ETHERNET FOR SOUNDING ROCKETS

Markus Wittkamp and Rainer Kirchhartz

Deutsches Zentrum für Luft- und Raumfahrt (DLR), Mobile Rocket Base, markus.wittkamp@dlr.de

ABSTRACT

The standard interface for experiment computers for forwarding low to medium data rates to the Service Module on a Sounding Rocket payload is still the asynchronous UART protocol on a RS422 physical layer. This interface is well known, easy to implement and debug. However, it is limited in speed and signal integrity when it comes to high bit rates. Further, the UART protocol cannot be routed without additional layers of communication, since it was designed to serve as a point to point protocol.

A common interface to overcome these problems is Ethernet together with the TCP/IP family of protocols. Some experimenters are already using this stack in their laboratory.

This paper presents a communication system which provides Ethernet on-board a Sounding Rocket to provide the flexibility of this standard system to the experiments. The architecture is based on a pair of gateways to adapt to the existing TM/TC systems without modifying them. The system requires transparent octet-streams on both TM and TC. It replaces the Ethernet frame and forwards data of the IP communication layer.

Due to the nature of IP based communication, the messages can be routed within networks and Firewalls might be used at the Gateways on both ends to protect the TM/TC connections from unwanted packets.

Within the paper we describe the details of the architecture, show an reference implementation of the two Gateways and results of tests during the MAIUS flight.

Key words: Ethernet, Gateway, Sounding Rocket, TM/TC, MAIUS.

1. INTRODUCTION

Modern Payload Systems on Sounding Rockets or Balloons require more flexibility on data distribution on board. Some of these systems also provide several different data streams and not all of them are directed to the ground stations but to other subsystems. These systems might depend on each other, they are able to cooperate and receive information from other systems.

One solution to realise such a scenario is to use the already known and commonly used TCP/IP family of protocols together with Ethernet. Compared to interfaces like UART on RS422, this stack of protocols was designed to handle very large networks, not just point-to-point connections. When it comes to higher data rates, Ethernet also works better in terms of EMC[1] and usability. The system is also more easy to install by using managed or unmanaged switches. Since the system is forming a star-topology, at least parts of the data-paths can be shared between devices and therefore the topology reduces the total number of wires in the harness and provides access-points to the network for monitoring and debugging of the network traffic at the same time.

From a communications engineering point of view, the UART protocol with start- and stop-bits, together with RS422 is operating on the physical layer (OSI Layer 1) [1, Chapter 9.1.4]. A message protocol using frames of octets can be treat as a communication system on the link-layer (OSI Layer 2). If such a protocol includes a destination address, a routing logic can be defined to forward single messages to a destination within a network of computers. This is a property of the network layer (OSI Layer 3).

The IP protocol, specified in [2] was designed as a network layer protocol, whereas Ethernet is providing only layer 1 and 2. TCP and UDP are transport protocols, related to the transport layer of the OSI model (layer 4).

This forms a stack of protocols where each protocol depends on the layer below. Technically a message on a specific layer is embedded in the next layer below, until the whole stack reaches the physical layer. That also means each protocol has to add the specific information for its layer and therefore the size of the message grows with each layer. However, a protocol on a specific layer usually does not need any information of any other layer. This allows to replace protocols with different, compatible ones of the same layer and adapt the stack to fit to different purposes.

To connect an onboard network with the ground stations, the IP data can be forwarded within the already existing

[1] EMC: Electro Magnetic Compatibility

up- and downlink infrastructure. Ethernet itself is not a suitable protocol for this task, since it already defines the physical layer and was not made for RF links. Further, the protocol specifies several services at layer 2 which are not required for forwarding of IP messages to a ground-station.

The subject of this paper is the description of the architecture and the implementation of the link to the existing up- and downlink infrastructure.

