A web based monitoring tool for AFDX networks

Rodrigo Coelho, Mark Szczepanski, Tarek Miari, Gerhard Fohler
Technische Universität Kaiserslautern, Germany
{coelho, szczepanski, miari, fohler}@eit.uni-kl.de

***Abstract*—AFDX is the de-facto network used in avionics systems. The configuration of a realistic AFDX network implementation requires the configuration of more than 100 devices and is error prone. In this paper we present a web based monitoring tool for AFDX networks, which allows for the detection of configuration errors and the visualization of “live” traffic information.**

Our monitoring tool provides a platform agnostic solution for the network visualization and does not require any specialized hardware to connect to the AFDX network. We make use of a network management system (NMS), a computer with COTS hardware, to request and store the SNMP data from the AFDX end-systems. The NMS further allows for users connected to a LAN to access the monitoring tool.

The SNMP traffic in the AFDX network is minimal, transmitted with the lowest frame priority and is independent of the number of users monitoring the AFDX network.

I. Introduction

Most current avionics systems make use of the Avionics Full-DupleX switched ethernet network (AFDX) [1] to replace previous point-to-point networks. Based on ethernet, AFDX addresses the intrinsic non-predictable characteristics of ethernet networks by ensuring bandwidth isolation based on the concept of virtual links (*vl*s). Each *vl* receives a share of the total link bandwidth. A *vl* further defines a logical path: one source end-system[1] (*ES*) and one or more destination ESs. The actual physical route of each *vl* is statically defined at design time.

In this paper we present a web-based AFDX monitoring tool that allows for multiple users to monitor, via web browser: *i)* errors caused during the network configuration, e.g. upload of wrong configuration files, and wrong cable connections during run-time and *ii)* “live” information about the current network traffic.

Our approach does not require any special hardware for collecting the AFDX traffic information, i.e. no need for network tap (test access point), extra end-systems etc. Instead, we make use of one free switch port of the AFDX network to connect the network management station (NMS) whose function is twofold: *i)* collecting AFDX traffic data from the end-systems, *ii)* serving as a web server hosting the monitoring tool and providing access to the monitoring tool to other computers connected to the LAN. Figure 1 depicts the role of the NMS for our monitoring tool.

Our monitoring tool is platform agnostic, providing access to the complete monitoring tool via web browser. The monitoring tool provides:

[1]End-systems are the entities responsible to connect a computing node to the AFDX network.

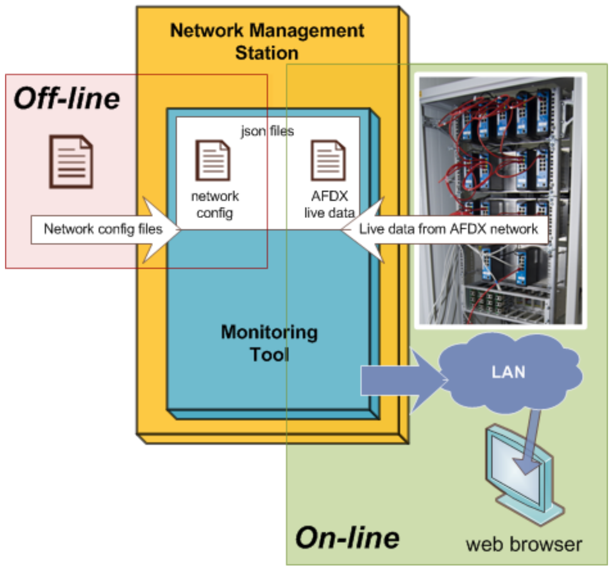


Fig. 1: Monitoring tool architecture.

- identification of disparities between the uploaded configuration files and the actual AFDX network traffic
- identification of possible physical connection errors in the network
- identification of defects on switches and nodes
- visualization of the actual network topology[2], connected nodes and *vl* configuration
- visualization of “live” traffic information

The proposed monitoring tool currently runs on the AVIonics NEtwork Lab of the chair of real-time systems at the university of Kaiserslautern (AVINEL). The network topology of the AVINEL replicates the topology of the commercial aircraft Airbus A380 AFDX network[2], featuring 16 AFDX-compliant switches and capacity for the connection of 85 end-systems[3] (25 currently running). Scientific contributions of the AVINEL include the comparison between practical and theoretical values of end-to-end frame latency and buffer backlog.

