OPC UA PubSub over TSN

Open Source with a Real-time Operating System

Markus Carlstedt, Yabing Liu and Ting Wang

Wind River Systems

markus.carlstedt@windriver.com, yabing.liu@windriver.com, christina.wang@windriver.com

November 2020

***Abstract* -- This paper presents OPC UA PubSub over Time-Sensitive Networking (TSN) with the VxWorks real-time operating system. We describe TSN features in VxWorks and the integration of open62541, an open source OPC UA project. Further, we evaluate the real-time performance of the system and present test results for time-aware scheduling and IEEE1588 time synchronization.**

I. INTRODUCTION

The advent of *OPC UA PubSub* in 2018 enabled OPC UA as a standard protocol for interoperability between high performance embedded devices in industrial real-time applications. While this market has been dominated by vendor or industry specific fieldbus technologies for decades, OPC UA PubSub in combination with *Time-Sensitive Networking (TSN)* provides cost savings and makes the connection to Industrial Internet of Things (IIoT) possible.

A typical use case for TSN is isochronous data transmission, that is, when packets are sent cyclic and with constant delay. For example, a sensor device in a control network sends new readings to the controller in every control loop cycle. OPC UA PubSub was designed particularly for this type of communication and adds very little protocol overhead. Furthermore, adding TSN allows for both hard real-time and competing best effort traffic to coexist on the same wires.

Typical *Key Performance Indicators (KPIs)* for industrial real-time applications are:

- Time synchronization convergence and stability
- Cycle time
- Packet delay variation (jitter)
- Interrupt latency

In realistic test scenarios KPIs should be measured on heavily loaded systems. A loaded system is one where competition exists for both CPU resources and network bandwidth between the OPC UA PubSub application and other applications on the device.

Previous publications on OPC UA over TSN evaluate performance when running on the Linux operating system [2] [3]. In this paper we describe the test process and present performance results when running on the VxWorks real-time operating system (RTOS). VxWorks, from Wind River Systems, is one of the most commonly used RTOSs. It operates in the Industrial, A&D, Space, Medical, Consumer and Telecom markets with billions of devices deployed over the last 30+ years. Table 1. lists the TSN related standards applied in this paper.

Table 1.

Standard	**Description**
IEEE 1588-2008	Precision Time Protocol
IEEE 802.1AS	Timing and Synchronization for Time-Sensitive Applications

IEEE 802.1Qbv	Enhancements for Scheduled Traffic
IEEE 802.1Qbu	Frame Preemption
IEEE 802.3br	Interspersing Express Traffic

A. OPC UA Implementation

Several stacks were evaluated before selecting an OPC UA implementation for integration with VxWorks TSN. The open62541 project at *https://open62541.org* provides a portable OPC UA stack written in ANSI-C which meets RTOS requirements. These requirements include:

- A non-GPL license that can be used with a commercial software license
- Multi-threading support
- Scalable
- Deterministic
- Secure

The open62541 project is maintained by several academic institutions and commercial enterprises. The source code is licensed under MPL 2.0 (Mozilla Public License Version 2.0). Because it meets the requirements, open62541 was chosen for the VxWorks TSN integration effort.

B. Time Synchronization

Naturally, the prerequisite for TSN is that all devices on the network must keep synchronized time in order to transmit scheduled traffic. This is achieved by using the Precision Time Protocol (PTP) on the network. The PTP protocol, also known as IEEE1588, holds the time in a network device register and uses it to generate receive- and transmit timestamps. The effect of varying queuing delays is therefore minimized, and a synchronization accuracy of under one microsecond can be achieved.

Two open source implementations of PTP with permissive licenses are available with VxWorks:

- PTPd (https://github.com/ptpd/ptpd)
- gPTP (https://github.com/Avnu/gptp)

gPTP implements the 802.1AS standard, which is an adaption profile of IEEE1588 geared towards industrial and automotive TSN applications.

C. Time-Aware Scheduling

The *time-aware scheduler (TAS)* is defined in the IEEE 802.1Qbv standard and was designed to separate Ethernet transmission into repeating time cycles of a fixed length. Each cycle can be divided into time slices, where each time slice is assigned to one or more traffic classes (TCs). During a time slice, 802.1Qbv capable hardware grants exclusive use to the transmission medium for those traffic classes that have been assigned to it. Figure 1 shows an example with two time slices per cycle and one TC per time slice. Typically, the VLAN priority bits are used to map the frame to a particular traffic class.

