BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 68, No. 4, 2020
DOI: 10.24425/bpasts.2020.134167

SPECIAL SECTION

Implementation of IEC 61850 power protection tester in Linux environment

K. KUREK*, Ł. NOGAL, R. KOWALIK, and M. JANUSZEWSKI

Faculty of Electrical Engineering, Warsaw University of Technology

Abstract. Software power protection tester implemented in a real-time operating system (RTOS) might replace the conventional testing setups in IEC 61850 protection systems. This paper describes an open power protection testing platform. Linux RT capabilities related to runtime environment for such a tester are examined and OS latency sources are identified and evaluated. An algorithm for a multithreaded tester operation is proposed, including Sampled Values (SV) publisher, GOOSE input/output and time synchronization. SV and GOOSE services implemented in RT Linux environment are evaluated in accordance with IEC 61850–5 transfer time requirements. Linux PTP time synchronization service of two similar systems controlling its electrical ports is evaluated in different synchronization scenarios. The developed tester is compared to an equivalent conventional setup during the test of IED over-current function. The conducted tests show that the Linux implementation of power protection tester in the case of scheduler latency, time synchronization accuracy and transfer time all meet the requirements of IEC 61850.

Key words: implementation in real time, power systems, protective relay testing, digital control systems, IEC 61850.

1. Introduction

IEC 61850-based control is becoming increasingly important as the power grid evolves [1]. As protection devices using digital measurement data streams compliant with IEC 61850 standard-9-2 appear, there arises a need to develop new power protection testers or adapt the existing solutions to support SV and GOOSE messages in fast protection and control data exchange. There are numerous commercial products available on the market at the moment. They are built as a stand-alone IEC 61850 testing device or as an extension to existing analog-based conventional testers, allowing for the synthesis of SV data stream that simulates faulty conditions of a power system. Such testers can also be implemented in Linux operating system as an application executed under real time CPU scheduling policy on regular PC hardware. This is possible because SV, GOOSE and time synchronization services (IEEE 1588 v2) are mapped onto an Ethernet protocol, which means measurement signals (current, voltages) as well as binary signals (e.g. Tripping) are meant to be sent through a substation intranet Ethernet network (process and station bus). The time synchronization method recommended in IEC 61850 edition 2 is IEEE 1588 v2 (PTP) protocol specified in IEC 61850-9-3 as the so-called power profile. All new devices available from European vendors support PTP that achieves sub-microsecond synchronization through a common Ethernet link.

The motivation for the work is to build an open and cost-effective platform for a power protection tester based solely on open source software, including Linux and libiec61850 library. This would facilitate testing IEC 61850 compliant devices without having to deploy specialized hardware and resorting to vendor-dependent solutions. Moreover, in contrast to devices available on the market, the proposed tester is open source software licensed under the GPL, which allows for unrestricted development and adaptation to specific testing scenarios.

A few approaches to build IEC 61850 protection tester software are suggested in the literature, but none of them proposed a comprehensive solution. In publication [2] authors use a Merging Unit (MU) simulator running in RTOS, which replays signals from a COMTRADE file, generated earlier in a PSCAD offline simulation. The MU simulator runs on a single machine with multiple Network Interface Controllers (NIC), with neither time synchronization to an external signal nor GOOSE input/output. A similar platform was developed in [3], with a difference that it can also work in an online mode and process real time data output from a simulation program.

2. Related work

The paper describes the implementation of IEC 61850 power protection tester [4] running as a RT application inside Linux OS. In sections 3 and 4, Linux runtime environment and its latency issues are discussed. Linux latency sources are identified and related adjustments are proposed and evaluated. A similar Linux RT scheduling latency test was carried out in [5]. The worst case of latency reported reaches 18 μs and is claimed to be measured in a non-latency optimized system.

A multithreaded operation algorithm for the tester, including the SV publisher, time synchronization and GOOSE input/output, is presented in section 5.

*e-mail: karol.kurek@ien.pw.edu.pl

Manuscript submitted 2019-11-29, revised 2020-03-11, initially accepted for publication 2020-03-16, published in August 2020

Linux PTP time synchronization is discussed and tested in section 6. Its performance is evaluated with two Linux RT systems controlling its electrical ports while being synchronized with a PTP master clock. This ensures the implementation conformity with IEC 61850-5 time synchronization requirements.

IEC 61850 Linux SV and GOOSE services performance evaluation in accordance with IEC 61850-5 is presented in section 7. A similar evaluation was conducted in [6]. It was made as a rough estimate which did not follow the IEC61850-5 guidelines. In this paper, background traffic tests were omitted because of numerous sources already reporting process bus performance under different load conditions [7–9]. Section 7 includes the comparison of a tester's performance when testing real world IED over-current function in combination with a conventional testing setup consisting of the Omicron CMC tester and MU.

