2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Accelerating Automotive Ethernet validation by leveraging Synopsys Virtualizer with TraceCompass

Ashish Gandhi, Synopsys, Ottawa, Canada (agandhi@synopsys.com)

Praveen Kumar Kondugari, Synopsys, Bengaluru, India (kpkumar@synopsys.com)

Sam Tennent, Synopsys, Livingston, Scotland (stennent@synopsys.com)

***Abstract*— Ethernet enables high-bandwidth and cost-effective data exchange and is therefore critical to cope with the ever-increasing demand for vehicle communications. To also provide the required reliability for in-vehicle communications, Ethernet has been enhanced with sophisticated protocols like Audio Video Bridging (AVB) and Time Sensitive Networking (TSN). This complexity substantially increases the effort to validate and test these software stacks. Automotive Tier-1s and OEMs are looking to Virtual Prototyping for enabling a shift-left of their products' time-to-market with early architecting, validation and reduced-cost regression frameworks. Debugging an Ethernet path on a virtual platform is challenging as there is so far no tool with a holistic view of the traversal of Ethernet transactions across multiple in-vehicle hops. This paper proposes a solution to simplify and enhance the validation and performance analysis of complex Ethernet scenarios for in-vehicle networks, through deep integration of Synopsys' Virtualizer and TraceCompass, an Eclipse-based plugin. This integration provides a comprehensive view across multiple Ethernet nodes, whilst allowing correlation of the Ethernet traffic with other hardware and software events in the individual ECUs.**

Keywords—Automotive ethernet, In-vehice networks, Virtualizer, simulation

I. INTRODUCTION

Automotive Ethernet is becoming the backbone for high bandwidth, safe, secure and reliable in-vehicle communication. New protocols like AVB and TSN provide the required Quality of Service (QoS), security, and safety for using Ethernet in automotive applications like Infotainment, ADAS, Power-Train, Domain Controller, etc. The next-gen Automotive platforms will typically have many Ethernet nodes and employ switch technology to route and manage the data flows.

Many Automotive vendors are employing virtual prototyping techniques to allow software development to progress before hardware is available and to deploy regression systems to manage testing of software variants and updates [1]. Virtual Prototyping tools provide many capabilities to help the debugging and testing of complex embedded Software.

Synopsys Virtualizer [2] is a SystemC-based virtual prototyping solution, which has the capability to trace transactions and events in software and the virtual hardware. Virtualizer's debug capabilities include tracing of registers, ports, software functions, TLM transactions, FastTrack logging, SystemC processes, etc.

However, debugging an Ethernet path in simulation is still challenging as there is no tool with a holistic view of Ethernet transactions traversal across multiple in-vehicle hops.

II. RELATED WORK

Integration with 3rd party tools like pcap [3] can provide even higher-level visibility for ethernet. Virtualizer uses the pcap capture with timestamping at ethernet interfaces in the platform. Tools like Wireshark [4] can help visualize the data for a single node but is limited for multi-node systems. However, in a typical Automotive Ethernet platform, there are many pcap captures which are hard to correlate with each other and with the rest of the system activity.

TraceCompass [5] allows multiple pcap captures to be viewed at once while also providing timing information and statistical analysis of traffic. However, TraceCompass lacks the ability to correlate this information with the hardware and software events in the system that can be captured by Virtualizer.

This paper proposes a solution to extend existing Virtualizer Tracing Analysis by combining with TraceCompass Network Analysis capabilities. This integration will provide a holistic view to analyze the multi-hop ethernet transactions traversal in a platform.

III. INTEGRATING TRACECOMPASS WITH VIRTUALIZER

The TraceCompass plugin provides several Eclipse-based time-synchronous views namely Pcap Trace Viewer, TimeChart, Histogram, State System Explorer. On the other hand, Virtualizer also provides Eclipse-based time-synchronous views of the Hardware and Software activity, e.g. function traces, registers, context switches, messages, etc. The integration of the two worlds ensure that the time updates of TraceCompass views are synchronized with the Virtualizer views. This is done by creating a new Eclipse plugin called CompassTimeSync.

