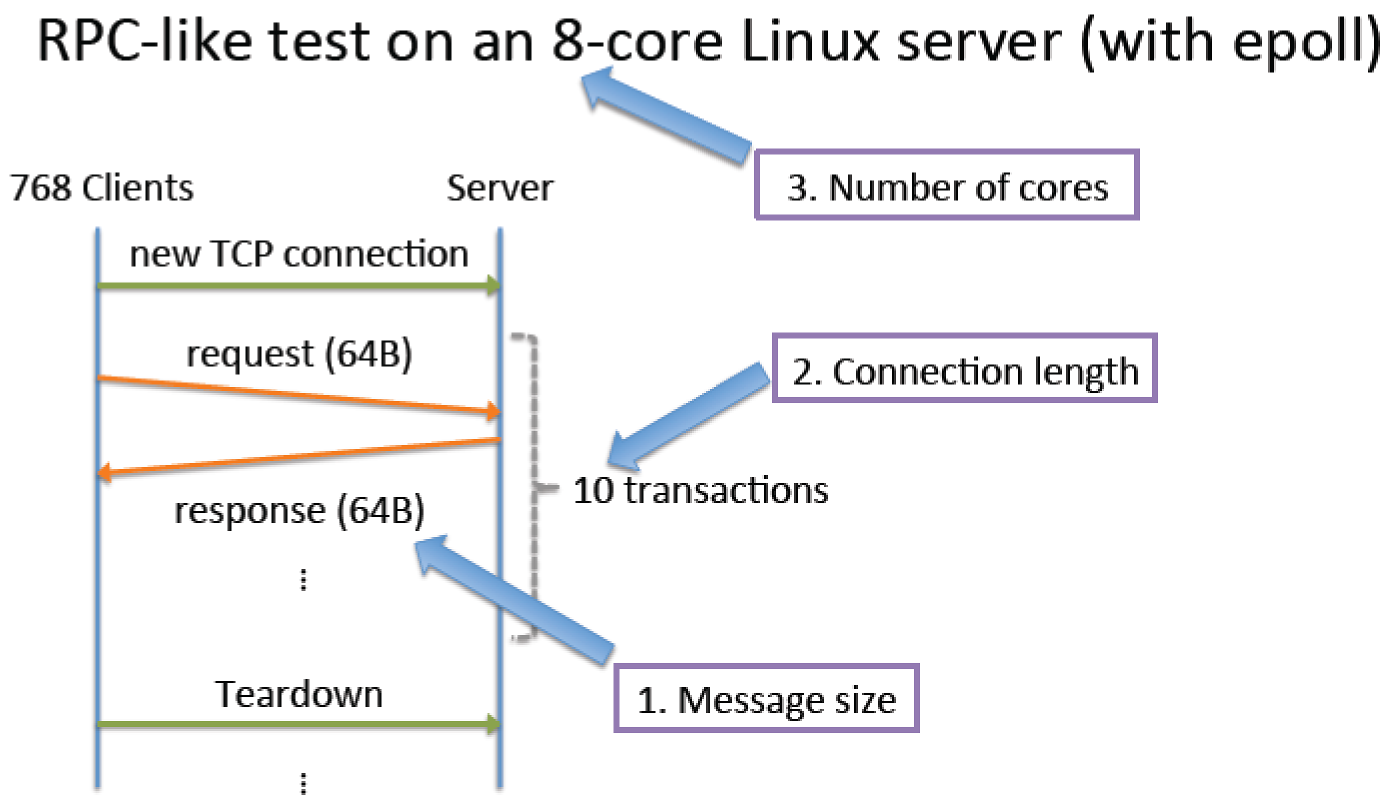
Background

- Message-Oriented Workload
  - Short connections or small messages
    - Examples: HTTP, RPC, DB
- Issues with message-oriented workloads
  - System call overhead
  - Shared listening socket
  - File abstraction overhead

