iperfTZ: Understanding Network Bottlenecks for TrustZone-Based Trusted Applications

Christian Göttel, Pascal Felber, and Valerio Schiavoni(✉)

University of Neuchâtel, Rue Emile-Argand 11, 2000 Neuchâtel, Switzerland
{christian.gottel,pascal.felber,valerio.schiavoni}@unine.ch

Abstract. The growing availability of hardware-based trusted execution environments (TEEs) in commodity processors has recently advanced support (*i.e.,* design, implementation and deployment frameworks) for network-based secure services. Examples of such TEEs include Arm TrustZone or Intel SGX, largely available in embedded, mobile and server-grade processors. TEEs shield services from compromised hosts, malicious users or powerful attackers. TEE-enabled devices are largely being deployed on the edge of the network, paving the way for large-scale deployments of trusted applications. These applications allow processing and disseminating sensitive data without having to trust cloud providers. However, uncovering network performance limitations of such trusted applications is difficult and currently lacking, despite the interest and reliance by developers and system deployers.

iperfTZ is an open-source tool to uncover network performance bottlenecks rooted at the design and implementation of trusted applications for Arm TrustZone and underlying runtime systems. Our evaluation based on micro-benchmarks shows current trade-offs for trusted applications, both from a network as well as an energy perspective; an often overlooked yet relevant aspect for edge-based deployments.

Keywords: Network · Performance · Bottleneck · Measurement · ARM TrustZone · OP-TEE

1 Introduction

Services are being moved from the cloud to the edge of the network. This migration is due to several reasons: lack of trust in the cloud provider [7], energy savings [19,24] or reclaiming control over data and code. Edge devices are used to accumulate, process and stream data [20,30]. The nature of such data can be very sensitive: edge devices can be used to process health-based data emitted by body sensors (*e.g.,* cardiac data [26]), data originated by smart home sensors indicating the presence of humans inside a household, or even financial transactions [16,28]. In this context, applications using this information must be protected against powerful attackers, potentially even with physical access to

M. Ghaffari et al. (Eds.): SSS 2019, LNCS 11914, pp. 178–193, 2019.
https://doi.org/10.1007/978-3-030-34992-9_15

the devices. Additionally, communication channels for inter-edge device applications must also be secured to prevent attacks such as man-in-the-middle attacks. Edge devices are low-energy units with limited processing and storage capacity. As such, it is unpractical to rely on sophisticated software-based protection mechanisms (*e.g.,* homomorphic encryption [22]), currently due to their high processing requirements and low performance [12]. Alternatively, new hardware-based protection mechanisms can be easily leveraged by programmers to provide prior protection guarantees. Specifically, *trusted execution environments* (TEEs) are increasingly made available by hardware vendors in edge-devices [29]. Several ARM-based devices, such as the popular Raspberry Pi[1], embed native support for TEEs called TRUSTZONE [4,23]. TRUSTZONE can be leveraged to deploy *trusted applications* (TAs) with additional security guarantees.

There exist several programming frameworks and runtime systems to develop TAs for TRUSTZONE with varying capabilities and different degrees of stability and support (*e.g.,* SierraTEE[2], OP-TEE[3], and [21]). While a few studies look at the interaction between TEEs and the corresponding untrusted execution environments [2,14], little is known on the network performance bottlenecks experienced by TAs on ARM processors. We fill this gap by contributing IPERFTZ, a tool to measure accurately the network performance (*e.g.,* latency, throughput) of TAs for TRUSTZONE. IPERFTZ consists of three components, namely *(1)* a client application, *(2)* a TA, and *(3)* a server. Our tool can be used to guide the calibration of TAs for demanding workloads, for instance understanding the exchanges with untrusted applications or for secure inter-TEE applications [28]. In addition, IPERFTZ can be used to study the impact of network and memory performance on the energy consumption of running TAs. By adjusting IPERFTZ's parameters, users evaluate the network throughput of their TAs and can quickly uncover potential bottlenecks early in the development cycle. For instance, internal buffer sizes affect the achievable network throughput rates by a factor of 1.8×, almost halving throughput rates.

The rest of the paper is organized as follows. Section 2 motivates the need for tools analyzing TAs. We provide an in-depth background on TRUSTZONE in Sect. 3, as well as covering details on the TRUSTZONE runtime system OP-TEE. In Sect. 4 we present the architecture of IPERFTZ and some implementation details in Sect. 5. We report our evaluation results in Sect. 6. We cover related work in Sect. 7 before concluding in Sect. 8.

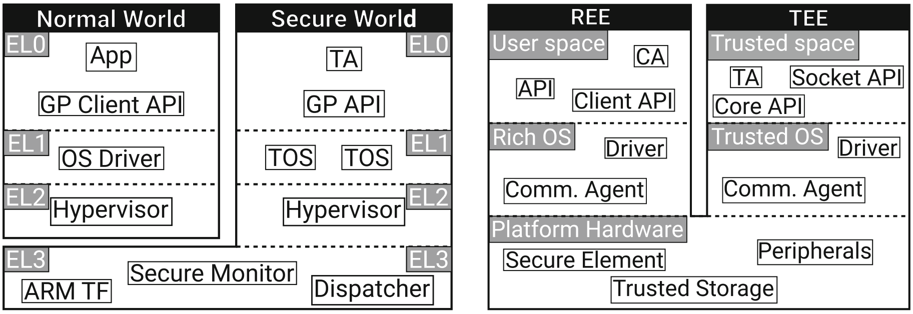
2 Motivating Scenario

We consider scenarios with simple yet practical services deployed as TAs. For instance, in [13] authors deploy key-value stores inside a TRUSTZONE runtime system. Benchmarks show a 12×–17× slowdown when compared to plain (yet

[1] https://www.raspberrypi.org, accessed on 30.07.2019.