3. Runtime environment

The most common way to build an IEC 61850 power protection tester and similar devices is the embedded system implementation, because of a high degree of control over the hardware it provides. Operating systems like Linux offer a level of abstraction in hardware and provide unified, shared access to underlying resources. This simplifies the development, increases the portability and works really well for regular applications that do not have to meet any time constraints. Linux default CPU scheduler distributes time slices evenly between all processes that demand processing power and, additionally, any regular application (user space) can be preempted during its execution by system processes (kernel space) whenever necessary (for example, to handle hardware interrupts from IO subsystem). The problem arises when the user space process needs to execute a periodic task at specific time intervals. In the case of a power protection tester, a publisher thread sends SV packets at 4 kHz rate (according to 61850-9-2, 80 samples per period, data stream for 50 Hz system). Despite the fact that the Linux kernel provides access to high resolution timers with 1 ns precision, there is no guarantee that user space process will acquire CPU at a desired time and will hold it until scheduled tasks are completed. This is where real time operating systems are useful. Such systems guarantee a response time within a given timeframe, usually a couple of microseconds. Linux, by its nature, is not a real time system. However, the *RT_PREEMPT* patch that converts Linux into a RTOS is developed in parallel to the mainline kernel. The main objective of the real time patch is to allow the user programs executed under real time scheduler policies (SCHED_FIFO, SCHED_RR) to preempt kernel space processes and reduce the time in which the kernel executes in a non-interruptible state. This gives the possibility to run applications in real time by acquiring the whole processing time of the CPU whenever needed with minimal latency. Real time applications with sufficient priority cannot be preempted even by the system kernel itself.

Another advantage of using Linux in the implementation of a power protection tester is the out of the box availability of other services like secure shell or web server for remote control, similar to those described in [10].

4. System latency

When implementing time critical services like SV or GOOSE, inability of precise timing can turn the whole implementation useless. The Linux RT_PREEMPT patch guarantees the scheduled processes to get CPU time in a strict timeframe. Ideally, the SV publisher thread should get CPU time once every 250 μs (50 Hz system) in order to publish new samples. When scheduled under real time policy, the SV publisher thread can preempt kernel processes and cannot be preempted by those with lower priority. In practice, latency in which CPU power is assigned to the demanding thread is variable and can be affected by multiple factors, for instance, in hardware service interrupts, the kernel remaining in a non-interruptible state or CPU switching to/from low CPU power state. The quantity that describes the Linux kernel RT capability is worst-case latency. It should not exceed the deadline for the RT application.

Latency sources on x86 PC running Linux and related corrections are presented below and evaluated at the end of this section.

4.1. Logical CPUs Logical CPUs are a feature allowing the operating system to address two logical CPUs for one physical core. For regular applications, this approach can increase the processor throughput by allocation of unused resources. In the RT environment the problem of shared resources is introduced. Two physical threads scheduled by the operating system for different logical CPUs compete to get processing time of one physical unit. The downside of logical CPUs is increased system latency, and therefore it should be disabled in latency demanding RT environments.

4.2. Frequency scaling Frequency scaling allows the processor to conserve energy by temporarily reducing its frequency when it is not fully utilized. This can lead to time-consuming switching, which should be avoided. Frequency scaling can be typically disabled at the hardware configuration level, at the kernel compile time or in user space by changing the scaling governor.

4.3. Real time throttling By default, to prevent system deadlocks, real time processes can be executed for up to 0.95 of CPU time. The remaining time is reserved for regular processes executed under regular default policy. This behavior can be changed to facilitate the exclusive use of the CPU by writing "*−1*" value to */proc/sys/kernel/sched_rt_runtime_us* file located in procfs.

4.4. Core isolation To get the lowest possible latencies it is recommended to designate one of the cores from a multiprocessor system for a time critical task. Three things need to be done to ensure proper core isolation:

- the isolated core should be excluded from CPU scheduling – no new user space processes should be scheduled on the core,
- hardware interrupts should be handled on different cores,
- kernel threads should be kept away from the isolated core,
- time critical process should be manually run on the selected core.

Core isolation from CPU scheduling can be done in a bootloader kernel command line by adding parameter *isolcpus* with the list of CPUs to isolate, e.g. *isolcpus* = *2.3*.

To ensure that a time-critical application gets the lowest possible latencies, hardware interrupts should not be serviced on the core it is running. Affinity of all interrupts in the system to CPU0 can be changed with a single line shell script:

for i in $(*seq 0 255*); *do echo 1 > /proc/irq/$i/smp_affinity; done*.

The script writes value *1* to *smp_affinity* files located in procfs filesystem, corresponding to every possible interrupt request (IRQ) present in the operating system. IRQ affinity can be observed in */proc/interrupts* file, which shows the overall count of all IRQs handled on a specific CPU.

Kernel threads (kthreads, workqueues, timers) can be kept away from isolated cores by setting *CONFIG_CPU_ISOLATION* option at the kernel compilation. Prior to Linux version 4.15 it is not possible to fully isolate the core from the kernel workload.

Running the real time task on a selected CPU can be done with the use of the taskset utility, available in most Linux distributions.