2. ARCHITECTURE

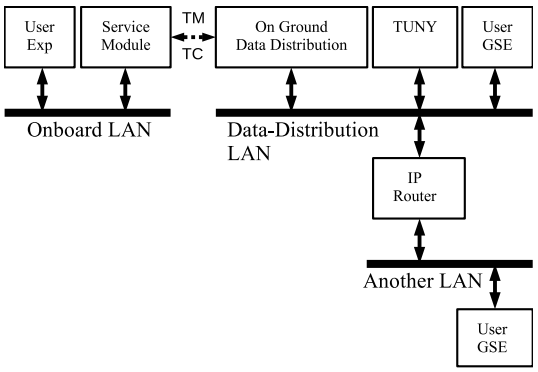

Figure 1. Dataflow overview of the connection between onboard LAN and the ground segment using standard TM/TC links.

For a short overview on the architecture of the proposed solution, Figure 1 is showing a typical scenario for a LAN[2] on a payload. One ore more user-experiments sharing this network and wants to send IP packets to a receiver in a different network on ground. It is assumed that the TM/TC-system within the service module already provides transparent up- and downlinks for streaming of octets.

In order to forward data to ground, the Ethernet protocol has to be replaced by a different and more robust one to make the new system compatible to that octet streaming service. The whole process doesn't modify the IP header nor any other information included in the packet. It basically adds synchronisation words and a protecting CRC[3].

On the ground segment the packets containing the IP data can be extracted and forwarded to their destination via IP routers as figure 1 shows.

The system has to know the length-information of the embedded IP packet for synchronisation purposes, because IP packets are variable in length. To avoid redundancy, it reads this information directly from the IP header and validating it with the IP header checksum. So it actually depends on the IP packet format and IP is not changeable anymore. Any other information in the IP header or the embedded payload within the IP packet doesn't matter for the forward process.

OSI Layer	Ethernet, TCP/IP	*IpOverDynStreams*-Stack
4	TCP, UDP, etc.	
3	IP	
2	Ethernet	*IpOverDynStreams*
1		TM/TC

Table 1. Layer Stack of TCP/IP together with Ethernet and IpOverDynStreams

This architecture results in two gateways, one on each end of the TM/TC-system. The gateways basically convert the packets from Ethernet to the internal format which was called "IP over Dynamic Streams" or *IpOverDynStreams* and back. To be more precise, the overall system is forming a layer 3 router, composed of two protocol-converters for the octet-streams on layer 2 and the actual routing logic. Table 1 shows an overview of the OSI layers to compare the TCP/IP stack with the one of the *IpOverDynStreams*.

Each of these gateways might be equipped with additional packet filtering firewalls. Such a firewall is checking both the IP- and the layer above, usually TCP or UDP, against a mission specific set of rules to forward or reject packets. This can be used to protect the up- and downlinks from unwanted packets, e.g. due to failures on a client firmware.

Due to the fact that the IP protocol itself doesn't need a response, the system can also be used in a one-way configuration, if one of the up- or downlink systems fails or simply doesn't exist by design.

3. IMPLEMENTATION

As stated in the architecture description above, the system is a composition of several sub-components: the *IpOverDynStreams* protocol on layer 2 to protect the IP packet and a gateway located in the service module close to the TM/TC system and another one in the ground segment.

The reference implementation has been done on a MFC2 computer board [3] which provides a Blackfin 561 processor together with a MAC chip for Ethernet as well as the interfaces for up- and downlink. An instance of the LwIP-stack [4] is running as a thread on the RODOS kernel [5] on the processor. This open source network stack is used to control the Ethernet port as well as the gateway to the telemetry. It is also responsible for all kind of communication on layer 3 (IP) and 4, like TCP and UDP (see also Table 1). To the LwIP stack the gateway behaves as any other network interface, since it uses the same software interface. The layers below the IP layer are performed using hardware driver. For Ethernet the MFC2

[2]Local Area Network
[3]Cyclic Redundancy Check

provides a dedicated Ethernet MAC controller. The hardware for sending and receiving octet-streams is implemented as FPGA-logic[4].

As for Ethernet, a MTU[5] size is used to determine maximum buffer sizes and optimise the link-layer protocol on the up- and downlink.