[2] https://www.sierraware.com/open-source-ARM-TrustZone.html, accessed on 30.07.2019.

[3] https://www.op-tee.org, accessed on 30.07.2019.



(a) ARMv8.4-A architecture [3] **(b)** GLOBALPLATFORM architecture [11]

Fig. 1. Block diagrams highlighting relevant software components

unsecure) deployments, due to shared memory mechanisms between the trusted and untrusted environments. As further detailed in Sect. 4, networking in OP-TEE is supported by similar shared memory mechanisms. Yet, we observe the lack of tools to clearly highlight the root causes of such bottlenecks. Further, in the TRUSTZONE ecosystem, there is a lack of proper tools to evaluate network bottlenecks contrary to untrusted environments (*e.g.*, `iperf3`[4], `netperf`[5], `nuttcp`[6]). The overhead originating from the shared memory mechanism can be identified by comparing the measured network throughput inside and outside the TEE. Measuring such overheads is of particular relevance in embedded, mobile and IoT environments. In those scenarios, devices are often battery powered, limited both in time and capacity. Hence, network performance tools should further highlight energy costs, pointing users to specific bottlenecks.

3 Background

This section provides a background on ARM TRUSTZONE (Sect. 3.1), the GLOBALPLATFORM specifications (Sect. 3.2) and OP-TEE, the TRUSTZONE runtime system used for IPERFTZ (Sect. 3.3). This background helps understanding technical challenges in our context and how IPERFTZ addresses them.

3.1 ARM TrustZone in a Nutshell

TRUSTZONE is a security architecture designed for ARM processors and was introduced in 2003 [3]. It partitions hardware and software resources into two worlds, *i.e.*, *secure* and *normal* world, as shown in Fig. 1a. A dedicated `NS` bit [4] drives this world separation and allows to execute secure (`NS` bit set low) or

[4] https://software.es.net/iperf/, accessed on 30.07.2019.
[5] https://hewlettpackard.github.io/netperf/, accessed on 30.07.2019.
[6] https://www.nuttcp.net/Welcome%20Page.html, accessed on 30.07.2019.

non-secure (NS bit set high) transactions on the system bus. In general, non-secure transactions cannot access system resource secured by a low NS bit. The TrustZone architecture spans beyond the system bus, including peripherals (*e.g.,* GPUs [31] and I/O). Every TrustZone-enabled processor is logically split into a secure and a non-secure (virtual) core, executing in a time-shared manner. Hence, accessible system resources are determined by the executing core: secure cores can access all system resources, while non-secure cores can only access non-secure ones. Arm processors embed one *memory management unit* (MMU) per virtual core in charge of mapping virtual addresses to physical addresses. The *translation lookaside buffer* (TLB) in the MMU is used to maintain the mapping translations from virtual to physical memory addresses. Tagging TLB entries with the identity of the world allows secure and non-secure address translation entries to co-exist. With tags the TLB no longer has to be flushed making fast world switches possible.

The implementation of TrustZone is organized into four *exception levels* (EL) with increasing privileges [5] (Fig. 1a). EL0, the lowest one, executes unprivileged software. EL1 executes operating systems, while EL2 provides support for virtualization. Finally, Arm Trusted Firmware is running at EL3 dispatching boot stages at boot time and monitoring secure states. Switches between the two worlds are supervised by a secure monitor [6]. It is invoked in two ways: *(1)* by executing a *secure monitor call* (SMC), or *(2)* by a subset of *hardware exception mechanisms* [4]. When invoked, the secure monitor saves the state of the currently executing world, before restoring the state of the world being switched to. After dealing with the worlds' state, the secure monitor returns from exception to the restored world.

3.2 The GlobalPlatform Standard

GlobalPlatform[7] publishes specifications for several TEEs (*e.g.,* Op-Tee and [21]). We provide more details on Op-Tee in Sect. 3.3 (an implementation of such specifications), while briefly explaining the terminology in the remainder to understand Fig. 1b. An *execution environment* (EE) provides all components to execute applications, including hardware and software components. A *rich execution environment* (REE) runs a rich OS, generally designed for performance. However, it lacks access to any secure component. In contrast, TEEs are designed for security, but programmers have to rely on a reduced set of features. A trusted OS manages the TEE under constrained memory and storage bounds. TEE and REE run alongside each other. In recent Arm releases (since v8.4), multiple TEEs can execute in parallel [3], each with their own trusted OS. TAs rely on system calls usually implemented by the trusted OS as specific APIs [10]. *Client applications* (CA) running in the rich OS can communicate with TAs using the *TEE Client API*. Similarly, TAs can access resources such as *secure elements* (*i.e.,* tamper-resistant devices), *trusted storage*, and *peripherals*, or send messages outside the TEE. *Communication agents* in the TEE and REE

[7] https://globalplatform.org, accessed on 30.07.2019.

mediate exchanges between TAs and CAs. Finally, the *TEE Socket API* can be used by TAs to setup network connections with remote CAs and TAs.

