EXPLORING MTCP BASED SINGLE-THREADED AND MULTI-THREADED WEB SERVER DESIGN

By
Pijush Chakraborty
Roll: 153050015

Guided By:
Prof. Sriram Srinivasan
Prof. Purushottam Kulkarni

User Space Network Stack

- Why do we need an user-space network stack?

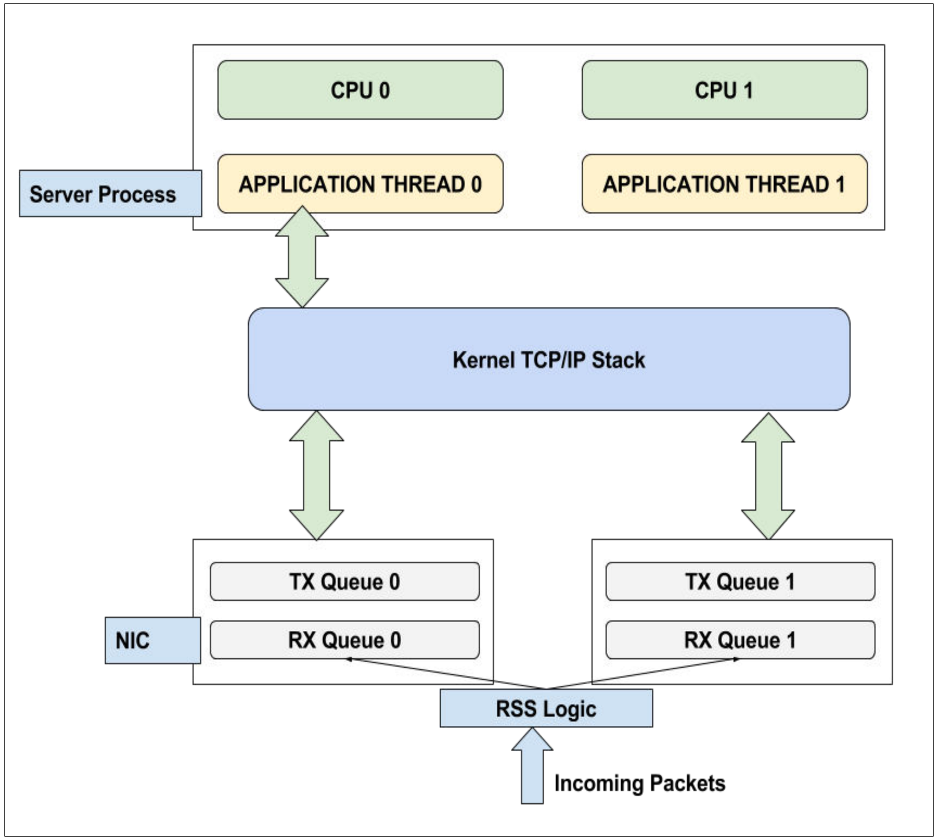


Figure: Kernel Network Stack

1. Single Kernel Stack
2. System Call Overhead
3. Heavy Shared Data-Structures

mTCP: Multicore User-Level TCP Stack

- Design of mTCP stack:
  - No Sharing Model
  - Per Core Independent TCP Stacks
  - Per Core Flow Affinity

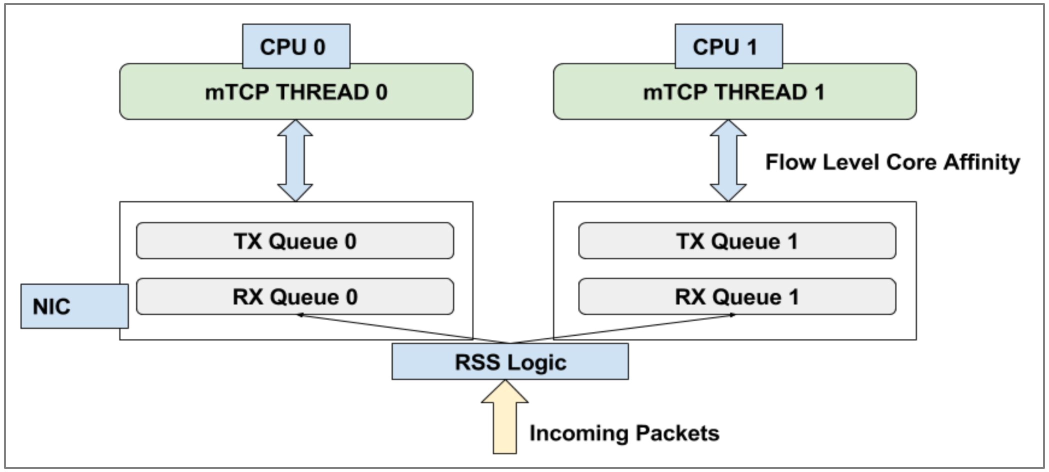
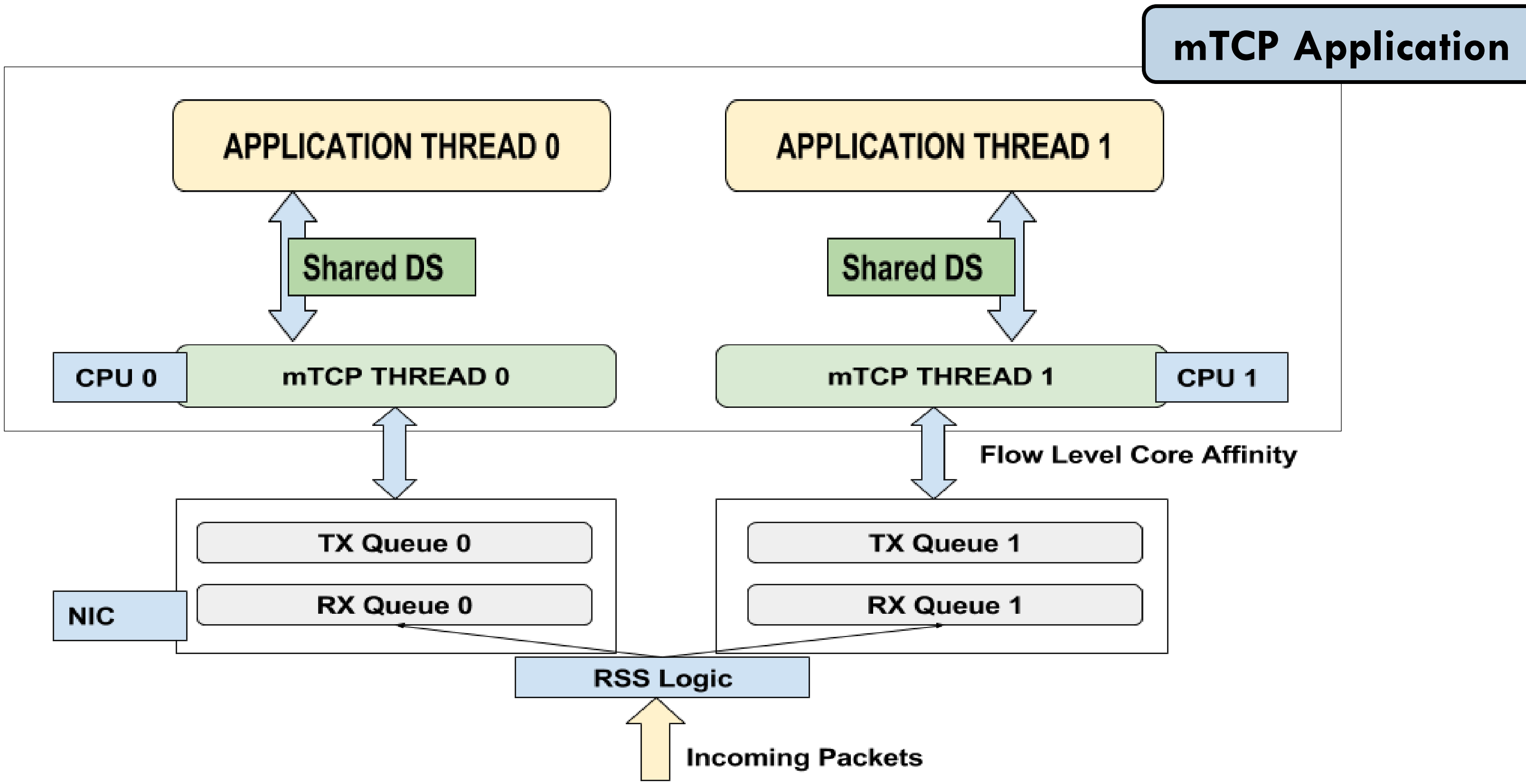


