SDC

STORAGE DEVELOPER CONFERENCE

SNIA ■ SANTA CLARA, 2015

Integrity of In-memory Data Mirroring in Distributed Systems

Tejas Wanjari

EMC Data Domain

Problem Definition

- In-memory data is changing
- Disk checksums are for the older state
- Mirroring cannot rely on disk checksums
- Undetected corruptions are not acceptable
- Reliability is prime (e.g. Backup/Recovery Systems)

Sources of Corruption during Mirroring

- System failure
  - “Clean” shutdown and reboot
- Hardware failure
  - Redundant failover
- Disks failure
  - Disk/filesystem checksums
- Process corruption
  - Avoiding copying without checksums
- *Network corruption*
  - *Application/protocol checksums*

TCP Checksum Vulnerability

- TCP Checksum: 4 bytes & weak
- Prone to False Positives (FPs)
  - Wrong data, correct checksum
- Failure probability: 1 in 16 million to 10 billion packets for 1526 bytes *[Reference: [1] Stone et. al., When the CRC and TCP checksum disagree]*
- *Implies 1 undetected TCP corruption in 20GB to 1.2TB data, approximately*

Strong checksum in Application?

- Performance overhead
  - Application data-structures different from network data-structures (e.g. B-tree data to fit into MTU)
  - “Interconnect or network” is the vulnerability, *not the application*
- End up reinventing transport protocol in application (over TCP!)
  - Handling retransmissions, in-order delivery, gaps, etc.

Ideal Solution

- Zero-copying: avoid multiple **copies without checksums**
- H/w redundancy for **hardware** failures
- Clean shutdowns on **system** failures
- Filesystem/block/disk checksums for **disk** reliability
- *Bridging the integrity gap in* ***network/interconnect***
  - Protection in transport protocol

Why reinvent the wheel?

- RFC 2385: TCP MD5 Signature Option
- Implemented in Linux Kernel as TCP_MD5SIG socket option
- Linux implementation:
  - Efficient compute (uses kernel crypto-engine)
- Retransmission on checksum mismatch
  - Implies seamless error-recovery
- Reduces syscalls by calling /dev/crypto from within kernel. Thus, lesser **copy_to_user/copy_from_user** and smaller memory footprint.