Figure 1 and Figure 2 depict the details of the CompassTimeSync class:

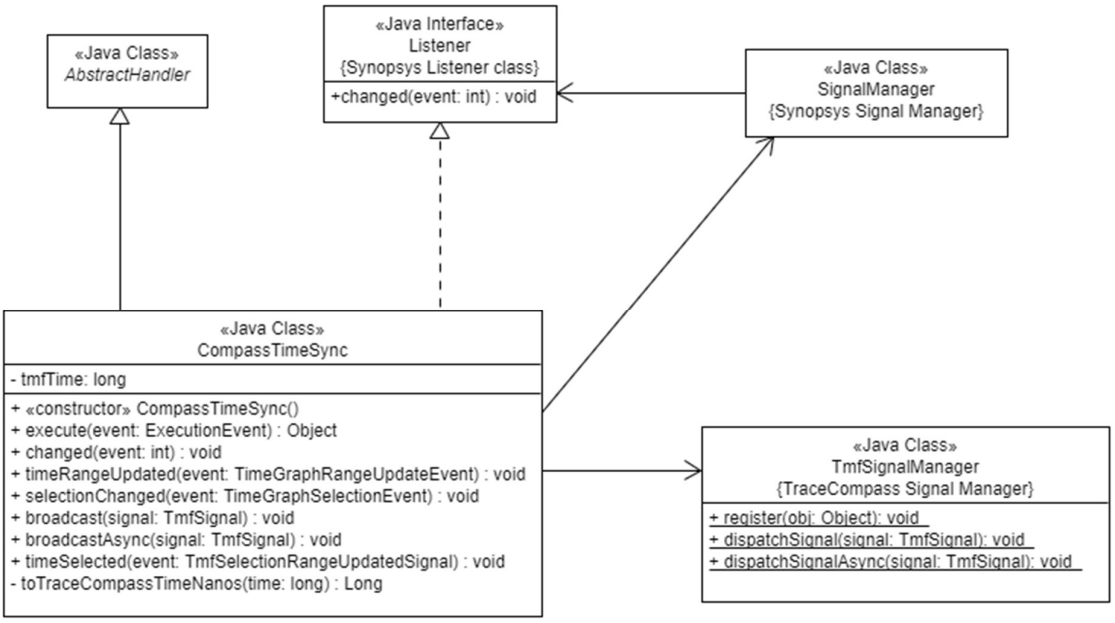

Figure 1: CompassTimeSync Class

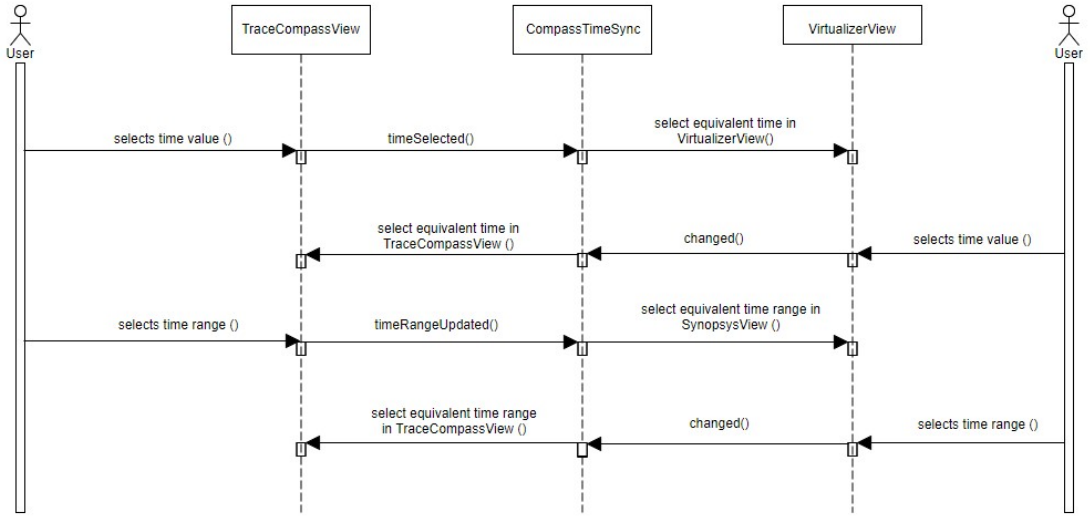

Figure 2: Events flow

This integration enables synchronization of the different views supported by both Virtualizer and TraceCompass Eclipse plugin respectively. This way we scale up the analysis capabilities to cope with the of the complexity Automotive Ethernet protocols.

IV. EXPERIMENTS AND RESULTS

A. An Automotive Ethernet scenario

Figure 3 represents a typical automotive ethernet platform containing multiple ECUs connected via Ethernet Switch. This example uses a Synopsys VDK containing 4x Virtual ECU subsystems with Ethernet Controllers connected using an Ethernet Switch.