Commercial tools providing AFDX monitoring capability exist on the market, e.g. AFDX Monitor (EC comp GmbH), CANoe.AFDX (Vector). Yet, these tools are expensive, some depend on vendor-developed hardware and are focused on timeliness properties of messages. Conversely, our tool does

[2]In the current implementation, our monitoring tool displays the topology used in the AVINEL

[3]The number of connected end-systems in the AVINEL (85) is, in fact, smaller than the one in the Airbus A380 (approximately 120).

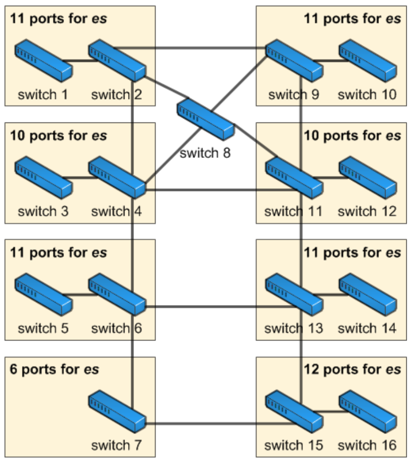


Fig. 2: AVINEL AFDX network topology.

not aim at providing end-to-end latency measurements or dissect message contents. The goal of our monitoring tool is to provide a platform agnostic tool for identification of errors on the network configuration and for providing visualization of the actual network traffic.

The paper continues as follows: In section II we present more details on the AFDX network installed in the AVINEL and used as test case for our monitoring tool. Sections III-A and III-B describe the features and the architecture of our AFDX monitoring tool, respectively. We analyze the impact of the tool in the network traffic in section IV. In section V, we discuss some properties of our monitoring tool in detail, and section VI presents the conclusion and future work.

II. Environment

The AFDX network of the AVINEL has a topology similar to the one used in the Airbus A380 [2]. Figure 2 depicts the AFDX network in AVINEL. In contrast to the 24-port switches used in the A380, in the AVINEL we make use of "TTE-Development Switches 100 Mbit/s"[4][3]. These switches feature eight 100 Mbit/s Ethernet ports supporting three types of traffic: time-triggered (TT), rate-constrained (AFDX) and standard Ethernet (COTS). These switches allow for the configuration of 4096 virtual links and 256 shared BAGs.

In order to simulate the behavior of a switch with more than eight ports, we group pairs of switches to form clusters and then replicate the topology of the A380.

In our network, we replace the AFDX-certified end-systems by the development boards mbed NXP LPC1768 Microcontroller. They feature an ARM Cortex-M3 core running at 96MHz and provide a 100Mbps ethernet interface. These nodes are capable of generating network traffic of multiple virtual links and of implementing the SNMP protocol required by our monitoring tool.

[4]In this setup, we configure the TTEthernet switches to only use the AFDX protocol, i.e. neither time-triggered messages nor TTEthernet synchronization protocol frames are transmitted.

Info type	Traffic	Information
		network overview
error	n.a.	possible switch errors
general	n.a	connected devices (end-systems and switches) and their interconnections
		end-systems
error	All	error during frame TX/ RX
error	AFDX	TX/ RX of an unexpected virtual link
error	n.a.	end-system is not running or disconnected
error	n.a.	end-system rebooted
general	n.a.	end-system up time
general	AFDX	for each expected virtual link: - summary of TX/ RX frames - number of lost frames (TX frames that did not reach their destination) - throughput
general	BE	summary of TX/ RX frames
		virtual links
error	AFDX	wrong path (wrong destination end-system)
error	AFDX	packet loss ratio
error	AFDX	wrong parameter: VLID, BAG, S_{max}
general	n.a.	visualization of physical path (including source and destination end-systems)

TABLE I: AFDX Network information accessible via the monitoring tool.

III. AFDX monitoring tool

This section presents, in detail, the features offered by our monitoring tool and the architecture used to implement it.

