SCONE: Secure Linux Containers with Intel SGX

Sergei Arnautov1, Bohdan Trach1, Franz Gregor1, Thomas Knauth1, Andre Martin1, Christian Priebe2, Joshua Lind2, Divya Muthukumaran2, Dan O'Keeffe2, Mark L Stillwell2, David Goltzsche3, David Eyers4, R¨udiger Kapitza3, Peter Pietzuch2, and Christof Fetzer1

1Fakult¨at Informatik, TU Dresden, christof.fetzer@tu-dresden.de
2Dept. of Computing, Imperial College London, prp@imperial.ac.uk
3Informatik, TU Braunschweig, rrkapitz@ibr.cs.tu-bs.de
4Dept. of Computer Science, University of Otago, dme@cs.otago.ac.nz

Saeid Mofrad

1-INTRODUCTION:

Linux Containers:

Containers use OS-level virtualization and they are popular for packaging, deploying and managing services such as key/value stores and web servers.
Unlike VMs, they do not require hypervisors or a dedicated OS kernel. Instead, they use kernel features to isolate processes, and thus do not need to trap system calls or emulate hardware devices.

- Container process can run as normal system process.

They are lightweight (they use the host OS for I/O operations, resource management, etc.) faster I/O throughput and latency than VMs
Isolation is weak since it is using software kernel mechanisms, make it easier for attackers to compromise the confidentiality and integrity of application data within containers.

- Docker and LXC are using for the packaging of the containers.
- Docker Swarm or Kubernetes are using for their deployment.

What is SCONE ?

SCONE is a Secure Container Environment for Docker that uses SGX to run Linux applications in secure containers.

Goal of SCONE:

1. Run unmodified Linux applications
2. In containers
3. In an untrusted cloud
4. Securely with acceptable performance

SCONE Properties:

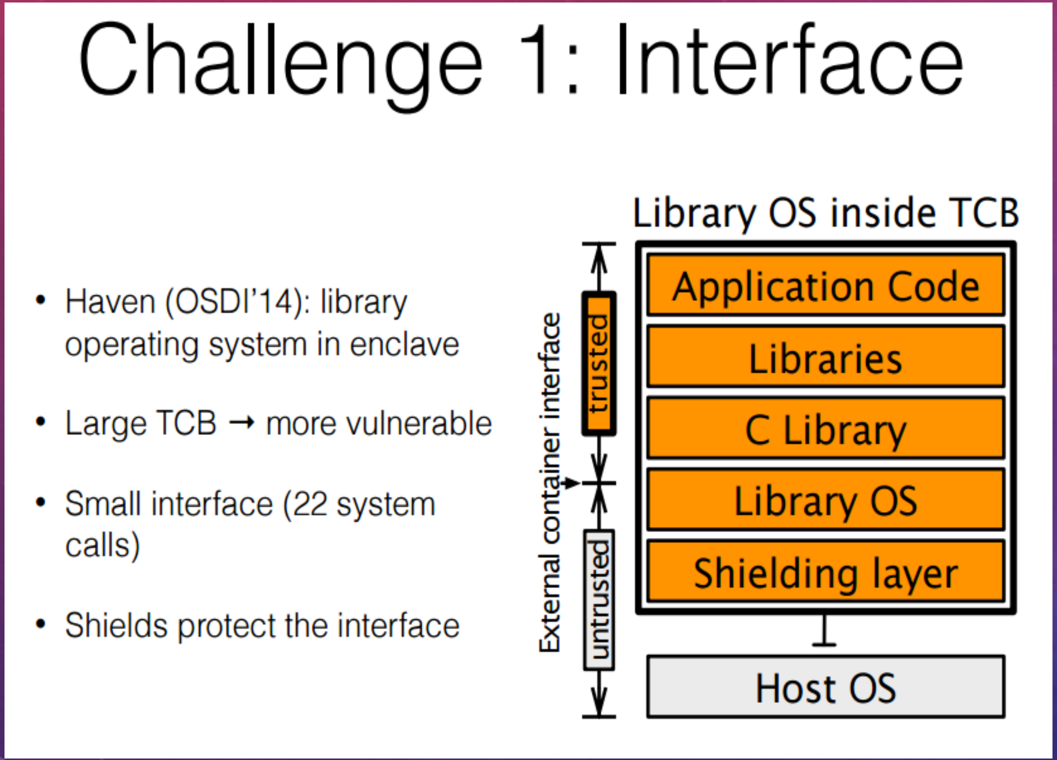
1. Secure containers have a small TCB.
2. Secure containers have a low overhead.
3. Secure containers are transparent to Docker.

Design trade-offs:

what system support should be placed inside an enclave to enable the secure execution of Linux processes in a container?
Challenges : Security decision about **the size of the TCB** and the exposed interface to the outside world and performance impact because of the SGX limitation).
TCB SIZE: Bigger TCB Larger Attack surface

External container interface: To execute unmodified processes inside secure containers, the container must support a C standard library (libc) interface. Since any libc implementation must use system calls, which cannot be executed inside of an enclave, a secure container must also expose an external interface to the host OS. As the host OS is untrusted, the external interface becomes an attack vector.

To justify the design of SCONE, They explored alternate design choices.