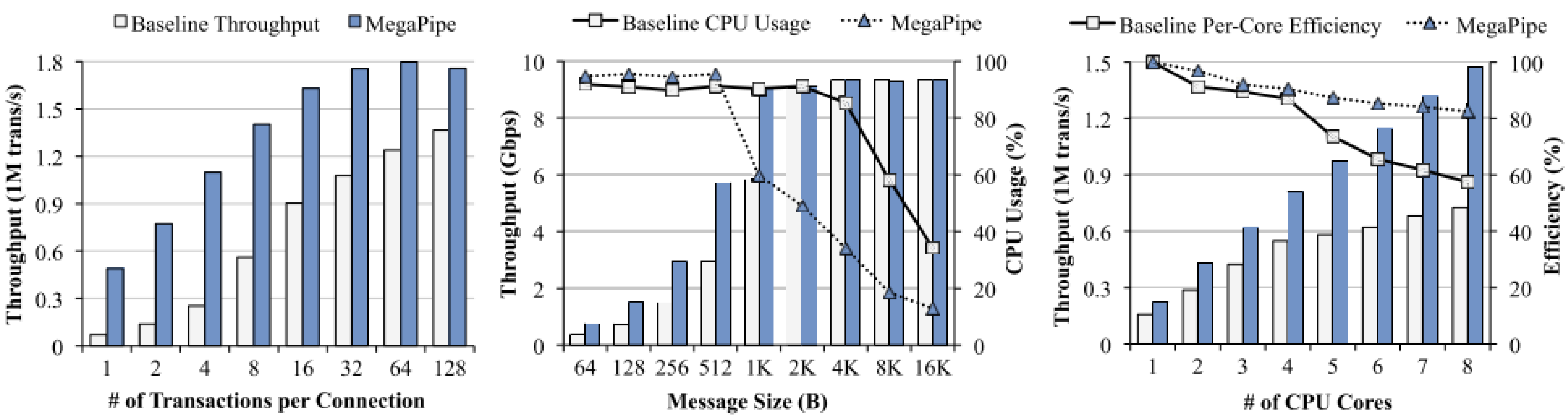
Solved Issues Comparison with mTCP & Fastsocket

- MegaPipe (OSDI, 2012)
    - System call overhead
    - Shared listening socket
    - File abstraction overhead
- mTCP (NSDI, 2014)
    - Shared resources
    - Broken locality
    - Per packet processing
- Fastsocket (ASPLOS, 2016)
    - Shared resources
    - Broken locality
    - Uncompatible API

Microbenchmark



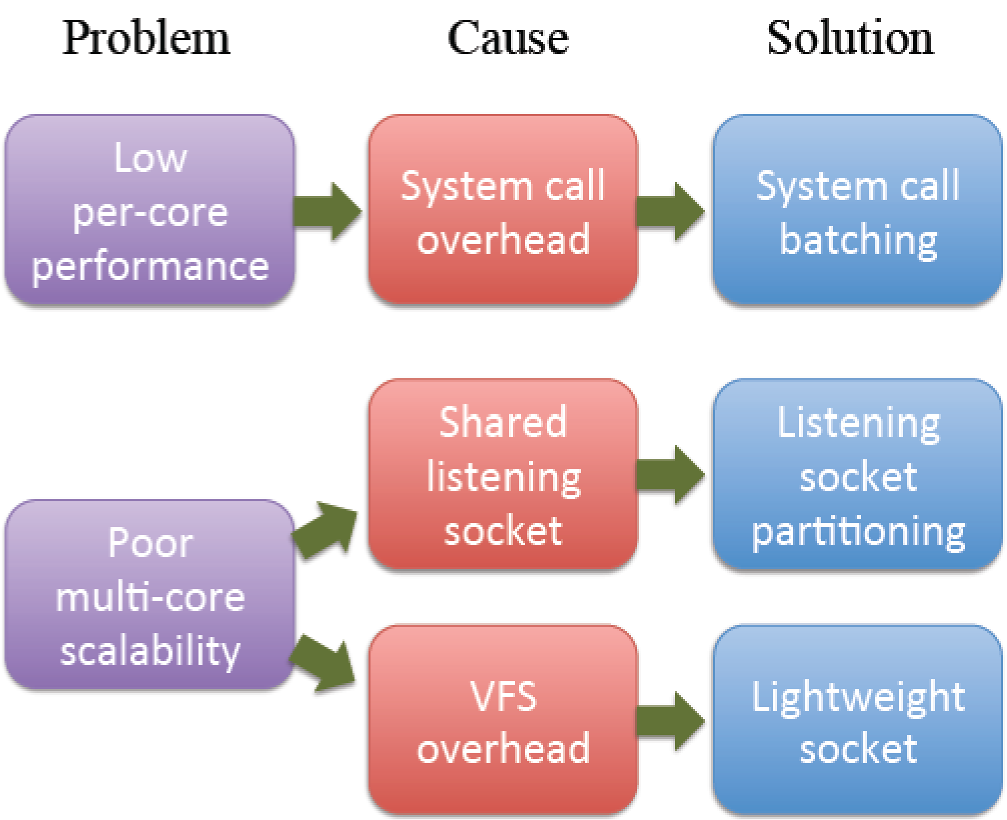
Performance of Message-Oriented Workloads