A. Features

Our monitoring tool allows for the visualization of AFDX relevant information via web browsers. Consequently, we can provide a platform agnostic visualization tool for AFDX networks. Moreover, our monitoring tool interfaces with the AFDX network via one single AFDX switch port and no special hardware is required for this interface, i.e. the network management station (NMS) connects to the AFDX network through a normal ethernet controller. The NMS makes use of another ethernet controller to allow for the LAN users to access the monitoring tool (see Figure 1). Some additional frames used for message exchange of the SNMP protocol lead to a minimal impact on the AFDX network (see section IV). This impact is indeed independent of the number of users accessing the monitoring tool.

Table I presents a list with all features available in the monitoring tool. We improve the readability of this list by splitting it into three sets of rows: network overview, end-system related, and virtual link related information. The first column classifies the type of available information into: error or general information. The second column depicts which type of traffic the information presented in the third column relates to: AFDX, best-effort(BE) or all (*n.a.* stands for *not applicable*). The third column presents the actual accessible information.

Our monitoring tool allows the user to configure the visualization refresh rate as well as to enable/ disable the display of the status of individual end-systems.

B. Architecture

We divide the analysis of our monitoring tool into two main parts: off-line, and on-line (interchangeably called run-time).

Figure 1 depicts the architecture of our monitoring tool. In the off-line phase, the user uploads the network configuration files to our monitoring tool which stores all relevant data in a set of files (*network config*). During run-time, the monitoring tool collects SNMP data containing “live” information about the network traffic and node status. A set of PHP and java scripts loaded on the user web browser is responsible for the visualization part of the tool (called visualizer). The visualizer then downloads the *json* files containing configuration and “live” data from the NMS and compares the respective data against each other.

These scripts then generate the visualization of general and error information (see section III-A), and additionally store the error information in a log file. In the next sections we present details on the implementation of the back-end and visualization (front-end) tool.

1) Back-end: During the off-line phase, the back-end parses the files uploaded into the network devices and stores the relevant network data (network topology, virtual links properties, MAC addresses for best-effort traffic) into the config files in *json* format. The TTEthernet toolchain generates device configuration files in two formats: binary and xml, and uploads the binary files into the switches. Since parsing binary files is extremely difficult, our tool parses the xml counterparts of those binary files instead.

In order to collect the “live” AFDX traffic information, we make use of the *simple network management protocol* (SNMP). This protocol is an open standard widely used in ethernet networks, implemented on the IP layer, and is supported by the AFDX standard [1]. Compared to other management protocols, e.g., Common Management Information Protocol (CMIP), SNMP requires significantly less resources (section A.2 of [4]). Four elements form the basis of the SNMP protocol [5]: network management station (NMS), management agent, management information base (MIB) and network management protocol. The NMS is the interface for the human network manager into the network management system. Management agents provide communication between NMS and the managed devices as well as access to the information in the MIB. MIB is a collection of information related to the managed device. The network management protocol defines how NMS and management agents communicate with each other.

The SNMP design permits the extension of the MIB, e.g. via private MIB, as well as the extension of management agents. These properties, consequently, allow for SNMP to handle information of specific applications, e.g. AFDX specific traffic information.

The AFDX network in AVINEL has two types of devices: TTEthernet 8-port development switches and the end-systems. Unfortunately, the TTEthernet 8-port development switches do not support the SNMP protocol. Therefore, we concentrate our efforts on collecting the traffic information on the embedded nodes end-systems (for the sake of simplicity, we will refer to them as *end-systems*). After collecting traffic data from end-systems we can infer information about the status of switches.

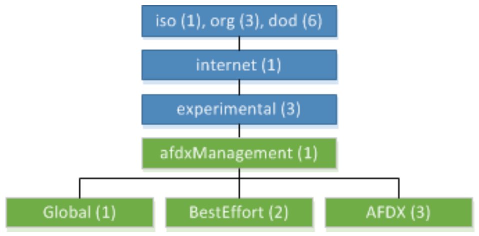


Fig. 3: Entry point of our private MIB.

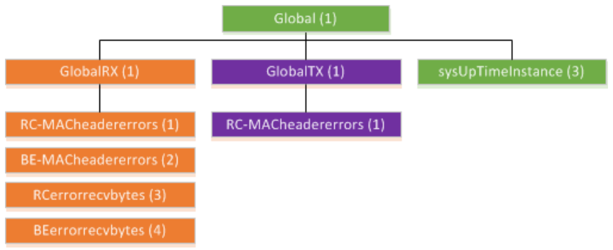


Fig. 4: Global information objects.