SDC 15

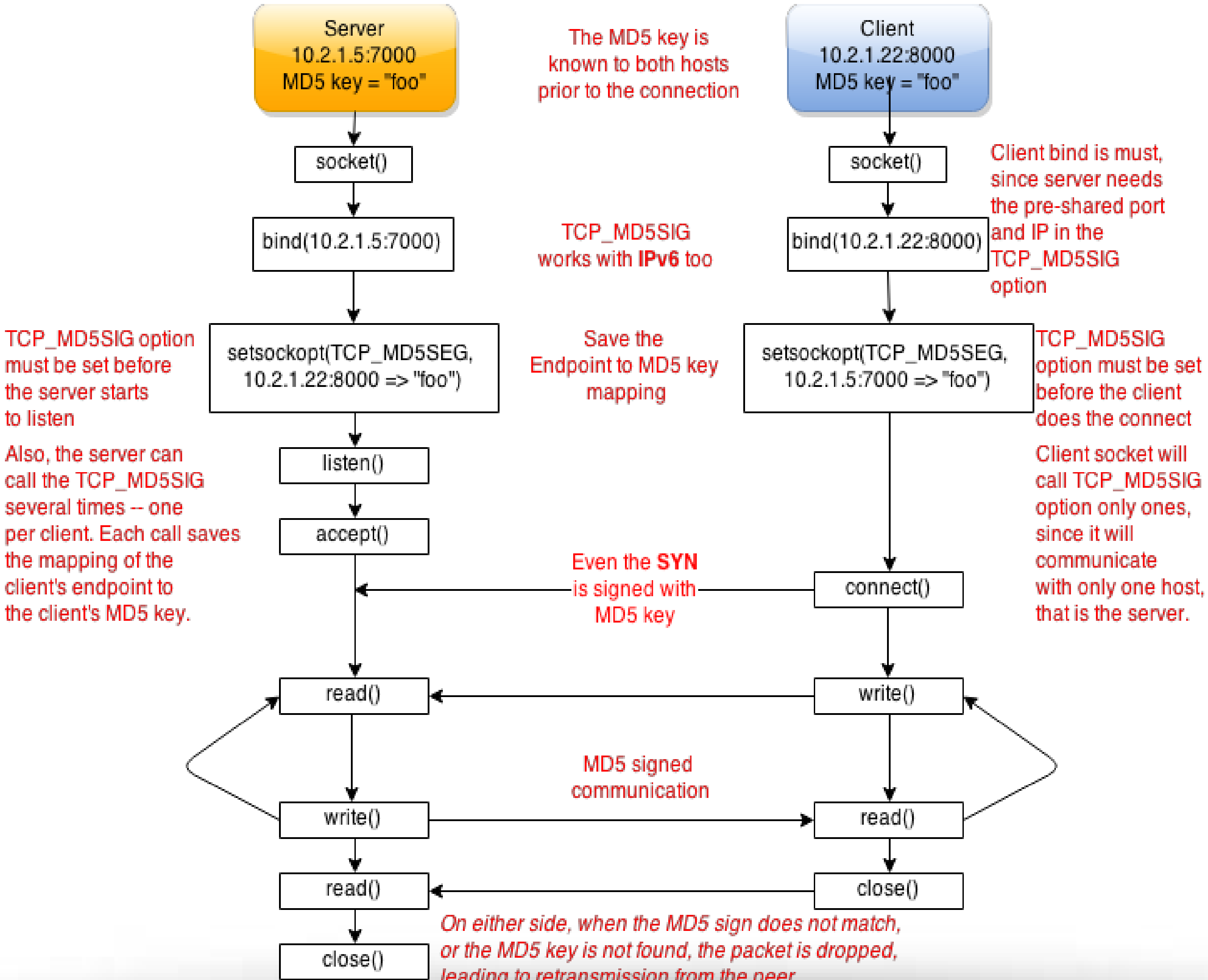
Working of TCP_MD5SIG socketopt

- Both client and server must know each others':
    - IP
    - Port
    - MD5 Key