Figure: mTCP Stack

mTCP Based Application Design



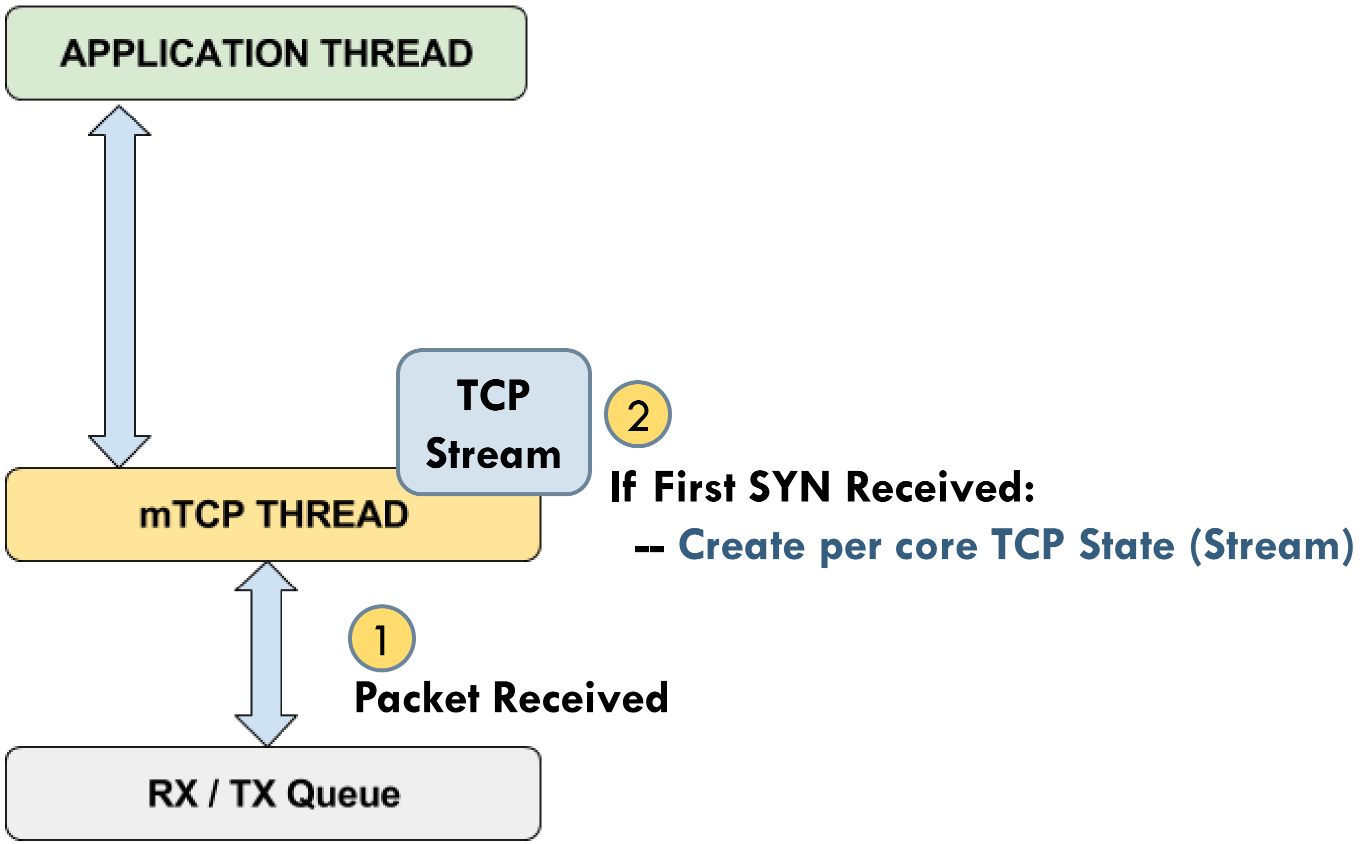
mTCP Application Design

Problem Statement

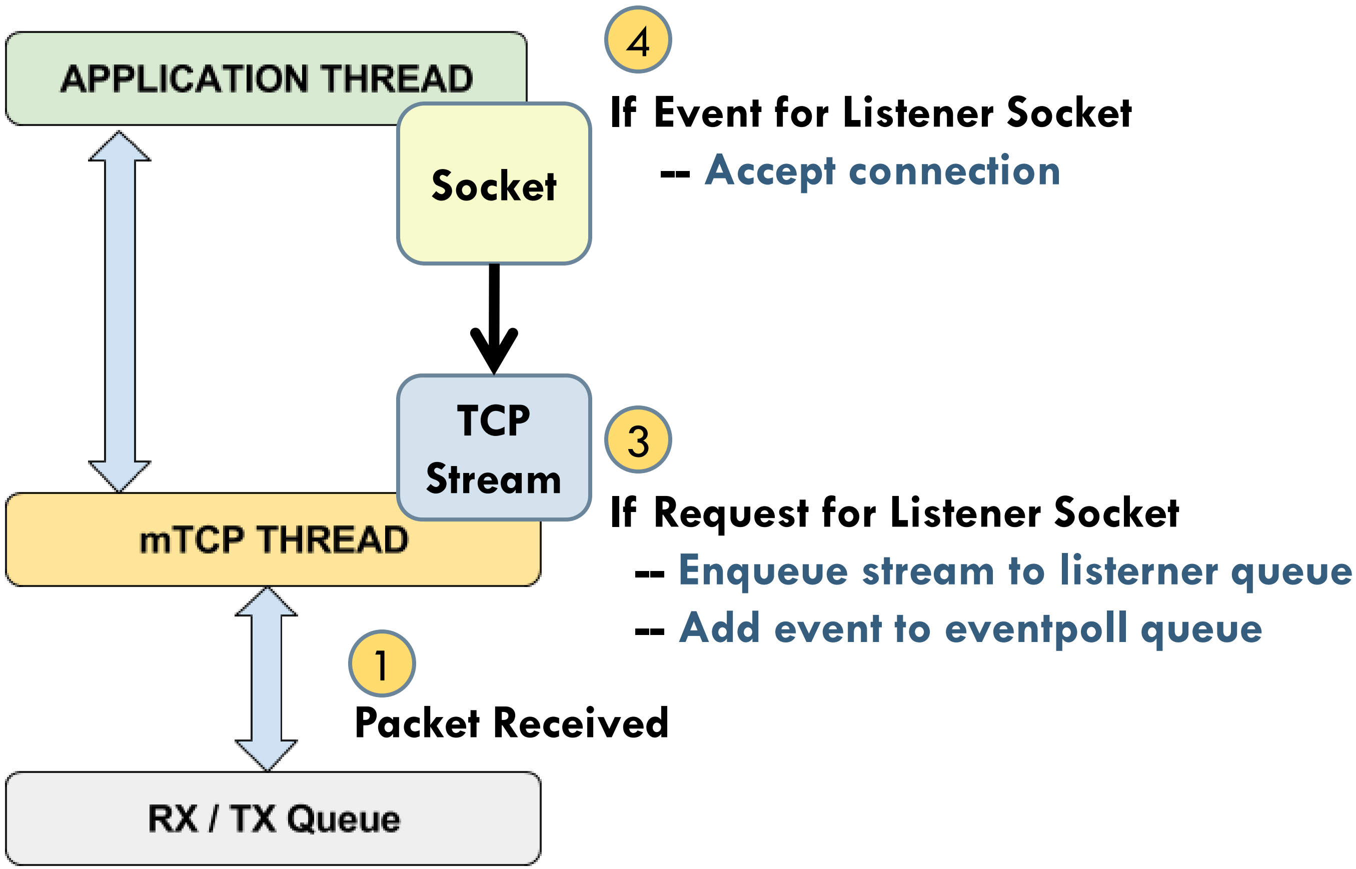
- Compare mTCP based single threaded and multithreaded web server designs

- Answers to find:
  - Can a master-worker web server design work with current mTCP architecture?
  - What is the performance overhead in having a shared mTCP architecture?