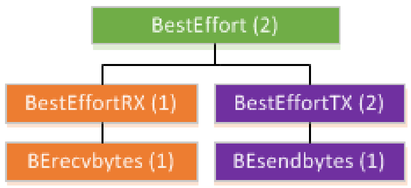


Fig. 5: Best-effort related objects.

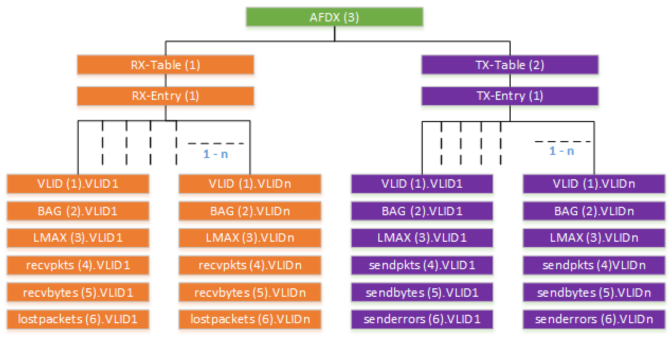


Fig. 6: AFDX related objects.

Applications on the end-systems access the ethernet port using the *lwIP* library [6]. This library is supported by the embedded nodes LPC1768 and implements the TCP/IP stack including the SNMP protocol with support for private MIB. In our approach, we implement the SNMP agents and extend the MIB tree in order for them to collect and store the required monitored traffic information.

Figures 3, 4, 5 and 6 depict the MIB objects used by the monitoring tool: Figure 3 depicts the entry point of our private MIB, Figures 4, 5 and 6 present the elements that store global, best-effort and AFDX information. At the start-up of each end-system (*ES*), we add our private MIB objects to the MIB tree: The *ES* checks the sending and receiving virtual links configured for the end-system and then creates one branch under *AFDX.TX-Table.TX-Entry* and *AFDX.RX-*

Table.RX-Entry for each configured virtual link. These objects store information about the transmitted and received frames of a virtual link, respectively. The information stored in each MIB object presented in figure 6 is self-explanatory. The explanation on how to compute the number of lost frames however is not straight forward: An AFDX *ES* inserts into every transmitted frame an incremental sequence number. Consequently, the monitoring tool checks the sequence number of each incoming frame and detects if some frames do not arrive at their destination. In this case, the SNMP agent counts the number of lost frames of the respective virtual link and stores it in the *lostpackets* MIB object.

Information regarding arrival of frames with unexpected destination MAC address and best-effort traffic is stored in the MIB objects presented in figure 4 and 5, respectively. Under the *BestEffort* branch, the SNMP agents account for the amount of data transmitted and received by all best effort traffic of an end-system. The SNMP agents store the number of frames and the amount of data received with unexpected destination address into the objects under the *GlobalRx* branch.

We add in the *lwIP* library the SNMP agents functions responsible to update our private MIB. For each transmitted or received message, we apply the algorithm presented in algorithm 1 or algorithm 2 respectively.

Algorithm 1: Transmitting frames

```
begin
    foreach frame to send do
        if successful transmission then
            $frame \leftarrow transmitted\_frame$;
            if frame.type == AFDX then
                if frame.VLID == expected VLID then
                    $sendpkts.VLID++$;
                    $sendbytes.VLID+= frame.length$
            else
                $BEsendbytes+= frame.length$;
end
```

The network management station collects the information stored in the private MIB of all selected end-systems. Due to the web based property of our monitoring tool, we make use of PHP library *Net-SNMP*[7] to establish the communication between the NMS and the managed devices (end-systems). We design our monitoring tool such that end-systems do not generate traps to notify changes on their state. Conversely, the monitoring tool periodically polls the end-systems private MIBs content. For this purpose, we use the *Net-SNMP* function *snmpwalk*, which permits to retrieve a private MIB sub-tree every time this function is called.

The monitoring tool refresh rate parameter determines how often two activities take place: The first is the polling interval used by the NMS to send SNMP commands to the selected end-systems and store the retrieved information in the *AFDX live data* json file, as depicted in figure 1. In section IV, we present the impact of the SNMP traffic on the overall network traffic. The second activity, is the refresh rate of the visualization described in section III-B2.