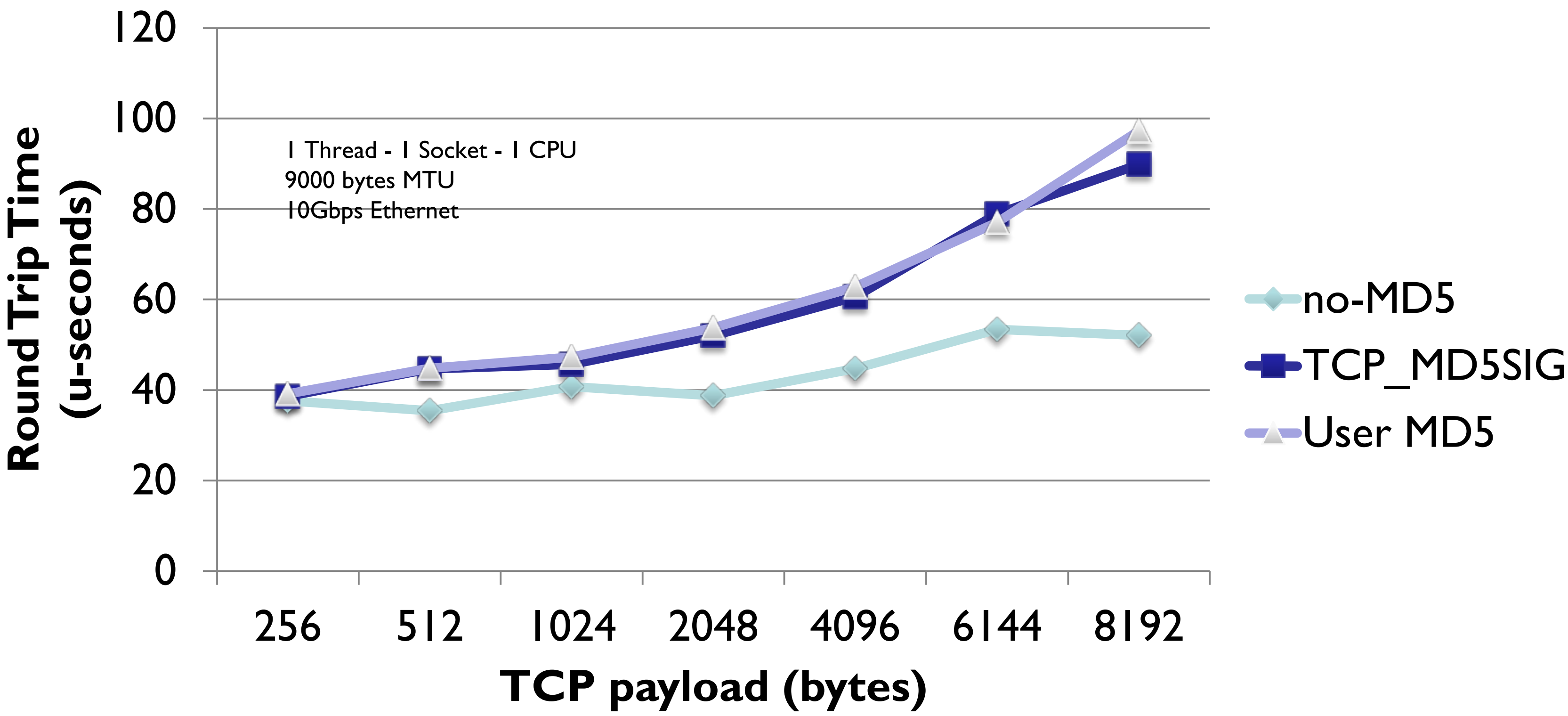
  before the connection is setup

- Client must bind() for the server to save the <IP,Port,MD5Key> mapping

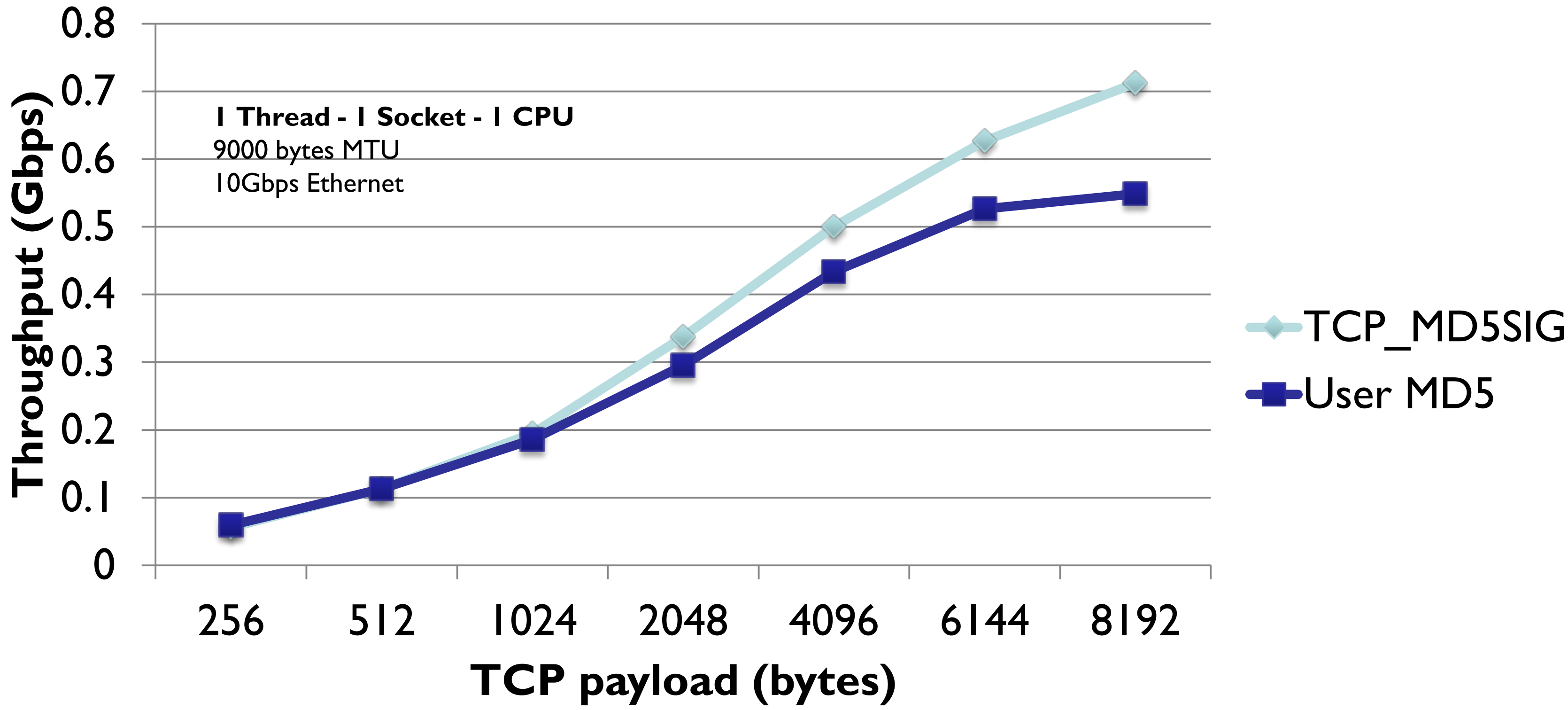
TCP_MD5SIG over Socket



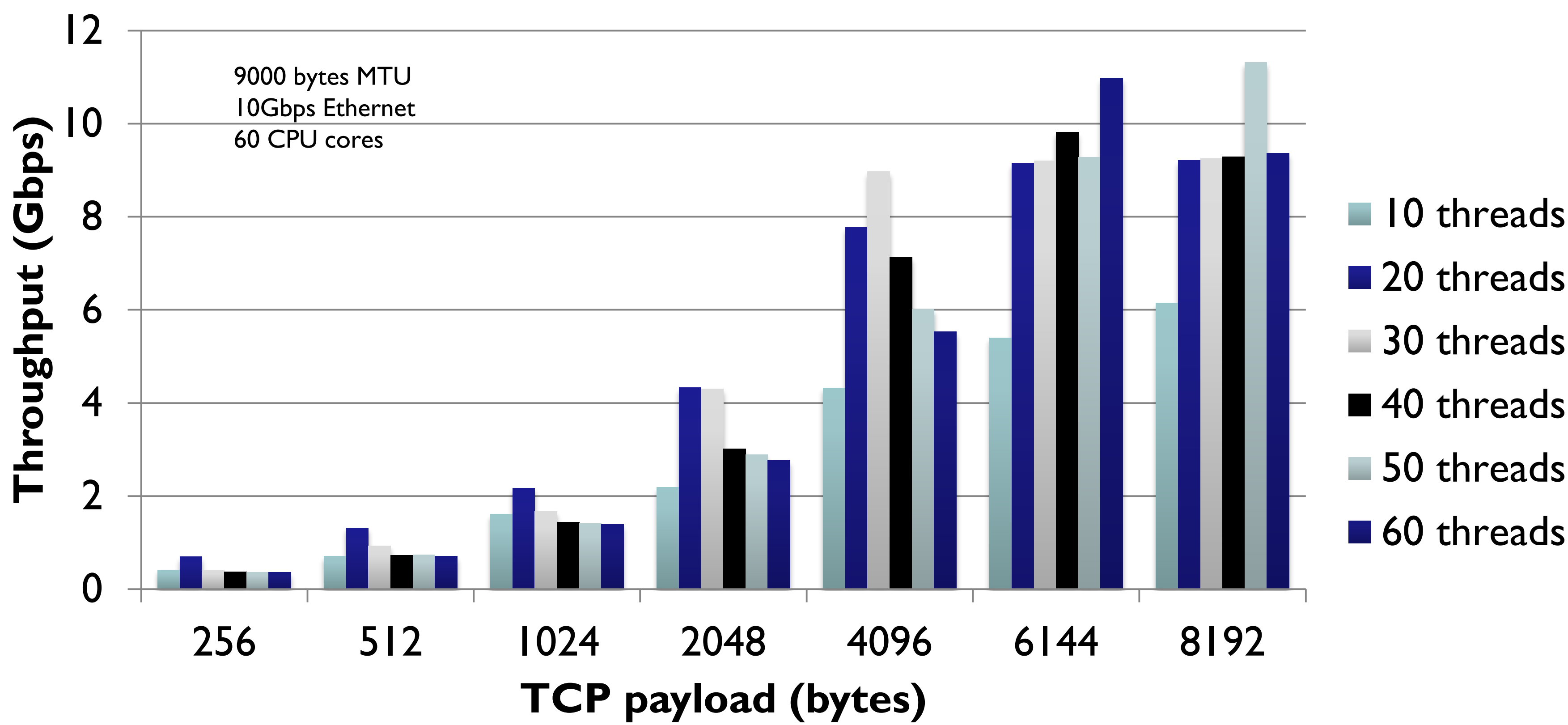