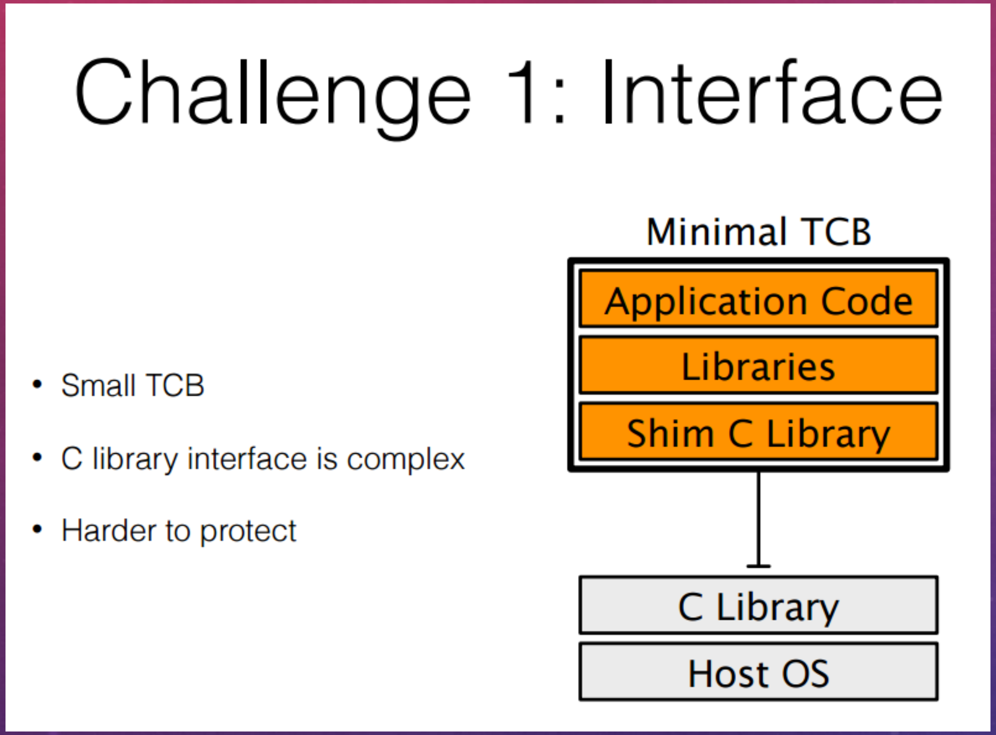


Challenge 1: Interface

- Haven (OSDI'14): library operating system in enclave
- Large TCB → more vulnerable
- Small interface (22 system calls)
- Shields protect the interface

Figure 1a shows a prior design point, as demonstrated by Haven, which minimizes the external interface by placing an entire Windows library OS inside the enclave.
A benefit of this approach is that it exposes only a small external interface with 22 calls because a large portion of a process' system support can be provided by the library OS. The library OS, however, increases the TCB size inside of the enclave.

To justify the design of SCONE, They explore alternate design choices.(cont.)


Challenge 1: Interface

- Small TCB
- C library interface is complex
- Harder to protect

Figure 1b shows the opposite, extreme design point:

the external interface is used to perform all libc library calls made by the application. This raises the challenge of protecting the confidentiality and integrity of application data whilst exposing a wide interface. For example, I/O calls such as read and write could be used to compromise data within the enclave, and code inside the secure container cannot trust returned data. A benefit of this approach is that it has minimal TCB inside the enclave—only a small shim C library needs to relay libc calls to the host libc library outside of the enclave.

To justify the design of SCONE, They explored alternate design choices.(Cont.)

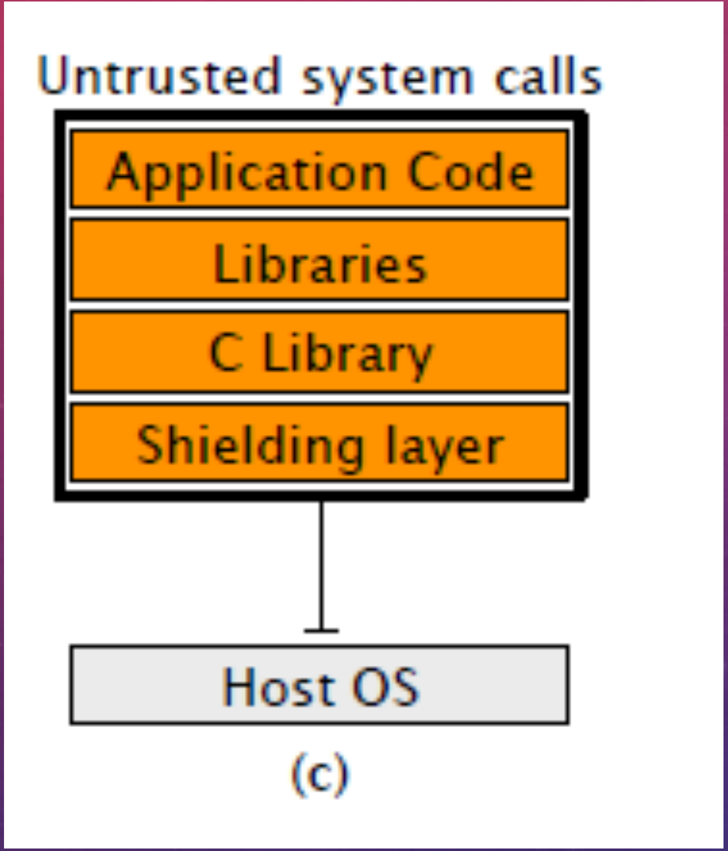
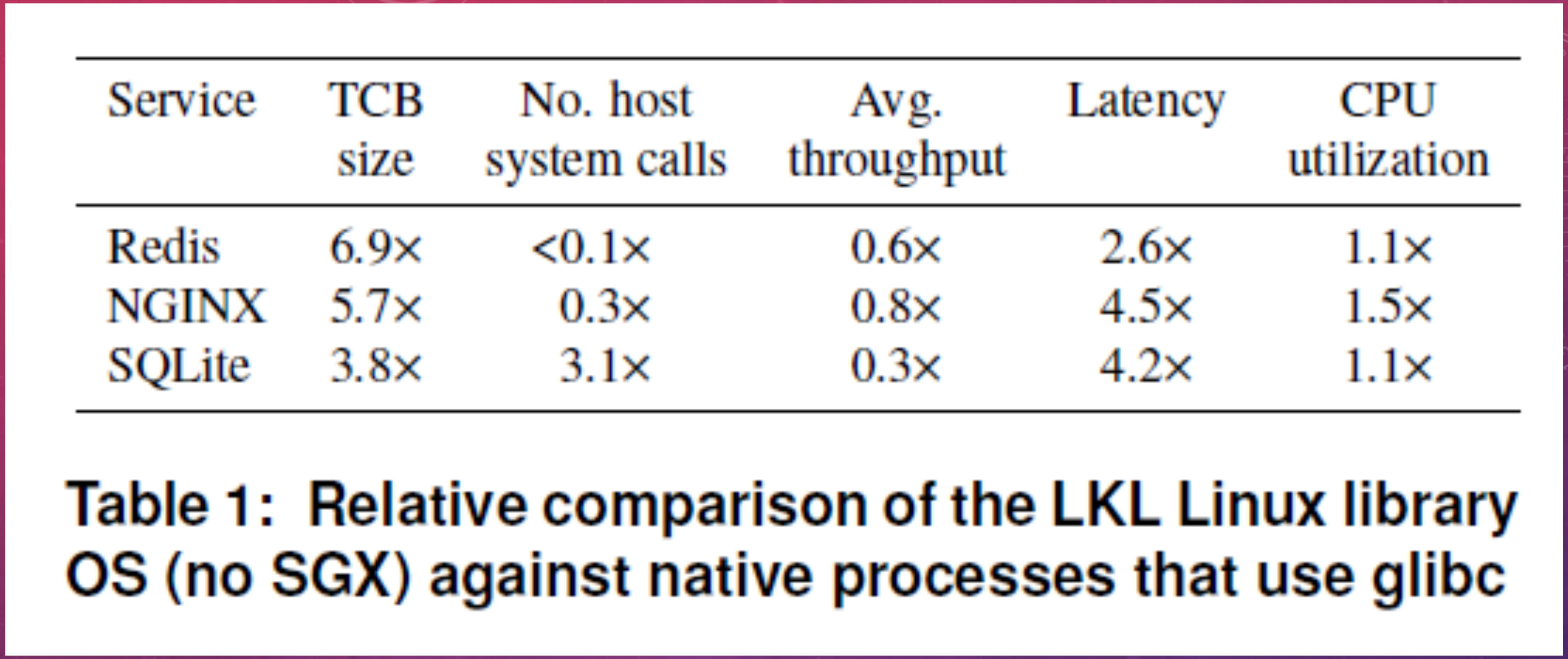