Algorithm 2: Receiving frames

```
begin
    foreach incoming frame do
        frame ← incoming_frame;
        if frame.type == AFDX then
            if frame.VLID == expected VLID then
                $recvpkts.VLID++$;
                $recvbytes.VLID+= frame.length$;
                if frame.seq_no is correct then
                    break;
                else
                    $lostpackets.VLID+= |actual\_seq\_no - expected\_seq\_no|$;
            else
                $RC\text{-}MACheadererror++$;
                $RCerrorrecvbytes+= frame.length$;
        else
            if dest MAC == expected_dest MAC then
                $BErecvbytes+= frame.length$;
            else
                $BE\text{-}MACheadererror++$;
                $BEerrorrecvbytes+= frame.length$;
        Forward frame to upper layer;
end
```

The traffic injected into the AFDX network by the monitoring tool does not increase if multiple users monitor the network status: the visualizer of only one user sends the SNMP requests for the NMS to poll the end-systems (traffic in the AFDX network). All other users access the information in the json files stored in the NMS (LAN traffic).

2) Visualization: The visualization of our monitoring tool is based on the *AJAX* [8] technique. This technique allows for the monitoring tool visualizer (user web browser) to send asynchronous requests to the NMS without interfering with the actual display content, e.g., the user does not need to click anywhere for the visualizer to send requests to the NMS. Once the monitoring tool viewer receives the response from the NMS, the web browser updates the visualization with the changes on the network state, i.e. there is no need to refresh the complete web page. We describe the time behavior of the monitoring tool visualization in the next paragraph.

First, after opening the monitoring tool web page, the visualizer executes a set of scripts that refreshes the visualization of the modified information and requests the NMS to retrieve the private MIB from the end-systems. In a second step, the NMS sends the request for the private MIB transmission to an end-system. The third step consists of the response from end-system with the expected MIB information. In case of no response after a time-out, the NMS polls another end-system.

In the last step, the NMS stores the received MIB into the *AFDX_live_data* json file. The NMS repeats these four steps for each end-system. On the next refresh period, the visualizer downloads *AFDX_live_data* and updates the visualization with the updated information. The NMS allows for only one user to change the configuration parameters: network configuration files, refresh rate and monitored end-systems.

Figures 7, 8a and 8b show screen shots of the monitoring tool main window, detailed information about a switch and an end-system, respectively.

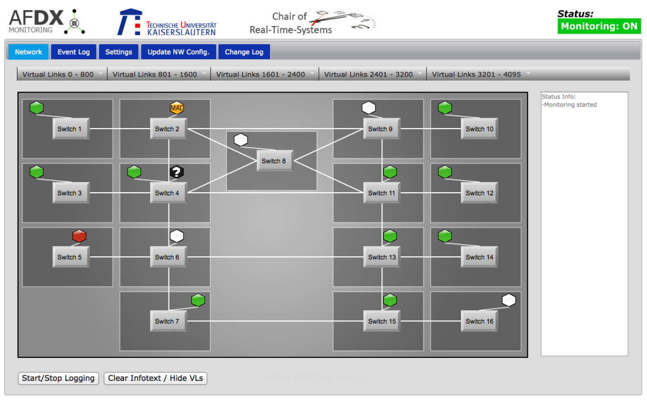


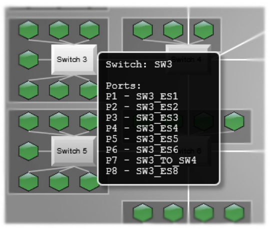
Fig. 7: Monitoring tool main window.

The graphical representation of the end-systems and switches change in the main window according to their states. Table II depicts the graphical representation of end-systems and switches states as well as how the monitoring tool identifies these states.

IV. Performance and Impact on AFDX Network

In our monitoring tool one NMS sequentially polls the traffic information stored in every *ES*. Consequently, the time required to poll the information of all nodes on the network depends on four factors: the number of *ES*s on the network, the amount of data stored in each *ES* (consequently the number of *vl*s on this *ES*), the time taken for the NMS to generate requests and the time taken for the *ES*s to respond to those requests.