4.5. Local timer interrupt frequency The Linux kernel operation is based around a timer tick. A local timer generates periodic interrupts at a predetermined frequency between 100 – 1000 Hz. Once every 1 ms (for 1000 Hz system), despite the tasks being executed, the CPU gets interrupted, so the kernel can do its housekeeping tasks and the CPU scheduler gets the opportunity to switch the currently running processes. Local timer frequency is set at the kernel compile time and cannot be changed later. Running lower timer frequency can decrease the interruption rate of real time processes scheduled on an isolated core. On the other hand, it can increase latency of processes executed under normal scheduling policies. It is also possible to disable timer tick on all cores except for CPU0 by setting the option *CONFIG_NO_HZ_FULL* at the compile time. The change is at the expense of latency during the kernel to user space context transition, therefore it is beneficial in real time applications not to switch contexts often. In the case of the SV publisher that uses POSIX timer interface, disabling timer interrupts effectively increases the worst-case latency.

4.6. Processor sleep states To conserve power during idle states, modern CPUs switch their C states. States are typically numbered starting from C0 to C6, where C0 means an operating state, and the higher the number, the deeper the sleep state is. Recovery from higher C-states is a time consuming process, so it can negatively impact the system latency. Therefore, it should be disabled by writing *processor.max_cstate* = *0* into the kernel command line at boot time. This will prevent the CPU from entering any state other than C0.

To evaluate the impact of the presented latency sources on the SV publisher thread, a test program was created in C. Its main loop sleeps in 250 μ s cycles using *clock_nanosleep* (*CLOCK_MONOTONIC*, *TIMER_ABS*). As soon as it wakes up, the current time is read and compared to the expected time as a consequence of a loop cycle. The time difference is then calculated as latency and saved to a pre-allocated array. Tests were carried out on the following hardware/software setup:

- AMD 3-core CPU 4 Ghz,
- Linux 4.19.59-rt24, timer frequency – 1000 Hz,
- high resolution timers, TSC time source,
- GCC 5.5.0.

Test results are presented in Table 1. Every test introduces a new change in the configuration (in brackets) as well as previous changes. All tests were conducted in a highly stressed system (*stress-ng* utility, 3 cpu threads, 2 IO and 2 VM threads). Every test consisted of one billion wakeups.

Table 1
Latency test results in different system configurations

Test	Min [μs]	Max [μs]	Avg [μs]	Std dev [μs]
Non RT	1.945	1080	2.928	4.661
RT patch	2.414	33.73	3.672	0.426
RT log, CPU, scal. (1, 2)	2.085	6.73	3.323	0.640
RT throttling (3)	2.022	6.19	2.937	0.527
RT core isol. (4)	2.239	3.477	2.451	0.100
RT 100 Hz (5)	1.801	4.004	2.401	0.076

The first test was meant as a reference – an unpatched non real time kernel. The worst-case latency reached 10 ms, which means the SV publisher thread would get scheduled on CPU 10 ms after the demand. The patching kernel with a RT patch reduced maximum latency to 33.73 μs. Turning off the logical CPU feature and frequency scaling decreased the maximum latency to 6.727 μs. Finally, the core isolation and elimination of RT throttling resulted in 3.477 μs worst-case latency and 2.451 μs average. Running the same test on a kernel with a local timer frequency set at 100 Hz resulted in a slightly lower minimum latency and a decreased standard deviation at the expense of 0.5 μs worst-case latency.

Considering the 80 samples/cycle SV stream, samples need to be sent every 208.3 or 250 μs (60 and 50 Hz system, respectively) and the presented worst-case latency of 4 μs meets the deadline requirements. In the case of latency Linux RT can be successfully considered as a viable environment for the implementation of 9-2LE power protection tester.

5. Tester implementation

The tester structure was implemented as shown in Fig. 1. Assumed functionality of the tester is as follows:

- SV stream generation according to 9-2LE. Input data as a COMTRADE (Common format for Transient Data Exchange) file or a predefined sequence of states,
- GOOSE inputs that allow for closed loop tests of IED,
- GOOSE outputs to simulate a binary data state change,
- time synchronization using PTP in master or slave mode,
- additional time synchronization of the SV data stream to 1PPS signal from an external source.

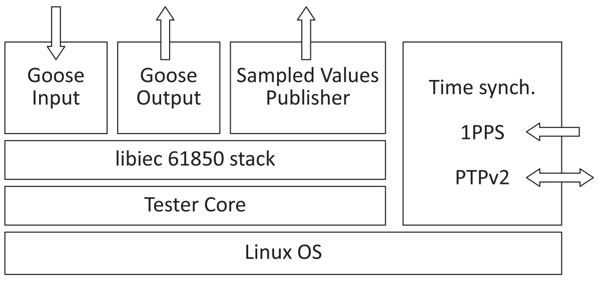

Fig. 1. IEC61850-9-2 software tester structure

SV processing should not be affected by non-deterministic events. Therefore, its thread should have the highest priority. When using a real time scheduler, thread prioritization can be used. Priority values range from 0 to 99, with 0 as the lowest priority and 50 as the priority of the kernel software interrupt handlers. Thread priorities are set in the following order (from the highest): SV publisher, 1PPS, GOOSE Output, GOOSE Input.

There are two types of sampled value streams defined in 9-2LE UCAiUG guidelines. The first one is dedicated for protection purposes (80 samples per nominal period) and another one for applications that require high sampling rates, e.g. power quality (256 samples per nominal period). The implemented tester uses the protection-dedicated stream in which every data unit consists of 16 INT32 variables – 8 channels (4 currents, 4 voltages) plus quality variable for every channel. Values are calculated for primary side and are represented in mA for currents and in 10 mV for voltages.