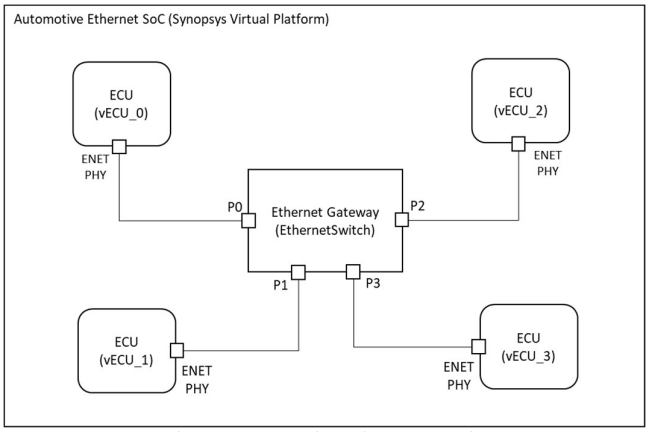

Figure 3: Automotive Ethernet scenario

Figure 4 and Figure 5 show the details of the experiment:

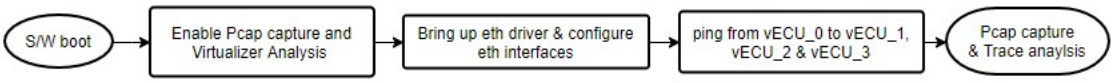

Figure 4: Setup flow

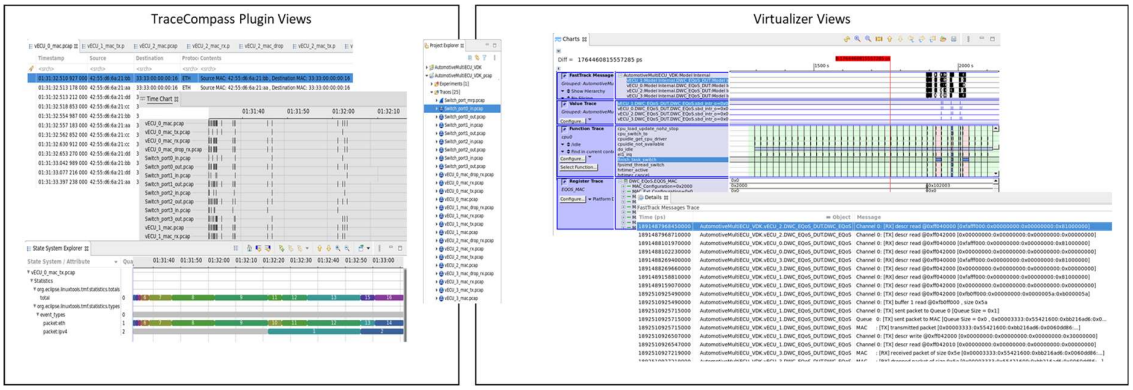

Figure 5: Individual Views from TraceCompass and Virtualizer

Figure 6 provide the following four synchronized views to analyze ethernet transactions:

1. For vECU_0, select a time value in the register trace, where the MAC Transmit Enable bit changes to value “ENABLE” (Virtualizer)
2. The cursor in the Pcap Trace Viewer jumps to selected time and show the frame being transmitted (TraceCompass)
3. Additionally, the State System Explorer (or Time Chart) view also jumps to the selected time and the corresponding stats can be seen (TraceCompass)
4. The Fast Track Messages view jumps to the selected time and the logging from virtual hardware is analyzed (Virtualizer)