3.1. IP over Dynamic Streams

Name	Offset	Length	Description
Sync	0	4	Const.: `0xfafbfcfd`
x/IP	4	...	start of IP packet
IP-tl	4	2	total length of IP
...	...	...	
State	*see text*	2	reserved for future use
CRC	*see text*	2	CCITT 16 polynomial

Table 2. Definition of the IpOverDynStreams-Format. The grey marked area shows the embedded IP packet.

The format of the layer 2 protocol to forward and protect IP data on RF-links of the telemetry and telecommand links is shown in table 2. As usual for this kind of protocols, all fields are transmitted in big-endian format, the network order of bytes.

Synchronisation To find the start of a *IpOverDynStreams* message, a four-byte synchronisation constant is leading the octets of the message.

Message Length The problem with IP messages is that the actual length is not known in advance. To overcome this, the "Total Length"-field of the IP packet is used directly. For IP version 4, this field is located at offset zero, relative to the IP packet, which is offset 4 in the *IpOverDynStreams* packet, because of the 4 byte synchronisation constant. The total length of the IP packet is the only information needed to forward it on layer 2.

State and CRC With the length of the message, the position of the State and the CRC field can be calculated immediately: It is the total length of the IP packet plus the size of the leading and trailing fields of the additional headers. Therefore the total length of a message is the size of the IP packet plus 8 octets.

The State-field is reserved for future use. It should be set to zero.

To avoid unnoticed errors during the transport process, a CRC 16 is appended to the message. The CCITT polynomial $x^{16}+x^{12}+x^5+1$ is used to calculate this checksum. It will be initialised with 0xffff.

[4]FPGA: Field Programmable Gate Array
[5]MTU: Maximum Transfer Unit

3.2. Service Module Gateway

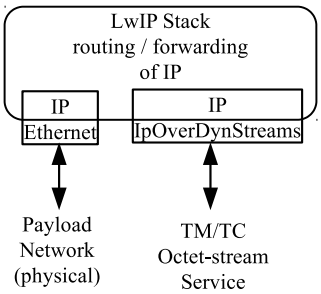

Figure 2. Routing and Forwarding Process within the LwIP stack.

LwIP Stack The on board side of the system is based on the already mentioned LwIP stack. This stack provides a rich set of functions to an application level program. Beside of this, it can handle more than one network interface and is able to forward traffic between these interfaces.

The LwIP stack itself is not subject of this paper, it is well documented in [4]. But the network interface to the TM/TC system in figure 2 uses the functions of that stack. It communicates with it by an internal software-interface for reading and writing of IP data. Since the stack was written in C, this communication is basically a passing of pointers to data-structures within the memory of the processor without copying of data (zero-copy policy).

To forward data in the scenario of this paper, it needs at least two interfaces, one to Ethernet and another one to the TM/TC system. Both of them need a full set of data: an IP Address, a Gateway address, the network mask, MAC Address, etc. One of the interfaces becomes the default gateway to the LwIP stack. Figure 2 is showing the process. It can be assumed that the default gateway points to the TM/TC system, because most other networks will be on ground instead of on payloads.

The routing logic of the stack will select a specific network interface either if it points directly to the destination network of the IP packet or it was defined to be the default gateway. This is a common way to forward packets. In this scenario the gateway will get all packets that cannot be forwarded to directly connected onboard networks.

All network interfaces controlled by the stack have to implement the same (software-) interface. Where other interfaces implementing drivers to control Ethernet chips, the interface to the TM/TC system was specially designed to connect the internal interface of the stack to the octet-streaming service. In case of the reference implementation, the streaming service consists of a set of registers and FIFOs[6] to exchange data directly with the TM/TC system.

[6]FIFO: First in, First out buffer

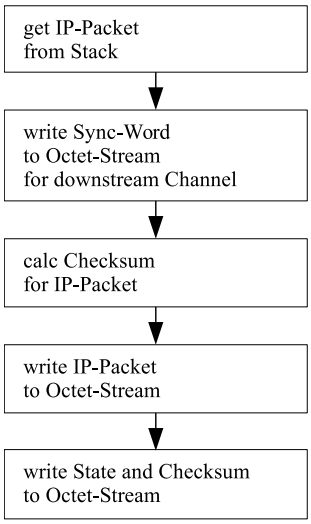

Figure 3. Flowchart of the transmitting process.