Evaluation: Latency



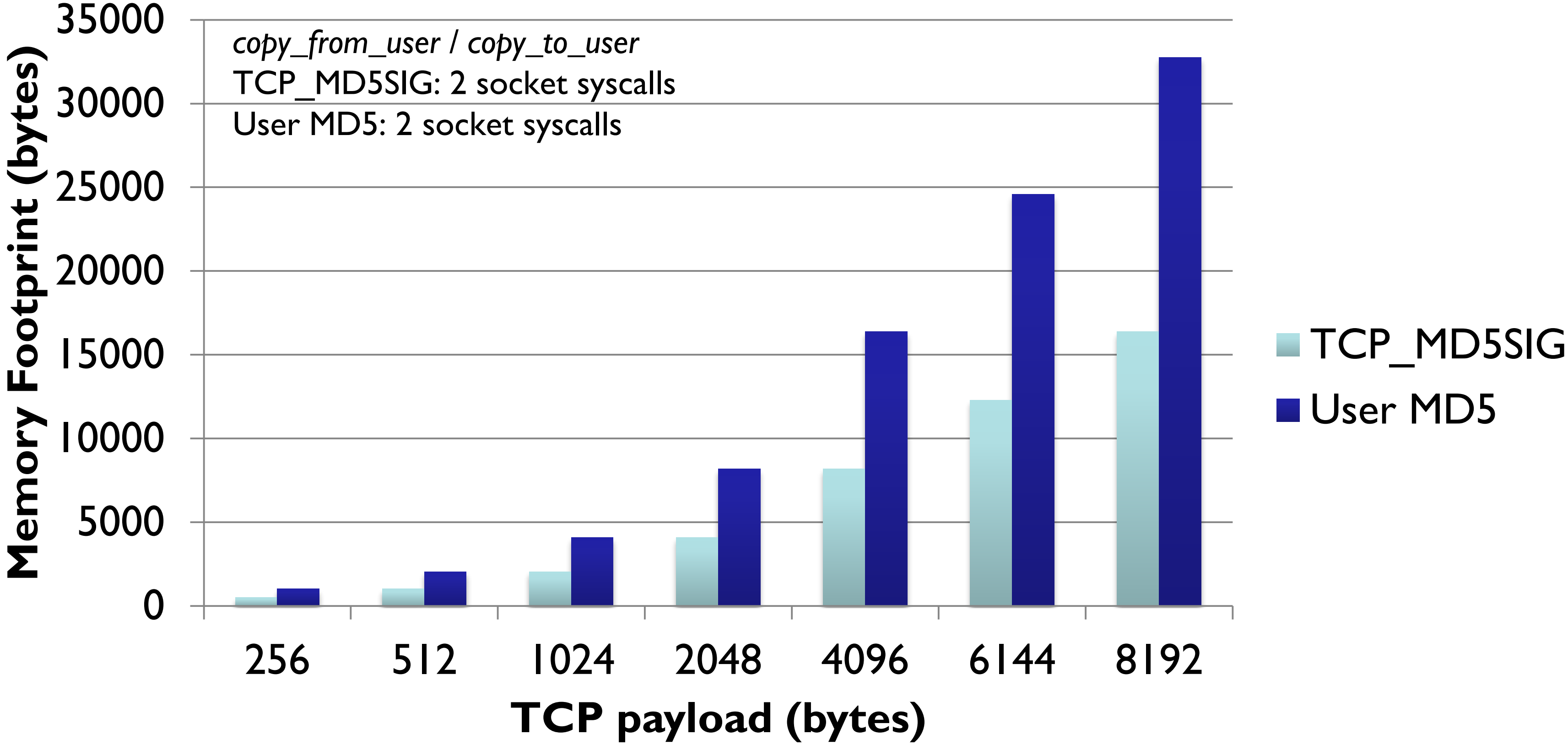
Evaluation: Throughput (Single-threaded)