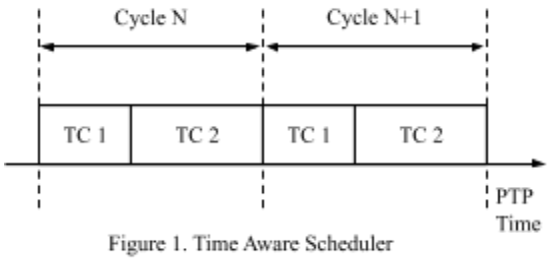

Figure 1. Time Aware Scheduler

Time Specific Departure (TSD) is a variant of TAS where, instead of assigning a traffic class to each frame, the application (or network stack) programs the requested launch time into the packet descriptor. TSN capable hardware will then hold the packet in the transmit queue until the PTP time has reached the launch time before sending it. Figure 2 shows an example where the launch time of the isochronous data is 70 µs into the TSN cycle.

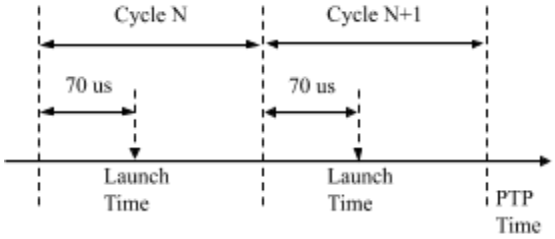

Figure 2. Time Specific Departure

In this paper we use TSD to control time-aware scheduling. There are two reasons for this choice. First, it gives the user exact control over when the OPC UA packet shall be sent. Second, TSD, allows us to evaluate the effect of frame preemption (802.1Qbu/802.3br) on the maximum receive packet jitter.

Previous Ethernet standards do not allow a frame to interrupt an already transmitting frame. This means if competing best-effort packets have begun transmission at the moment the real-time OPC UA packet must be sent, the real-time packet is delayed. Since the typical Maximum Transfer Unit (MTU) on an Ethernet network is 1500 Bytes this delay is up to 12 µs on a 1 Gb/s link. On a 100 Mb/s link the delay is up to 120 µs. To overcome the inherent problem with competing traffic, two new standards were developed, 802.1Qbu (Frame Preemption) and 802.3br (Interspersing Express Traffic). These standards allow preemption of an already transmitting Ethernet frame by a higher priority (express) frame. Figure 3 shows how the preempted frame continues after the express frame has completed transmission. 802.1Qbu and 802.3br are not supported on legacy Ethernet hardware.

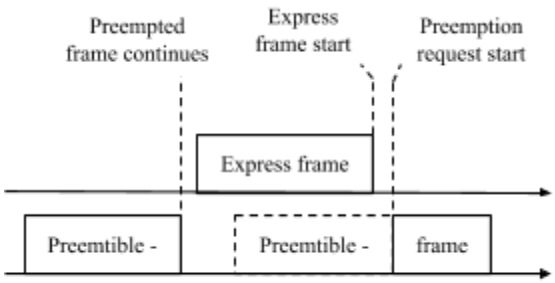

Figure 3. Frame Preemption

D. TSN Implementation

Any operating system must be configured to enable Time-Sensitive Networking features and connect the OPC UA PubSub application to the requested cycle time and transmit schedule. In VxWorks we used the following TSN features to perform the measurements:

- TSN interrupt and affinity
- TSN streams
- SO_X_QBV and SO_X_STACK_IDX

TSN interrupt and affinity

Usually, IEEE1588 hardware can generate either one-shot or periodic interrupts based on the PTP clock. The network devices have their internal clocks synchronised by the PTP protocol, so we can use the clock interrupt to generate a synchronous event triggering isochronous OPC UA PubSub transmission. The example code snippet below shows how to use the VxWorks API to configure a periodic tick timer based on the 1588 VxWorks interrupt. The timer will expire every 250 µs and call the user defined routine *usrTsnClockIsr*.

```
cid = tsnClockIdGet("enetc",0,0);
if (cid != 0)
{
 ret = tsnTimerAllocate (cid);
 if (ret == OK)
 {
  ret = tsnClockConnect (cid,
          usrTsnClockIsr, 0);
  if (ret == OK)
   ret = tsnClockRateSet(cid, 250);
 }
}
```

Note that while open62541 can be adapted to transmit packets directly in the interrupt service routine (ISR), we decided to only use the ISR to signal a semaphore. This mimics a realistic RTOS programming scenario. The semaphore will unblock the OPC UA PubSub publisher task which in turn prepares the OPC UA packet for transmission.