3.3 Op-Tee: Open Portable Trusted Execution Environment

Op-Tee is an open-source implementation of several GlobalPlatform specifications [8–11] with native support for TrustZone. The Op-Tee OS manages the TEE resources, while any Linux-based distribution can be used as rich OS alongside it. Op-Tee supports two types of TAs: *(1)* regular TAs [11] running at EL0, and *(2) pseudo TAs* (PTAs), statically linked against the Op-Tee OS kernel. PTAs run at EL1 as secure privileged-level services inside Op-Tee OS's kernel. Finally, Op-Tee provides a set of client libraries to interact with TAs and to access secure system resources from within the TEE.

4 Networking for Trusted Applications

For networked TAs, *i.e.,* generating or receiving network traffic respectively from and to TAs, runtime systems must provide support for sockets and corresponding APIs. To do so, either *(1)* the TEE borrows the network stack from the REE, or *(2)* the TEE relies on *trusted device drivers.* The former solution implies leveraging *remote procedure calls* (RPC) to a `tee-supplicant` (an agent which responds to requests from the TEE), and achieves a much smaller *trusted computing base.* The latter allows for direct access to the network device drivers for much lower network latencies. Furthermore, it simplifies confidential data handling as the data does not have to leave the TEE. The former requires developers to provide data confidentiality before network packets leave the TEE, for instance by relying on encryption.

iperfTZ leverages `libutee`[8] and its socket API, supporting streams or datagrams. The socket interface exposes common functions: `open`, `send`, `recv`, `close`, `ioctl` and `error`. The GlobalPlatform specification allows TEE implementations to extend protocol-specific functionalities via command codes and `ioctl` functions. For example, it is possible to adjust the receiving and sending socket buffer sizes with TCP socket or changing the address and port with UDP sockets.

The `libutee` library manages the lifecycle of sockets via a TA session to the socket's PTA. The socket PTA handles the RPC to the `tee-supplicant`, in particular allocating the RPC parameters and assigning their values. Afterwards, a SMC instruction is executed to switch back to the normal world. The `tee-supplicant` constantly checks for new service requests from the TEE. Once a new request arrives, its arguments are read by the `tee-supplicant` and the specified command is executed. Finally, when the data is received by the `tee-supplicant`, it is relayed over Posix sockets to the rich OS. In essence, when data is sent or received over a socket, it traverses all exception levels, both secure (from EL0 up to EL3) and non-secure (from EL2 to EL0 and back up).

[8] https://optee.readthedocs.io/architecture/libraries.html#libutee, accessed on 30.07.2019.

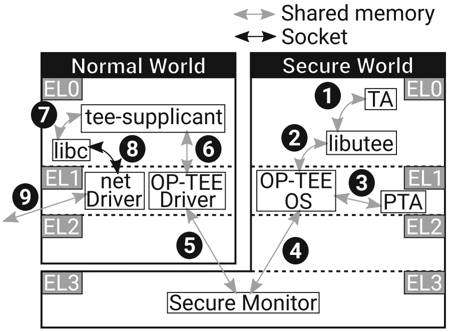


Fig. 2. Execution flow inside Op-Tee.

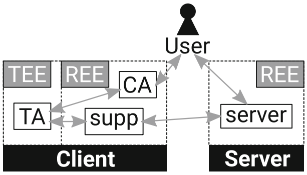


Fig. 3. Interaction of iperfTZ's components in the client-server model.

Figure 2 summarizes the previous paragraphs and shows the interaction between the secure and normal worlds in Op-Tee. The secure world hosts the TA, which interacts directly with `libutee` (Fig. 2-❶). When using GlobalPlatform's Socket API, `libutee` does a system call (Fig. 2-❷) to Op-Tee. Op-Tee then delegates the request to the socket PTA (Fig. 2-❸). The secure monitor is invoked through a SMC (Fig. 2-❹), which maps the data from the TEE to the REE's address space. From there execution switches into the normal world and the Op-Tee driver (Fig. 2-❺) resumes operation. Requests are then handled by the `tee-supplicant` (Fig. 2-❻) over `ioctl` system calls. The agent executes system calls using `libc` (Fig. 2-❼) to directly relate the underlying network driver (Fig. 2-❽) over the Posix interface. Once data reaches the network driver, it can be sent over the wire (Fig. 2-❾).

4.1 Threat Model

For our threat model we consider a malicious user that has physical access or is able to obtain remote access on the devices used to deploy iperfTZ as depicted in Fig. 3. By gaining access to the network or devices connected to it, the malicious user can break security by either compromising these devices or exploiting iperfTZ for *denial-of-service* (DoS) attacks. We assume that the REE, which includes the rich OS and the user space, cannot be trusted. However, we consider that the devices and the TEE, which includes dispatcher, Op-Tee, and secure monitor, can be trusted. As also stated in [4], side-channel attacks are out of scope of our threat model. We also point out that some ARM *systems on a chip* (SoCs) are affected by the Meltdown [18] and Spectre [15] attacks[9].