TX process Figure 3 is showing a simplified flowchart of the process of sending data to the TM/TC system, which is called the downlink direction. The stack is calling a *write* function of the gateway driver and passing a pointer to an IP packet.

The first step is to send the constant synchronisation word to the octet-stream of the TM/TC system. The next step is to calculate the CRC of the IP packet part of the message (see table 2) and the state information and write both out to the octet-stream. Finally the result of the CRC is written to the stream.

RX process Since the driver doesn't know in advance how many octets are to read for the current packet, the process is split into two parts, as Figure 4 shows in a simplified flowchart.

The first part of the process is waiting for the synchronisation words and then read the IP header of the packet. A check of the header is performed with the checksum field of the header, to ensure the integrity of the length information. If this fails, the driver will drop the already received octets and start a new read process.

With a verified IP header, the length information is known and the number of bytes left to read can be computed. The system is then changing its state, starting another read process to get the rest of the packet and start waiting for the next receive-interrupt to proceed. This interrupt is generated as soon as the requested number of octets are received in the hardware-buffer in the TM/TC system. They are not yet available to the processor.

The second part of the read process is starting with reading the rest of the IP packet from the buffer of the TM/TC system, including the tailing fields of the *IpOverDynStreams* message. When the full packet has been read, the CRC is checked and compared with a locally computed one. If this fails, the driver drops the packet and will restart the process.

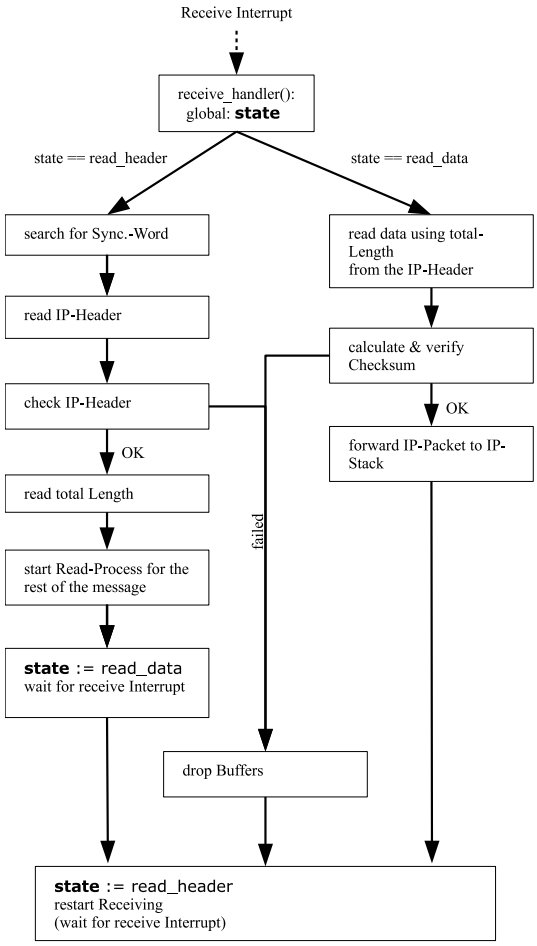

Figure 4. Flowchart of the receiving process.

If the CRCs are equal, a pointer to the IP packet is passed to the LwIP stack for further processing. The additional fields of the *IpOverDynStreams* are not needed anymore.

The receive process will restart immediately with a fresh buffer to get the next packet.

3.3. TUNY

On ground, the gateway was implemented as a Linux program, it was called *TUNY*. The Linux kernel provides an interface to its routing mechanism either on IP level (tun) or directly on Ethernet (tap) [6]. A program running with root-privileges is able to open such a driver and communicate on the selected layer. The driver becomes available as a virtual network device.