Affinity of the TSN interrupt can be controlled with the VxWorks API *tsnClockIntReroute*. This API selects the CPU core that will process the TSN interrupt. In order to minimize interrupt and task latency, it is recommended to use the same core for the ISR and the OPC UA publisher and isolate this core from other interrupts and tasks.

TSN streams

TSN streams are used to configure TSN scheduling in VxWorks. The network stack is aware of TSN streams and will apply scheduling parameters to outgoing packets as defined by the stream. Additionally, the network stack will apply VLAN tag, PCP bits and MAC address according to the TSN stream. TSN streams can be created by a programming API or by a special tool called *tsnconfig* which takes a JSON formatted file as input. For the tests in this paper we used the following configuration file:

```
{
  "schedule": {
    "cycle time": 250000,
    "start sec": 0,
    "start nsec": 0
  },
  "stream objects": [
    {
      "stream": {
        "name": "flow1",
        "dst mac": "01:00:5E:00:00:01",
        "vid": 3000,
        "pcp": 7,
        "tclass": 7,
        "tx time": {
          "offset": 200000
        }
      }
    }
  ]
}
```

The *schedule* object defines the TSN schedule cycle time in nanoseconds and the schedule start time (0 means immediately). A list of streams is defined by the object *stream_objects*, where each stream object is defined by:

- Name (ascii string)
- Destination MAC address
- VLAN id
- PCP bits
- Traffic class
- Tx time

Tx time defines the offset into the schedule when this TSN stream transmits (launch time). The JSON file supports many other configuration parameters which are not discussed here.

SO_X_QBV and SO_X_STACK_IDX

The OPC UA PubSub over Ethernet implementation in open62541 uses a raw packet socket (AF_PACKET) for transmission. Integration was simple because VxWorks already supports the AF_PACKET socket domain. Connecting open62541 to VxWorks TSN required a single code addition. The socket option SO_X_QBV was set on the OPC UA PubSub socket. In VxWorks, SO_X_QBV connects a socket to a TSN stream which means all data that is sent through the socket will be scheduled according to the stream. Another VxWorks socket option that optimizes latency is *SO_X_STACK_IDX*. This socket option locks a socket to a specific network stack instance which in turn can be affinitized to a specific core. In our tests we assign a dedicated network instance to the same core that processes the TSN interrupt and runs the OPC UA publisher task. The example code below shows how to use SO_X_QBV and SO_X_STACK_IDX:

```
setsockopt (fd, SOL_SOCKET, SO_X_QBV,
"flow1", TSN_STREAMNAMSIZ);

setsockopt(fd, SOL_SOCKET, SO_X_STACK_IDX,
&stkIdx, sizeof(stkIdx))
```

II. TEST SETUP

A. Hardware Setup

Figure 4 shows the hardware setup used in the tests, consisting of two NXP LS1028A reference design boards. The LS1028A is a dual core ARM Cortex-A72 processor system running at 1.3 GHz.

The board carries an Ethernet controller (ENETC) and an Ethernet switch, both of which are TSN capable. In this test we use the ENETC to connect the boards peer to peer at a link speed of 1 Gb/s with a standard Ethernet cable.

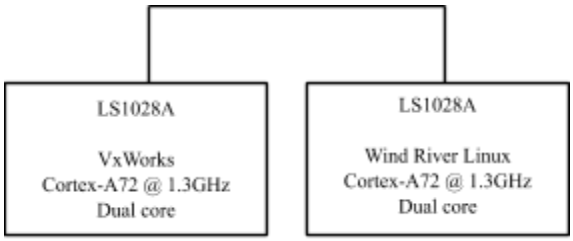

Figure 4. Hardware setup

The first system is the VxWorks device under test (DUT) and runs the OPC UA publisher. The second system is the receiver and runs Wind River Linux. The receiver is any computer that supports PTP and is capable of capturing hardware timestamps of incoming packets. Specifically for PTP testing, we connected two intermediate Cisco IE-4000 industrial Ethernet switches between the two systems.

B. Software setup

The software used in the tests is listed in table 2. It consists of both Wind River proprietary software packages and Open Source packages.

Table 2.

Software package information	
VxWorks	**Version**
VxWorks 7	SR0660
ptpd	v2.3.1
gPTP	v1.0.0
open62541	v1.1.2
iperf3	v3.7
Linux	**Version**
Wind River Linux	LTS 19.45 Update 10
ptp4l	v2.0
iperf3	v3.7

In addition to the software packages provided by the operating systems, we developed a receiver application for Linux called *tsnPerfLxRx*. This application captures all incoming OPC UA PubSub packets with their receive timestamps based on the IEEE1588 clock. The receiver will dump the packet log into a text file which can be used to analyze the performance of the DUT. For this purpose we developed a Python tool called *tsnPerfAnalyzer* which will compute packet delay and jitter for the OPC UA PubSub traffic. The tool will also plot graphs of test results.