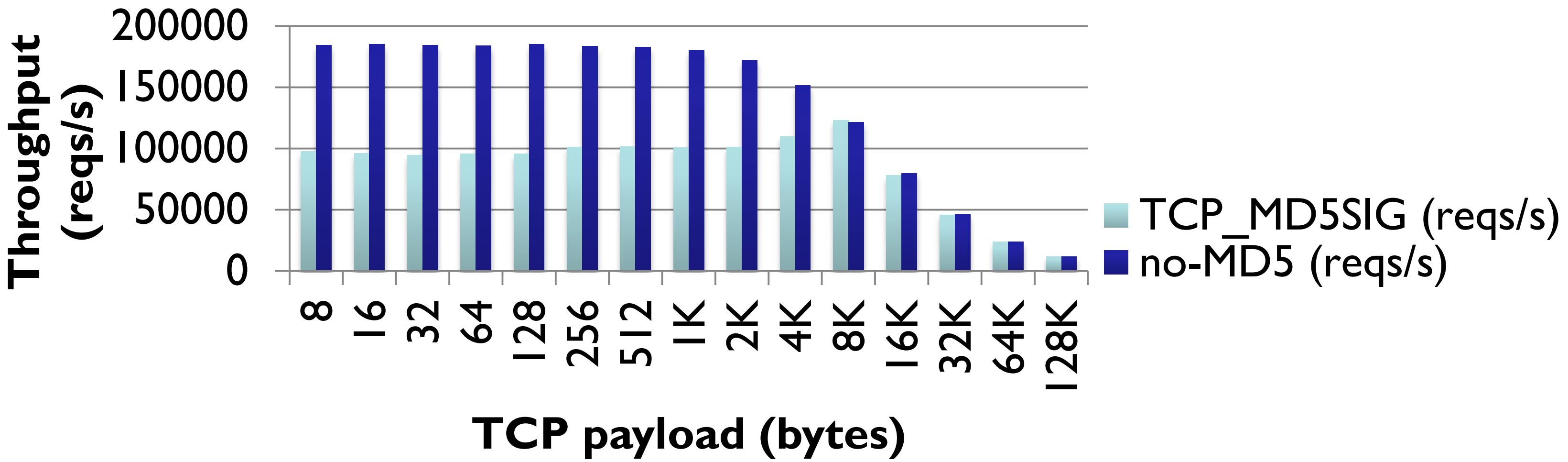
Evaluation: Throughput - TCP_MD5SIG (Multi-threaded)