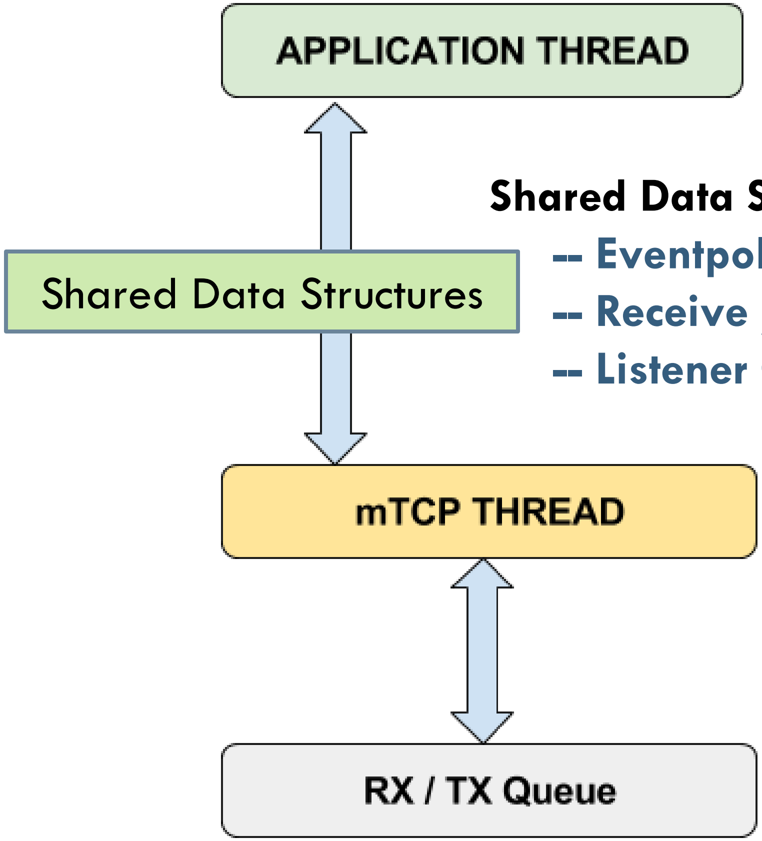
mTCP Single-Threaded Web Server



mTCP Single-Threaded Web Server



mTCP Single-Threaded Web Server



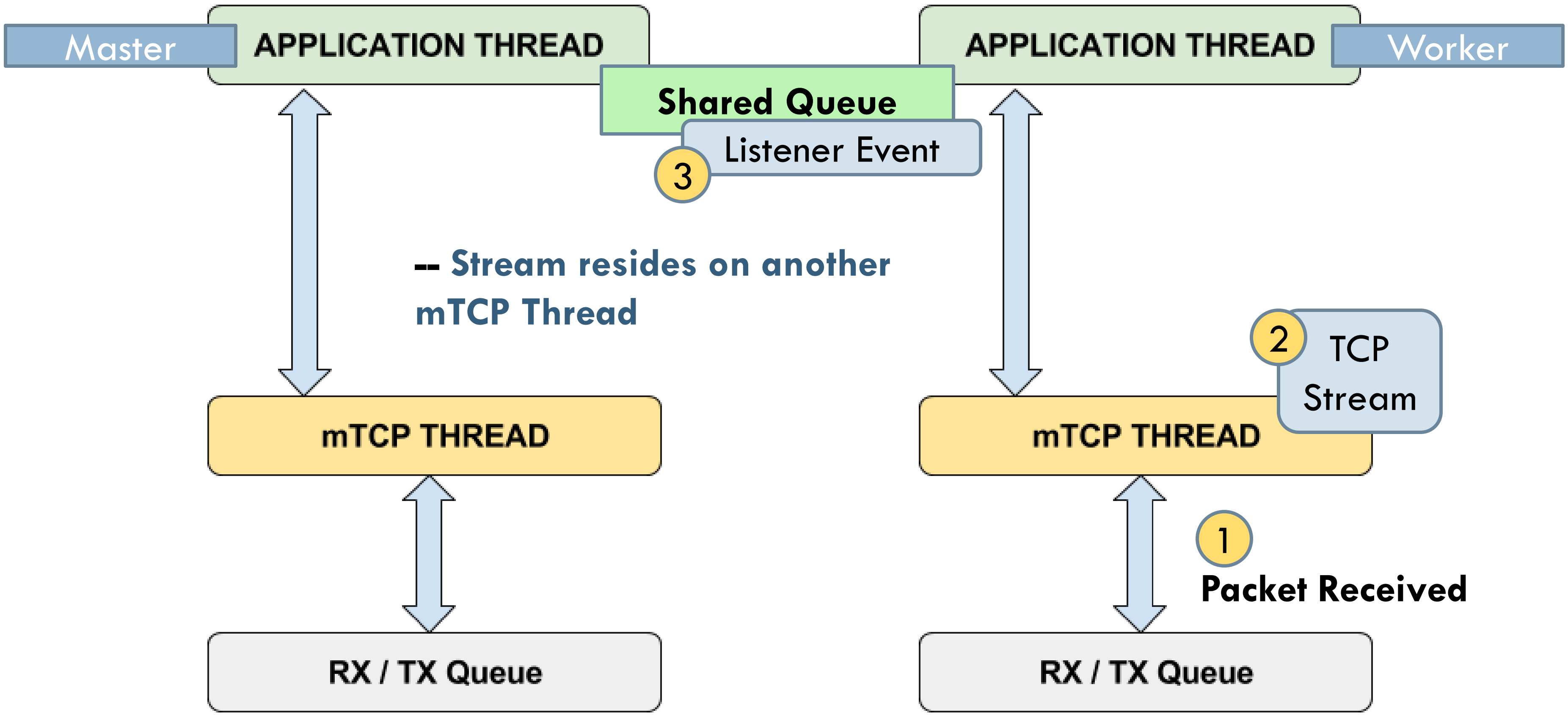
Shared Data Structures:

-- Eventpoll Queue

-- Receive / Send Buffer

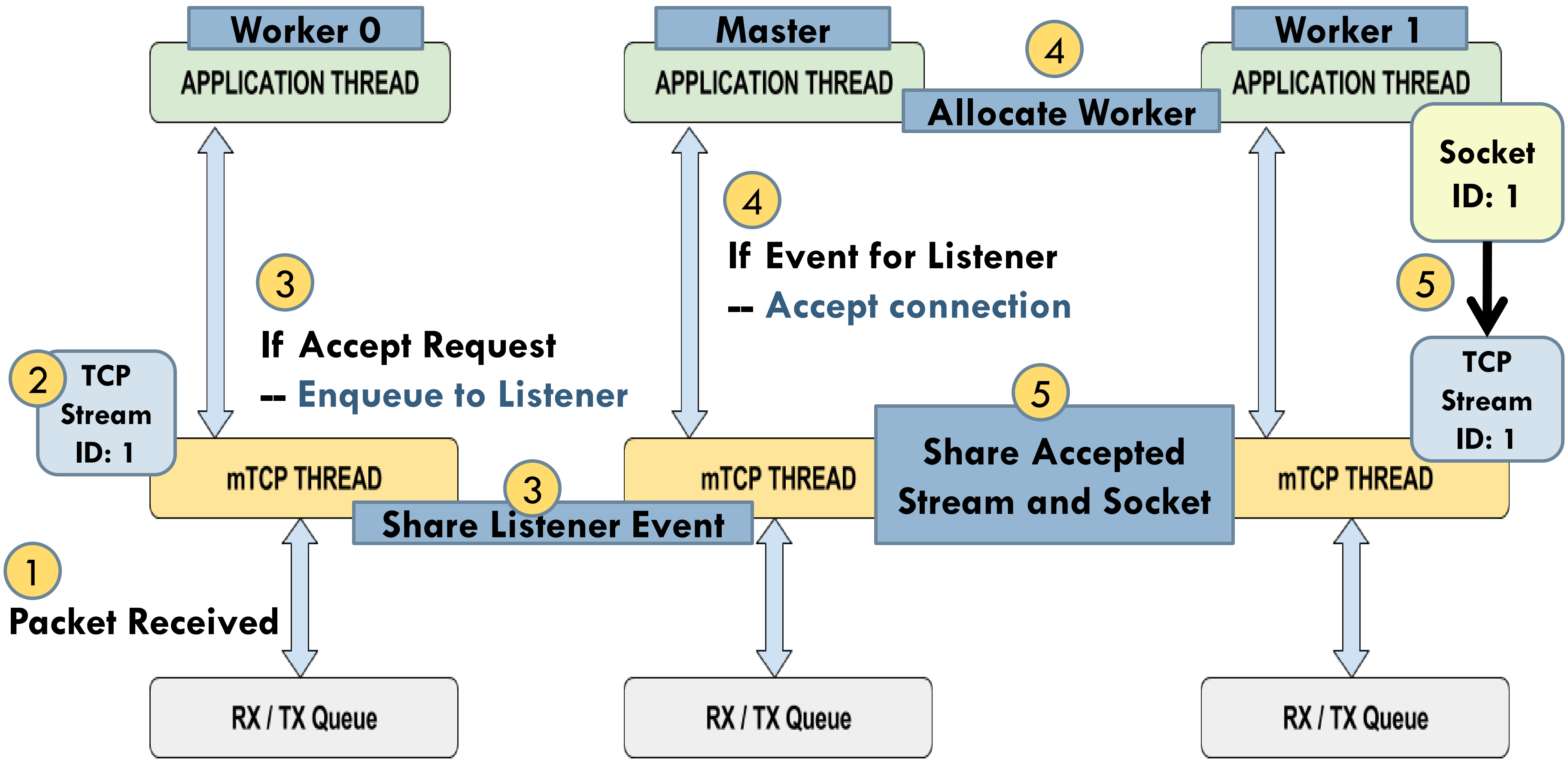
-- Listener Queue

mTCP Multithreaded Web Server

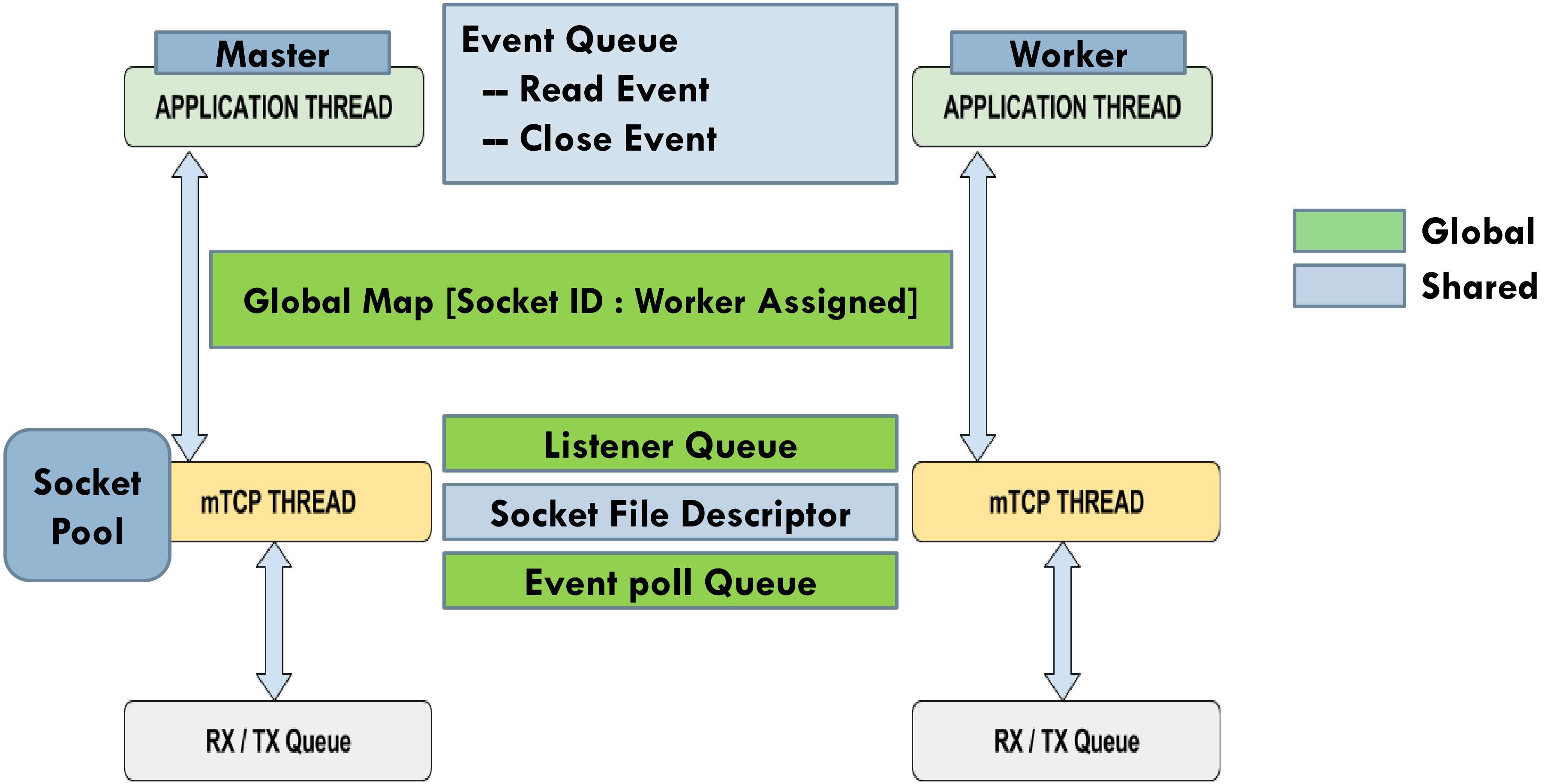


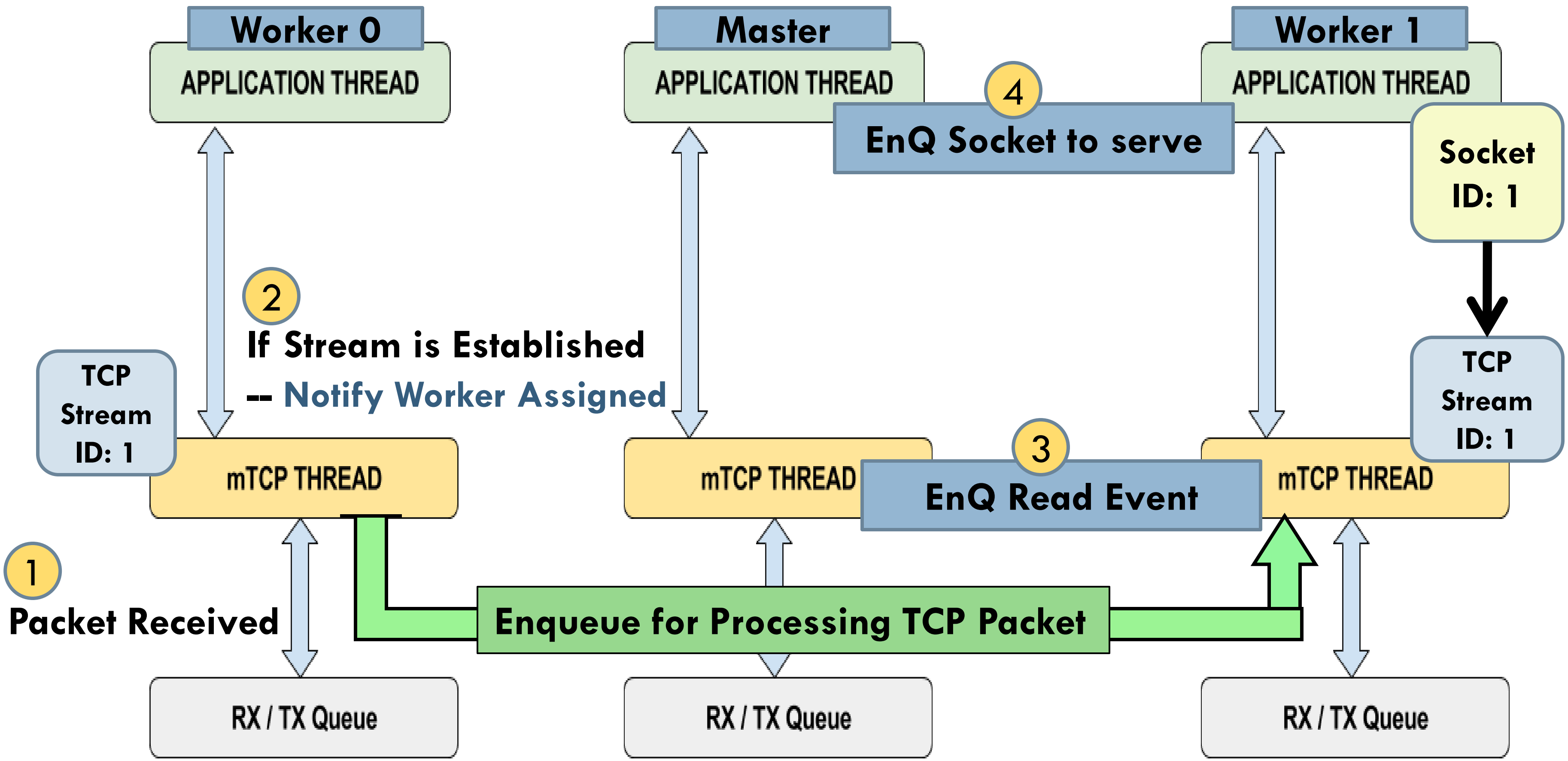
Shared mTCP Network Stack Design

- mTCP based Master-Worker designs:
  - Master controls all events [Read and Accept]