III. TEST RESULTS

A. Precision Time Protocol

The aim of initial testing was verification of time synchronization accuracy and stability for the IEEE1588 clock on the VxWorks device as controlled by PTP. Isochronous real-time communication with TSN relies on highly accurate network device clock synchronization. Depending on the type of industrial application, the required synchronization accuracy can be as low as 250 ns [5]. The endpoints in industrial Ethernet networks are usually connected with PTP capable switches or bridged endpoints. Even if the PTP protocol allows for compensation of switch queueing delays, some degradation of synchronization accuracy is expected for each switch hop. In this paper we investigate the effect on synchronization accuracy by adding two intermediate switches. Figure 5 shows the test setup with two switches. We configured the PTP protocol such that the Linux device will become the master and the device under test will become the slave.

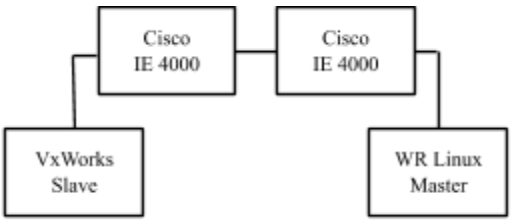

Figure 5. Time synchronization test setup

We also tested the convergence time of clock synchronization, meaning the time it takes for a new network device to reach IEEE1588 clock synchronization to the master device. The convergence time includes both the time to determine the best master clock (BMC) as governed by the 802.1AS protocol and the actual time to synchronize the time by tuning the IEEE1588 clock frequency. Convergence time is an essential factor for some TSN applications as it affects system startup and repair.

Table 3. summarizes the test results for time synchronization. For 80000 samples taken over a 22-hour period, the maximum offset was 29 ns with no switch hops and 37 ns with two switch hops. The offset is sampled by calculating the root mean square value (rms) of the offset of the last eight PTP sync messages received from the master. The convergence time was between 2 and 3 seconds in both cases.

Table 3.

Switch Hops	**PTP Profile**	**Conv-ergence Time (s)**	**Offset from Master (ns)**	
			Max	**Average**
0	802.1AS	2 - 3	29	4
2	802.1AS	2 - 3	37	6

Figure 6 shows a histogram of synchronization accuracy for the two hops case where the bucket size is 10 ns. The first bucket counts samples between 0 and 10 ns, the second between 10 and 20 ns and so forth. It is evident that the majority of samples (>90%) are in the first bucket which means the offset is less than 10 ns. Sampling starts when the time synchronization has converged to avoid skewing the data with initial large offsets.

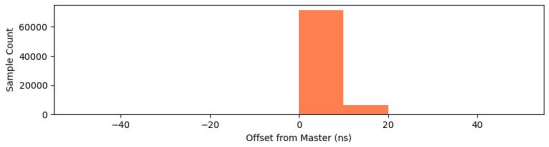

Figure 6. Time synchronization accuracy with two switch hops

Figure 7 shows the convergence time for the two switch hops case. The graph begins after the best master clock algorithm (BMC) has completed and the slave starts to tune the frequency of the IEEE1588 device clock to keep in sync with the master.

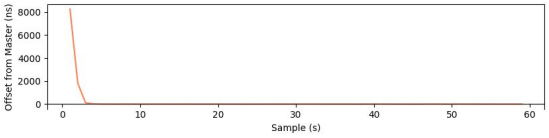

Figure 7. Convergence time with two switch hops

B. Isochronous transmission

The next phase of testing was to evaluate the performance of isochronous packet transmission by OPC UA PubSub. Several factors must be considered when determining the sustainable publishing rate of a particular hardware device and operating system. A few examples are listed below:

- CPU frequency and number of cores
- Interrupt latency
- Context switch latency
- Network stack
- OPC UA stack
- Application code
- Background traffic
- Ethernet transmission time

The packet delay and jitter can be measured at the receiver by taking a hardware timestamp of all incoming packets using the IEEE1588 clock. Since the publisher's clock is synchronized with the master, and OPC UA packets can only be transmitted once

per cycle at the programmed offset, we know that the time difference between successive OPC UA packets must be equal to the cycle time. If the OPC UA packets do not arrive in every cycle the publisher is not able to sustain the requested cycle time. Some jitter will naturally be introduced by factors such as variation in time synchronization accuracy, and background traffic which is competing for the transmission media. The OPC UA traffic can be referred to as critical and the background traffic as best effort. In order to understand the latencies introduced by the operating system, we added instrumentation code in the execution path of the OPC UA publisher application. We sample the IEEE1588 clock at these instrumentation points and embed the timestamps in the OPC UA user data. By this method, the timing characteristics can easily be analyzed in the receiver application and compared with the cycle time. Figure 8 shows three time intervals labeled T1, T2 and T3 used in our tests.