The main SV publisher thread is based on POSIX timers – the *clock_nanosleep* system function is used. When configured to use TSC (time stamp counter) as time base, 1 ns granularity is achieved. TSC is an internal processor register that counts CPU cycles. The above-mentioned function is used to measure the delay between sending the subsequent samples. It allows for the measurement of time as an absolute value (*TIMER_ABS* parameter) instead of a fixed delay. *Clock_nanosleep* can use different type of clocks and two main options are *CLOCK_REALTIME* and *CLOCK_MONOTONIC*. Both of them can use a TSC time source with a difference. The real time method represents the current time and is a subject of time adjustment by NTP and PTP services, while the monotonic method represents the relative time from some unspecified point of time and is not adjusted during runtime. Timestamps used by the function are held in the *timespec* structure which consists of seconds since the start of the Unix epoch and nanoseconds (for real time method). The *clock_nanosleep* function wakes up the publisher thread every 250 µs and designates the time moments when a sampled value frame is copied into the send buffer. The values of samples to be published are read from a predefined samples array, built from a COMTRADE file or calculated from input data before the fault simulation starts (sequence of states). This approach minimizes the calculations that need to be done during the publisher thread execution. An operation algorithm for tester threads is presented in Fig. 2.

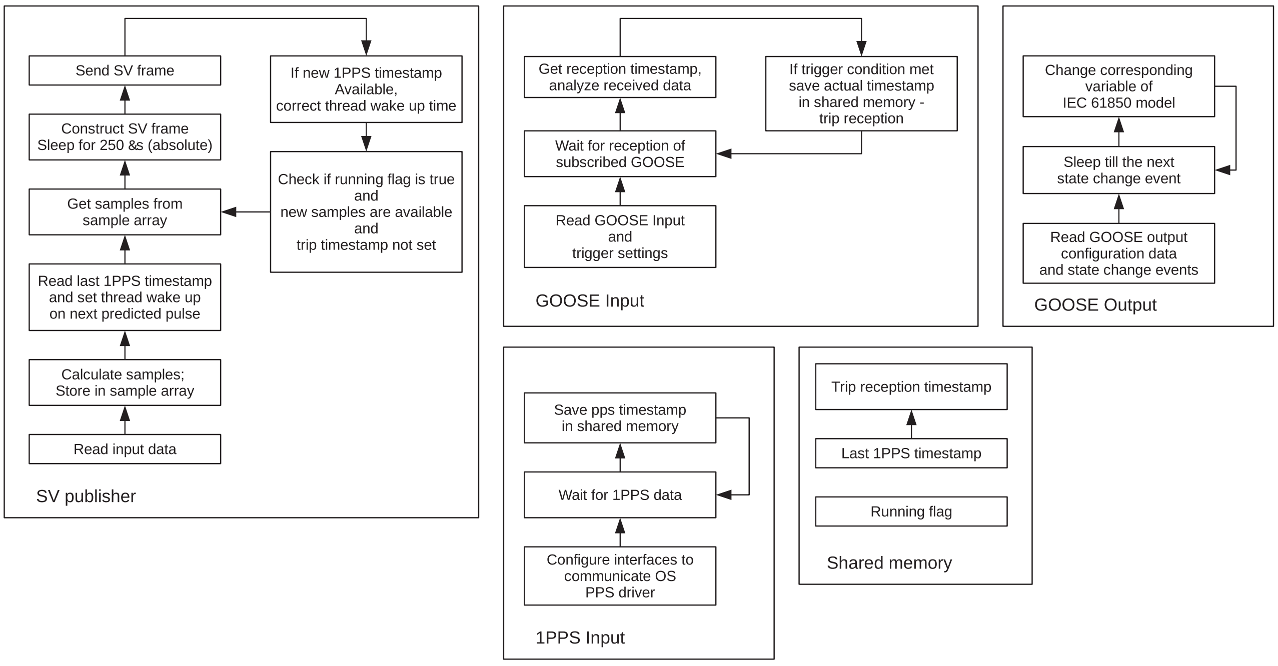
Send SV frame
If new 1PPS timestamp Available, correct thread wake up time
Construct SV frame Sleep for 250 &s (absolute)
Check if running flag is true and new samples are available and trip timestamp not set
Get samples from sample array
Read last 1PPS timestamp and set thread wake up on next predicted pulse
Calculate samples; Store in sample array
Read input data
SV publisher
Get reception timestamp, analyze received data
If trigger condition met save actual timestamp in shared memory - trip reception
Wait for reception of subscribed GOOSE
Read GOOSE Input and trigger settings
GOOSE Input
Change corresponding variable of IEC 61850 model
Sleep till the next state change event
Read GOOSE output configuration data and state change events
GOOSE Output
Save pps timestamp in shared memory
Wait for 1PPS data
Configure interfaces to communicate OS PPS driver
1PPS Input
Trip reception timestamp
Last 1PPS timestamp
Running flag
Shared memory

Fig. 2. Simplified operation algorithm for tester threads

A basic testing procedure consists of the concurrent execution of SV publisher GOOSE input and GOOSE output thread. The SV thread sends samples using a selected generation algorithm (COMTRADE or sequence of states), while the GOOSE input thread is constantly listening to Ethernet interface for GOOSE messages from a specified multicast address and GOCB. When it detects a state change of a variable marked as a trip signal, a trip reception timestamp is saved in the shared memory and a running flag is cleared. The SV publisher is notified of the trip reception and can simulate a post fault state. The result of a simulated test is the tripping time of the tested device. It is calculated as a difference between the reception time of the GOOSE message that carries a trip signal and the start of a simulated fault state.