    - Master does the polling and allocates workers
  - Master controls only Listener Accept Events
    - Master only sets the initial connection
    - Worker works independently after that

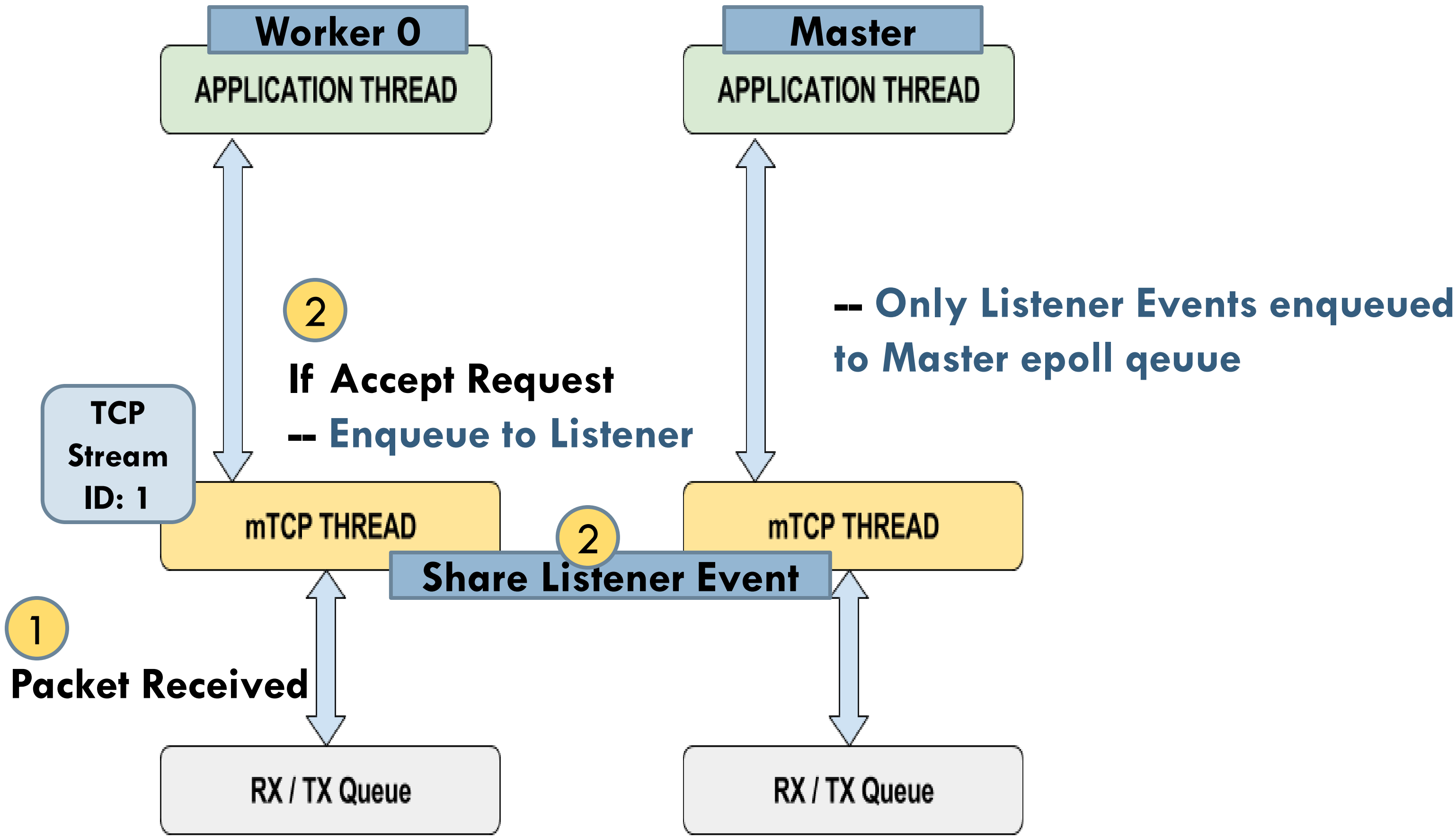


Worker 0
APPLICATION THREAD
Master
APPLICATION THREAD
4
Allocate Worker
Worker 1
APPLICATION THREAD
Socket
ID: 1
4
If Event for Listener
-- Accept connection
3
If Accept Request
-- Enqueue to Listener
5
2
TCP
Stream
ID: 1
5
Share Accepted
Stream and Socket
TCP
Stream
ID: 1
mTCP THREAD
mTCP THREAD
mTCP THREAD
3
Share Listener Event
1
Packet Received
RX / TX Queue
RX / TX Queue
RX / TX Queue