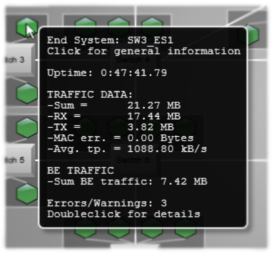
For the AFDX network in AVINEL, in a configuration with 10 *ES*s configured with 5 *vl*s each (380 SNMP objects), it takes approximately 3.5 seconds for the monitoring tool to poll all *ES*s. Our tests can verify that the polling time scales approximately linearly with number of *ES*s and number of *vl*s.



```
Switch: SW3

Ports:
P1 - SW3_ES1
P2 - SW3_ES2
P3 - SW3_ES3
P4 - SW3_ES4
P5 - SW3_ES5
P6 - SW3_ES6
P7 - SW3_TO_SW4
P8 - SW3_ES8
```

(a) Detailed view of switch status



```
End System: SW3_ES1
Click for general information

Uptime: 0:47:41.79

TRAFFIC DATA:
-Sum =      21.27 MB
-RX =       17.44 MB
-TX =        3.82 MB
-MAC err. =  0.00 Bytes
-Avg. tp. = 1088.80 kB/s

BE TRAFFIC
-Sum BE traffic: 7.42 MB

Errors/Warnings: 3
Doubleclick for details
```

(b) Detailed view of end-system status

Fig. 8: Detailed information about network devices.

Visualization	How detected
⬢ no error	no error detected
⟳ reboot	current end-system up time smaller than a previous value
⬢ shut down	current end-system not available
MAC MAC error	frames arrive with unexpected destination MAC address
⬢ $< 2\%$ ⬢ $> 2\%$, $< 10\%$ ⬢ $> 10\%$ virtual link loss ratio	computing the ratio between the number of transmitted frames in the source *ES*, against the number of lost frames in the destination *ES*. This symbol appears on the destination *ES*.
⬢ no frame of some virtual link(s) arrived at the destination	comparing the number of transmitted frames in the source *ES*, against the number of arrived frames in the destination *ES* (the lost frame counter has value zero). This symbol appears on the destination *ES*.
⛔ wrong virtual link parameter(s)	comparing the *vl* parameters (BAG, S_{max}) from live AFDX data against the configuration parameters *ES*.
? error on a virtual link source	if all receiving *ES* of a virtual link present errors, the error origin is possibly the source *ES*.This symbol appears on the source *ES*
▭ no error	all *ES* connected to a switch present low packet loss ratio.
🟥 possible error	all *ES* connected to a switch present high packet loss ratio.

TABLE II: End-system and switches states.

It is important to mention that, since the goal of AVINEL is to analyse the network traffic, end-systems run only low level applications for frame transmission and reception, i.e. no application process the received data. In section V we discuss implementation related reasons for this polling time and present suggestions on how to shorten it.

The impact of the SNMP protocol in the AFDX network traffic is minimal. We measure the bandwidth required, on the NMS link, to retrieve the SNMP objects considering the time interval between the first NMS request and the response from the last *ES*. The bandwidth required is very low: approximately 160 Kbps, i.e. 0.16% of the total bandwidth. Further, this bandwidth requirement does not increase with the number of polled SNMP objects: experiments in which the number of SNMP messages range from 780 to 1980 messages, require approximately the same amount of network bandwidth.

V. Discussion

In contrast to most of the commercial AFDX monitoring tools, our monitoring tool does not provide detailed information on the end-to-end delay or content of individual messages. One of our goals is to provide an AFDX monitoring tool that does not require special hardware. Since most COTS ethernet controllers do not allow for time-stamping precision in sub microsecond range, we believe that precise timing analysis should be deferred to another tool.

Our monitoring tool allows for the comparison of data used in the configuration of the network devices against the actual network traffic collected with the SNMP protocol. One possible strategy to use our monitoring tool is, to upload the configuration files into the network devices and run a set of test applications on the end-systems that make use of all virtual links on that *ES*. In this scenario, our monitoring tool can detect disparities between the uploaded configuration files and the actual AFDX network traffic as well as identify physical connection errors in the network. After this initial test phase, the actual AFDX application is loaded into the end-systems and our monitoring tool can identify errors on switches, nodes and cables (caused after the test phase), errors on the transmitted virtual links as well as display the state of the network nodes.