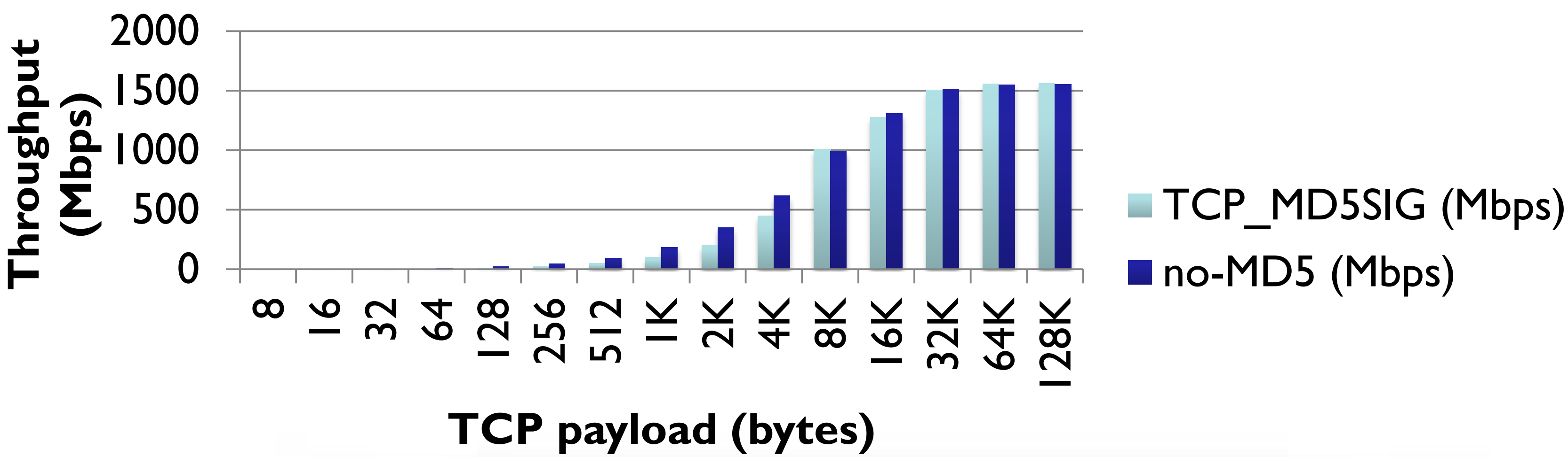


Evaluation: Memory Footprint
(accessing */dev/crypto* from userspace)



Use-Case: NVM Mirroring





Conclusion

- Not a generic solution
  - First try other fits:
    - TCP checksum not good enough for the application?
    - Disk/filesystem checksum
    - Disk/flash mirroring
- But very effective for typical usecases
  - For line-speed mirroring of in-memory data:
    - Better throughput, memory footprint and same latency
  - Error detection and recovery seamless to application
- Future prospects: Persistent Memory

References

[1] Jonathan Stone and Craig Partridge. 2000. When the CRC and TCP checksum disagree. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (SIGCOMM '00). ACM, New York, NY, USA, 309-319. DOI=10.1145/347059.347561 http://doi.acm.org/10.1145/347059.347561

[2] TCP_MD5SIG: An Undocumented Socket Option in Linux. http://criticalindirection.com/2015/05/12/tcp_md5sig/

[3] Iperf3 with TCP_MD5SIG (Patch being submitted): https://github.com/tejaswanjari/iperf

[4] Linux Kernel Source-tree www.kernel.org

Thank-you!

- Questions?