The GOOSE Output thread can simulate messages from a user-specified Control Block and dataset with desired variable values that change over time. This not only allows the tester to generate changes in measured values but also in binary signals. All the threads are stopped when the SV publisher or GOOSE Input thread sets the running flag to false.

6. Time synchronization

Every device that acts as the SV publisher is required to do the sampling synchronously. It is especially vital during the implementation of differential protection, when time synchronization inconsistencies can lead to tripping in normal conditions. The tester's main time synchronization source are IEEE 1588 v2 (PTP) protocol. PTP synchronization can work directly on an Ethernet network or on the top of UDP protocol (L3). To achieve the lowest latency possible, an Ethernet layer (L2) and a multicast mode should be used (Power profile). The use of PTP for SV synchronization is described in [11]. The expected time synchronization accuracy defined in 61850-5 for protection classes (P2 and P3) is 4 μs (class T4). Class T5 (1 μs) is required for applications where extra precision is needed. However, as shown in [8], an error of 100 μs in one MU sampling process does not affect the operation of the tested differential function. Authors suggest the time synchronization precision of 100 μs should be enough for 80 samples/cycle and 40 μs for 256 samples/cycle SV streams, respectively.

Linux support for PTP is provided by two system services: ptpd and linuxptp. Only the latter supports hardware timestamping and is preferred for power protection applications. Jitter in time offset between the grand master clock and ptpd working in a slave mode, as measured in [6], is 23 μs. To achieve better time synchronization, NIC should support hardware timestamping. In this mode onboard NIC hardware clock is synchronized with the grand master clock. The clock is used to timestamp the incoming and outgoing packets. The system clock can then be synchronized with the NICs clock. Ptpd uses software time stamping and timestamps are generated in the kernel space with much poorer performance [12]. NICs PTP clock support needs to be enabled in the kernel configuration by enabling:

- CONFIG_PTP_1588_CLOCK,
- NETWORK_PHY_TIMESTAMPING

at the kernel compile time.

To evaluate linuxptp performance on IEEE 1588v2 supported NIC (82576 Intel chipset), a laboratory setup was proposed as shown in Fig. 3.

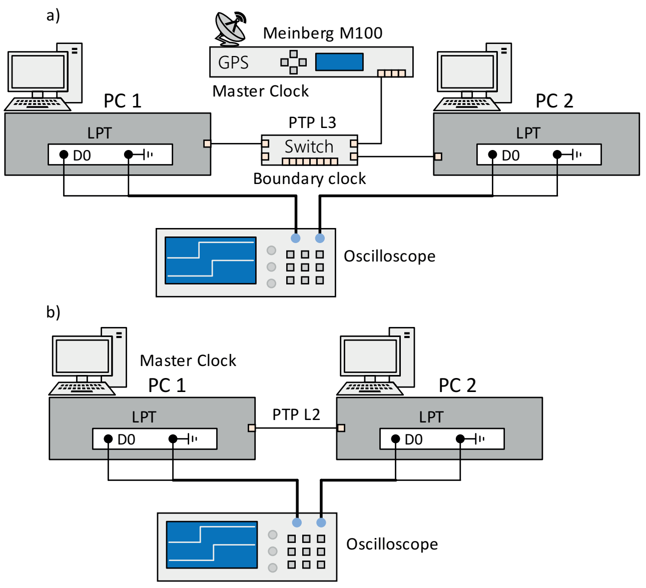

Fig. 3. Laboratory setup to evaluate linuxptp performance

Two identical PCs are synchronized with Meinberg M100 grandmaster clock. Real time test application on both PCs changes the state of the D0 data pin of the parallel port at 250 μs intervals. The test was repeated in two configurations: PCs connected to a boundary clock on top L3 (Edge to Edge) as shown in Fig. 3a) and another reference configuration – direct Ethernet connection, synchronization on top of L2 as shown in Fig. 3b). In the second test, the PCs were synchronized without an external clock, with one of them acting as a master clock. Two signals from D0 pins are observed on the oscilloscope. The signals are floating around zero phase with maximum magnitudes of approximately:

- 1.6 μs – switch, L3 PTP synchronization through a boundary clock (Fig. 4),

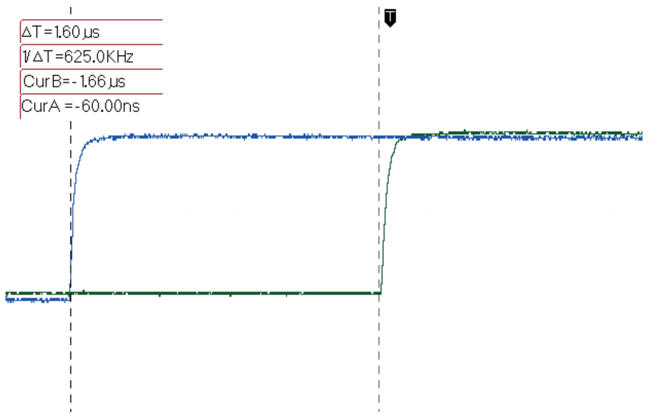

Fig. 4. Delta of D0 signals – switch, L3 synchronization with boundary clock

- 100 ns – direct connection, L2 PTP synchronization (Fig. 5).