For use of iperfTZ in production, we recommend hardcoding network test parameters in the TA and disabling any argument passing to reduce the potential of DoS attacks. Furthermore, the signing key used for TAs should be kept

[9] https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability, accessed on 30.07.2019.

confidential as it allows the malicious user to modifiy TA binaries and create authentic TA binaries. Assuming the TrustZone-enabled device is equipped with an *embedded MultiMediaCard* (eMMC), then TAs can be securely stored on the eMMC and the malicious user cannot tamper with a TA's binary. In development use, manipulation of the CA's parameters by the malicious user to exploit a buffer overflow can be excluded. During a network bandwidth measurement, the malicious user can run a (distributed) DoS attack to reduce the network bandwidth, such that a lower network throughput is measured and reported by iperfTZ. At the time of writing, Op-Tee does not provide support for the TLS protocol which renders secure connections unusable. Although irrelevant to iperfTZ but applicable in general to networked TAs, the malicious user could run a man-in-the-middle attack, either directly within the REE or on the network, and intercept the traffic exchanged between the two devices.

5 Implementation

We describe the implementation challenges of the three components included in iperfTZ,[10] namely *(1)* a CA acting as proxy for iperfTZ's *(2)* TA, and *(3)* the server component which the TA is interfacing. All components are implemented in the C language, and consists of 927 lines of code: 243 for the client, 314 for iperfTZ's TA, and 430 for the server.[11]

5.1 iperfTZ: Client Application

When the CA starts, the TEE context is initialized (`TEEC_InitializeContext`) using the file descriptor fetched from the Op-Tee driver. Two distinct dynamic shared-memory areas are allocated (`TEEC_AllocateSharedMemory`) at this time, to *(1)* exchange arguments passed over the *command line interface* with the TA (see Sect. 5.2) and *(2)* to retrieve metrics gathered by the TA during the network measurement. Several arguments (*e.g.*,, IP of the target server node, dummy data size, socket buffer size) are written in the shared memory area. The dummy data size is used by the TA to read/write data to the interface socket. Both shared memory areas get registered with the operation data structure (`TEEC_OpenSession`) before calling the `TEEC_InvokeCommand` function. The executing thread in the CA is blocked until the TA completes. The execution inside the TEE is resumed at the TA's main entry point upon world switch. Once the TA completes, an `SMC` instruction drives the CPU core to switch back into the normal world, where execution is resumed. The metrics gathered from the TA are available to the user as persistent files.

5.2 iperfTZ: Trusted Application

The iperfTZ TA is the primary executing unit. It takes the role of the client in the client-server model. The TA allocates a buffer for the dummy data on the

[10] https://github.com/ChrisG55/iperfTZ.

[11] Numbers for individual components include local header lines of code.

Table 1. Comparison of evaluation platforms.

Device	QEMU	Raspberry
CPU Model	Intel Xeon E3-1270 v6	Broadcom BCM2837
CPU Frequency	3.8 GHz	1.2 GHz
Memory Size	63 GiB DDR4	944 MiB LPDDR2
Memory data rate	2400 MT/s	800 MT/s
Disk Model	Samsung MZ7KM480HMHQ0D3	Transcend micro SDHC UHI-I Premium
Disk Size	480 GB	16 GB
Disk Read Speed	528.33 MB/s	90 MB/s
Network Bandwidth	1 Gbit/s	100 Mbit/s

heap, filled with random data generated by Op-Tee's Cryptographic Operations API [10]. With the information from the arguments, the TA finally sets up a TCP interface socket and opens a client connection before assigning the socket buffer sizes. Our implementation relies on the Time API [10] to measure the elapsed time during the network throughput measurement inside the TEE. Op-Tee computes the time value from the physical count register and the frequency register. The count register is a single instance register shared between normal and secure world EL1. The network throughput measurement is then started while either maintaining a constant bit rate, transmitting a specific number of bytes or running for 10 seconds. During the measurement, the TA gathers metrics on the number of transmit calls, *i.e.*, `recv` and `send`, bytes sent, time spent in the transmit calls and the total runtime. Upon completion, results are written to the shared memory area and the execution switches back to the normal world.

5.3 iperfTZ: Server

The server component is deployed and executed inside the normal world. This is used to wait for incoming TCP connections (or inbound UDP datagrams) from iperfTZ's TA. While executing, it gathers similar network metrics as the other components. Additionally, this component collects TCP specific metrics, such as the smoothed *round trip time* or the *maximum segment size*. This TCP specific data is not accessible for TAs and can only be retrieved on the server side using a `getsockopt` system call.

6 Evaluation

In this section we will demonstrate how iperfTZ can measure the network throughput. We further draw conclusions regarding hardware and software implementation designs. We report that it is particularly challenging to assess network throughput, given the remarkable diversity one can find on embedded and mobile Arm systems.

Evaluation Settings. We deploy iperfTZ on the Raspberry Pi platform. Due to the limited network bandwidth of Raspberry Pi devices supported by Op-Tee, we also include results under emulation using QEMU.[12] With QEMU we can run the same evaluation as on the Raspberry Pi and we also profit from a higher network bandwidth. Table 1 compares in detail the two setups. For both setups we use the same machine as server, on which we collect power consumptions and run the iperfTZ server component.