Figure 1c shows a middle ground by defining the external interface at the level of system calls executed by the libc implementation.

- shield libraries can be used to protect a security-sensitive set of system calls: file descriptor based I/O calls, such as read, write, send, and recv, are shielded by transparently encrypting and decrypting the user data.



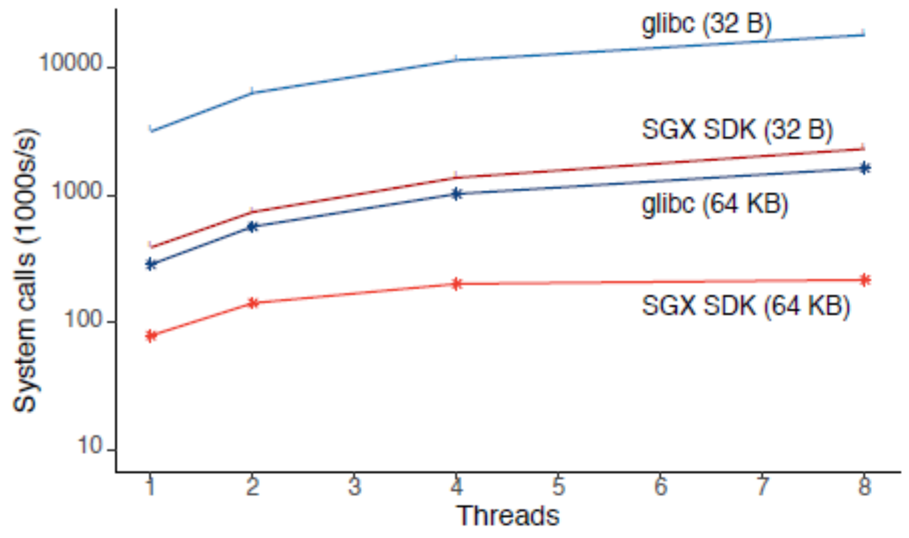
Service	TCB size	No. host system calls	Avg. throughput	Latency	CPU utilization
Redis	6.9×	<0.1×	0.6×	2.6×	1.1×
NGINX	5.7×	0.3×	0.8×	4.5×	1.5×
SQLite	3.8×	3.1×	0.3×	4.2×	1.1×

Table 1: Relative comparison of the LKL Linux library OS (no SGX) against native processes that use glibc

Table 1 Shows the performance and resource metrics for each service using the Linux library OS compared to a native glibc deployment. On average, the library OS increases the TCB size by 5x, the service latency by 4x and halves the service throughput.

Observation: System call overhead and Memory Access Overhead.

A micro-benchmark on an Intel Xeon CPU E3-1230 v5 at 3.4 GHz measuring the maximum rate at which pwrite system calls can be executed with and without an enclave.