Design Goal

- API, applicable to existing event-driven server applications with moderate efforts
- Unified interface for various I/O types, TCP connection, UNIX domain sockets, disk files…
- Low overhead & multi-core scalability

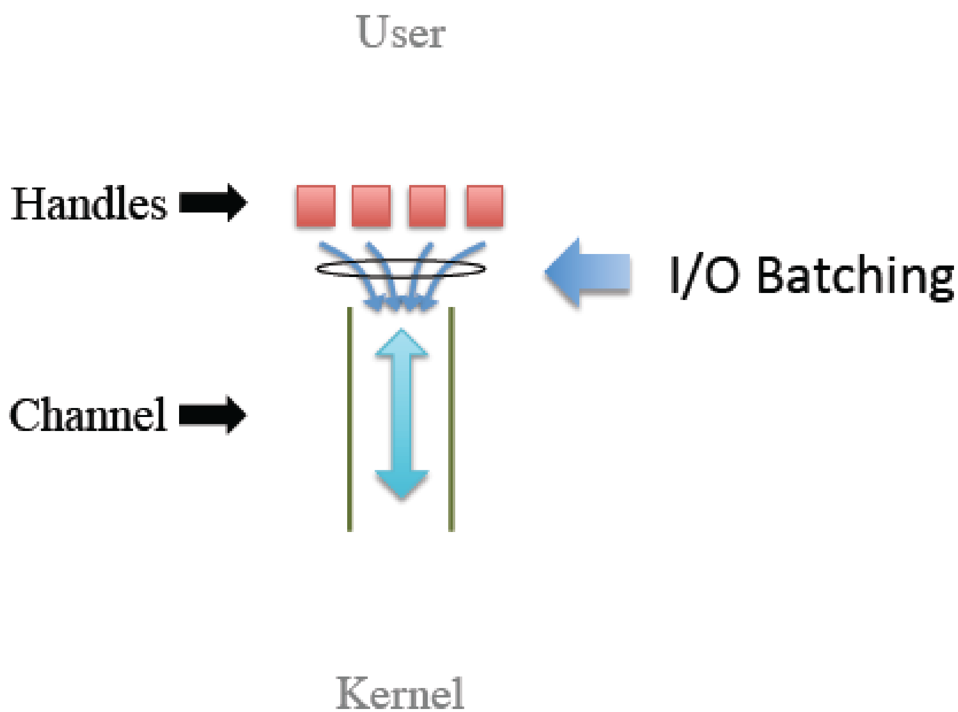
Overview



Problem
Cause
Solution
Low per-core performance
System call overhead
System call batching
Shared listening socket
Listening socket partitioning
Poor multi-core scalability
VFS overhead
Lightweight socket

Key Primitives

- ***Handle***
  - Similar to file descriptor
    - But only valid with in a channel
  - TCP connection, pipe, disk file…
- ***Channel***
  - A per-core, bi-directional pipe between the kernel and user
  - Multiplexes I/O operations of its handles



Completion Notification Model

- Application issue asynchronous I/O commands
- Kernel notifies the application when the commands are complete
- Why CNM?
  - CNM allows transparent batching of I/O commands and notifications
  - It is compatible with not only sockets but also disk files
  - Simplify the complexity of I/O multiplexing

```
epoll_ctl(fd1, EPOLLIN);
epoll_ctl(fd2, EPOLLIN);
epoll_wait(…);
…
ret1 = recv(fd1, …);
…
ret2 = recv(fd2, …);
…
```