Server. The server is connected to a Gigabit switched network, with access to power meter measurements. The nodes being measured are at a single-hop from the server. During the micro-benchmarks server components will be deployed on the server with fixed dummy buffer and socket buffer sizes of 128 KiB. This allows creating an accurate time series of the recorded throughput, latency and power metrics by concentrating the data acquisition on a single node.

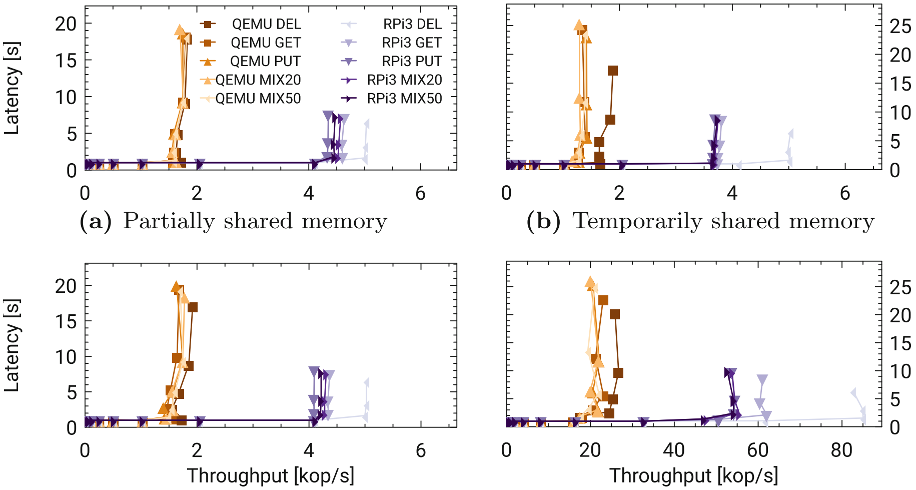
QEMU. We deploy Op-Tee with QEMU v3.1.0-rc3 running on a Dell PowerEdge R330 server. The Op-Tee project has built-in support for QEMU and uses it in system emulation mode. In system emulation mode QEMU emulates an entire machine, dynamically translating different hardware instruction sets when running a virtual machine with a different architecture. In order to provide full network capability, we replace the default SLiRP network[13] deployed with Op-Tee by a bridged network with a tap device.

Raspberry Pi. Op-Tee only supports the Raspberry Pi 3B. We deploy Op-Tee on a Raspberry Pi 3B v1.2 equipped with a Broadcom BCM2837 SoC. The SoC implements an ARM Cortex-A53 with ARMv8-A architecture. The BCM2837 chip lacks support for cryptographic acceleration instructions and is not equipped with TrustZone *Protection Controller* (TZPC), TrustZone *Address Space Controller* (TZASC), *Generic Interrupt Controller* (GIC) or any other proprietary security control interfaces on the bus [27]. The Raspberry Pi 3B lacks an on-chip memory or eMMC to provide a securable memory. We take these limitations into account in our evaluation, and leave further considerations once a more mature support for the Raspberry Pi platform is released.

Power Measurement. To measure the power consumption of the two platforms, we connect the Dell PowerEdge server to a LINDY iPower Control 2 × 6M *power distribution unit* (PDU) [17] and the Raspberry Pi 3B to an Alciom PowerSpy2 [1]. The LINDY PDU provides a HTTP interface queried up to every second with a resolution of 1 W and a precision of 1.5%. Alciom PowerSpy2 devices rely on Bluetooth channels to transfer the collected metrics. Both measuring devices collect voltage, current and power consumption in real time.

[12] https://www.qemu.org, accessed on 30.07.2019.

[13] https://wiki.qemu.org/Documentation/Networking#User_Networking_.28SLIRP.29, accessed on 30.07.2019.



(a) Partially shared memory

(b) Temporarily shared memory

(c) Whole shared memory

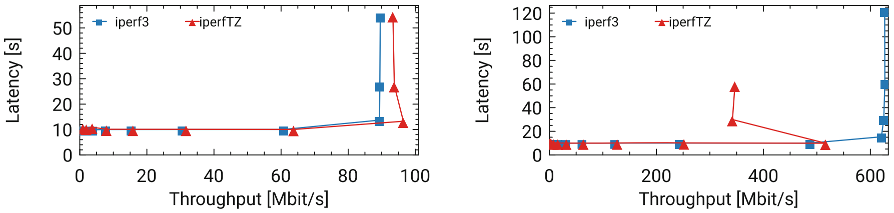
(d) CA in the REE

Fig. 4. Throughput-latency plots for different kinds of shared memory.