Since the gateway is acting on layer 3, a *tun*-type interface has been chosen to extract and inject IP packets to the kernel. An overview is given in figure 5. On the other

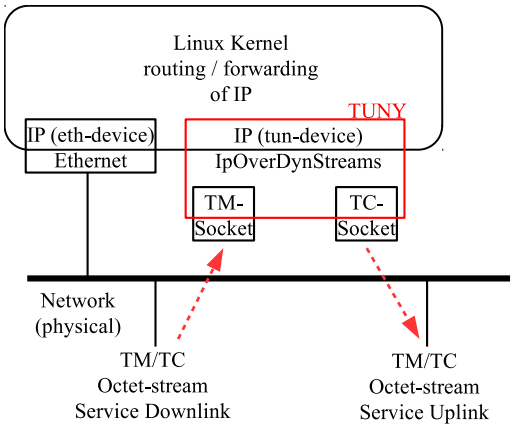

Figure 5. Routing and Forwarding Process of TUNY.

end of the communication, the program is reading and writing the octet-stream from and to the TM/TC data distribution in the ground segment using standard application level sockets.

Interface to the Linux Kernel The virtual interface *tun* allows the kernel to pass IP packets to *TUNY*. It behaves like drivers for Ethernet-cards or other devices but on layer 3 only. Therefore it needs the full set of information: IP address, network mask and gateway settings. [6] provides a documentation of the driver.

TUNY registers such a device on the kernel and provides network settings for it. The kernel is using the device from that point on until *TUNY* is closing the driver for exit.

Interface to the TM/TC system *TUNY* is designed to connect to the octet-streams for up- and downlink by UDP/IP. That implies the TM/TC-system on ground is already providing such an interface, as the reference implementation does. Figure 5 shows that connectors as sockets at application level, one for each direction of dataflow. The dataflow itself is marked with dashed lines.

This architecture of *TUNY* allows most flexible network designs in the ground segment. The network can even be distributed over several physically separated segments as long as a route exists and is known to forward the packets.

TX, Uplink process The uplink-process on ground is quite similar to the downlink process on the payload and analogously to the simplified flowchart in figure 3. *TUNY* is reading an IP packet from the *tun* device, adds the fields of the *IpOverDynStreams* packet including CRC and send it as an octet-stream to the uplink-socket. The socket forwards the octet-stream to an instance of e.g. a telecommand-encoder.

RX, Downlink process As the uplink process, also the downlink is working similar as the receiving process on board (Figure 4). The receiving UDP socket on application level (Figure 5) is reading the incoming packets byte-by-byte and searching for the synchronisation constant of the *IpOverDynStreams* header.

The next step is to read and verify the IP header to get the total-length of the packet. With the knowledge of that length, the rest of the packet can be read and the CRC can be checked.

If the packet was received without errors, the fields of the *IpOverDynStreams* are removed and the IP packet is passed to the *tun* device to be forwarded towards its destination within the ground segment by the Linux kernel. In case of receiving errors, the process is dropping the buffer and continue searching for the next packet header.

4. RESULTS AND MEASURES

4.1. Functional Test: ICMP

A first test of all IP network related implementations is the use of the ICMP protocol [7]. The popular command *ping* sends so called *ECHO Requests* to a receiver which answers with *ECHO Reply*. Further, the request frequency of the command can be adjusted and the program provides a switch to append an optional payload of almost any size. With this tool it is possible to generate arbitrary load on a network.

The response time for the command in this scenario is mainly affected by the speed and the load of the uplink. For the test the speed was set to 230.4 kBit/s for a shared uplink of 16 transparent octet-streams simultaneously. In result the response time was about 18 ms to several 100 ms, depending on the load of the link.

4.2. Performance during Flight of MAIUS

The main experiment on MAIUS was connected to a local network on board the payload. The implementation presented in this paper has been used during flight to forward the data to the ground segment and transport telecommands the other way.

Figure 6 shows the load of the octet-stream over flight-time. This curve is generated by the telemetry statistic module of the telemetry generator. In this case, the statistic is generated by counting bytes over a full second and writing the result to a housekeeping message. The message is generated synchronously to the measure. Depending on the operational state, the experiment computer is generating burst of different types, length and contents. As the figure shows, the sum of octets of this messages ranges from zero to over 80 kByte/s.