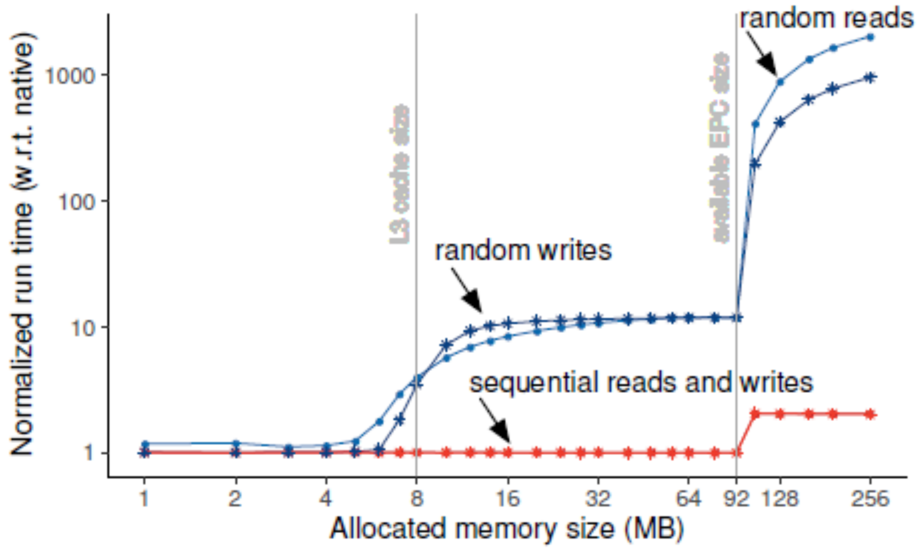
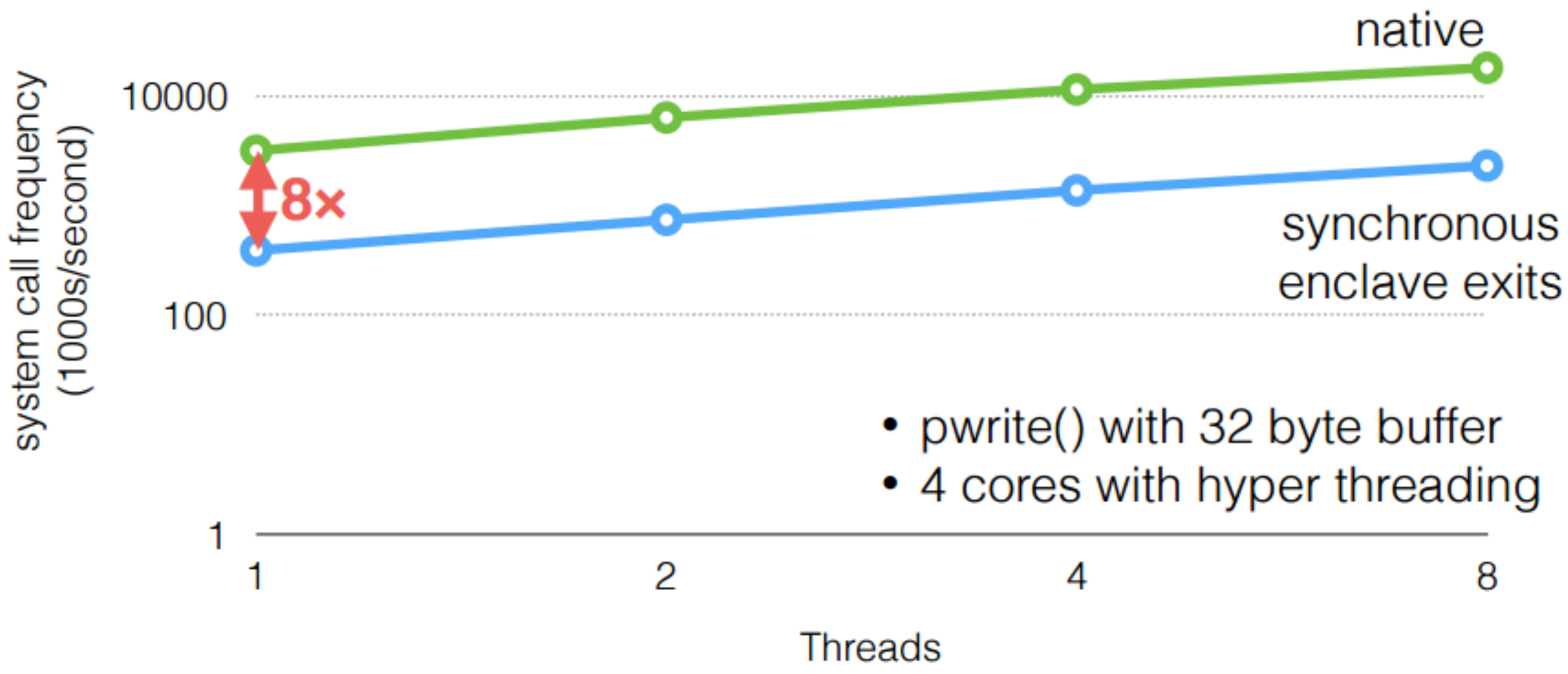


Figure 2: Number of executed `pwrite` system calls with an increasing number of threads

Figure 3: Normalized overhead of memory accesses with SGX enclaves

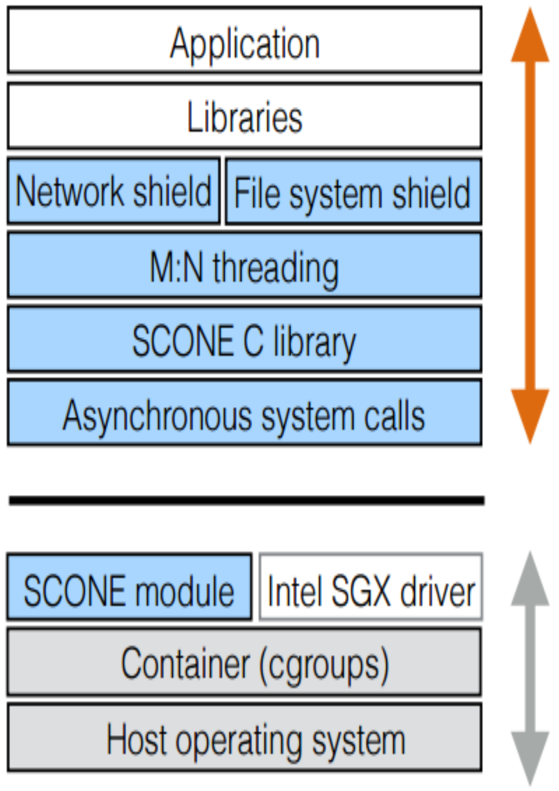
Challenge 2: Performance



- pwrite() with 32 byte buffer
- 4 cores with hyper threading

SCONE Architecture

- Enhanced C library → small TCB (Challenge 1)
- Asynchronous system calls and user space threading **reduce** number of **enclave exits** (Challenge 2)
- Network and file system shields **actively** protect user data