Memory Bandwidth. We use an existing key-value store TA [13] to evaluate the overhead of the different types of shared memory. The hash-table at the core of the key-value store uses separate chaining for collision resolution and implements modular hashing. The GLOBALPLATFORM specification defines three different types of shared memory: *whole* (an entire memory area), *partial* (a subset of an entire memory area with a specified offset), and *temporarily* (a memory area within the REE with an optional offset). The temporarily shared memory area is only shared with the TA for the duration of the TEE method invocation; the two others get registered and unregistered with the TEE session. The key-value store supports common operations such as `DEL`, `GET` and `PUT` on key-value pairs. We benchmark each operation in isolation as well as combining `GET` and `PUT` operations (`MIX`ed benchmark). The benchmarks operate as follows: for whole and partially shared memory, the CA will request a shared memory region of 512 KiB from the TEE and fills it with random data from `/dev/urandom`. With temporarily shared memory, the CA will allocate a 512 KiB buffer and initialize it similarly with random data. Before invoking a key-value operation a chunk size of 1 KiB is selected as data object at a random offset in the shared memory respectively buffer. The random offset is then used as key and every operation is timed using `CLOCK_MONOTONIC`.[14] During the benchmark 256 operations are issued at a fixed rate between 1 and 32768 operations per second. Figure 4 shows the throughput-latency plots for each type of shared memory as well as for running the key-value store as a CA in the REE.

Compared to the Raspberry Pi, the results on QEMU are predominantly superposed and only achieve about half the throughput. We believe this is due

[14] Manual page: `man time.h`.



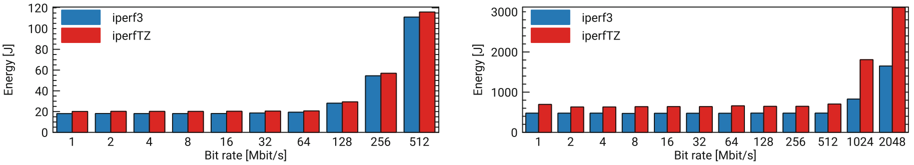
(a) Raspberry Pi **(b)** QEMU

Fig. 5. TCP network throughput measurements for 128 KiB buffer sizes.

to an I/O bound from the Arm instruction and TrustZone emulation using QEMU. We further observe with QEMU that the DEL benchmark for temporarily shared memory (Fig. 4b) and as CA (Fig. 4d) is clearly distinguishable from the other benchmarks. On the Raspberry Pi platform the graphs are well separated and ranked according to our expectations (lowest to highest throughput): PUT, MIX50, MIX20, GET, and DEL. The PUT operation has the lowest throughput because of memory allocation, memory copy and object insertion in the TA. The GET operation looks up the data object and copies it to the shared memory resulting in a higher throughput than the PUT operation. The mixed benchmarks show a similar behavior: the higher the PUT ratio, the lower the throughput. Hence, the MIX50 (50% PUT operations) has a lower average throughput than MIX20. The DEL operation avoids any time intensive memory operation and only has to free a data object after looking it up in the store. An interesting observation is made when comparing the memory throughput of the benchmarks executed in the REE against the benchmarks executed in the TEE. Key-value store operations executed inside TAs experience a 12×-14× overhead with QEMU and a 12×-17× overhead on the Raspberry Pi. This overhead is due to the world and context switches associated to TA method invocations.

Network Bandwidth. This micro-benchmark compares the network throughput measured with iperfTZ in Op-Tee to the network throughput measured with iperf3 in Linux. We deploy both programs with the same set of parameters, *i.e.,* 128 KiB socket and dummy buffer sizes. Upon each iteration the data send is doubled starting at 1 MiB up to 10 GiB. We allocate not more than 512 KiB for the dummy data on the TA's heap, since TAs are by default limited in Op-Tee to 1 MiB in size. Linux has two kernel parameters which limit the maximum size of read and write socket buffers: /proc/sys/net/core/rmem_max and /proc/sys/net/core/wmem_max. These kernel parameters can be changed at runtime using sysctl, in order to allocate larger socket buffers.

As shown in Fig. 5, iperfTZ generally exceeds the network throughput of iperf3 in both setups. On the Raspberry Pi 3B we cannot observe any degradation of the network throughput due to an overhead from frequent world switches. This result does not come as a surprise. The memory bandwidth benchmark operates at a throughput of several hundred MB/s, while the network bandwidth benchmark operates at about 10 MB/s. There is a gap of one order of magnitude



(a) Raspberry Pi

(b) QEMU

Fig. 6. Energy consumption during TCP network throughput measurements. Bit rates on the x-axis are given in logarithm to base 2.

in throughput between the two benchmarks, which we assume to be sufficient for the overhead not to arise. However, on QEMU we observe a serious degradation of the network throughput, when trying to achieve Gbit/s bit rate with Op-Tee. Remarkably, high throughput rates are strongly affected by the world switching overhead, even degrading beyond unaffected throughput rates. Our measurements indicate that network throughput beyond 500 Mbit/s is affected by a 1.8× world switching overhead, almost halving the network throughput.

Energy. During the network bandwidth benchmark, we recorded the power consumed by both setups. The LINDY iPower Control and the Alciom PowerSpy2 both record the timestamp as Unix time in seconds and the instantaneous power in watts. We use those units to execute a numerical integration over time using the trapezoidal method to obtain the total energy consumed by both setups during a benchmark run. Figure 6 shows these results. The total energy on the y-axis (in joule) is consumed by the device while executing a benchmark run for a specific bit rate on the x-axis (as binary logarithmic scale in Mbit/s). On the Raspberry Pi (Fig. 6a) we observe that before reaching saturation, iperfTZ is consuming about 2 J (11%) more than `iperf3`. In the highly saturated range, the energy doubles with the throughput. However, with QEMU (Fig. 6b), the energy difference between the execution in the REE and the TEE is significant. Given that QEMU is running on an energy-demanding and powerful server, iperfTZ consumes about 173 J (36%) more before the overhead arises than `iperf3` in the REE. We can clearly attribute this additional energy consumption observed on both setups to the execution of iperfTZ in the TEE. Certainly, the world switching overhead also contributes to an increase of the energy consumption with QEMU. By assuming a similar behavior for the energy consumption on QEMU as in the saturated range on the Raspberry Pi, we obtain a 1.6× energy overhead due to world switching.