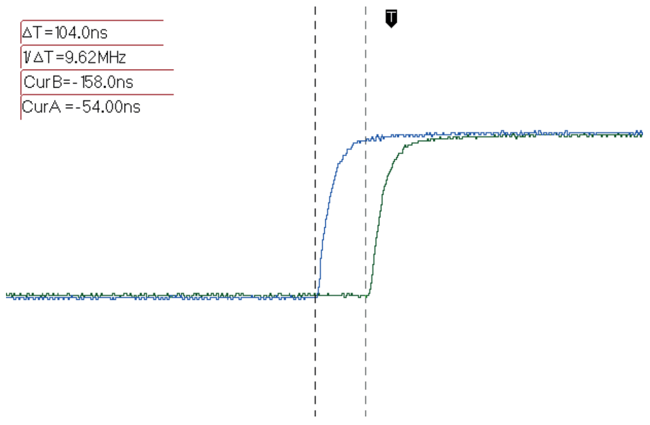

Fig. 5. Delta of D0 signals – direct connection, L2 synchronization

Assuming synchronization errors are equal on both sides, the accuracy for L3 synchronization is below 1 μs, which meets T4 and T5 class requirements. When using L2 IEC 61850-9-3 power profile supporting network architecture, synchronization error can be expected to fall below 100 ns.

It is worth noting that the SV data stream synchronization is based on the *smpCnt* counter. There are no dedicated timestamps included within the SV frame. *SmpCnt* is a part of every SV frame. Its operation is controlled by the *smpMod* attribute (part of the SV Control Block) and by default it counts frames per second. Considering 80 samples per period stream, *smpCnt* changes from 0 to 3999. For a synchronized stream, the zero frame should always correspond to the start of a full second. The synchronization of the SV stream is achieved using the *CLOCK_REALTIME* clock in SV publisher, GOOSE Input and Output threads. Timestamps obtained this way are subject to constant time adjustments from linuxptp daemon.

7. Tester evaluation

IEC 61850 part 5 defines performance classes for different type of messages. Every class specifies a required overall transfer time, in which transfer time is defined as a complete transmission time of a message including its handling at both ends (sender, receiver). It is calculated from the moment the sender copies the data into a send buffer and up to the moment when the receiver extracts the data from its receiver buffer. Coding and decoding data is also accounted for overall transfer time.

GOOSE message that carries trip information is the most important fast message in the substation. Therefore, it should meet the highest performance constraints. The same applies to SV data. Transfer time should be so small that no negative impact on the application function is experienced. Both types of messages are specified in P1 and P7 (equivalent to P1) performance classes, respectively, which should meet the constraints of the highest TT6 transfer time class – less or equal to 3 ms.

In [8] performance of SV and GOOSE was examined with respect to network load and latency. Part 10 of 61850 standard defines network delay as accounting for 20% of overall transfer time and internal processing as accounting for 40% [8]. To keep network delays to a minimum, broadcast traffic should be kept disabled and multicast addresses should be carefully selected to avoid processing the redundant frames [13]. In the field of background traffic affecting operation of a single device, the tests carried out in [8] showed that properly configured 20 MUs on 100 Mbit process bus have a negligible impact on IED performance. The same test conducted when using the same multicast address in every SV stream showed performance degradation of IED at 14 concurrent streams. For a reliable operation, the ability to filter multicast traffic at IED NIC hardware level is required. Without this mechanism, IED should be equipped with at least 1 Gb/s Ethernet ports [7]. Authors of [9] tested the limit capacity of 100 Mb/s process bus, which is 21 concurrent SV streams (80 spp) while still maintaining reasonable performance without data loss. Performance Impact of Parallel Redundancy Protocol (PRP) on the process bus was examined in [14].

In order to confirm the proper operation of the described tester in the context of IEC 61850 transfer time requirements, SV and GOOSE evaluation tests were proposed. The final test proves the correct operation of the system as a whole. The tester was used to test the over-current protection of ABB REL670 relay. The same tests were repeated using a standard OMICRON CMC/SAM600 MU set as a reference.

In the first step of SV and GOOSE services evaluation, the SV/GOOSE subscriber was implemented. Both the publisher (tester) and the subscriber are implemented in PCs running Linux RT and are synchronized to external clock through PTP L3 Boundary clock. For the sake of evaluation, every published data frame contains a timestamp taken just before copying the data to a transmission buffer. The timestamp is read at subscriber's side and compared with the timestamp taken just after the data is passed to its application layer. The time difference is logged to a pre-allocated array and in the end dumped to a file. Because the timestamps are measured in the application layer of both systems, the calculated time difference consists of:

- processing time in system A,
- processing time in system B,
- transmission time through the network,
- time synchronization latency (below 1 μs).