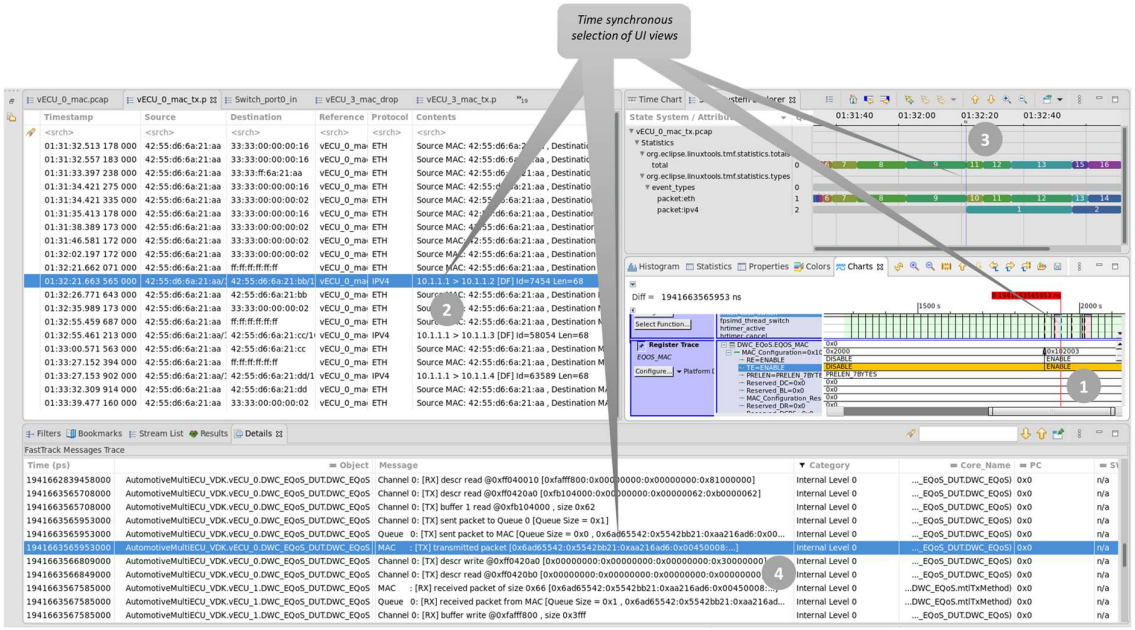

Figure 6: Integrated Analysis views showing time synchronization

In the above experiment, the integration is successfully verified. As a result, the user can correlate any ethernet traffic analysis from TraceCompass with the analysis offered by Virtualizer.

B. Ethernet packet loss analysis in an Automotive platform using AUTOSAR

This is a real example from a production automotive platform with 2 ECUs and an external interface to host ethernet adapter. We observe the packet losses between a VLAN application running under Linux on the host communicating over ethernet with one of the ECU running AUTOSAR [6].

To debug this issue, we use the ping6 application on the host. This way we can replicate the packet loss scenario as shown in Figure 7.

```
agandhi@automotive-platform:~$ ping6 -w4000  -I tap0.14 fd53:abcd:123:b::11 -c20
PING fd53:abcd:123:b::11(fd53:abcd:123:b::11) from fd53:abcd:123:b::999 tap0.14: 56 data bytes
64 bytes from fd53:abcd:123:b::11: icmp_seq=2 ttl=64 time=192 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=3 ttl=64 time=200 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=4 ttl=64 time=57.5 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=5 ttl=64 time=84.8 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=6 ttl=64 time=311 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=7 ttl=64 time=153 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=9 ttl=64 time=683 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=13 ttl=64 time=399 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=16 ttl=64 time=1024 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=19 ttl=64 time=466 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=20 ttl=64 time=781 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=21 ttl=64 time=681 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=22 ttl=64 time=528 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=24 ttl=64 time=366 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=25 ttl=64 time=218 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=26 ttl=64 time=980 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=28 ttl=64 time=647 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=32 ttl=64 time=176 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=34 ttl=64 time=591 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=35 ttl=64 time=68.5 ms

--- fd53:abcd:123:b::11 ping statistics ---
35 packets transmitted, 20 received, 42% packet loss, time 34413ms
rtt min/avg/max/mdev = 57.570/430.846/1024.870/291.081 ms, pipe 2
```

Figure 7: ping6 output showing packet losses

The SystemC TLM model of the Ethernet MAC component has the capability to dump the inbound and outbound ethernet traffic into separate files as well as a superset pcap file with the timing information. Also, Virtualizer Studio allows to dump the AUTOSAR Software Function Trace.

Both the traditional and the proposed approaches of analyzing the issue are explained in the following paragraphs.

B.1. Traditional Approach

Without this proposed solution, the issue was investigated as below:

Step 1: Analyze the dumped pcap files in Wireshark and confirm missing ping replies for some of the request packets.

This indicates the following possible issues:

1. the request/reply packets are dropped by the simulated hardware or
2. the simulated hardware is forwarding request packets to the software with delays (could be related to Scheduling, Latencies, etc) or
3. the software is dropping the request/reply packets or
4. the software is not responding to some of the request packets

Step 2: Analyze the recorded Function Trace from VP Explorer to look at software functions of the AUTOSAR stack as shown in Figure 8 and Figure 9.