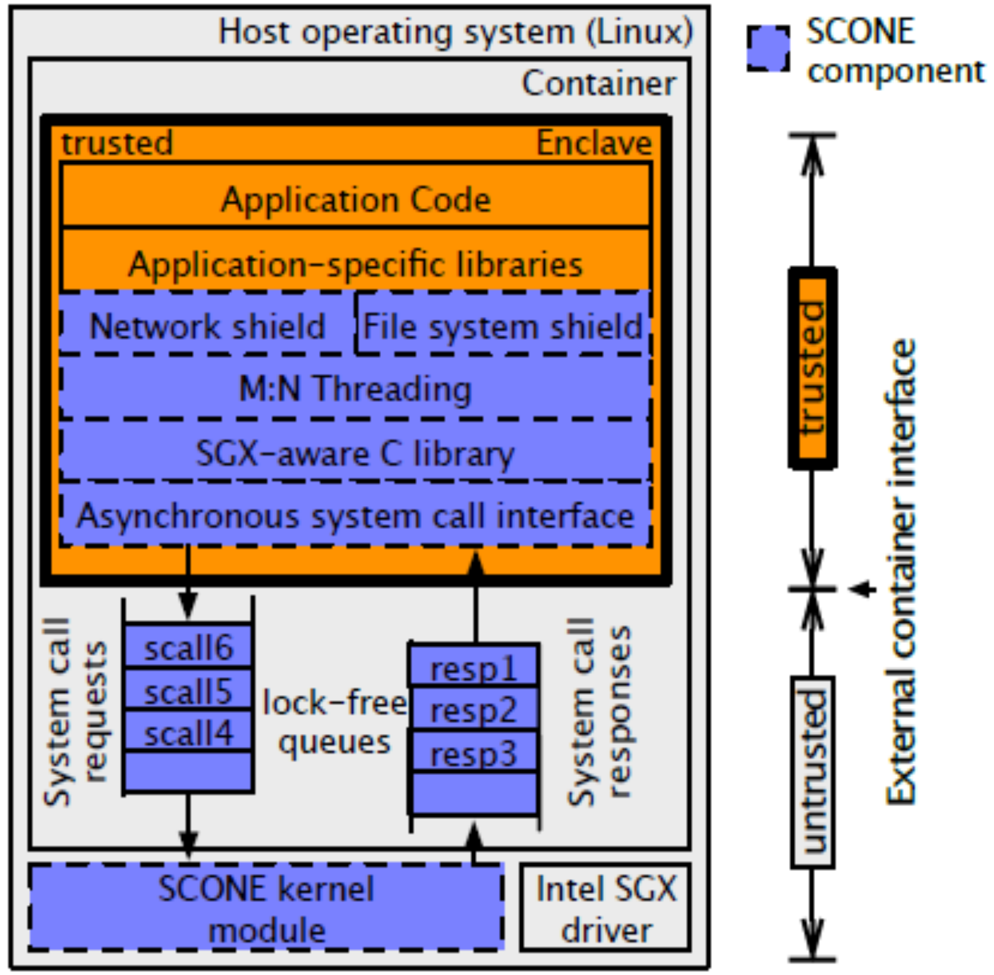


Figure 4: SCONE architecture

3.2 External interface shielding:

SCONE supports a set of shields. Shields focus on:
(1) Preventing low-level attacks, such as the OS kernel controlling pointers and buffer sizes passed to the service
(2) ensuring the confidentiality and integrity of the application data passed through the OS.
SCONE supports shields for:
(1)the transparent encryption of files
(2) the transparent encryption of communication channels via TLS
(3) the transparent encryption of console streams.

A shield also has configuration parameters, which are encrypted and can be accessed only after the enclave has been initialized.

File system shield:

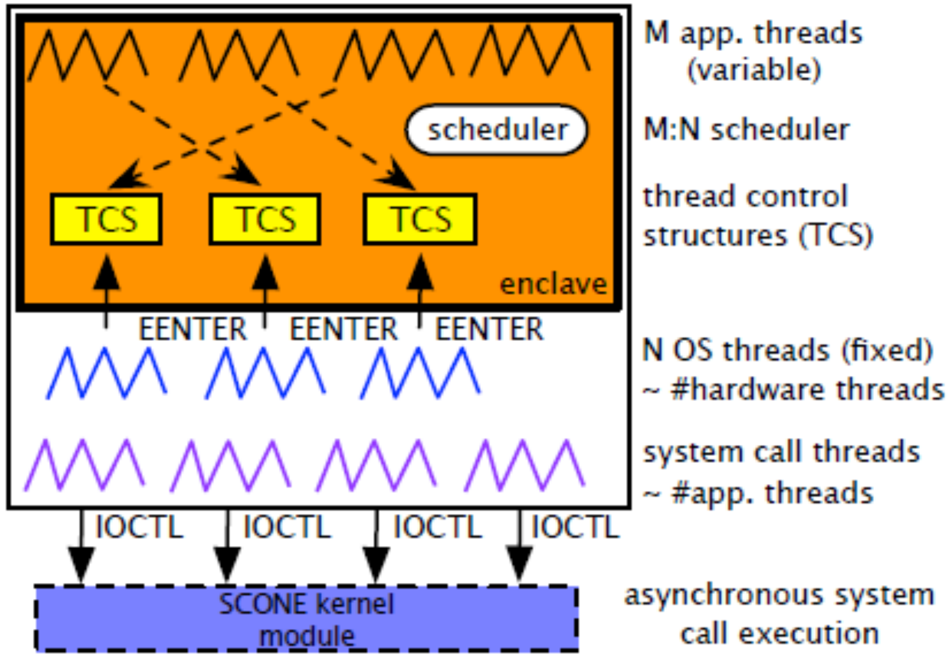
The file system shield protects the confidentiality and integrity of files.
Container image creator must define three disjoint sets of file path prefixes:
(1) Unprotected files,
(2) encrypted and authenticated files,
(3) authenticated files.
-Processes in a secure container have access to the standard Docker tmpfs, but it is costly as lightweight alternative SCONE supports a secure ephemeral file system through its file system shield.
the ephemeral file system maintains the state of modified files in non-enclave memory and it is faster than tmpfs.

The ephemeral file system is resilient against rollback attack: after restarting the container process, the file system returns to a preconfigured startup state that is validated by the file system shield, and therefore it is not possible for an attacker to rollback the file system to an intermediate state. This is also true during runtime, since the metadata for files' blocks resides within the enclave.

Network shield:
SCONE permits clients to establish secure tunnels to it and wraps all socket operations and redirects them to a network shield.
The network shield, upon establishing a new connection, performs a TLS handshake and encrypts/decrypts any data transmitted through the socket.
The private key and certificate are read from the container's file system. Thus, they are protected by the file system shield.
Console shield:
Container permit authorized processes to attach to the stdin, stdout, and stderr.
SCONE supports transparent encryption for them. The symmetric encryption key is exchanged between the secure container and the SCONE client during the startup procedure.
A console shield encrypts a stream by splitting it into variable-sized blocks.
A stream is protected against replay and reordering attacks by assigning each block a unique identifier, which is checked by the authorized SCONE client.