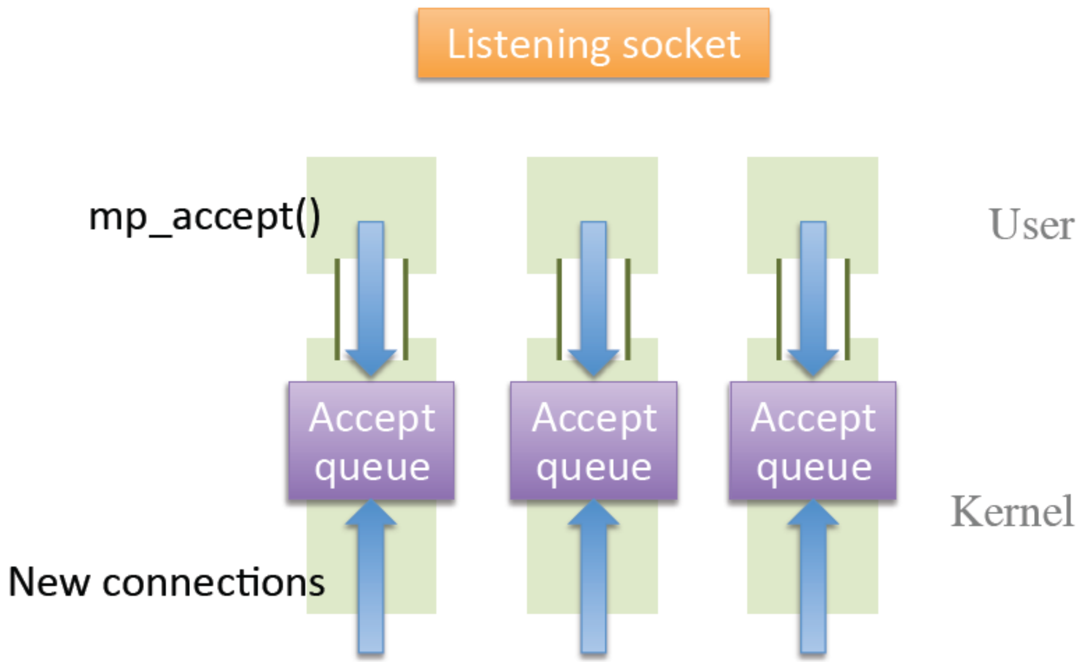
Readiness Model

```
mp_read(handle1, …);
mp_read(handle2, …);
…
ev = mp_dispatch(channel);
…
ev = mp_dispatch(channel);
…
```

CNM

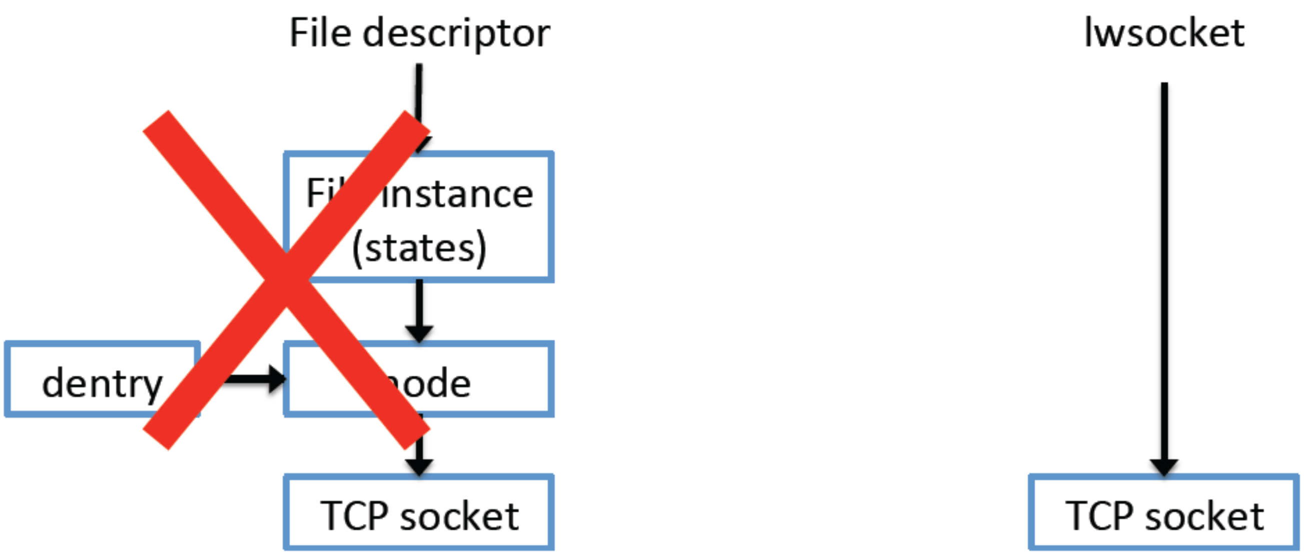
Listening Socket Partitioning

- Per-core accept queue for each channel
  - Instead of the globally shared accept queue