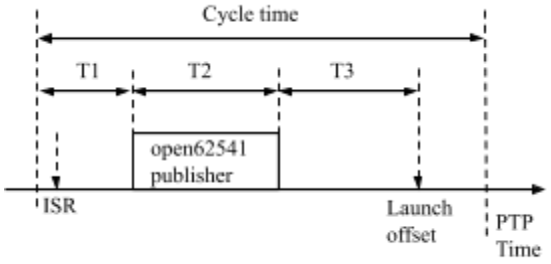

Figure 8. OPC UA publisher timing

T1 is the time from the start of the cycle until the OPC UA PubSub task begins to execute. It consists of interrupt and context switch latency induced by the RTOS. T2 is the runtime of the OPC UA publisher task and consists of the open62541 PubSub logic to setup and transmit the packet. T3 is the time between the end of the OPC UA publisher task and the programmed launch offset in the cycle. T3 can be interpreted as the headroom available for additional user code in the context of the publisher task. Be aware that T3 must also allow for any additional CPU time required for the packet to traverse the network stack and reach the transmission queue in the device before the programmed launch time. In order to minimize latency and optimize the cycle time we made some specific configurations in the VxWorks RTOS as shown in figure 9.

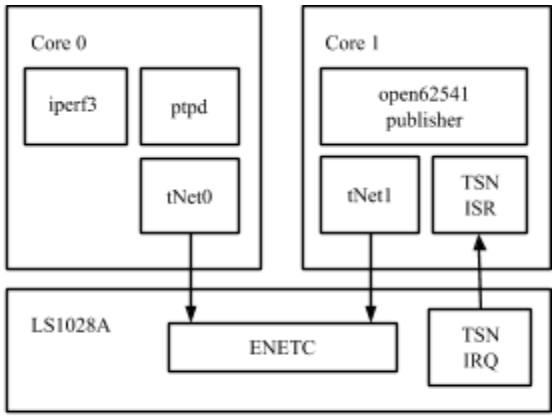

Figure 9. LS1028A core configuration

The LS1028A was configured with core affinity so that the first core handles the PTP protocol and best effort traffic. Best effort traffic was simulated by *iperf3* which is a common tool for network throughput testing. The second core handles OPC UA PubSub and also processes the TSN interrupt. Furthermore, we created two network stack instances, one for each core, which means the OPC UA packets can pass independently and at a higher priority through the network stack. We used different transmission queues in the Ethernet controller for the best effort and OPC UA traffic types where the latter has higher priority. Figure 10 shows the traffic types.

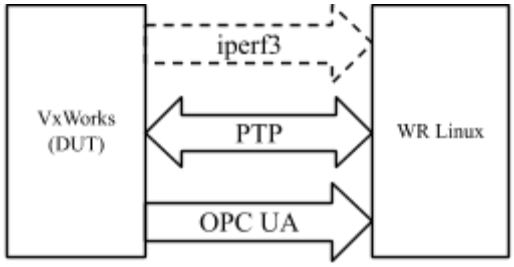

Figure 10. Traffic types

Our testing of isochronous packet transmission by OPC UA PubSub comprised four test cases:

A. OPC UA PubSub without TSN and no best effort traffic

B. OPC UA PubSub with TSN (802.1Qbv) and no best effort traffic
C. OPC UA PubSub without TSN but with simultaneous best effort traffic.
D. OPC UA PubSub with TSN (802.1Qbv) and with simultaneous best effort traffic

In each test case we measured the packet delay and calculated the packet delay variation (jitter). The packet delay is defined as the difference in receive time between two consecutive packets which nominally should be equal to the cycle time. The jitter is defined as the absolute value of the variation in packet delay for each packet. For example, if the cycle time is 250 µs and a packet is received 248 µs after the previous packet, the jitter for that packet is 2 µs. Further, we also verified that all OPC UA PubSub packets transmitted by the sender were received.