3.3 Threading model:

SCONE supports an M:N threading model in which M application threads inside the enclave are mapped to N OS threads. ->fewer enclave transitions.



M app. threads (variable)
scheduler
M:N scheduler
TCS
TCS
TCS
thread control structures (TCS)
enclave
EENTER
EENTER
EENTER
N OS threads (fixed) ~ #hardware threads
system call threads ~ #app. threads
IOCTL
IOCTL
IOCTL
IOCTL
SCONE kernel module
asynchronous system call execution

Figure 5: M:N threading model

-Multiple OS threads in SCONE can enter an enclave. Each thread executes the scheduler.
Scheduler checks if: (1) an application thread needs to be woken due to an expired timeout or the arrival of a system call response; or (2) an application thread is waiting to be scheduled. In both cases, the scheduler executes the associated thread.

-The number of OS threads inside the enclave is typically bound by the number of CPU cores.

-The system call threads reside in the kernel indefinitely to eliminate the overhead of kernel mode switches.

-When there are no pending system calls, the threads back-off to reduce CPU load.

3.4 Asynchronous system calls:

This interface consists of two lock-free, multi-producer, multi-consumer queues: a request queue and a response queue.

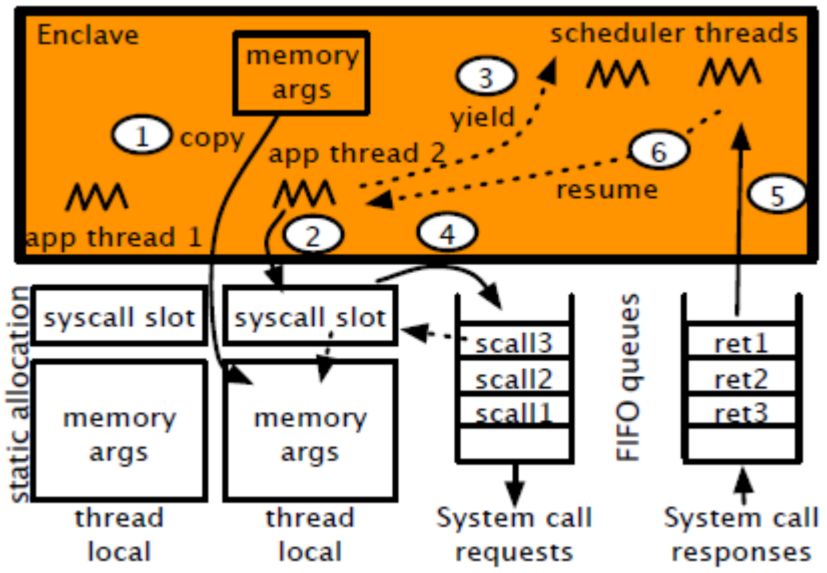


Figure 6: Asynchronous system calls

1. When system call happens copies memory-based arguments outside of the enclave
2. adds a description of the system call to a syscall slot data structure containing the system call number and arguments. The syscall slot and the arguments use thread-local storage.
3. Next the application thread yields to the scheduler ,which will execute other application threads until the reply to the system call is received in the response queue.
4. The system call is issued by placing a reference to the syscall slot into the request queue.
5. When the result is available in the response queue , buffers are copied to the inside of the enclave, and all pointers are updated to point to enclave memory buffers.
6. The associated application thread is scheduled again.

3.5 Docker integration:

The integration of secure containers with Docker requires changes to the build process of secure image, and change to client-side extensions.

SCONE does not require modifications to the Docker Engine or its API.

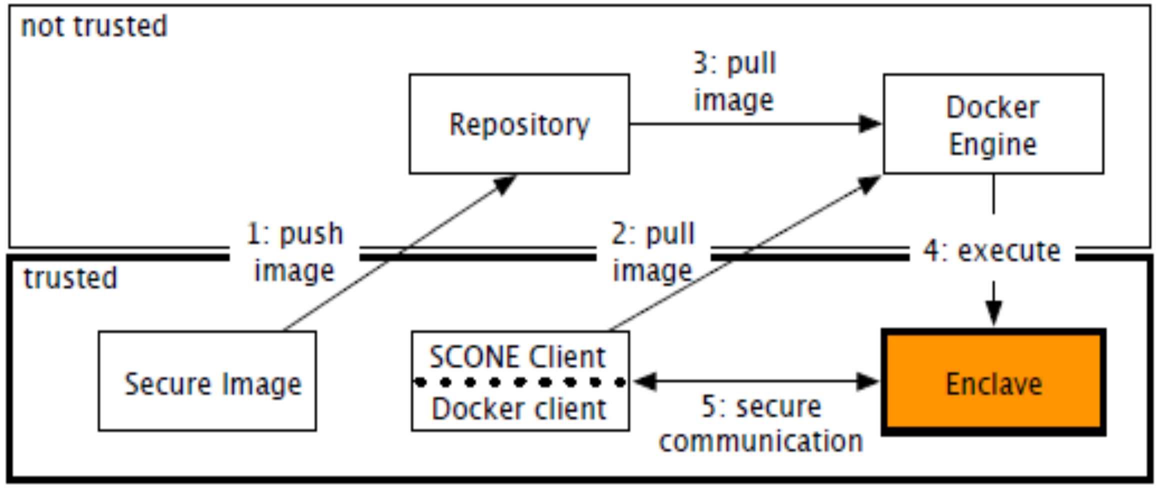


Figure 7: Using secure containers with Docker

Container startup: Each secure container requires a startup configuration file (SCF). The SCF contains keys to encrypt standard I/O streams, a hash of the FS protection file and its encryption key.

Since SGX does not protect the confidentiality of enclave code, putting the startup configuration in the enclave itself is not an option.

Instead, after the executable has initialized the enclave, the SCF is received through a TLS protected network connection, during enclave startup.

Evaluation:

They Used Two web servers, Apache, and NGINX ,Memcached ; Redis ; and SQLite.