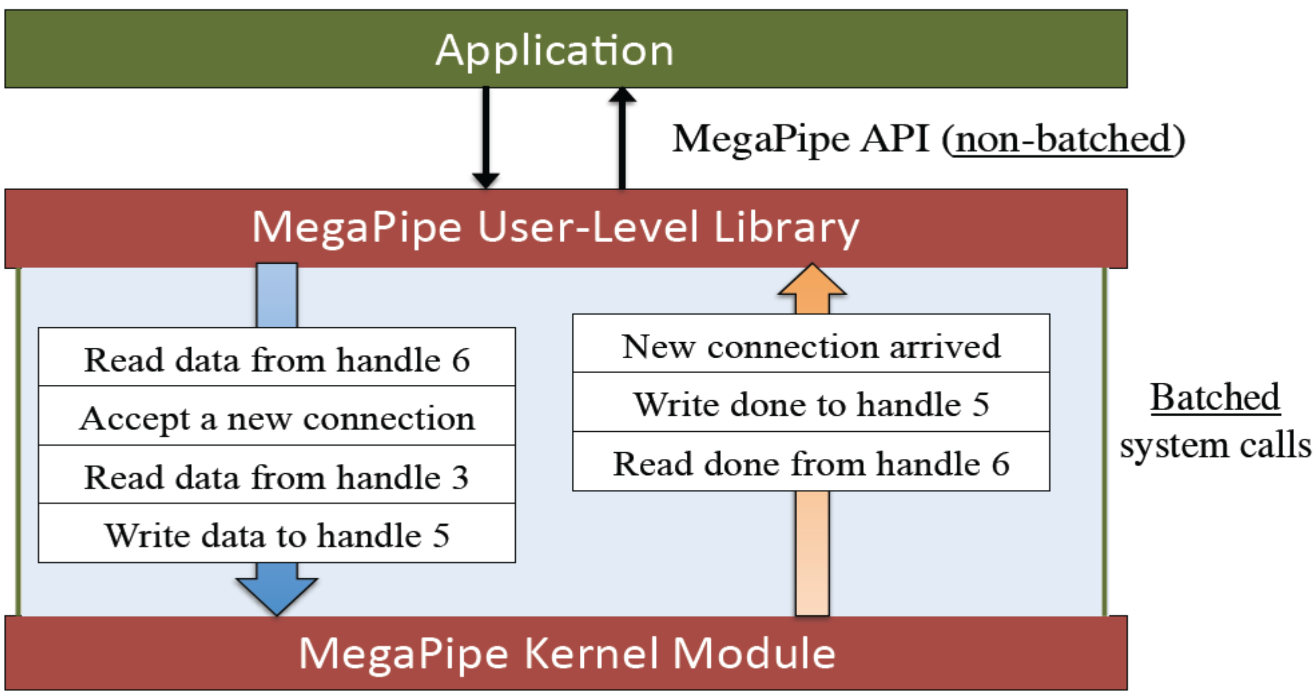
Lightweight Socket

- Sockets are ephemeral and rarely shared
  - Bypass the VFS layer
  - Convert into a regular file descriptor only when necessary



System Call Batching

- System calls are expensive due to cost of mode switching and bad cache locality
- Transparent batching