All tests were run with a cycle time of 250 µs and a transmission offset for the OPC UA PubSub packet of 200 µs. The packet length was 64 bytes. In case *A* and *C* which do not use TSN, transmission is still driven by the TSN interrupt but the packet is not assigned a launch time. Best effort traffic was generated by *iperf3* which sources TCP data without rate limiting. Therefore, *iperf3* uses all available bandwidth.

For each test we collected 1 million samples. By comparing the packet delay and jitter for test case *A* and *B,* we demonstrate the general effect of TSN on packet delay and jitter. Comparing *A* and *C* shows the additional jitter of critical traffic induced by lower priority traffic without the support of TSN. Comparing case *C* and *D* proves packet delay and receive critical traffic jitter are stable with TSN when lower priority traffic competes for the transmission media. Further, case *D* validates that the system can handle the requested cycle time while all critical packets arrive at the receiver with the expected delay and jitter. Case *D* should be seen as the realistic use case that will be deployed in industrial applications because it introduces competition for both CPU and network resources in the system.

Figure 11a shows packet delay and jitter for one million samples in case *A* which corresponds to about four minutes of runtime at 250 µs cycle time.

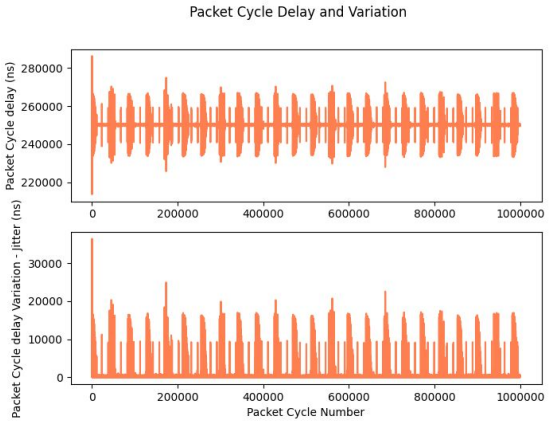

Figure 11a. Packet delay and jitter without TSN

The average jitter for case *A* is 117 ns while the maximum jitter is 36 µs. Figure 11b shows packet delay and jitter for the first 1000 samples in case *B*.

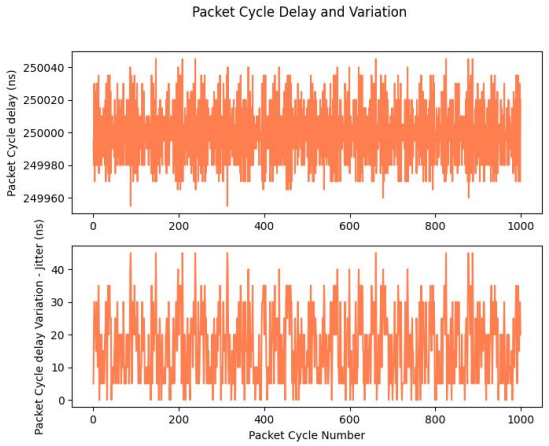

Figure 11b. Packet delay and jitter with TSN (802.1Qbv)

The average jitter for case *B* is 16 ns while the maximum jitter is 45 ns. Comparing *A* and *B* shows significant jitter reduction by the addition of TSN. Extending the test duration to one hour generates similar data.

Figure 11c shows packet delay and jitter for the first 1000 samples in case *C*.

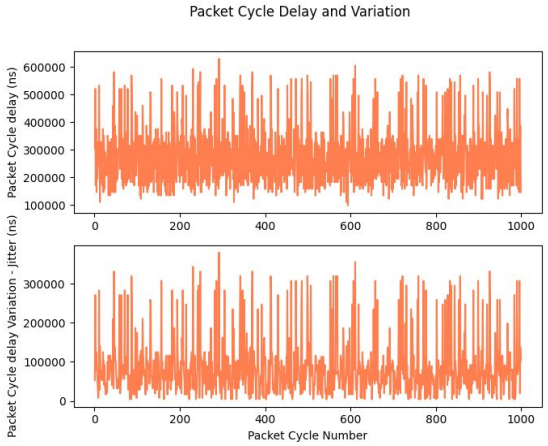

Figure 11c. Packet delay and jitter without TSN but with simultaneous best effort traffic

The average jitter for case *C* is 85 µs while the maximum jitter is 380 µs. In this case the OPC UA PubSub packets are heavily delayed by lower priority packets and fail to transmit in the intended cycle. Extending the test duration to one hour shows even worse jitter.

Figure 11d shows packet delay and jitter for one million samples in case *D*.