The applications include a mix of compute (e.g., SQLite) and I/O intensive (e.g., Apache and Memcached) workloads.

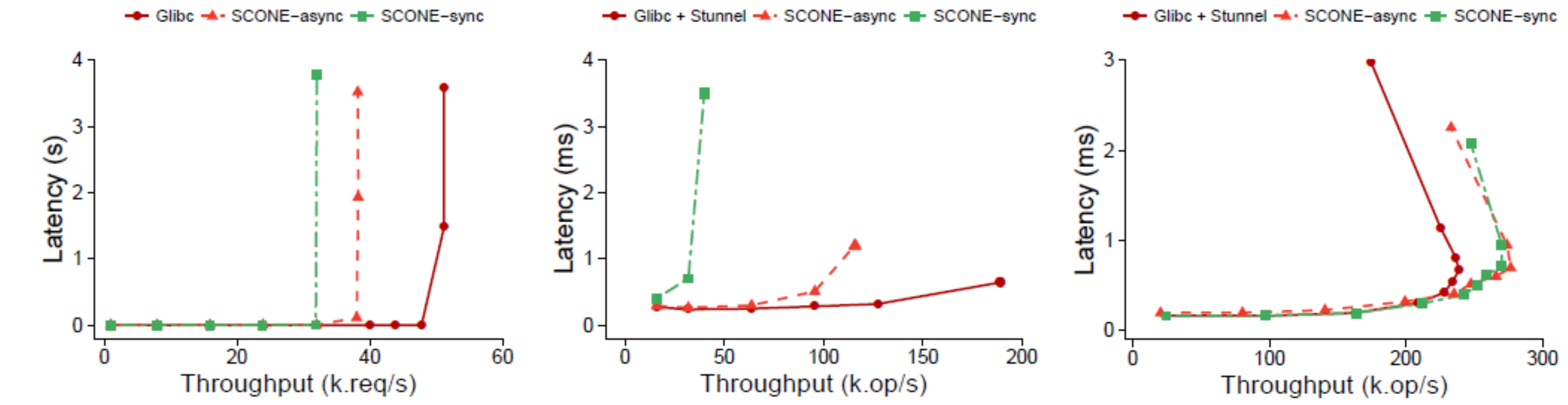
Three variants for each application:

1- one built with the GNU C library (glibc);

2- one built with the musl C library adapted to run inside SGX enclaves with synchronous system calls (SCONE-sync);

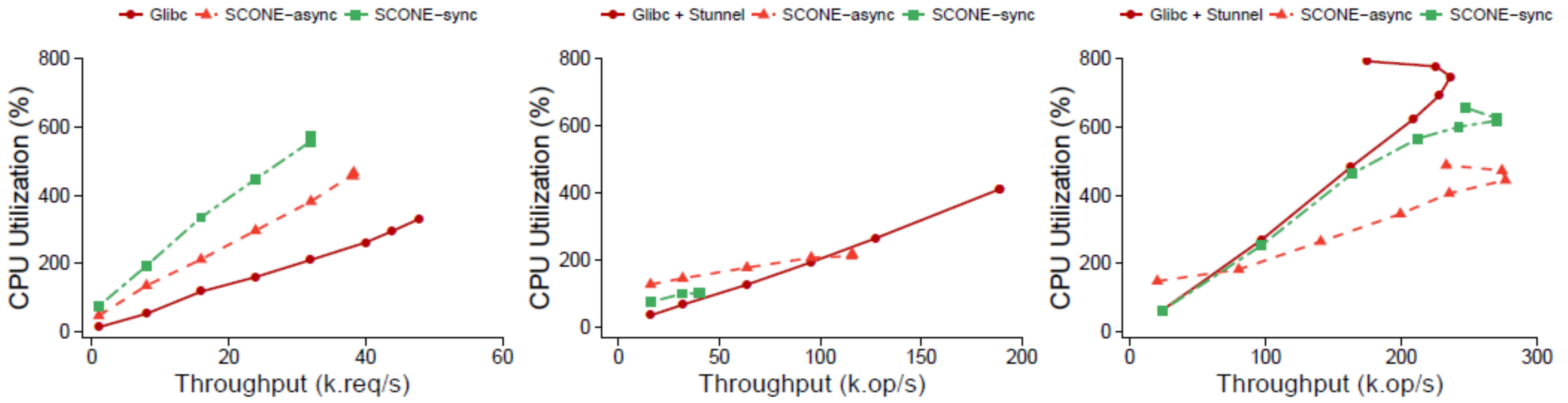
3- one built with the same musl C library but with asynchronous system calls (SCONE-async).

For applications that do not support encryption (e.g.,Memcached and Redis), they use Stunnel to encrypt their communication in the glibc variant.



