NetSD:
Remote Access to Integrated SD Cards of Embedded Devices

Valentin Schröter*, Arne Boockmeyer† and Lukas Pirl†
Professorship for Operating Systems and Middleware
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany

Email: *valentin.schroeter@student.hpi.uni-potsdam.de, †{firstname.lastname}@hpi.uni-potsdam.de

Abstract—Digitalization continuously pervades all areas and
the Internet of Things (IoT) is still on the rise. This leads to an
increased need for efficiency in the development of embedded
devices and systems composed thereof. Hybrid testbeds are
common environments to representatively assess, e.g., hardware-
software interaction, interoperability, and scalability. Although
automation is inevitable to achieve efficiency, not all devices offer
interfaces to be fully software-controlled. Most notably, block
devices tend to be inaccessible for software outside a Device under
Test (DuT), especially when the latter is in a dysfunctional state.

This paper introduces the Networked SD card (NetSD) which
enables remote access to removable block devices. The proposed
system consists of a hardware part, which enables multiplexed
access to a block device (e.g., an SD card) and a software part
which enables remote access to the block device (e.g., via HTTP or
network block device). NetSD thus adds testing and automation
opportunities to DuTs without the need to modify their hard- or
software. During the hardware design, we fund that different SD
transfer modes and access profiles (read or write focus) benefit
from different pull-up resistor configurations for the data lines.

Index Terms—Hardware, Testing, Testbeds, Remote Storage
Access, Internet of Things, Fault Injection

I. INTRODUCTION

In modern days, all areas of daily life experience shift to-
wards more digitalization. Systems are increasingly computer-
aided and interconnected. This also affects safety-critical areas,
like the railway industry. Besides the positive effects like
increased efficiency and comfort, the aforementioned shift also
comes with the challenge of more complexity and thus more
potential causes for defects. This indicates that especially in
the area of safety-critical systems, testing the systems under
development and their interplay with other systems need to be
a first-class concern throughout their whole life cycle.

Especially for railway infrastructure, large-scale testing of
the interoperability of digital devices is a relatively novel
challenge. So far, there were only a few players on the
infrastructure market selling their products. For this small
amount of devices and interfaces, manual interoperability
testing was feasible. But due to new open standards, like from
the EULYNX initiative‡, the market opens for new players
and their devices. This circumstance increases the need for
automated testing, as the large number of devices, interfaces,
inter-dependencies and synergistic effects make manual testing
insensible.

A common way to test the interoperability of software
together with the underlying hardware is testing in hybrid
testbeds. A testbed like Marvis [1] provides a representative
and scalable execution environment for hybrid tests. Therefore,
techniques like simulations, virtualization, and hardware-in-
the-loop (HIL) are used [2]. A device under development
becomes the Device under Test (DuT) to interact with a
set of other devices, in order to test if the behavior and
communication are interoperable. Together with the traffic
simulator SUMO [2], Marvis can execute railway scenarios, each
containing a new configuration and a new set of connected
devices to test various workflows for the DuT.

This requires, that the DuT is easily reconfigurable (e.g.,
firmware version, network configuration, software version).
The reality shows, that easy reconfiguration is mostly not
supported by embedded systems. It is rather the contrary, that
remote access to the configuration is prevented by the system
hardware and software design. This reduces the risk for attacks
but also limits the capabilities for automated testings.

To add the capability of automated configuration for de-
vices, which are by default not able to be reconfigurable in an
automated way but contain a block storage device for configu-
ration, we suggest the Networked SD card (NetSD), a remotely
accessible block storage device. The project arose from the
use case of improving the automated testing capabilities of
a railway axle counter, which stores its configuration data on
integrated block storage that is not accessible remotely. NetSD
makes the block storage device — including the possibly
contained file systems and files — remotely accessible without
physical access to the device itself. Besides remote access,
NetSD also enables the injection of faults at hardware level,
which opens new possibilities for testing storage hosts.