The above mentioned elements correspond to the overall transfer time defined in IEC 61850-5, except for the time synchronization error, which is known. Tests were repeated twice, once for the SV data stream and once for GOOSE messages to prove that SV and GOOSE Linux implementation meets the P1 class for sampling and trip signal requirements. During the SV evaluation, the tester sent frames at 250 μs intervals, the subscriber side processed the frames and calculated the transfer delay and saved the data in its memory. While testing the GOOSE mechanism, the same setup was used, with the addition of changes to the variables of the data model, which resulted in generating a GOOSE message and were introduced once every 500 μs on the subscriber side. While still generating the SV stream, the publisher processed the incoming GOOSE messages and calculated the transfer delay with the same principle as the subscriber did with SV frames. The test results are presented in Table 2.

Table 2
Transfer time of SV and GOOSE Linux implementations

Service	Min [µs]	Max [µs]	Avg [µs]	Std dev [µs]
SV	21.091	396.16	29.745	4.971
GOOSE	26.656	281.36	36.838	5.025

To achieve those results libiec61850 library was patched to support higher resolution timekeeping in its internal functions – with the minimum quantum time of 1 ms, which was changed to 10 µs with the use of the *nanosleep* function. The evaluated publisher – subscriber setup meets the IEC 61850 TT6 transfer time requirements, leaving extra time for additional network delays and extra processing.

The final evaluation of the tester was conducted using a REL670 v2.1 protection relay as a test object. It was equipped with a 61850-9-2 digital input card synchronized to a 1PPS external signal. The over-current function was set as follows:
- *IEC Definite Time* characteristic,
- threshold value of 2 A,
- 0 ms delay.

Tests were carried out in two comparable setups:
- software tester – the tester and REL670 were connected to the ABB ASF675 switch. 1PPS signal from the Arbiter clock synchronized with GPS was supplied to IEC 61859-9-2LE card. The tester PC was synchronized with PTP protocol to Meinberg M1000 clock. A trip signal was output from the IED as a GOOSE message (Fig. 6).
- CMC/SAM600 – OMICRON CMC256+ test set and ABB SAM600 MU were used as the SV source. The SAM600 internal clock was used as a 1PPS synchronization signal for REL670. Because of the lack of binary inputs in SAM600, a hardwired trip signal from the REL670 was directly connected to the CMC binary input (Fig. 7).

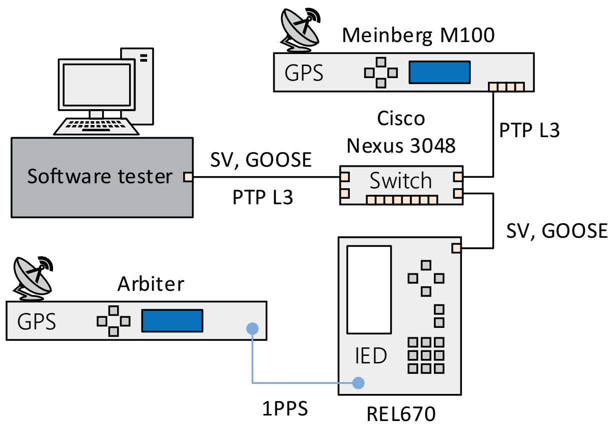

Fig. 6. Laboratory setup for IED evaluation – software tester

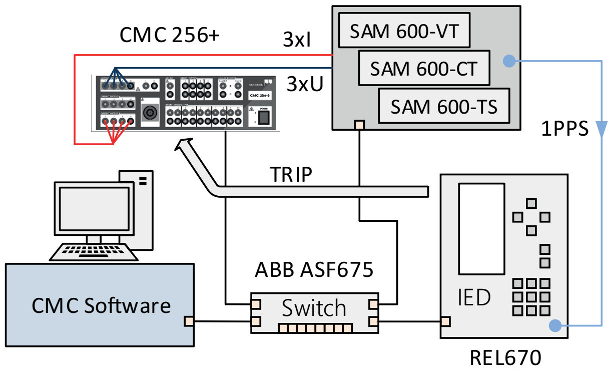

Fig. 7. Laboratory setup for IED evaluation – CMC/MU

The data in Table 3 presents average tripping times (10 shots) for different test currents and test setups. It is worth noting the software tester measured times are approximately 5 ms lower than those measured in the CMC setup. This is the difference between the relay output closing time and the processing time of the GOOSE message.

Table 3
Mean tripping times of over current function

	Software tester			CMC/SAM600		
Test current	2 A	3 A	5 A	2 A	3 A	5 A
Average trip time [ms]	32.02	15.31	7.11	32.56	20.02	15.46

Table 4 shows the results of a threshold value determination test for the over-current function. The test attempts to find the lowest value of the threshold fed from the tester's side that results in tripping of the over-current function. The software tester is approximately 10 times more accurate thanks to its digital and synthetic nature. The CMC/SAM600 setup introduces two levels of Analog/Digital conversions, which in turn results in higher level of errors.