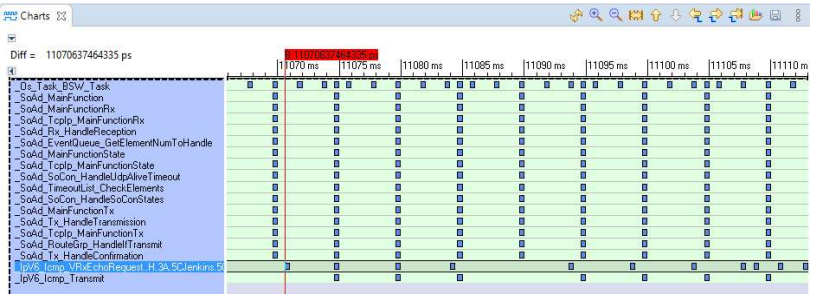

Figure 8: AUTOSAR Software stack Function Trace

Function	Self (%)	TISP	TISCP	Called	Instructions
_SoAd_MainFunction	1.858	2762058	191644332	51	663
_SoAd_EventQueue_GetElemen...	14.862	22096...	22096464	255	5304
_Os_Task_BSW_Task	29.475	43822...	7715840...	0	10519
_IpV6_Icmp_VRxEchoRequest.....	5.786	8602790	29745240	35	2065
_IpV6_Icmp_Transmit	1.289	1916360	232071196	20	460

Figure 9: Details View showing Function calls statistics

This way we can rule out issues 1 and 3 from the list above.

Step 3: Correlate the packets in the pcap file to the function calls based on the simulation time.

Figure 10 indicates the way to look at the time in Wireshark for a specific ping request and then manually zoom and pan to the same value on the Function Trace view inside VP Explorer.

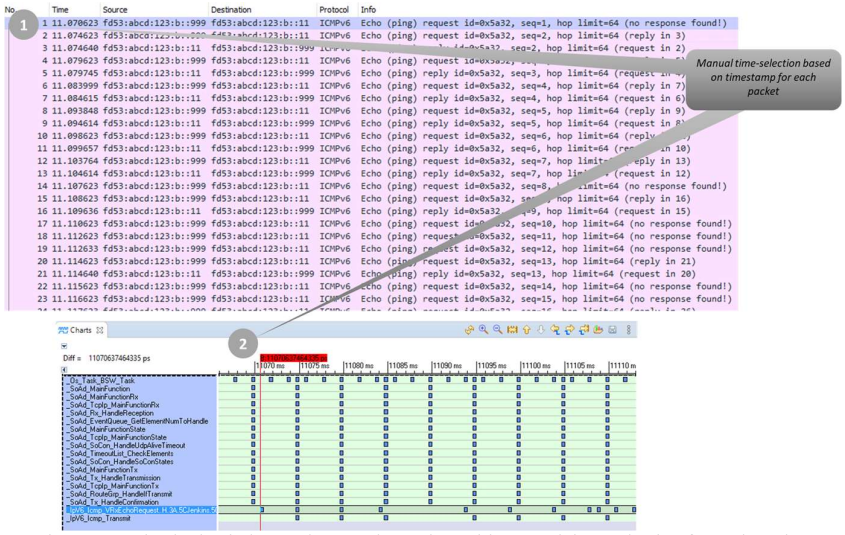

Figure 10: Wireshark window and VPExplorer view with manual time-selection for each packet

It is a tedious and time-consuming effort to manually establish the correlation of 55 packets. After matching up the timestamps for all the packets in the captured pcap file to the same time in the software invocation, the issue 2 is ruled out.

Upon further analysis of the function traces and its correlation with the pcap file, it is observed that there is only 1 ping request packet received within subsequent invocations of the "_SoAd_MainFunction" AUTOSAR runnable.

B.2. Proposed Approach

With the proposed solution, the above steps change as follows:

Step 1: From the TraceCompass Plugin, we use the Statistics window and the Pcap Trace Viewer to check the number of receive and transmit packets. As shown in Figure 11, there are missing ping replies for request packets.