The following paper is structured as follows: section [II]
provides an overview of other solutions of remotely acces-
sible block storage devices. Section [III] describes the details
about our hardware implementation. Section [IV] evaluates
the performance of the previously described implementation.
Section [V] shortly illustrates the software enabling remote
access. Section [VI] summarizes the capabilities of the imple-

https://www.eulynx.eu (accessed 2021-07-07)

https://sumo.dlr.de (accessed 2021-07-07)
Remote SDs only work when the host device is powered. It would not be possible at all to access the SD when the host device is switched off.

- They are running a custom and inaccessible firmware. Modifications would require reverse engineering.
- The connection settings like IP addresses, the WiFi SSID, and passwords are stored on the SD itself. For any changes, it is required to remove the SD from the DuT.
- On the other hand, WiFi SDs could not grant exclusive data access nor the guarantee, that there is no data corrupted during the operation of embedded systems.

WiFi SDs could be sufficient for specific applications. However, the option to access data while the device is turned off is already missing for our use case. Apart from this, we would not have any possibility to expand the functionality regarding enhanced testbed features that are presented in section V.

Conclusion and Future Work

Besides WiFi SDs, to the best of our knowledge, we haven’t found any closely related publications concerning direct network-accessible block storage devices.

III. Implementation

The project goal includes, that NetSD enables remote access to the block storage without the need for modifying the software of the DuT. This aspect of the project is especially important for complex systems like railway infrastructure devices, that are not open source or easily adaptable. NetSD must not interfere with the actual operation in any unintentional way. Thus, while expanding the SD with remote capabilities, it should appear unchanged and be controllable from the DuT via the same interface.

This leads to the two relevant aspects of the project:
- Implement a device enabling network access to an SD.
- Implement a mechanism to enable shared access to the SD from both our remote access gateway and the DuT while maintaining the same interface for the DuT.

For simplicity, we limit our system to SDs, although embedded devices might also use other technologies, such as CompactFlash (CF) cards. However, some embedded devices, like the axle counter from the initial use case of the project, store its configurations on CF cards. Nowadays, SDs appear to be the most common and various adapters to convert between the different interfaces are available, e.g., CF-to-SD. The decision to base NetSD on SD technology aims to make the project applicable for a wide range of devices — be it directly or through adapters. In the following, we use the term SD to not only refer to SD cards themselves but also to refer to block storage technologies having adapters to SDs.

Figure 1 shows the system architecture. Our device, depicted on the right, implements both the remote and the shared access part.

Since NetSD is intended for development, fault injection, test automation, and alike purposes, neither hardware nor software security are of concern in this project.

A. Remote Access Gateway

To allow remote access, NetSD is based on a microcontroller with network capabilities. Nowadays, there are various microcontrollers and single-board computers offering onboard WiFi, like the Raspberry Pi 4, the ESP32 microcontroller or the wireless STM32 series. For this project, we decide to use an ESP32 that can connect to existing networks as well as creating an own access point. As a small-sized, low-level microcontroller with 4 MByte Flash, 320 KByte RAM, and a 160 MHz dual-core processor, it has enough performance to handle simple storage tasks on the SD and also offers fast WiFi performance. In contrast to a Raspberry Pi, the ESP32 also provides real-time capabilities out of the box which is important for timing aspects of, e.g., storage protocols.

B. Physical Shared Access

The shared access implementation is the main part of our contribution. The following problems arise if multiple devices should have simultaneous access to the same SD:

- The shared access must support the DuT and NetSD having different voltage levels. As the SD interface is specified to operate between 1.7 V and 3.6 V, we cannot assume both devices having the same voltage.
Advantages
- Fast control speed and high signal transmission frequencies.
- Non-exclusive shared access can create data conflicts, as simultaneous operations from both devices can lead to failed transmissions.

Disadvantages
- Slow control speed. Switching needs a lot of energy. Big component size.
- Decrease in voltage affects the interface. Complex electrical network to allow bidirectional switch.
- Complex electrical network to allow bidirectional switch.

Due to these problems, a simple electrical connection of all SD data lines between the DuT and our device would be neither safe nor reliable in operation. To ensure a functioning shared access, we introduce the SD switch. This part of NetSD can grant explicit and exclusive access to the SD, either to the DuT or to our remote access gateway. Therefore it is possible to let the DuT access the SD by default and just revoking the access for a short time if we have to change data by our remote access gateway.