(a) Apache (b) Redis (c) Memcached

Figure 8: Throughput versus latency for Apache, Redis, and Memcached



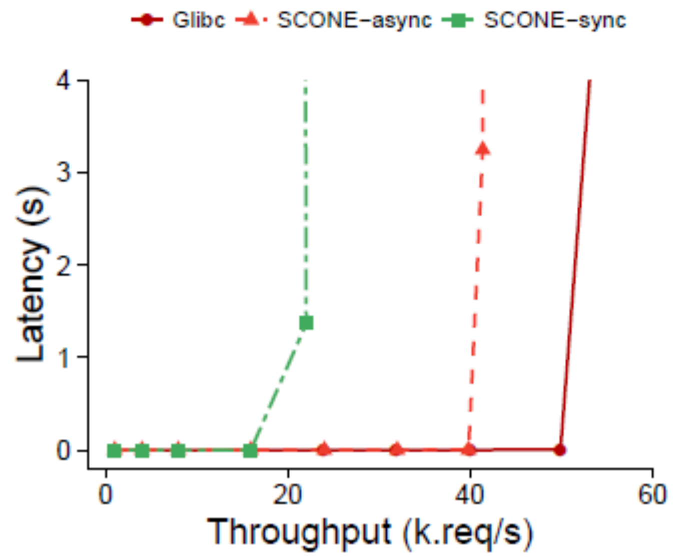


Figure 10: Throughput versus latency for NGINX

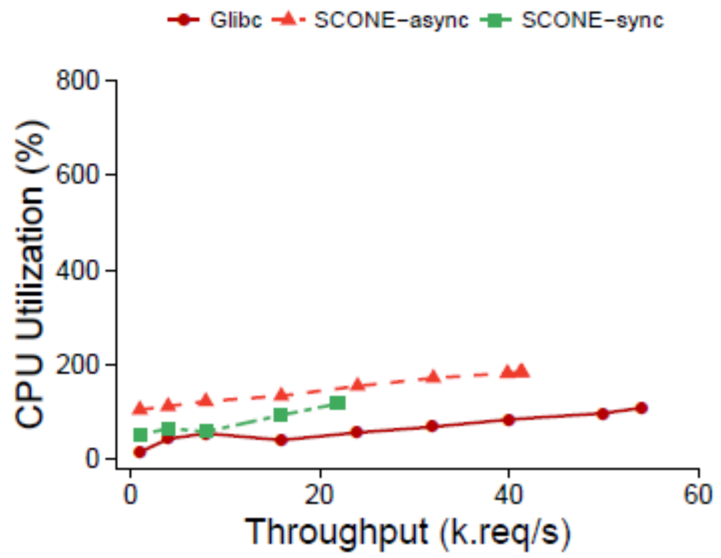


Figure 11: CPU utilization for NGINX

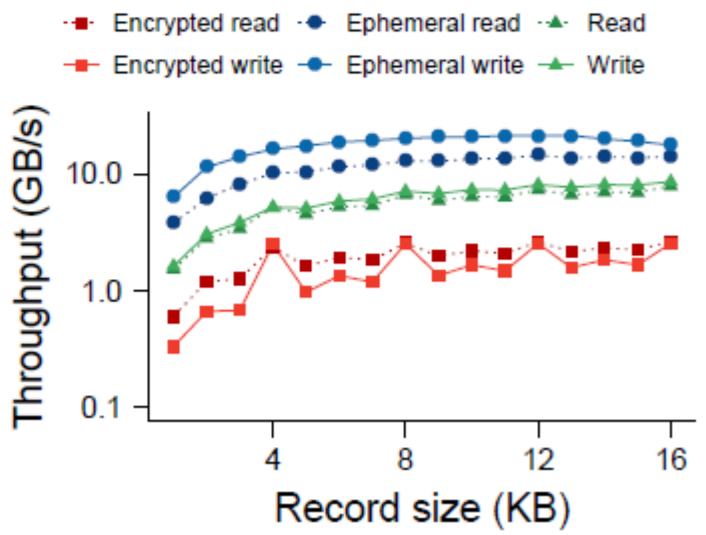


Figure 12: Throughput of random reads/writes with ephemeral file system versus `tmpfs`

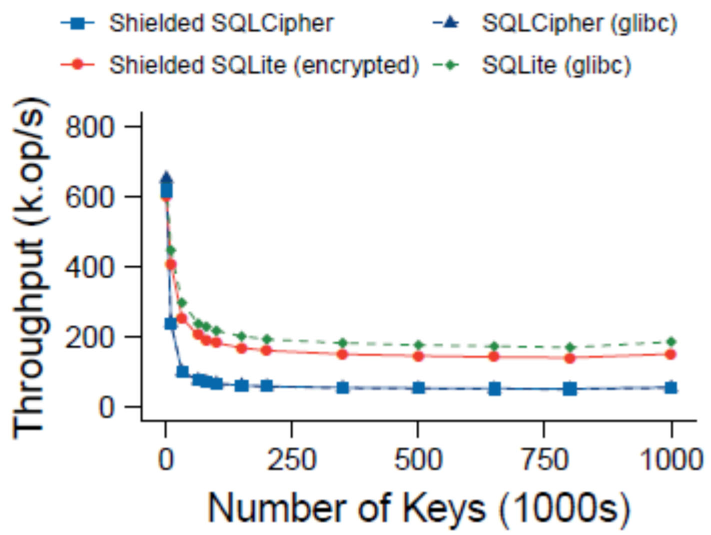


Figure 13: Throughput of SQLite and SQLCipher with file system shield

FIGURE 14 SHOWS HOW MANY PWRITE() CALLS CAN BE EXECUTED BY SCONE-ASYNC, SCONE-SYNC AND NATIVELY.

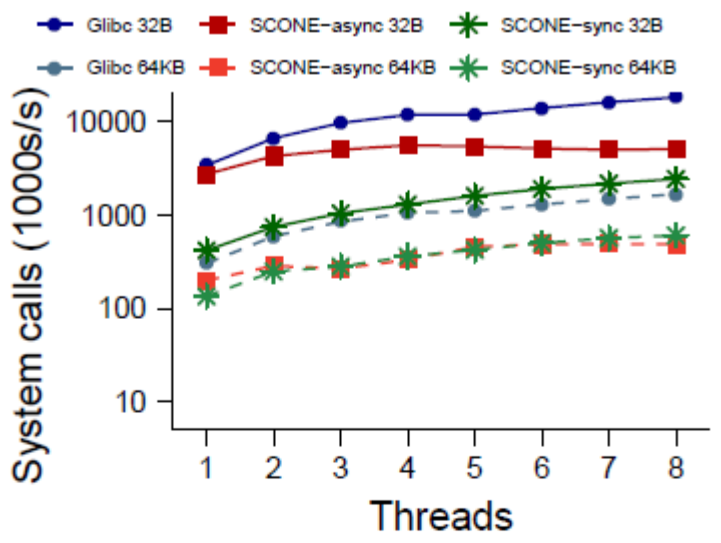


Figure 14: Frequency of system calls with asynchronous system call support

CONCLUSION

- SCONE increases the confidentiality and integrity of containerized services using Intel SGX.
- TCB is between 0.6–2 the application code size and are compatible with Docker.
- asynchronous system calls and a kernel module make SGX overhead less.
- For all evaluated services, they achieved at least 60% of the native throughput;

REFERENCE:

- https://www.ibr.cs.tu-bs.de/users/goltzsch/papers/osdi2016scone-preprint.pdf
- https://www.usenix.org/sites/default/files/conference/protected-files/osdi16_slides_knauth.pdf