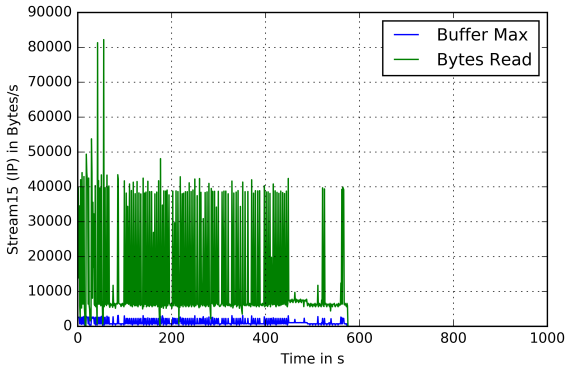

Figure 6. Load of the octet stream reserved for IP data during flight of MAIUS.

The blue curve in figure 6 shows the maximum number of bytes used from the telemetry buffers within the second of sampling. This is merely a measure of burst sizes to the octet buffer to the telemetry system. The maximum value for the buffer was 3130 Bytes. It shows that the telemetry system was filled with a quite high octet-rate, compared to the rate of reading octets from that buffer to the actual telemetry transfer frame.

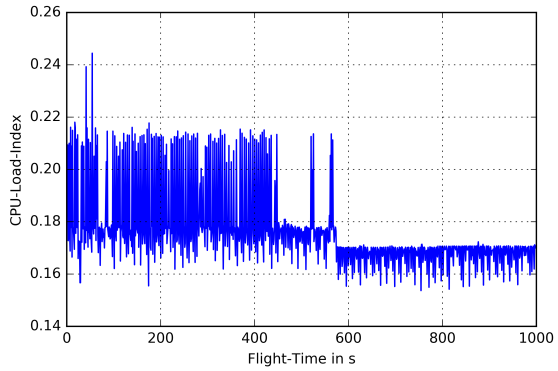

Figure 7. CPU Load Index on MAIUS during flight.

A descriptive plot of the load-index of the CPU is shown in figure 7. It shows the same period of time as in figure 6, to demonstrate the correlation of network load to CPU load. This index is derived from a counter running in the idle thread of the RODOS kernel. It's proportional to the CPU load; 100% of load results in an index value of 1. The index is computed once a second and is also part of the already mentioned housekeeping message.

Beside the handling of the LwIP stack and forwarding of data from the payload to the ground segment, the CPU was used for short, periodic tasks with high frequencies. As expected, this generates a base load to the processor with the additional load induced by network traffic on top. The system performed well during the flight of MAIUS, no data was lost and the CPU and buffer load was as expected.

5. CONCLUSION

Within this paper an architecture and an implementation of a set of routers was shown to forward IP data on up- and downlinks on sounding rockets or balloons. This provides a solution to replace serial RS422 connections with Ethernet networks to provide higher data rates and more flexibility to the users.

The architecture fits easily to existing octet-streaming TM/TC systems. It has been used successfully for the MAIUS mission and will be used also for future projects.

REFERENCES

[1] Dietmar Lochmann. *Digitale Nachrichtentechnik*. Verlag Technik, 3. auflage edition, 2002.

[2] Jon Postel. Internet protocol. Technical report, University of Southern California, 1981. `https://tools.ietf.org/html/rfc791`.

[3] Markus Wittkamp and Anderson Cattelan Zigiotto. A very high performance multi purpose computing card for tm/tc and control systems. *European Test and Telemetry Conference (ETTC'09), Toulouse*, 6 2009.

[4] Adam Dunkels and Leon Woestenberg. Lightweight ip stack. `http://www.nongnu.org/lwip/2_0_x/index.html`.

[5] S. Montenegro and F. Dannemann. RODOS - Real Time Kernel Design for Dependability. In *DASIA 2009 - Data Systems in Aerospace*, volume 669 of *ESA Special Publication*, page 66, May 2009.

[6] Maxim Krasnyansky. Universal tun/tap device driver, 2000. Part of the linux documentation at kernel.org `https://www.kernel.org/doc/Documentation/networking/tuntap.txt`.

[7] Jon Postel. Internet control message protocol. Technical report, University of Southern California, 1981. `https://tools.ietf.org/html/rfc792`.