An SD contains 6 data lines and 2 power lines [3, p. 12]. The SD switch allows to switch on and off bidirectional communication on all data lines as well as switching the power source so that the respective connected device can perform hard resets of the SD. For switching, multiple electrical components can be considered, while mainly relays, transistors, and MOSFETs are relevant. Table I lists the advantages and disadvantages of each technology.

| Component | Advantages | Disadvantages |
|-----------|------------|---------------|
| Relays    | Easy to design bidirectional switch. Certainty of electrical separation. | Slow control speed. Switching needs a lot of energy. Big component size. |
| Transistors | Fast control speed and high signal transmission frequencies. Small component size. | Decrease in voltage affects the interface. Complex electrical network to allow bidirectional switch. |
| MOSFETs  | Fast control speed switching and high signal transmission frequencies. No changes in signal voltage. Small component size. | Complex electrical network to allow bidirectional switch. |

The simplest way to implement an electrical switch would be to use relays. However, relays as mechanical components are comparatively large, which would accumulate for six data lines per connected device. A MOSFET on the other side is a non-mechanical, cheap and small-sized component. An important requirement for switching bidirectional interface lines is the consistency of the voltage levels. In contrast to a normal transistor, a MOSFET exactly offers this voltage consistency. That leads to using a MOSFET as the preferred choice for the SD switch basis. Nevertheless, to ensure a safe electrical separation if the SD switch should disconnect a single device, multiple MOSFETs and other components must be combined in a complex electrical circuit. But, as analog switching is a default task in electronics, there exist Integrated Circuits (ICs) offering the desired bidirectional switching functionality with safe electrical separation. For this project, we chose the 74LV1G66-Q100 circuit from Nexperia.

Figure 2 depicts how analog switches are used to ensure electrical separation for both devices. All signals of the SD are connected to two sets of switches, one set for the DuT on the right and one set for our remote access gateway on the left. Both switch sets contain a single switch for each of the six data lines from the SD. All switches are controllable by our remote access gateway, which takes care of which side currently has granted access to the SD.

C. SD extender cable

To relocate the physical SD to NetSD, an extension cable is used, that simulates the SD in the actual slot of the DuT. The extension cable uses crosstalk avoiding signal arrangement, which can be seen on the right side of fig. 3. This arrangement inserts a GND line between each data line, minimizing the mutual influence of signals.

For low-frequency signals (around 10 MHz) it is possible to use the exact same arrangement as the contacts from the SD, which can be seen on the left side of fig. 3. However, see fig. 3 for an overview of the SD data lines.
since the native SD interface is specified to transfer data with up to 208 MHz [3, p. 1] these direct adjacency of signal lines leads to interference in the transmission. Thus the crosstalk avoiding signal is important to meet tighter timings and allow reliable transmissions from host devices to the SD.

D. Software

To enable accessing data on SD via our remote access gateway with software, NetSD needs to offer software interfaces to the network. We implemented two different software interfaces for remote access which are described below. The ESP32 itself can communicate with the SD through an Serial Peripheral Interface (SPI) and through the native SD interface [3, p. 12 ff.]. Both interfaces allow unrestricted access to the SD.

1) REST API: : The first method is a simple REST API. The API offers request endpoints the cover all basic file level operations. This enables simple and platform-independent communication with the connected SD.

2) Network block device: : The second method for remote access is the Network Block Device (NBD) protocol. This network protocol allows Linux systems to mount remote storage devices as virtual block devices. Once mounted, an SD can be accessed like normal other mounted devices at a block level. Therefore using NBD we can work with the remote SD like a normal directory in our local file system, but also flush full images to the SD. One drawback of NBD is the dependence on using Linux as the operating system. However, NBD is also available on Windows Subsystem for Linux (WSL). Furthermore, the data can also be made available to other platforms via protocols such as WebDAV.

3) System flow chart: : Figure 4 illustrates the normal operation process that allows exclusive access to the block storage. By default, the DuT has access to the SD. Our system is waiting continuously for incoming requests. On a request, the access to the SD is switched to our remote access gateway, enabling the execution of requested operations. After these operations are finished, the access is switched back to the DuT. Between each change in access to the SD, it is repowered to allow reinitialization to the communication interface. This hard reset would not be necessary if concurrent access is implemented like described in section V Conclusion and Future Work. For the project’s use case with the axle counter, the device itself will not be turned on until corresponding operations on SD are executed by our device, since the axle counter only read configuration files when starting it. This procedure however is implemented by the main tested application that embeds NetSD.