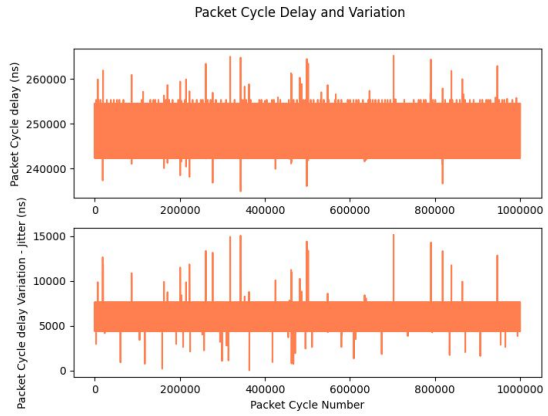

Figure 11d. Packet delay and jitter with TSN (802.1Qbv) and with simultaneous best effort traffic

The average jitter for case *D* is 6 µs while the maximum jitter is 15 µs. These jitter numbers clearly show that the system can handle the 250 µs cycle time. Despite time-aware scheduling lower priority packets can still delay the transmission of the critical packet up to 15 µs. While the average jitter was only 16 ns in case *B* it increased to 6 µs in case *D* due to best effort traffic occupying the transmission media for most critical traffic.

For case *D* we analyzed the timing data of the OPC UA publisher by plotting the T1 and T2 time intervals. Figure 12a and 12b show 3D histograms with 3000 sets of 400 samples which equals 1.2 million samples in total. Each repetition plots logarithmic latency bars for greater visibility. The meaning of T1 and T2 are defined in figure 8.

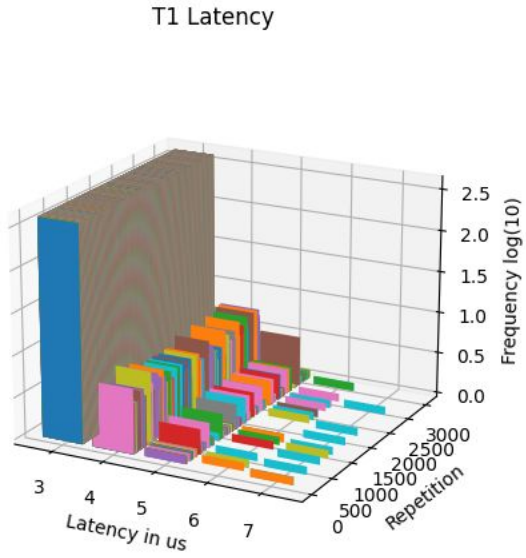

Figure 12a. Latency between cycle start until OPC UA publisher task begins

Figure 12a shows latency for T1, that is, the delay between the start of the cycle until the publisher task begins to execute. The worst case T1 latency is 7 µs which happens eight times out of all samples.

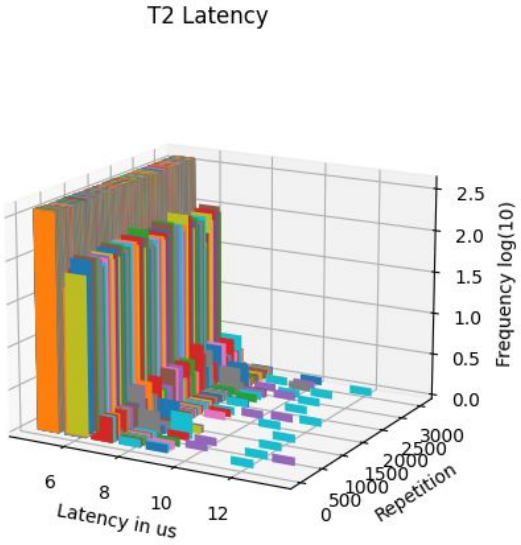

Figure 12b. OPC UA publisher runtime

Figure 12b shows latency for T2, that is, the execution time of the publisher task. The worst case T2 latency is 13 µs which happens once. It should be noted that the measured latencies include some overhead added by reading the IEEE1588 clock which we estimate to at most 10%.

Summing T1 and T2 equals 20 µs which means there are 230 µs left in the 250 µs cycle for additional user application code in the worst case. However, the network stack consumes additional CPU cycles when the packet is sent to the device, which should be taken into account when selecting the transmission offset.