Table 4
Determination of threshold value of over-current function

Software tester		CMC/SAM600	
Test current [A]	Trip [ms]	Test current [A]	Trip [ms]
1.9998	no trip	1.995	no trip
1.9999	30.58	1.998	no trip
2.00	31.92	1.999	35.10
Abs.error	0.005%	Abs.error	0.05%

8. Conclusions

IEC 61850 communication mechanisms allow for the reduction of IED hardwiring. The same applies to power protection testing systems, which can be successfully reduced to a single PC running RTOS. The proposed open testing platform uses Linux and libiec61850, which are available as open source software. This not only allows for reducing the testing system to a single PC without the need for dedicated hardware, but also can be freely changed and adapted to specific needs, which gives the possibility to overcome the constraints imposed by vendors of closed solutions available on the market. The platform was thoroughly tested in terms of latencies, transfer times and time synchronization accuracy. The conducted tests show a guaranteed Linux RT scheduler worst case latency of 4 µs. PTP time synchronization error below 100 ns on L2 and below 1 µs on L3 is sufficient for power protection applications. SV and GOOSE services implemented in Linux RT environment conform to the most constrained IEC 61850 transfer time requirements – class P1 (worst overall transfer time for SV is below 400 µs and below 300 µs for GOOSE). Linuxptp implementation of the PTP protocol meets the T5 time synchronization requirements (error below 1 µs). When comparing the proposed tester platform to the conventional analog testing setups, an increase in accuracy and major reduction of hardware is achieved. Hardware reduction also applies when compared to the digital IEC 61850 testers available.

Conclusions can be drawn that obtained results scale to the subscriber side of SV and GOOSE services implementation. This would allow for the implementation of power automation devices purely inside the Linux operating system and move protection functions and systems into a virtual environment as described in [15].

References

[1] M. Parol, Ł. Rokicki, and R. Parol, "Towards optimal operation control in rural low voltage microgrids", *Bull. Pol. Ac.: Tech.* 67 (4), 799–812 (2019).

[2] M.R.D. Zadeh, T.S. Sidhu, and A. Klimek, "Implementation and Testing of Directional Comparison Bus Protection Based on IEC61850 Process Bus", *IEEE Trans. Power Deliv.* 26 (3), 1530–1537 (2011).

[3] Y. Wu, N. Honeth, L. Nordström, and Z. Shi, "Software MU based IED functional test platform", in *IEEE Power Energy Society General Meeting 2015*, Denver, USA, 2015, pp. 1–5.

[4] K. Kurek, M. Januszewski, and R. Kowalik, "IEC61850 compliant protective device software tester using Sampled Values data stream", *Przegląd Elektrotechniczny* 94, 11–16 (2018).

[5] M.E. Hernandez, G.A. Ramos, M. Lwin, P. Siratarnsophon, and S. Santoso, "Embedded Real-Time Simulation Platform for Power Distribution Systems", *IEEE Access* 6, 6243–6256 (2018).

[6] S. Rinaldi, P. Ferrari, and M. Loda, "Synchronizing low-cst probes for IEC61850 transfer time estimation", in *IEEE International Symposium on Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2016*, Stockholm, Sweden, 2016, pp. 1–6.

[7] A. dos Santos et al., "Characterization of Substation Process Bus Network Delays", *IEEE Trans. Ind. Inf.* 14 (5), 2085–2094 (2018).

[8] D.M.E. Ingram, P. Schaub, R.R. Taylor, and D.A. Campbell, "System-Level Tests of Transformer Differential Protection Using an IEC 61850 Process Bus", *IEEE Trans. Power Deliv.* 29 (3), 1382–1389 (2014).

[9] D.M.E. Ingram, P. Schaub, R.R. Taylor, and D.A. Campbell, ' 'Performance Analysis of IEC 61850 Sampled Value Process Bus Networks", *IEEE Trans. Ind. Inf.* 9 (3), 1445–1454 (2013).

[10] K. Kulikowski, M. Korzeniewski, J. Zakis, M. Jasinski, and A. Malinowski, "Implementation of a Web-based remote control system for qZS DAB application using low-cost ARM platform", *Bull. Pol. Ac.: Tech.* 64 (4), 887–896 (2016).

[11] D.M.E. Ingram, P. Schaub, and D.A. Campbell, "Use of Precision Time Protocol to Synchronize Sampled-Value Process Buses", *IEEE Trans. Instrum. Meas*. 61 (5), 1173–1180 (2012).

[12] P. Orosz and T. Skopko, "Performance Evaluation of a High Precision Software-based Timestamping Solution for Network Monitoring", *Int. J. Adv. Software* 4 (1 & 2), 181–188 (2011).

[13] R. Kuffel, D. Ouellette, and P. Forsyth, "Real time simulation and testing using IEC 61850", in *Modern Electric Power Systems 2010*, Wrocław, Poland, 2010, pp. 1–8.

[14] X. Chen, H. Guo, and P. Crossley, "Interoperability Performance Assessment of Multivendor IEC61850 Process Bus", *IEEE Trans. Power Deliv.* 31 (4), 1934–1944 (2016).

[15] R. Wójtowicz, R. Kowalik, and D.D. Rasolomampionona, "Next Generation of Power System Protection Automation—Virtualization of Protection Systems", *IEEE Trans. Power Deliv.* 33 (4), 2002–2010 (2018).