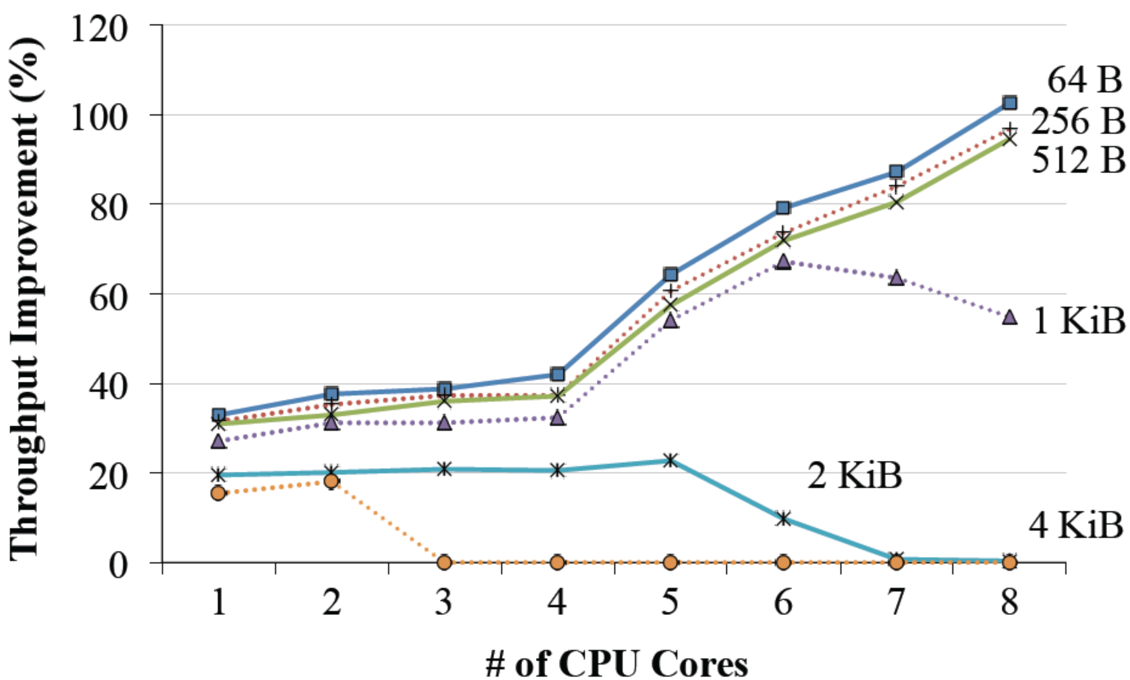


Implementation

- Kernel
  - One kernel module (~1800 lines)
  - Kernel itself (~400 lines)
- User-Level Library
  - ~400 lines
- Application
  - Supportive to event-driven server
  - ~hundreds of lines