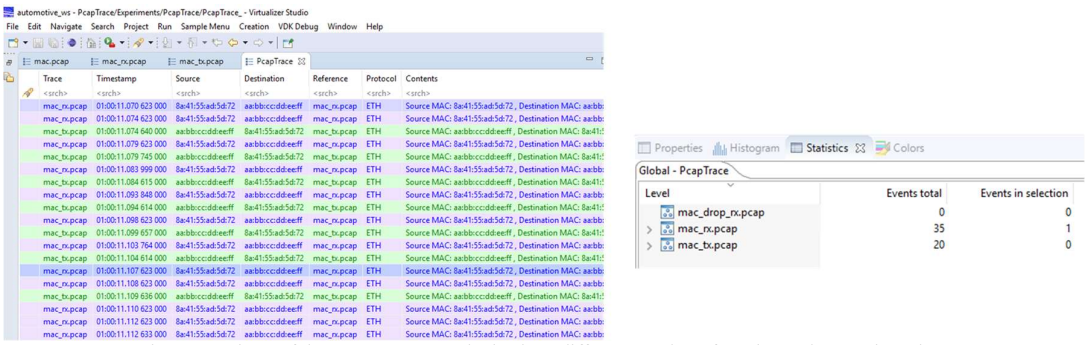

Figure 11: views of the TraceCompass Plugin show different number of receive and transmit packets

Step 2: Same as approach B.1, confirming all the request/reply packets are accounted for in the software stack.

Step 3: Correlate the packets in pcap file to the Function call based on simulation time.
Open Software Function Trace using VPExplorer's Charts view. Select the packets inside the TraceCompass' Pcap Trace Viewer and the VPExplorer's Function Trace auto selects the selected packet's simulation time value as shown in Figure 12 (marked by x).

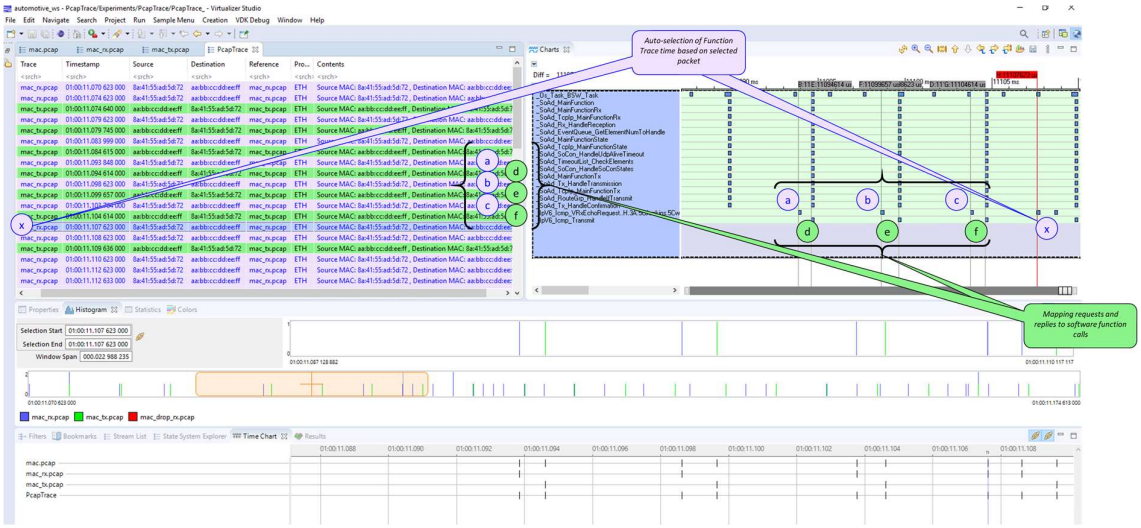

Figure 12: Single UI Window with auto-selection of Function Trace time based on selected based

We confirm that there is no delay from the time when hardware received the packet to the time when software received it. In other words, RxEchoRequest and Icmp_Transmit getting called at the same time as the respective receive and transmit packets.

The issue of missing replies is clearly analyzed in the Figure 12, where the Function Trace shows that there is only 1 ping request packet received within subsequent invocations of the _SoAd_MainFunction AUTOSAR runnable for the successful request-reply pairs (marked by a-d, b-e, c-f) as compared to unsuccessful request (marked by x).