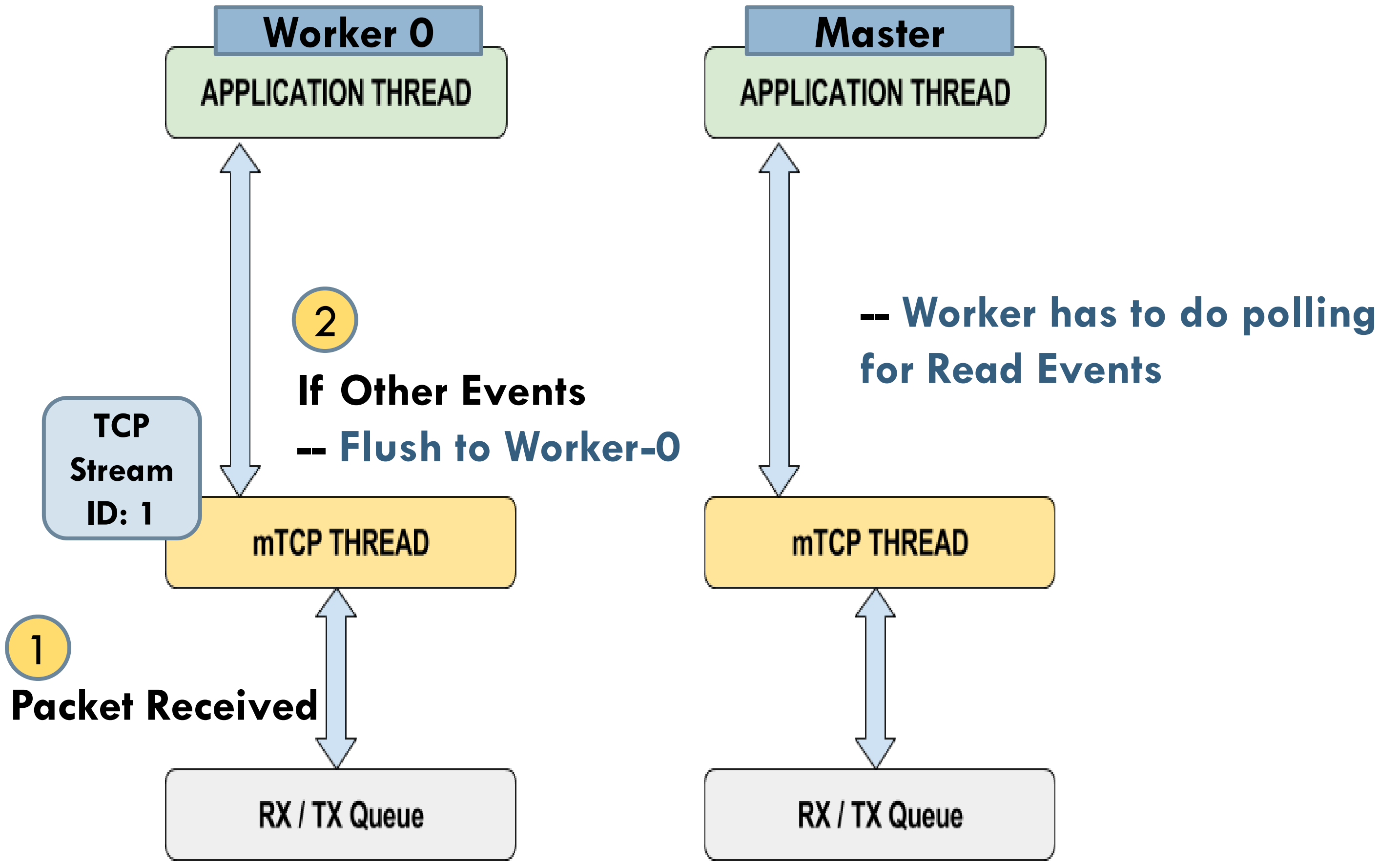

Master
APPLICATION THREAD
Event Queue
-- Read Event
-- Close Event
Worker
APPLICATION THREAD
Global
Shared
Global Map [Socket ID : Worker Assigned]
Socket Pool
mTCP THREAD
Listener Queue
Socket File Descriptor
Event poll Queue
mTCP THREAD
RX / TX Queue
RX / TX Queue


Worker 0
APPLICATION THREAD
Master
APPLICATION THREAD
Worker 1
APPLICATION THREAD
4
EnQ Socket to serve
Socket ID: 1
2
If Stream is Established
-- Notify Worker Assigned
TCP Stream ID: 1
mTCP THREAD
mTCP THREAD
3
EnQ Read Event
mTCP THREAD
TCP Stream ID: 1
1
Packet Received
Enqueue for Processing TCP Packet
RX / TX Queue
RX / TX Queue
RX / TX Queue

Another Master-Worker Design



Another Master-Worker Design



Comparing mTCP based Web Servers

- Single Threaded Design
  - Independent server threads working in parallel
  - Every thread has to do polling
- Multi Threaded Design [Master has Full Control]
  - Only master thread has to do the epoll
  - Workers depend on master thread's policies
- Multi Threaded Design [Master Initiates Worker]
  - Worker too has to do epoll for sockets allocated to it
  - Worker has full control of connections allocated to it

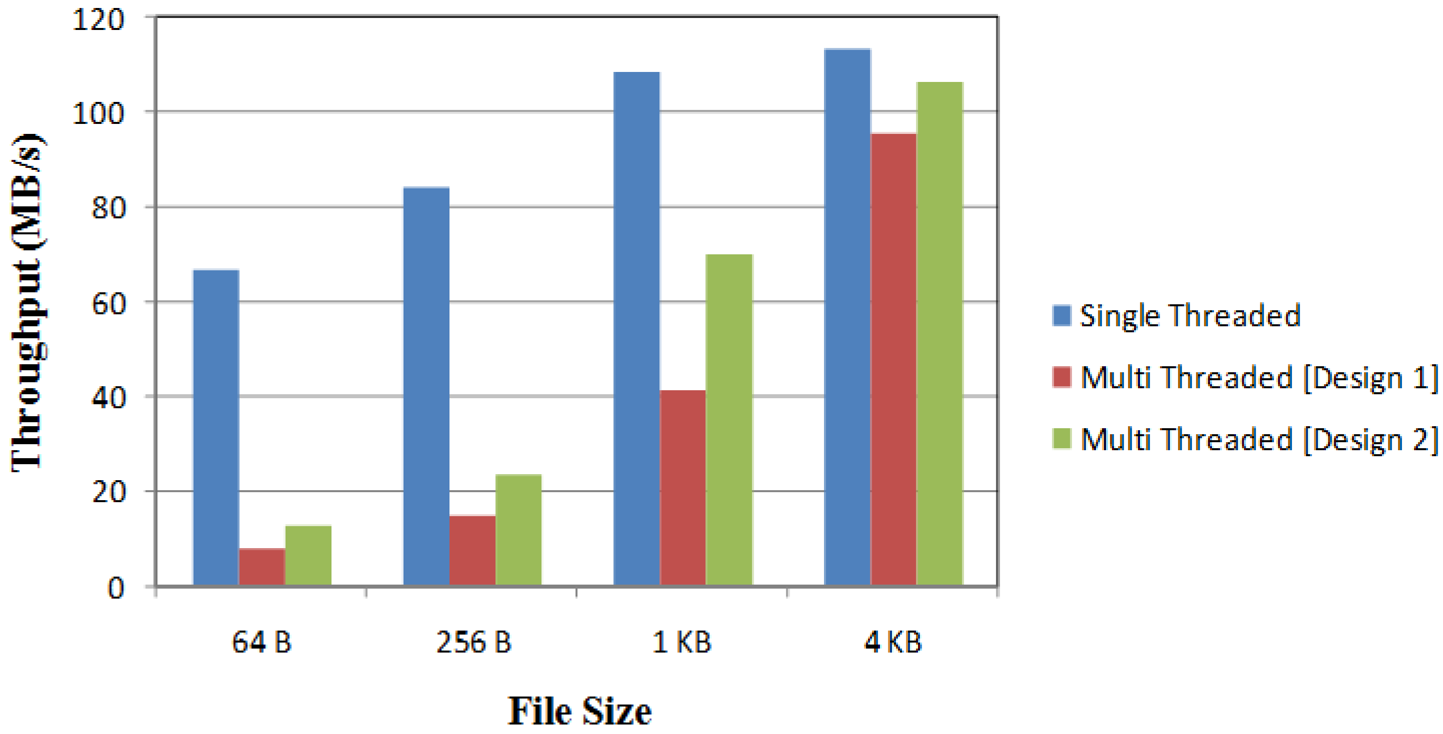
Experimental Setup

- System: 16 Core Xeon Blade Server, 32GB Ram, 1GB NIC
- Experiments to be Done:
  - Server Throughput based on varying file size requests from client
  - Server Throughput based on varying no of cores assigned to server