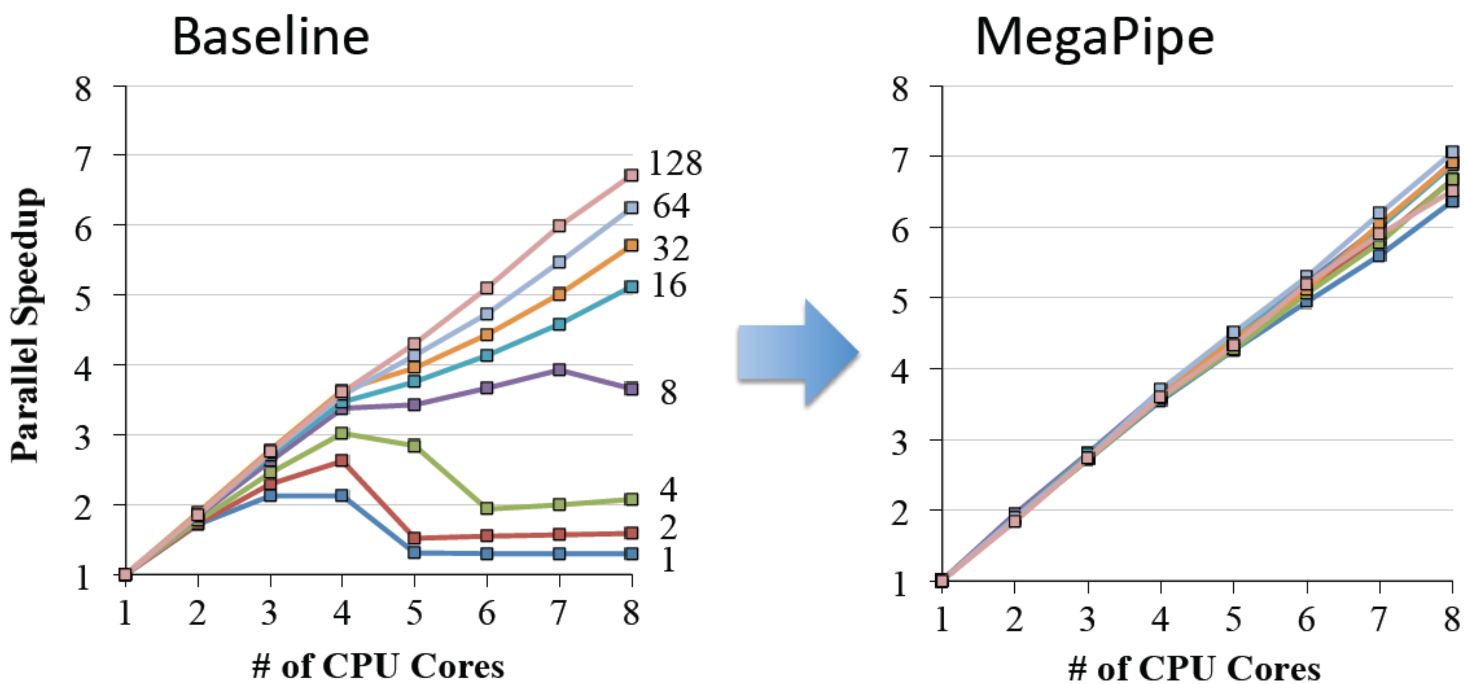
Evaluation

- Multi-core scalability
  - Throughput improvement with various message sizes



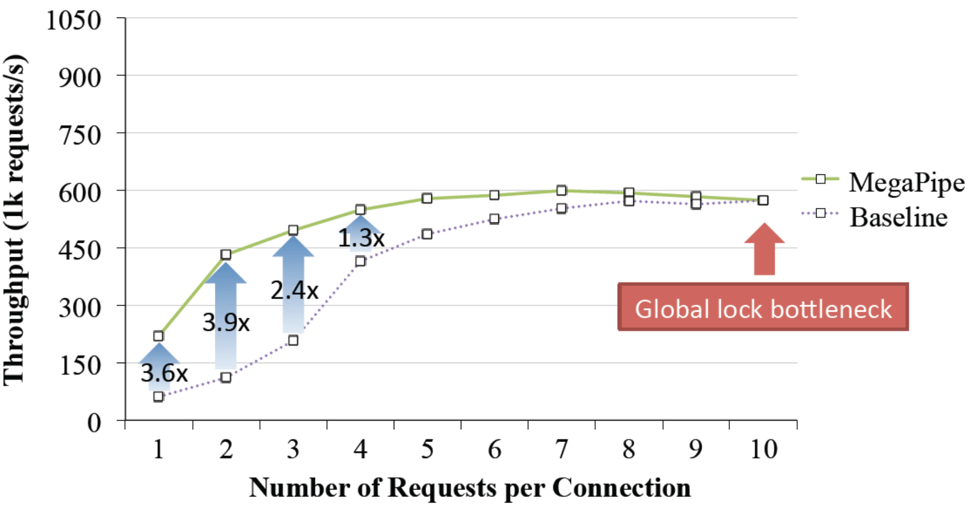
Evaluation

- Multi-core scalability
  - Throughput improvement with various connection lengths (# of transactions)