As a fix, we have the following the options:

- Increase the frequency of scheduling of AUTOSAR Ethernet task or
- Slow down the subsequent ping requests at the origin to ensure that only 1 packet is received within the subsequent invocations of the _SoAd_MainFunction AUTOSAR runnable

For this example, we used the second option. It resolves the packet loss as shown in Figure 13.

```
agandhi@automotive-platform:~$ ping6 -w4000 -i5 -I tap0.14 fd53:abcd:123:b::11 -c20
PING fd53:abcd:123:b::11(fd53:abcd:123:b::11) from fd53:abcd:123:b::999 tap0.14: 56 data bytes
64 bytes from fd53:abcd:123:b::11: icmp_seq=1 ttl=64 time=3393 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=2 ttl=64 time=2060 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=3 ttl=64 time=872 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=4 ttl=64 time=212 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=5 ttl=64 time=2356 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=6 ttl=64 time=255 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=7 ttl=64 time=1010 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=8 ttl=64 time=2019 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=9 ttl=64 time=2978 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=10 ttl=64 time=1071 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=11 ttl=64 time=855 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=12 ttl=64 time=1948 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=13 ttl=64 time=1737 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=14 ttl=64 time=415 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=15 ttl=64 time=1791 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=16 ttl=64 time=878 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=17 ttl=64 time=021 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=18 ttl=64 time=1073 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=19 ttl=64 time=965 ms
64 bytes from fd53:abcd:123:b::11: icmp_seq=20 ttl=64 time=4262 ms

--- fd53:abcd:123:b::11 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 95057ms
rtt min/avg/max/mdev = 212.059/1539.087/4262.651/1055.295 ms
```

Figure 13: ping6 output showing no packet losses with additional time delay option

Using the proposed solution, a significant time and effort is saved in Step 3, resulting in a much more efficient analysis.

C. *Software Bug analysis for faster debug*

The stmmac [7] module is a Linux ethernet driver containing support for Designware Ethernet Controllers. It had a bug for the DMA programming sequence, which was causing random DHCP failure on an ARMv8 based virtual platform. Debug required manually comparing the pcap files and the simulation traces of the pass v/s fail scenario. This solution provides mechanism to view multiple pcap traces over the same TimeChart (TraceCompass) and trace back to the software programming sequence by using Register Tracing and Fast Track capabilities (Virtualizer).

D. *Enhanced Scheduling Traffic (EST) visualization for Time Sensitive Networking*

Enhanced Scheduling Traffic (IEEE 802.1Qbv-2015) [8] is one of the critical standards to ensure that the ethernet traffic is forwarded from one ECU to another with a deterministic delay. Typically, an Automotive software developer programs virtual communication channels for the required priority of the traffic. For validating and optimizing the programmed priority traffic reservation in time, experiments performed can be visualized to analyze the configured scheduled traffic windows against the actual transmitted priority traffic.

V. CONCLUSION

This solution extends the current individual capabilities of the TraceCompass eclipse plugin and virtual platform tools like Synopsys Virtualizer by cross-synchronization of their views. This enables the holistic visualization of ethernet transactions in conjunction with hardware and software events across a virtual platform. The experiments covered in this paper provide clear differentiation of the value added by the solution. It accelerates the software debugging and performance analysis of complex Automotive Ethernet platforms. The TraceCompass plugin can be further enhanced to identify various protocols like AVB, TSN, PTP, ARP, ICMP, BOOTP/DHCP, IPv6, etc. and implementing the Raw Data view. This would further accelerate investigating Automotive Ethernet in virtual platforms.

VI. REFERENCES

[1] Wei Liu, "Shifting Left with Virtualizer: End-to-end Virtual Prototype for Automotive Ethernet Switch", Presented at virtual Synopsys Users Group Conference, Silicon Valley, 2020.
[2] Synopsys Virtualizer: www.synopsys.com/verification/virtual-prototyping/virtualizer.html
[3] pcap: Packet CAPture, application programming interface (API) for capturing network traffic, www.tcpdump.org
[4] wireshark: www.wireshark.org
[5] tracecompass: www.eclipse.org/tracecompass
[6] Concept and Implementation of AUTOSAR compliant Automotive Ethernet stack on Infineon Aurix Tricore board: https://monarch.qucosa.de/api/qucosa%3A20582/attachment/ATT-0/
[7] stmmac: https://git.kernel.org/pub/scm/linux/kernel/git/netdev/net-next.git/tree/drivers/net/ethernet/stmicro/stmmac
[8] EST:https://en.wikipedia.org/wiki/Time-Sensitive_Networking#IEEE_802.1Qbv_Enhancements_to_Traffic_Scheduling:_Time-Aware_Shaper_(TAS)