Performance on Varying File Sizes

- Client: 2000 open sockets on client side which sends requests of varying sizes for 20 seconds
- Server: The server applications have 7 cores assigned to it

Performance on Varying File Sizes

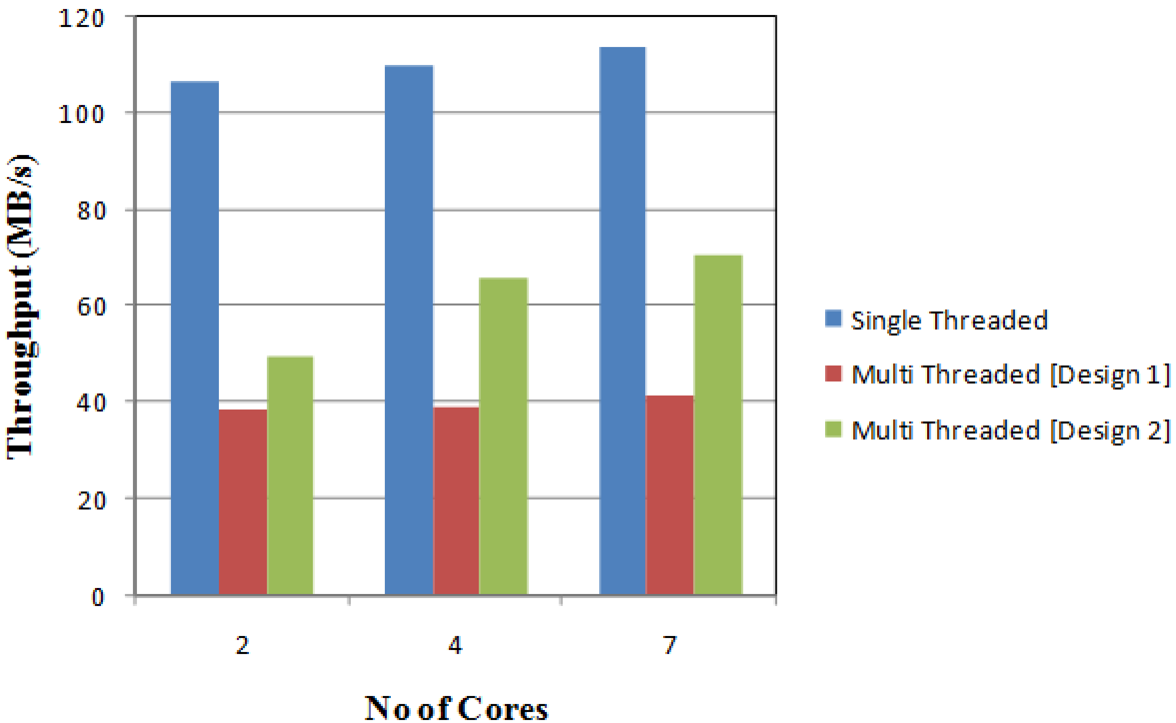


Performance based on Varying File Sizes

Performance on Varying No. of Cores

- Client: 2000 open sockets on client side which sends requests of 1KB file for 20 seconds
- Server: The server application have varying cores assigned to it based on the experiment

Performance on Varying No. of Cores



Performance based on varying Number of Cores

Read Copy Update

- Never Block Readers
- Writer synchronization left to the developers
- Use of multiple versions of the same data structure

Ref: Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems, 15(6):491–504, June 2004.

Read Log Update

- Provides a per thread log for concurrent writers
- Easy to use writer synchronization
- Provide multiple updates using a single operation

Ref: Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems, 15(6):491–504, June 2004.

Comparing RCU and RLU

- Is RLU better than RCU?
  - RCU provides better results for Hash-Lists
- Does Node-Size matter in the comparisons?
  - The relative performance remains the same

Ref: Paul E. McKenney. Read-mostly research in 2015: https://lwn.net/articles/667593. 2015.

Some Other Questions Answered

- Do writers need help in RCU protect data structures in the Linux Kernel?
  - Writer Contention is quite low in the given time frame

- Can Array Based RCU Design help Readers?
  - For small size objects it performs quite well but the performance drops significantly after the given cache size.

References

- [1] Timothy L. Harris. A pragmatic implementation of non-blocking linked-lists. Proceedings of the 15th International Conference on Distributed Computing, pages 300–314, October 2001.
- [2] Alexander Matveev, Nir Shavit, Pascal Felber, and Patrick Marlier. Read-log-update: a lightweight synchronization mechanism for concurrent programming. 2015.
- [3] Paul E. McKenney. Read-copy update (RCU) usage in Linux kernel. Available: http://www.rdrop.com/users/paulmck/RCU/linuxusage/rculocktab.html, October 2006.
- [4] Paul E. McKenney. What is rcu: https://lwn.net/articles/262464/. 2007.
- [5] Paul E. McKenney. Rcu linux usage log: http://www.rdrop.com/ paulmck/rcu/linuxusage/linux- 4.3.rcua. 2015.
- [6] Paul E. McKenney. Read-mostly research in 2015: https://lwn.net/articles/667593. 2015.
- [7] Paul E. McKenney. Some more details on read-log-update: https://lwn.net/articles/667720. 2015.
- [8] Paul E. McKenney and John D. Slingwine. Read-copy update: Using execution history to solve concurrency problems. In Parallel and Distributed Computing and Systems, pages 509–518, Las Vegas, NV, October 1998.
- [9] Maged M. Michael. Hazard pointers: Safe memory reclamation for lock-free objects. IEEE Transactions on Parallel and Distributed Systems, 15(6):491–504, June 2004.
- [10] John D. Valois. Lock-free linked lists using compare-and-swap. , Proceedings of the 14th annual ACM symposium on Principles of distributed computing, pages 214–222, August 1995.

Questions?