IV. EVALUATION OF THE HARDWARE PROTOTYPE

A switch circuit between the physical SD and an accessing device inevitably influences the electrical properties of the data lines. Especially for DuTs with fast data transfer requirements, this influence must be as low as possible. This section presents the hardware quality for the following test conditions:

- SD: Intenso microSDHC, 8 GByte, Class 10 (min. 20 MByte/s read throughput, min. 12 MByte/s write throughput).

Fig. 4. Flowchart of normal operation of NetSD. In normal operation, only one device is given explicit access to the SD at a time.

- Test tool: MiniTool Partition Wizard[7] on Windows 10 operating system testing with different data block sizes. Since the evaluation does not test the remote data throughput, but the signal quality for a DuT connected to an SD via the switching circuit, no NBD is required.
- Baseline test: Windows 10 computer as DuT with the SD directly plugged in, also without an extension cable.
- Test of our SD switch: Windows 10 computer as DuT with access to the SD through our SD switch.
- SD switch specification: 74LVC1G66 as switch ICs, 48cm long extension cable with crosstalk avoiding signal arrangement, and two different pull-up resistor configurations for the hardware (see below).

SD interfaces operate with open-drain I/Os[7]. These open-drain I/Os require a default voltage level on the signal line induced by the usage of pull-up resistors. Normally, pull-up resistors are present on the side of the host device. However, since the physical SD is relocated by an extension cable and the switch ICs into NetSD, these host pull-up resistors are not sufficient to guarantee steep signal edges. This consideration leads to two different hardware configurations, either by just using the already existent pull-up resistors of the host device (visualized on the bottom of fig. 5) or by explicitly adding pull-up resistors on NetSD to improve signal quality (visualized on the top of fig. 5). To test and debug these different hardware configurations we created an evaluation board that is visible in fig. 8.

The first test evaluates the read throughput of the SD. Therefore we compare the read throughput through our switch with and without an explicitly added pull-up resistor with the baseline read throughput, where the SD is plugged directly into the host device. Figure 6 visualizes the following read throughput results:

- The read throughput generally increases with larger block sizes (with a cap at >64 KByte block size).
- The read throughput without pull-up resistors is proportional to the baseline throughput.

[7]Open-drain I/Os do not actively set both signal states (low and high voltage levels for digital 0 and 1). In the case of SDs, signal lines are only actively drawn to GND for digital 0. For digital 1, the signal line voltage is not actively set to a high level, but just released to a default state.
The read throughput test shows an opposite behavior. Up to a block size of 32 KByte, the throughput without pull-up resistors is slightly better, but afterward, the write throughput with pull-up resistors becomes much better. This can be explained by the initial thought of signal degradation if no additional pull-up resistors are installed. Without pull-up resistors, the UHS mode at 1.8 V is used. At this low voltage, signal edges become unclean through the long extension cable. Especially at higher block sizes, this degraded signal quality leads to a higher rate of transmission errors. These failures result in the data rate cap at a block size of 32 KByte. On the other side, even though the host does not use the UHS mode if explicit pull-up resistors are added, the signal quality becomes much better, allowing higher write throughput.

In summary, different hardware configurations can take into consideration for different settings. If the DuT does not support the UHS mode at 1.8 V, you should add 3.3 V pull-up resistors to improve overall signal quality. However, if the DuT supports the UHS mode, the main operation mode is decisive for the decision, whether to use or not to use pull-up resistors. If primarily read operations are executed, omit the pull-up resistors to achieve higher read throughput. If primarily write operations are executed, use the pull-up resistors to enforce the 3.3 V mode to achieve higher write throughput.
V. CONCLUSION AND FUTURE WORK

NetSD has been developed to fulfill our mentioned use case with the axle counter. The concept proved to be useful and further, NetSD can aid development practices and test automation of embedded systems in general.