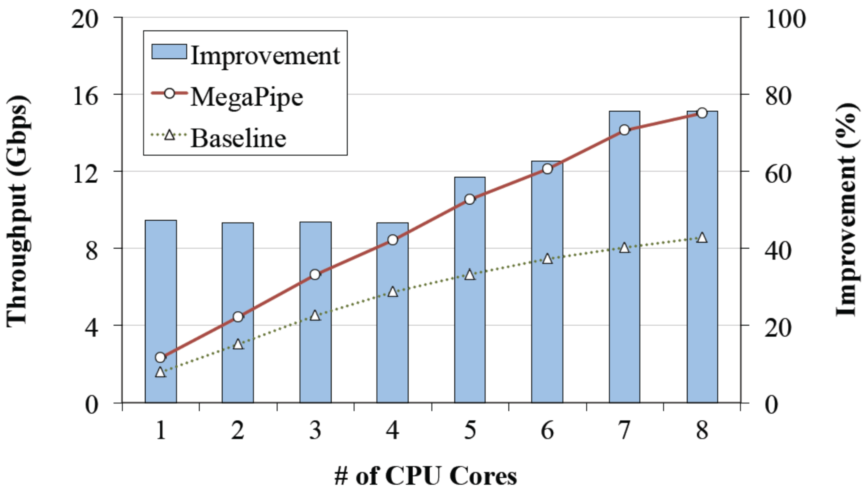


Application Evaluation

- Memcached



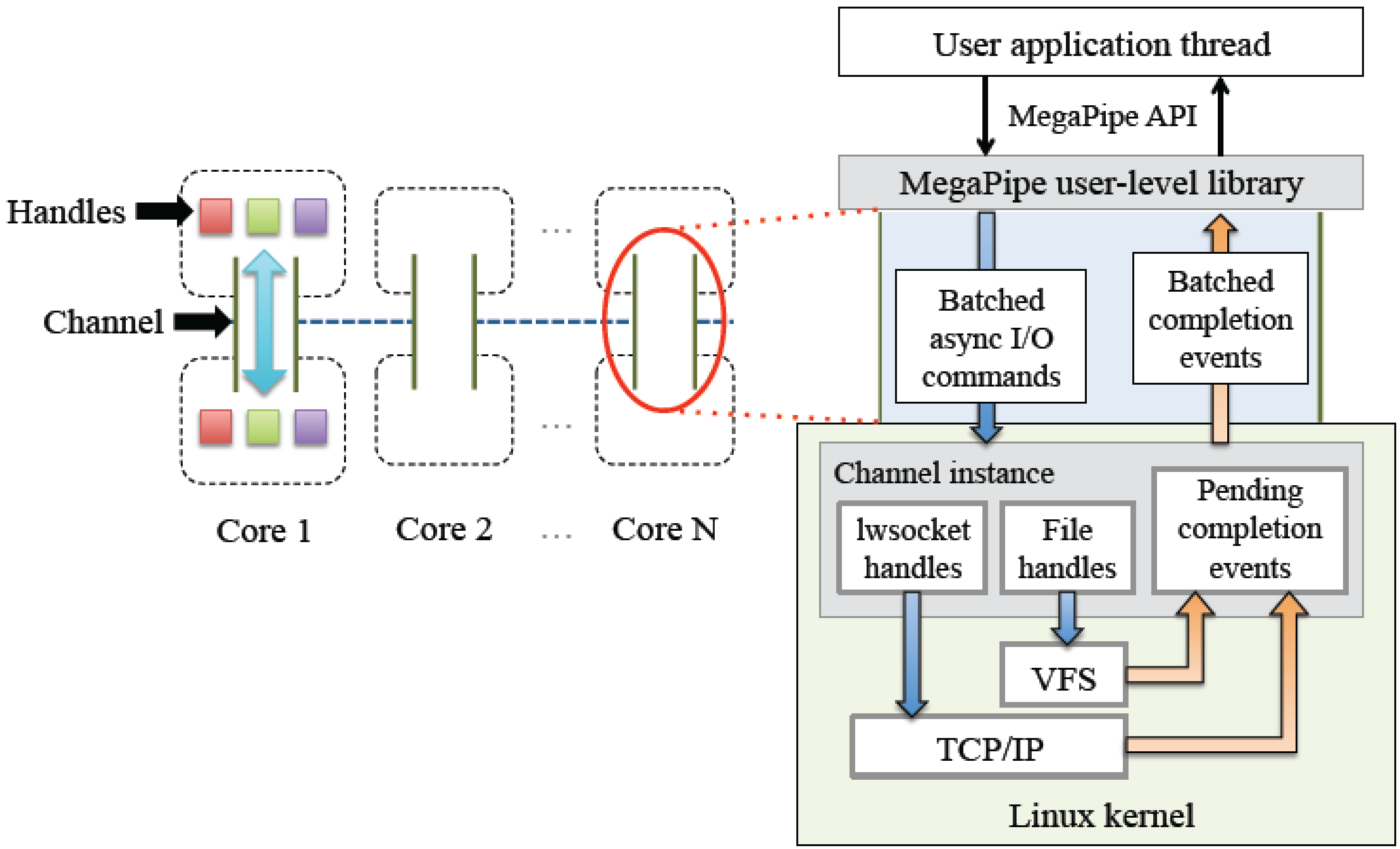
- nginx



Conclusion

- MegaPipe
  - Key abstraction: per-core channel
  - Enabling three optimization:
    - Batching, partitioning, lwsocket
  - Performance improvement in multi-core scalability and application

Comments About This Paper



Thank you!