In the current implementation, our monitoring tool does not display a topology different than the one used in AVINEL. Another limitation of the current version is that only configuration files used by the TTEthernet toolchain are supported. One can address these limitations as follows: Displaying a different topology requires the generation of the HTML content based on the information parsed from the network configuration files; Using the toolchain from another provider requires another parser to collect the relevant information from the configuration files.

As mentioned in section III-B1, the TTEthernet 8-port development switches used in the AVINEL do not support the SNMP protocol, and consequently the state information displayed for each switch is currently inferred from the end-systems information. This limitation on the used switches does not imply a limitation on our monitoring tool. On the contrary, despite the lack of SNMP support on the switches our monitoring tool can still provide some information about the switch state. In case SNMP support becomes available in future versions of the switches, our monitoring tool can provide more precise and detailed information about their states.

In order to keep the impact of our monitoring tool in the AFDX traffic minimal, we transmit all SNMP-related messages as best-effort traffic. BE messages have the lowest priority on a TTEthernet network and thus are only transmitted if no AFDX frame is queued for transmission.

Section IV presented the approximate time to poll 10 end-systems with 5 virtual links each. The largest impact on this time is caused by end-system response to the SNMP request, i.e., the time the node takes to process the NMS request and send the response. Our analysis points to the *lwIP* library as the source of this long response time. We compared the performance of *lwIP* against another library *mbed library* using raw ethernet frames, and we could conclude that *lwIP* is significantly slower. We believe that we can decrease the overall time to poll all end-systems once we run SNMP on another ethernet library. Two other points may also contribute to decrease the overall time to poll all end-systems: *i)* implementation of a multithreaded application to poll *ES* simultaneously and *ii)* using more than one NMS to request and store information from the *ES*.

VI. Conclusion and future work

In this paper we presented a web-based monitoring tool for AFDX networks that allows for multiple users to monitor, via a web browser both "live" information of the current network traffic and detect errors caused during the network configuration phase. We detect errors in the network configuration by comparing the files used to configure the network devices against the actual network traffic. Our monitoring tool does not require any specialized hardware and provides a platform agnostic AFDX visualization tool.

We make use of the AFDX network in the avionics network lab at the chair of real-time systems of the university of Kaiserslautern (AVINEL) to demonstrate the applicability of this tool. We collect "live" network information by means of the SNMP protocol (supported by the AFDX standard). In order for the end-systems to collect this information, we extended the SNMP implementation on the *es* to add support of AFDX traffic information. Then a computer connected to the AFDX and LAN networks serves as network management station (NMS) requesting and storing traffic information from the end-systems. This computer further serves as web server hosting the monitoring tool and providing access to the monitoring tool to other users connected to the LAN.

As explained in section IV, the impact of the monitoring tool in the AFDX traffic is minimal and independent of the number of users monitoring the AFDX network.

Future work includes the performance evaluation of other SNMP compatible ethernet libraries available for the nodes used as end-systems in the AVINEL. Further, we plan to modify the application that polls the end-systems by adding multithreaded capabilities and allow for parallel polling of SNMP data. Additionally, we plan to distribute this tool from our website.

References

[1] "ARINC specification 664 P7-1. Aircraft Data Network Part-7 Avionics Full-Duplex Switched Ethernet Network," September 2009.

[2] J.-B. ITIER, "A380 integrated modular avionics. the history, objectives and challenges of the deployment of ima on a380," http://www.artist-embedded.org/docs/Events/2007/IMA/Slides/ARTIST2_IMA_Itier.pdf.

[3] TTTech, "$^{\mathrm{TTE}}$ Development Switch 100 Mbit/s," http://www.ttagroup.org/ttethernet/doc/TTTech-TTE-Development-Switch-100Mbps.pdf.

[4] E. T. S. Institute, "Satellite earth stations and systems (ses); broadband satellite multimedia (bsm); management functional architecture," http://www.etsi.org/deliver/etsi_ts/102600_102699/102672/01.01.01_60/ts_102672v010101p.pdf.

[5] W. Stallings, *SNMP,SNMPV2,Snmpv3,and RMON 1 and 2*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[6] "lwip a lightweight tcp/ip stack," http://savannah.nongnu.org/projects/lwip/.

[7] "Net-snmp," http://www.net-snmp.org/.

[8] "Asynchronous javascript technology and xml (ajax) with the java platform," http://www.oracle.com/technetwork/articles/java/ajax-135201.html.