A. Remote access for general purposes

Any embedded device using an SD (or block storage device with a corresponding adapter available) but not providing remote access to that storage device could be extended by NetSD. As explained above, NetSD is not intended to replace existing remote technologies that are integrated into systems. NetSD rather aims to expand existing systems that are not capable of remote storage access, without changing hardware or software of such systems.

B. Storage multiplexing for embedded systems

The introduced switch part of NetSD allowing shared access to the SD could serve as a basis to multiplex a single block storage device between two or more hosts. Multiple bus switches (one set of switches for each device) would allow safe access to the same physical SD for all connected systems. This could be used to accumulate data from multiple hosts or to load the same data (e.g., configurations) to multiple hosts.

C. Fault Injection

The capabilities of the NetSD hardware prototype offer unprecedented prospects for software-controllable Hardware-Implemented Fault Injection (HWIFI) [4]. The forceful insertion (injection) of suspected error causes (faults) is a well-established approach in testing hardware and — increasingly — software. Having control over all data lines independently can be leveraged to test hardware (e.g., SDs, host controllers) and software (e.g., device and file system drivers). Single data lines can be disconnected for short amounts of time to test hardware and software fault tolerance (e.g., error-correcting codes). Our hardware implementation also allows for modification of the signals on the data lines, such as replaying or corrupting data. Experiments can thus not only assess crash fault models, but more complex fault models, including computation, omission and timing faults [5]. It can be suspected, that with more general error classes, more hardware and software misbehavior can be identified.

D. Future work

The original concept of shared access in NetSD implied mutually exclusive access: if one device has access to the SD, the other does not. Originally, this also implied a hard reset to reset the interface settings of the SD after each access switch.

With knowledge about the host device and the use of the same interface settings, this principle could be expanded for enabling parallel access. Instead of switching all lines (incl. the power lines) after switching to the other host device, the system could just switch the data lines without resetting the SD. However, besides the need to have the same interface settings for both host devices, this method can be expected to only work in highly controllable environments. E.g., since commonly used file systems do not support parallel access at block level, this approach carries the risk of data corruption.

Instead of implementing parallel access to the SD on a hardware level, it could be realized in software to circumvent the aforementioned shortcomings. For such a software SD to work, a microcontroller could emulate an SD interface via its GPIO lines. The microcontroller would have exclusive access to the physical SD, so it can safely handle parallel requests from the DuT and the remote access gateway. The challenges with this approach are the implementation of the SD protocol stack and a microcontroller offering IO capabilities fast enough to create the illusion for the hosts to communicate with a physical SD. Since the SD interface is specified to transfer data at up to 208 MHz [3, p. 1] the currently selected ESP32 microcontroller would be too slow for higher frequency modes and therefore not suitable for this approach. As an extension, the physical SD could be replaced with a network share.

ACKNOWLEDGMENT

The authors would like to thank Tobias Zagorni for his active development and testing of NetSD.

REFERENCES

[1] Jossekin Beilharz, Philipp Wiesner, Arne Boockmeyer, Florian Brokhausen, Ilja Behnke, Robert Schmid, Lukas Pirl, and Lauritz Thamsen. Towards a staging environment for the internet of things. In 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), pages 312–315. IEEE, 2021.

[2] Muhammad Abbas, Robbert Jongeling, Claes Lindskog, Eduard Paul Enoiu, Mehrdad Saadatmand, and Daniel Sundmark. Product line adoption in industry: an experience report from the railway domain. In Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A, pages 1–11, 2020.

[3] SD Association. Sd specifications part 1 physical layer simplified specification. version 8.00. https://www.sdcard.org/downloads/pls/pdf/?p=Part1_Physical_Layer_Simplified_Specification_Ver8.00.pdf&f=Part1_Physical_Layer_Simplified_Specification_Ver8.00.pdf&i=EN_SSI_8.2020. Accessed: 2021-05-26.

[4] Pedro Yuste, Juan Carlos Ruiz, Lenin Lemus, and Pedro Gil. Non-intrusive software-implemented fault injection in embedded systems. In Latin-American Symposium on Dependable Computing, pages 23–38. Springer, 2003.
[5] Flaviu Cristian, Houtan Aghili, Ray Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. *Information and Computation*, 118(1):158–179, 1995.