C. *Frame preemption*

The third phase of testing was to investigate the effect of frame preemption (802.1Qbu and 802.3br) in the context of OPC UA PubSub over TSN. It is evident from figure 11d that competing best effort traffic can delay time critical traffic by up to 15 µs, which may be an unacceptable jitter for some industrial applications. In order to enable frame preemption, we added a section in the TSN configuration file as below:

```
{
...
  "preemption": {
    "tclass mask": 1,
  },
...
}
```

The *tclass_mask* key specifies which traffic classes are preemtible and which are express traffic. In our test case the best effort traffic is preemptible while OPC UA packets are express traffic. Figure 13 shows packet delay and jitter for the first 1000 samples when frame preemption is enabled in addition to time-aware scheduling. As in case *D* best effort traffic is also present on the wire.

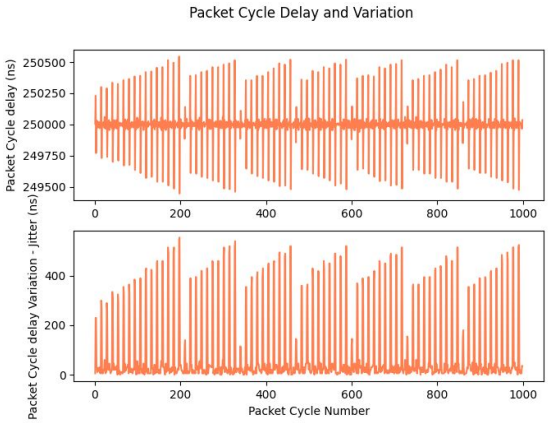

Figure 13. Packet delay and jitter with TSN (802.1Qbv+802.1Qbu) and with simultaneous best effort traffic

The average jitter with preemption enabled is 78 ns while the maximum jitter is 630 ns. Comparing this test case to case *D* clearly shows that the maximum jitter has been reduced by applying frame preemption. Further analysis of test data shows that over 90% of OPC UA PubSub packets preempted a low priority packet. Table 4. summarizes the test results for time-aware scheduling.

Table 4.

Test Case	**Packet Jitter**	
	Max	**Average**
OPC UA PubSub	36 µs	117 ns

OPC UA PubSub +802.1Qbv	45 ns	16 ns
OPC UA PubSub +Best effort	380 µs	85 µs
OPC UA PubSub +Best effort +802.1Qbv	15 µs	6 µs
OPC UA PubSub +Best effort +802.1Qbv +802.1Qbu	630 ns	78 ns

IV. CONCLUSION

First, based on the test results we conclude that OPC UA PubSub over TSN is a viable protocol for industrial real-time communication. Second, we have proven that the VxWorks RTOS provides the required performance, and implements the required TSN standards to build OPC UA publisher applications that meet highly deterministic industrial automation requirements as outlined in [5]. Applying time specific departure to critical traffic can guarantee a maximum receive jitter close to the Ethernet transmission time for the MTU, which is 12 µs on a 1 Gb/s link with 1500 bytes MTU size. We learned the jitter is caused by competing best effort traffic on the same wire. Also, adding 802.1Qbu (frame preemption) can reduce the jitter to less than one microsecond for critical traffic while the remaining bandwidth of the link is fully utilized.

V. FUTURE DIRECTIONS

The implementation and measurements described in this paper cover the real-time properties of an OPC UA publisher device on a TSN network. In addition to isochronous transmission performance, a device must also meet certain performance criteria when acting as a subscriber receiving time scheduled packets. Future experiments should focus on OPC UA PubSub roundtrip measurements, including both a publisher and subscriber.

Future work could also experiment with optimization of the minimal TSN cycle time that can be sustained without packet loss or missed cycles. While our current implementation is interrupt driven it would be possible to use a busy wait loop to drive isochronous transmission in order to reduce interrupt latency. Another optimization that could be considered is to use the lower level VxWorks API *muxSend* to transmit OPC UA PubSub packets. Currently, the implementation uses a network stack raw socket which adds some latency compared to writing frames directly to the driver layer in VxWorks.

OPC UA security is an important factor to consider in any network. At the time of writing, secure OPC UA PubSub is not supported in open62541. When it becomes available, it would be relevant to test the effect that security has on the cycle time and jitter. This is because cryptography is a relatively expensive CPU operation.

VI. REFERENCES

[1] E. Gardiner, Avnu Alliance: “Theory of Operation for TSN-enabled Systems”

[2] Pfrommer, et al: “Open Source OPC UA PubSub over TSN for Realtime Industrial Communication”

[3] Gopiga, et al: “Real-time Open Source Solution for Industrial Communication Using OPC UA PubSub over TSN”

[4] Bruckner, et al: “A new Solution for Industrial Communication”

[5] IEB Media GbR: “Industrial Ethernet Book”, Issue 106 / 15