7 Related Work

There exists a plethora of network benchmarking and tuning tools. We note that the implementation of iperfTZ is heavily inspired by the well-known `iperf` tool.

In this sense, iperfTZ supports a subset of its command-line parameters, for instance to facilitate the execution of existing benchmarking suites.[15]

The ttcp (Test TCP) tool was one of first programs implemented to measure the network performance over TCP and UDP protocols. Lately, it has been superseded by nuttcp.[16] A tool with similar features is netperf.[17] Unlike the aforementioned tools, tcpdump[18] is a packet analyzer that captures TCP packets being sent or received over a network. iperfTZ does not provide packet analysis tools. Instead, it does offer client and server-side measurements both for TCP and UDP data flows. More recently, iperf integrated most of the functionalities of ttcp, extending it with multi-threading capabilities (since iperf v2.0) and allowing bandwidth measurements of parallel streams. While it would be possible to provide similar support in iperfTZ, the execution of code inside the TAs is currently single-threaded, hence limiting the achievable outbound throughput. The most recent version of iperf (v3.0) ships a simplified (yet single-threaded) implementation specifically targeting non-parallel streams. Flowgrind[19] is a distributed TCP traffic generator. In contrast, iperfTZ follows a client-server model, with traffic generated between a server and a TA. StreamBox-TZ [25] is a stream analytics engine, which processes large IoT streams on the edge of the cloud. The engine is shielded from untrusted software using TrustZone. Similar to iperfTZ, StreamBox-TZ runs on top of Op-Tee in a TA. Yet, iperfTZ does not process data streams but can generate and measure network performance of those streams.

To summarize and to the best of our knowledge, iperfTZ is the first tool specifically designed to run as a TA for TrustZone that can measure the achievable network throughput for such applications.

8 Conclusion and Future Work

The deployment of TAs is becoming increasingly pervasive for the management and processing of data over the network. However, due to constraints imposed by the underlying hardware and runtime system, network performance of TAs can be affected negatively. iperfTZ is a tool to measure and evaluate network performance of TAs for Arm TrustZone, a widely available TEE on embedded, IoT and mobile platforms. We implemented the iperfTZ prototype on top of Op-Tee and we evaluated it on the Raspberry Pi platform. Our experimental results highlight performance and energy trade-offs deployers and programmers are confronted with both on hardware and emulated environments. We believe the insights given by our work can be exploited to improve design and configuration of TEEs for edge devices handling real-world workloads for TAs.

[15] Full compatibility with iperf would require substantial engineering efforts that we leave out of the scope of this work.

[16] See footnote 6.

[17] See footnote 5.

[18] https://www.tcpdump.org, accessed on 30.07.2019.

[19] www.flowgrind.net, accessed on 30.07.2019.

We intend to extend our work to support different types of sockets (*e.g.*, datagram sockets) and to leverage on-chip cryptographic accelerators. This would allow us to provide TLS-like channels for TAs, a feature that has not yet been implemented in Op-Tee. Finally, we aim for supporting various kinds of TEEs, especially in the context of embedded platforms and SoC, such as Keystone[20] for RISC-V processors.

Acknowledgments. The authors would like to thank the anonymous reviewers for their helpful comments and suggestions. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the LEGaTO Project (legato-project.eu), grant agreement No. 780681.

References

1. Alciom: PowerSpy2, 1.01 edn, 4 March 2013
2. Amacher, J., Schiavoni, V.: On the performance of ARM TrustZone. In: Pereira, J., Ricci, L. (eds.) DAIS 2019. LNCS, vol. 11534, pp. 133–151. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22496-7_9
3. Arm Limited: Isolation using virtualization in the Secure world. https://developer.arm.com/-/media/Files/pdf/Isolation_using_virtualization_in_the_Secure_World_Whitepaper.pdf?revision=c6050170-04b7-4727-8eb3-ee65dc52ded2
4. Arm Limited: ARM Security Technology: Building a Secure System using TrustZone Technology, April 2009. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf. Accessed 30 July 2019
5. Arm Limited: Arm Cortex-A53 MPCore Processor: Technical Reference Manual, 8 February 2016. https://developer.arm.com/docs/ddi0500/g. Revision: r0p4
6. Arm Limited: Fundamentals of ARMv8-A, March 2017. https://static.docs.arm.com/100878/0100/fundamentals_of_armv8_a_100878_0100_en.pdf. Accessed 30 July 2019
7. Baumann, A., Peinado, M., Hunt, G.: Shielding applications from an untrusted cloud with Haven. ACM Trans. Comput. Syst. **33**(3), 8:1–8:26 (2015). https://doi.org/10.1145/2799647
8. GlobalPlatform, Inc.: TEE Client API Specification Version 1, July 2010
9. GlobalPlatform, Inc.: TEE Sockets API Specification Version 1.0.1, January 2017
10. GlobalPlatform, Inc.: TEE Internal Core API Specification 1.1.2.50, June 2018
11. GlobalPlatform, Inc.: TEE System Architecture Version 1.2, November 2018
12. Göttel, C., et al.: Security, performance and energy implications of hardware-assisted memory protection mechanisms on event-based streaming systems. In: 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), pp. 264–266 (2018). https://doi.org/10.1109/SRDS.2018.00042
13. Göttel, C., Felber, P., Schiavoni, V.: Developing secure services for IoT with OP-TEE: a first look at performance and usability. In: Pereira, J., Ricci, L. (eds.) DAIS 2019. LNCS, vol. 11534, pp. 170–178. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22496-7_11
14. Jang, J.S., Kong, S., Kim, M., Kim, D., Kang, B.B.: SeCReT: secure channel between rich execution environment and trusted execution environment. In: NDSS, pp. 1–15 (2015)

[20] https://keystone-enclave.org, accessed on 30.07.2019.

15. Kocher, P., et al.: Spectre attacks: exploiting speculative execution. In: 40th IEEE Symposium on Security and Privacy (S&P 2019) (2019)
16. Lind, J., Eyal, I., Pietzuch, P., Sirer, E.G.: Teechan: payment channels using trusted execution environments. ArXiv preprint arXiv:1612.07766 (2016)
17. Lindy Electronics Ltd.: iPower Control 2x6M/2x6XM, 1 edn, June 2015
18. Lipp, M., et al.: Meltdown: reading kernel memory from user space. In: 27th USENIX Security Symposium (USENIX Security 2018) (2018)
19. Lyu, X., et al.: Selective offloading in mobile edge computing for the green internet of things. IEEE Netw. **32**(1), 54–60 (2018). https://doi.org/10.1109/MNET.2018.1700101
20. Mäkinen, O.: Streaming at the edge: local service concepts utilizing mobile edge computing. In: 2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies, pp. 1–6 (2015). https://doi.org/10.1109/NGMAST.2015.35
21. McGillion, B., Dettenborn, T., Nyman, T., Asokan, N.: Open-TEE - an open virtual trusted execution environment. In: 2015 IEEE Trustcom/BigDataSE/ISPA, TRUSTCOM 2015, vol. 1, pp. 400–407. IEEE Computer Society, Washington, DC (2015). https://doi.org/10.1109/Trustcom.2015.400
22. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW 2011, pp. 113–124. ACM, New York (2011). https://doi.org/10.1145/2046660.2046682
23. Ngabonziza, B., Martin, D., Bailey, A., Cho, H., Martin, S.: TrustZone explained: architectural features and use cases. In: 2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), pp. 445–451 (2016). https://doi.org/10.1109/CIC.2016.065
24. Ning, Z., Kong, X., Xia, F., Hou, W., Wang, X.: Green and sustainable cloud of things: enabling collaborative edge computing. IEEE Commun. Mag. **57**(1), 72–78 (2019). https://doi.org/10.1109/MCOM.2018.1700895
25. Park, H., Zhai, S., Lu, L., Lin, F.X.: StreamBox-TZ: secure stream analytics at the edge with TrustZone. In: 2019 USENIX Annual Technical Conference (USENIX ATC 2019), pp. 537–554. USENIX Association, Renton, July 2019. https://www.usenix.org/conference/atc19/presentation/park-heejin
26. Segarra, C., Delgado-Gonzalo, R., Lemay, M., Aublin, P.-L., Pietzuch, P., Schiavoni, V.: Using trusted execution environments for secure stream processing of medical data. In: Pereira, J., Ricci, L. (eds.) DAIS 2019. LNCS, vol. 11534, pp. 91–107. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22496-7_6
27. Sequitur Labs Inc.: Easing Access to ARM TrustZone - OP-TEE and Raspberry Pi 3, 26 September 2016
28. Shepherd, C., Akram, R.N., Markantonakis, K.: Establishing mutually trusted channels for remote sensing devices with trusted execution environments. In: Proceedings of the 12th International Conference on Availability, Reliability and Security, ARES 2017, pp. 7:1–7:10. ACM, New York (2017). https://doi.org/10.1145/3098954.3098971
29. Shepherd, C., et al.: Secure and trusted execution: past, present, and future - a critical review in the context of the internet of things and cyber-physical systems. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 168–177 (2016). https://doi.org/10.1109/TrustCom.2016.0060
30. Varghese, B., Wang, N., Barbhuiya, S., Kilpatrick, P., Nikolopoulos, D.S.: Challenges and opportunities in edge computing. In: 2016 IEEE International Confer-

ence on Smart Cloud (SmartCloud), pp. 20–26 (2016). https://doi.org/10.1109/SmartCloud.2016.18

31. Volos, S., Vaswani, K., Bruno, R.: Graviton: trusted execution environments on GPUs. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, pp. 681–696, OSDI 2018. USENIX Association, Berkeley (2018). http://dl.acm.org/citation.cfm?id=3291168.3291219